In App Purchaseプログラミングガイド

Developer

目次

図、表

はじめに

In-App Purchaseを使うと、アプリケーション内に直接販売機能（Store）を埋め込むことができます。In-App Purchaseは、Store Kitフレームワークを使用してアプリケーションに実装します。Store Kitは、アプリケーションに代わってApp Storeに接続し、ユーザからのペイメント（支払い）を安全に処理します。Store Kitは、ユーザにペイメントの承認を求めた後、ユーザが購入したアイテムを提供できるようにアプリケーションに通知します。このアプリケーション内の支払い機能を使用して、アプリケーションの拡張機能や追加コンテンツに対するペイメントを受け取ることができます。

たとえば、In-App Purchaseを使用して次のようなシナリオを実装できます。

- 基本バージョンにプレミアム機能が追加されたアプリケーション
- ユーザが新しい本を購入してダウンロードできるブックリーダーアプリケーション
- 新しい環境（レベル）の購入を提案するロールプレイングゲーム
- プレーヤーが仮想の不動産を購入できるオンラインゲーム

Important　In-App Purchaseが行うのはペイメントの受け取りのみです。デベロッパは、組み込み機能のロック解除や独自サーバからのコンテンツのダウンロードなど、追加の機能をすべて提供する必要があります。このドキュメントでは、アプリケーションへStoreを追加するための技術的な要件について詳しく説明します。In-App Purchaseを使用する際のビジネス要件については、App Store Resource Centerを参照してください。また、販売できるプロダクトや、アプリケーションでそれらを提供するために必要な手順などについては、ライセンス契約にある明確な取扱規定を確認する必要があります。

対象読者

このドキュメントは、アプリケーション内でユーザに対して有料の機能追加を提供することを目的とするデベロッパを対象としています。

この書類の構成

このドキュメントは、次の章で構成されています。

- “In-App Purchaseの概要”（7 ページ）では、In-App Purchaseが提供する機能について説明します。
- “プロダクト情報の取得”（14 ページ）では、アプリケーションが提供するプロダクトについての情報をApp Storeから取得する方法について説明します。
- “購入を処理する”（17 ページ）では、App Storeからのペイメントをアプリケーションで要求する方法について説明します。
- “アプリケーションへのStoreの追加”（20 ページ）では、アプリケーションへStoreを追加する手順を説明します。
- “Storeレシートの確認”（26 ページ）では、App Storeから受領したレシートをサーバで確認する方法を説明します。
- “Storeのテスト”（29 ページ）では、サンドボックス環境を使用してアプリケーションをテストする方法を説明します。
- “自動更新購読（Auto-Renewable Subscriptions）”（31 ページ）では、アプリケーションがApple Storeを使用して更新処理を管理することで、定期購読を実装する方法を説明します。

関連項目

App Store Resource Centerでは、ビジネス面におけるIn-App Purchaseの使用と、アプリケーションでプロダクトを販売するために必要なステップについて説明しています。

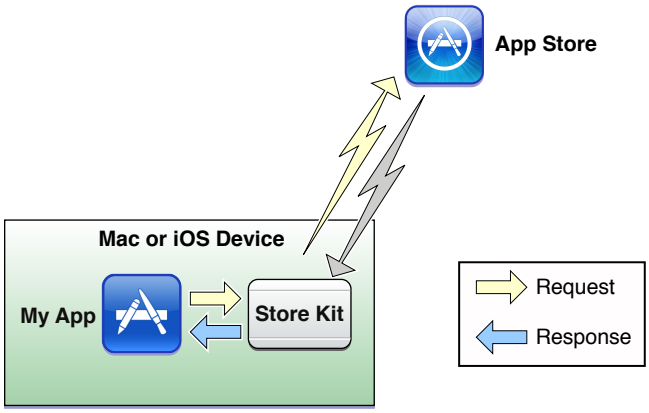
『*iTunes Connect Developer Guide*』では、App Storeでのプロダクトおよびアカウントテストの設定方法について説明しています。

『*Store Kit Framework Reference*』では、App Storeとやり取りするためのAPIを説明しています。

In-App Purchaseの概要

Store Kitは、アプリケーションに代わってApp Storeとやり取りします。アプリケーションはStore Kitを使用して、アプリケーションで提供するプロダクトについてのローカライズされた情報をApp Storeから受け取ります。アプリケーションはこの情報をユーザに表示し、ユーザがアイテムを購入できるようにします。ユーザがアイテムを購入すると、アプリケーションはStore Kitを使用してユーザからペイメントを受け取ります。図 1-1に基本的なStoreモデルを示します。

図 1-1 App Storeモデル



Store Kit APIは、アプリケーションにStoreを追加する処理の一部分にすぎません。デベロッパは、提供するプロダクトをどのように追跡するか、アプリケーションでユーザにStoreをどのように示すか、Storeでユーザが購入したプロダクトをアプリケーションでどのように提供するかを決定する必要があります。以降この章では、プロダクトを作成し、アプリケーションへStoreを追加するプロセスについて説明します。

プロダクト

アプリケーションのStoreで販売する機能はすべて**プロダクト**です。プロダクトは、デベロッパが新規アプリケーションを作成するのと同じ方法で、iTunes Connectを通してApp Storeに関連付けられます。In-App Purchaseを使用して販売できるプロダクトは、次の4つがサポートされています。

- コンテンツ（Content）。アプリケーションで提供できる電子書籍、雑誌、写真、アート作品、ゲームのレベルと登場人物、その他のデジタルコンテンツを含みます。

- *機能*プロダクト（Functionality Product）。アプリケーションですでに提供済みの機能をロック解除したり、拡張したりします。たとえば、あるゲームを出荷する際に、ユーザが購入できる、より小さな単位のゲームを複数同梱して出荷することが考えられます。
- *サービス*（Service）。たとえば、音声のテキスト化などの一度限りのサービスをユーザに請求できます。1回のサービス利用につき、個々の購入が発生します。
- *定期購読*（Subscription）。 コンテンツやサービスを継続的に利用できるようにします。たとえば、アプリケーションから金融情報やオンラインゲームのポータルへのアクセスを1ヶ月単位で提供するなどが考えられます。

In-App Purchaseは、プロダクトを作り出すための汎用のメカニズムを提供しますが、プロダクトの実装方法の詳細はデベロッパに委ねています。ただし、アプリケーションの設計においては、以下に示すように留意すべき重要なガイドラインがいくつかあります。

- アプリケーションの中で提供できるのは電子商品または電子サービスに限られています。In-App Purchaseを使って実物の商品やサービスを販売することはできません。
- 中間通貨のようなアイテムを提供することはできません。これは、特定の商品やサービスを購入するということがユーザに認識されることが重要であるためです。
- 販売用に提供するアイテムには、ポルノ、誹謗、中傷、ギャンブル（ギャンブルのシミュレーションであれば問題ありません）や、それらに関連するようなものを含めないようにします。

In-App Purchaseを使用して提供できるものの詳細については、ライセンス契約を参照してください。

App Storeへのプロダクトの登録

Storeで提供するプロダクトはすべて、最初に、iTunes Connectを通してApp Storeに登録する必要があります。プロダクトを登録する際は、プロダクト名、説明、価格とともに、App Storeとアプリケーションで使用するその他のメタデータを提供します。

特定のプロダクトは、**プロダクトID**と呼ばれる一意の文字列を使用して識別します。Store Kitを使用してApp Storeとやり取りする際に、アプリケーションはプロダクトIDを使用して、デベロッパがプロダクトに指定した設定データを取得します。その後、ユーザがプロダクトを購入する際に、アプリケーションはプロダクトIDを使用して購入されるプロダクトを識別します。

App Storeはさまざまな種類のプロダクトに対応しています。

- **Consumable（消耗型）**プロダクト。必要なときにユーザが毎回購入する必要があるプロダクトです。たとえば、一度限りのサービスは一般的に消耗型プロダクトとして実装します。

- **Non-consumable（非消耗型）**プロダクト。特定ユーザが一度だけ購入するプロダクトです。非消耗型プロダクトは、一度購入すると購入したユーザのiTunesアカウントに関連付けられたすべてのデバイスに提供されます。Store Kitには、非消耗型プロダクトを複数のデバイスでリストア（復元）するためのサポート機能が含まれています。
- **Auto-renewable subscriptions（自動更新購読）**。非消耗型プロダクトと同様の方法でユーザのすべてのデバイスに送付されます。しかし、自動更新購読はほかの点で異なっています。iTunes Connectで自動更新購読を作成した場合、購読の期間を選択します。App Storeは、期限が切れるたびに自動的に定期購読を更新します。ユーザが定期購読の更新を許可しないを選択すると、定期購読の有効期限が切れた後、ユーザの定期購読へのアクセスは取り消されます。アプリケーションは、定期購読が現在アクティブで、最新のトランザクションに対する更新されたレシートも受け取れることを検証する責任があります。
- **Free subscriptions（無料購読）**。無料購読できるプロダクトをNewsstandに置きます。サインアップしたユーザは、Apple IDに関連付けられたどのデバイスからでも購読できます。期限切れになることはありません。また、購読にはNewsstand対応アプリケーションが必要です。
- **Non-renewing subscriptions（非更新購読）**。一定の期間だけ提供するプロダクトを作成するための仕組みです。非更新購読には、自動更新購読といくつか主要な違いがあります。
    - 定期購読の期間は、iTunes Connectでプロダクトを作成するときに宣言されていません。この情報をユーザに提供することは、アプリケーションの責任です。ほとんどの場合、プロダクトの説明に購読期間を含めます。
    - 非更新購読は、複数回購入（消耗型プロダクトのように）される可能性があり、App Storeが自動的に更新することはありません。必要ならば、更新処理はアプリケーション内に別途実装する必要があります。特に、定期購読の有効期限を確認し、プロダクトの購入を促進するようにアプリケーションで行う必要があります。
    - ユーザが保持するすべてのデバイスに非更新購読を送る必要があります。非更新購読を、Store Kitがすべてのデバイスに自動的に同期することはありません。このインフラストラクチャを自分で実装する必要があります。たとえば、ほとんどの定期購読は外部のサーバから提供されます。サーバはユーザを識別するメカニズムを実装して、定期購読とこれを購入しているユーザを関連付ける必要があります。

App Storeへのプロダクトの登録方法の詳細については、『*iTunes Connect Developer Guide*』を参照してください。

機能の配布

ユーザにプロダクトを提供するためにアプリケーションで使用する配布メカニズムは、アプリケーションの設計と実装に大きく影響します。プロダクトの配布に使われる、デベロッパが想定すべき基本的な2つの方法として、組み込みモデルとサーバモデルがあります。どちらのモデルでも、デベロッパはStoreで提供したプロダクト群を追跡し、ユーザによって問題なく購入されたプロダクトを配布します。

組み込みプロダクトモデル

組み込みプロダクトモデルでは、プロダクトの配布に必要なすべてをアプリケーションに組み込みます。このモデルは、アプリケーション内の機能をロック解除するためにもっともよく使われます。アプリケーションのバンドル内で提供したコンテンツを配布するために、このモデルを使用することもできます。このモデルの重要な利点は、プロダクトをアプリケーションからすぐにユーザに配布できる点です。組み込みプロダクトのほとんどは非消耗型にするべきです。

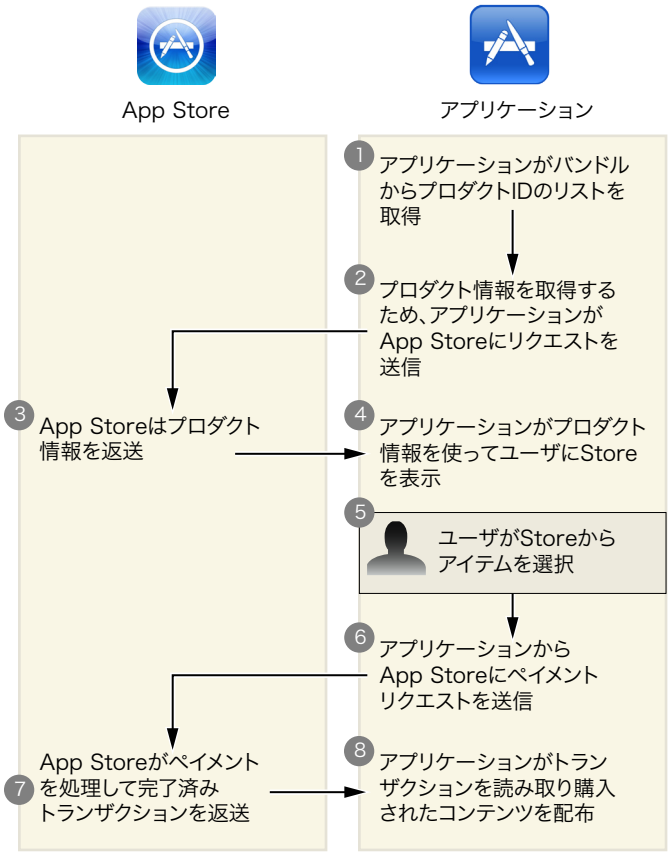
Important　In-App Purchaseは、問題なく購入された後のアプリケーションにパッチを適用する機能を提供しません。プロダクトでアプリケーションのバンドルを変更する必要がある場合、App Storeにアプリケーションの更新版を配布しなければなりません。

プロダクトを識別するために、アプリケーションではアプリケーションバンドルにプロダクトIDを格納します。Appleは、プロパティリスト（plist）を使用して組み込み機能のプロダクトIDを追跡することをお勧めします。コンテンツ駆動型のアプリケーションでは、これを使用してアプリケーションのソースを変更せずに新しいコンテンツを追加できます。

あるプロダクトが問題なく購入された後は、アプリケーションでは機能のロック解除を行い、それをユーザに配布しなければなりません。機能をロック解除するもっとも簡単な方法は、アプリケーションの環境設定を変更することです。“App-Related Resources”を参照してください。アプリケーションの環境設定は、ユーザがiOSベースのデバイスをバックアップするときにバックアップされます。購入したものが失われないようにアイテムの購入後にデバイスをバックアップするよう、ユーザに勧めるとよいでしょう。

図 1-2に、アプリケーションが組み込みのプロダクトを配布する一連の動作を示します。

図 1-2 組み込みプロダクトの配布

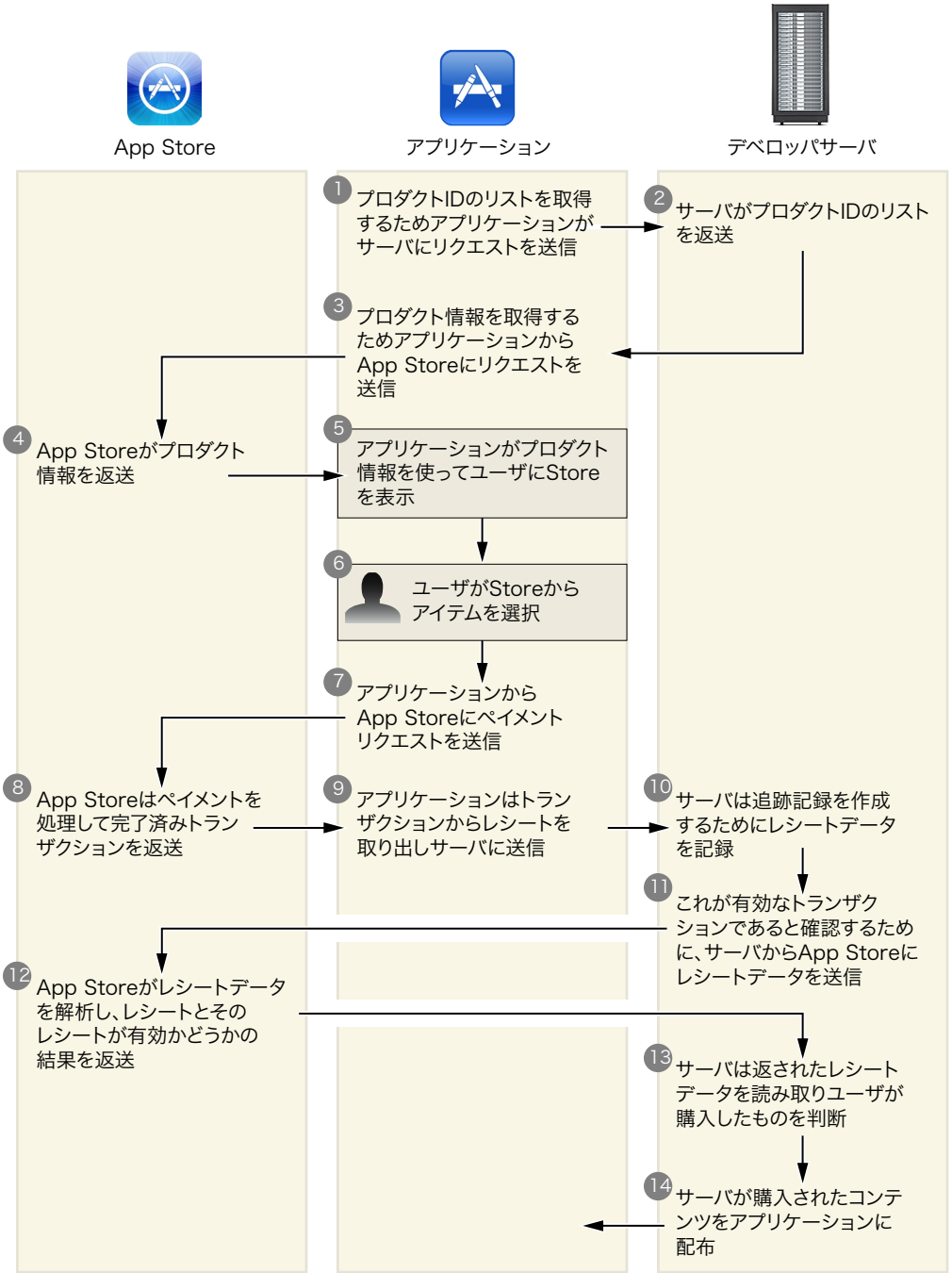


サーバプロダクトモデル

サーバプロダクトモデルでは、アプリケーションにプロダクトを配布するサーバをデベロッパが別途用意します。サーバによる配布は、アプリケーションバンドルを変更せずにデータとしてプロダクトを配布できるため、定期購読、サービス、およびコンテンツに適しています。たとえば、ゲームでは新しいプレイ環境（パズルやレベルなど）をアプリケーションに配布することができます。Store Kitはサーバの設計や、アプリケーションとのやり取りについては定義していません。*アプリケーションとデベロッパのサーバ間のすべてのやり取りについての責任はデベロッパにあります。*加えて、Store Kitは特定のユーザを識別するメカニズムを提供していません。設計によっては、ユーザを識別するメカニズムをデベロッパが提供する必要があります。アプリケーションでこうしたメカニズム（たとえば、特定のユーザにどの定期購読が関連付けられているかを追跡するなど）を必要とする場合、デベロッパ自身でこれを設計し、実装する必要があります。

図 1-3に組み込みモデルを展開して、サーバとのやり取りを示します。

図 1-3 サーバプロダクトの配布



Appleは、プロパティリストにプロパティIDを含めるよりサーバからそれを取得することをお勧めします。こうすることで、アプリケーションを更新せずに新しいプロダクトを柔軟に追加できるようになります。

サーバモデルでは、アプリケーションは、トランザクションに関連付けられた署名付きレシートを取得し、それをサーバに送信します。サーバはそこでレシートの有効性確認とデコードを行い、どのコンテンツをアプリケーションに配布するかを判断できます。この処理については、"Storeレシートの確認"（26 ページ）で詳しく説明します。

サーバモデルには、セキュリティと信頼性という懸念事項が伴います。セキュリティ上の脅威がないか環境全体をテストする必要があります。『*Secure Coding Guide*』で追加の推奨事項を説明しています。

非消耗型プロダクトはStore Kitの組み込み機能を使ってリストアできますが、非更新購読型はサーバからリストアしなければなりません。非更新購読型についての情報を記録し、ユーザにそれらをリストアするのはデベロッパの責任です。任意で、消耗型プロダクトをサーバから追跡することもできます。たとえば、消耗型プロダクトがサーバによって提供されるサービスの場合、リクエストの結果をユーザが複数のデバイス上で取得できるようにすることもできます。

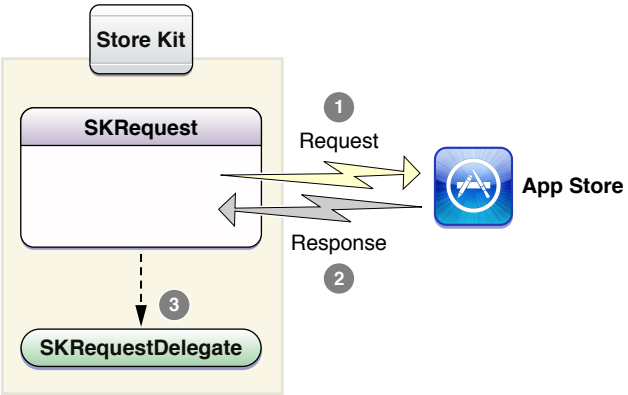
プロダクト情報の取得

アプリケーションでユーザにStoreを表示する準備ができたら、ユーザインターフェイスにApp Storeからの情報を埋める必要があります。この章では、App Storeからのプロダクトの詳細を要求する方法を説明します。

App Storeへのリクエストの送信

Store Kitは、App Storeから情報を要求する一般的なメカニズムを提供しています。アプリケーションはリクエストオブジェクトを作成して初期化し、デリゲートをアタッチしてリクエストを開始します。リクエストを開始するとリクエストがApp Storeに送信され、処理が行われます。App Storeがリクエストを処理すると、リクエストのデリゲートが非同期で呼び出されてアプリケーションへその結果を返します。図 2-1にリクエストモデルを示します。

図 2-1 Store Kitリクエストモデル



リクエストの処理中にアプリケーションが終了した場合、アプリケーションはそれを再送する必要があります。

SKRequest

`SKRequest`はStoreに送信されたリクエストの抽象ベースクラスです。

SKRequestDelegate

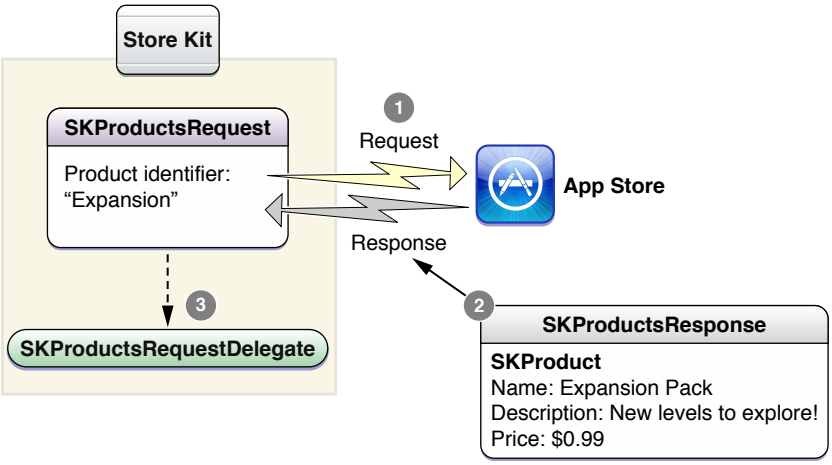
`SKRequestDelegate`は、正常に終了したリクエストと、エラーによって失敗したリクエストを処理するためにアプリケーションが実装するプロトコルです。

プロダクトに関する情報のリクエスト

アプリケーションは、**プロダクトリクエスト**を使用してプロダクトに関するローカライズされた情報を取得します。アプリケーションは、一連のプロダクトID文字列を含むリクエストを作成します。前述したように、アプリケーションではプロダクトIDをアプリケーション内に組み込むか、外部サーバからIDを取得します。

プロダクトリクエストを開始すると、プロダクトID文字列はApp Storeに送信されます。App Storeは、デベロッパが事前にiTunes Connectに入力したローカライズされた情報を返します。デベロッパはこれらの情報を使ってStoreのユーザインターフェイスを埋めます。図 2-2にプロダクトリクエストを示します。

図 2-2　ローカライズされたプロダクト情報のリクエスト



Important　ユーザに対してプロダクトの購入を許可する前に、特定のプロダクトID向けのプロダクトリクエストを行わなければなりません。App Storeからプロダクト情報を取得することで、iTunes Connectで販売できるとマークしたプロダクトに対する有効なプロダクトIDを使用していることが保証されます。

SKProductsRequest

`SKProductsRequest`オブジェクトは、Storeで表示するプロダクトのプロダクトID文字列のリストと共に作成されます。

SKProductsRequestDelegate

`SKProductsRequestDelegate`プロトコルは、アプリケーション内のオブジェクトによって実装され、Storeからのレスポンスを受信します。このプロトコルは、リクエストが正常に処理された場合に非同期にレスポンスを受け取ります。

SKProductsResponse

`SKProductsResponse`オブジェクトは、オリジナルのリクエスト内の有効なプロダクトIDごとに1つの`SKProduct`を持ち、Storeが認識できなかったプロダクトIDのリストも持ちます。Storeは、次のようないくつかの理由でIDを認識できない場合があります。たとえば、プロダクトIDのスペルが間違っている、非売品としてマークされている、あるいはiTunes Connectでデベロッパが行った変更がすべてのApp Storeサーバに伝えられていないなどの理由があります。

SKProduct

`SKProduct`オブジェクトは、App Storeに登録したプロダクトについてローカライズされた情報を提供します。

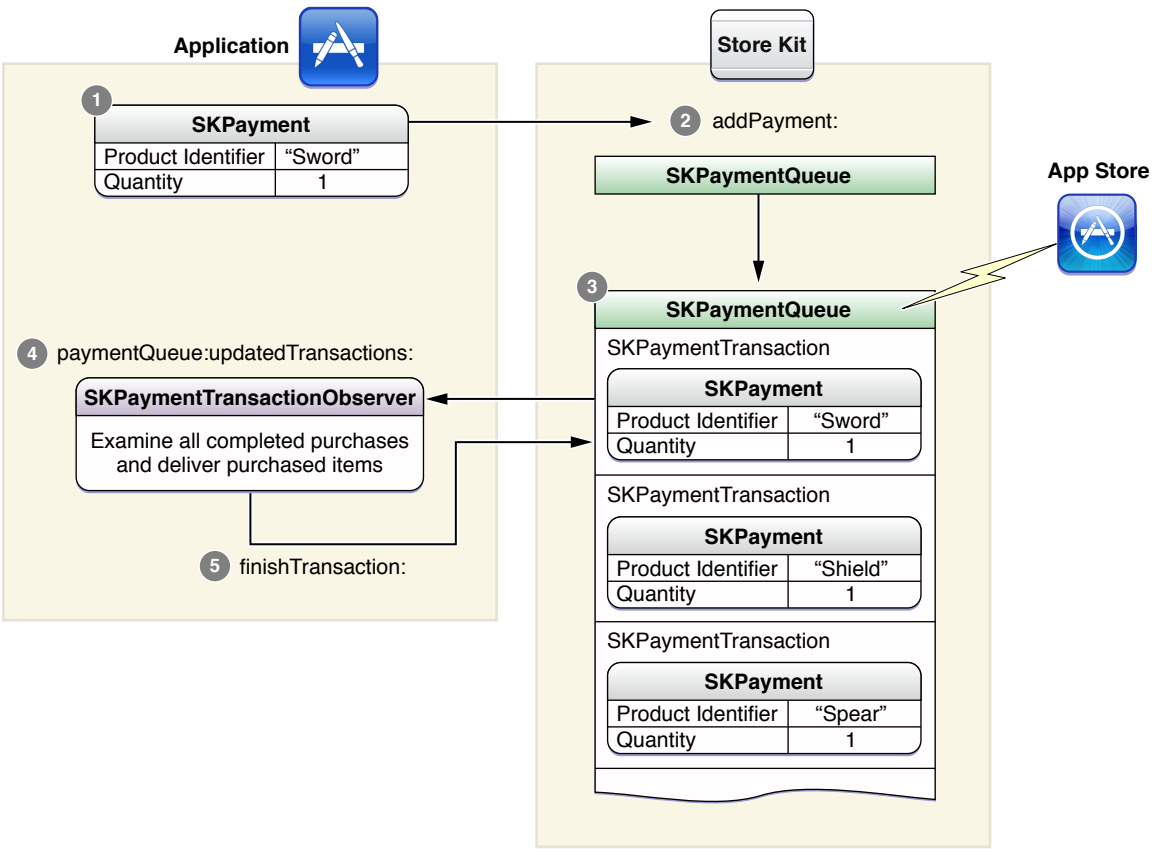
購入を処理する

ユーザがアイテムを購入する準備が整うと、アプリケーションはApp Storeにペイメントの受け取りを依頼します。アプリケーションからペイメントの依頼が発生すると、App Storeは持続的なトランザクションを作成し、ユーザがアプリケーションを終了して再起動したとしてもペイメント処理を続けます。App Storeは処理中のトランザクションのリストをアプリケーションと同期し、これらのトランザクションのいずれかの状態が変化したらアプリケーションに最新情報を送信します。

ペイメントの受け取り

ペイメントを受け取るには、図 3-1に示すように、アプリケーションでペイメントオブジェクトを作成し、オブジェクトをペイメントキューに入れます。

図 3-1 ペイメントリクエストのキューへの追加



ペイメントがペイメントキューへ追加されると、持続的なトランザクションが作成され保持されます。ペイメント処理が完了すると、ペイメントの受け取りに関する情報でトランザクションが更新されます。アプリケーションは、トランザクションが更新された際にメッセージを受け取るオブザーバを実装します。オブザーバは、購入されたアイテムをユーザに提供し、ペイメントキューからトランザクションを削除する必要があります。

SKPayment

ペイメントの受け取りはペイメントオブジェクトで開始します。ペイメントオブジェクトにはプロダクトIDと、必要に応じて、購入されるプロダクトの数量が含まれます。同一のペイメントを2回以上キューに入れることもでき、ペイメントオブジェクトをキューに入れるたびにそれぞれがペイメントの別個のリクエストになります。

ユーザは、購入機能を「設定(Settings)」アプリケーションで無効にできます。購入処理をキューに入れる前に、アプリケーションでまずペイメント処理を続けて良いか確認すべきです。これは、ペイメントキューの`canMakePayments`メソッドを呼び出して行います。

SKPaymentQueue

ペイメントキューはApp Storeとの通信に使用します。ペイメントがキューに追加されると、Store KitはApp Storeへリクエストを送信します。Store Kitは、ユーザにペイメントを承認するように求めるダイアログを表示します。完了したトランザクションはアプリケーションのオブザーバに返されます。

SKPaymentTransaction

トランザクションはキューに追加されたすべてのペイメントごとに作成されます。それぞれのトランザクションは複数のプロパティを持ち、トランザクションの状態をアプリケーションが判断できるようになっています。ペイメントを受け取ると、トランザクションには正常に終了したトランザクションについての詳細情報が追加されています。

アプリケーションでペイメントキューに対して、処理中のトランザクションのリストを求めることも可能ですが、ペイメントキューが更新されたトランザクションのリストをペイメントキューのオブザーバに通知するまで待つのが一般的です。

SKPaymentTransactionObserver

アプリケーションはオブジェクト上に`SKPaymentTransactionObserver`プロトコルを実装し、ペイメントキューにオブザーバとして追加します。オブザーバの主要な責任は、完了したトランザクションを検査し、正常に購入されたアイテムを引き渡し、これに関する一連のトランザクションをペイメントキューから削除することです。

アプリケーションは、ユーザがアイテムを購入しようとするまで待つのではなく、起動時にオブザーバをペイメントキューに関連付けるようにします。トランザクションはアプリケーションの終了時にも失われません。次のアプリケーション起動時に、Store Kitはトランザクションの処理を再開します。アプリケーションの初期化中にオブザーバを追加することで、すべてのトランザクションが確実にアプリケーションに返されます。

トランザクションのリストア

トランザクションが処理されキューから削除されると、通常は、アプリケーションに再度表示されることはありません。しかし、アプリケーションがリストア可能であるべきプロダクト型をサポートするならば、購入済みのプロダクトをリストアするためのインターフェイスを組み込む必要があります。ユーザはこのインターフェイスを使って、ほかのデバイスにプロダクトを追加したり、元のデバイスから消えてしまった場合そこにリストアしたりすることができます。

Store Kitは、非消耗型、自動更新購読型、無料購読型のプロダクト用に、トランザクションをリストアするための組み込み機能を提供します。トランザクションをリストアするには、アプリケーションでペイメントキューの`restoreCompletedTransactions`メソッドを呼び出します。トランザクションをリストアするため、ペイメントキューはApp Storeにリクエストを送信します。これに応じて、すでに完了したトランザクションごとに、App Storeが新しくリストアトランザクションを生成します。リストアトランザクションオブジェクトの`originalTransaction`プロパティは、オリジナルのトランザクションのコピーを保持しています。アプリケーションは、オリジナルトランザクションを取得して購入したコンテンツのロック解除にそれを使うことでリストアトランザクションを処理します。Store Kitは以前のトランザクションをすべてリストアした後、ペイメントキューオブザーバの`paymentQueueRestoreCompletedTransactionsFinished:`メソッドを呼び出してオブザーバに通知します。

リストア可能なプロダクトを（リストアインターフェイスを使わずに）ユーザが購入しようとすると、アプリケーションは、リストアトランザクションではなくそのアイテムの通常のトランザクションを受信します。ただし、そのプロダクトについてユーザに二重請求することはありません。アプリケーションでは、オリジナルトランザクションとまったく同じようにこれらのトランザクションを扱う必要があります。

非更新購読型および消耗型プロダクトは、Store Kitで自動的にリストアされません。しかし、非更新購読型プロダクトは、リストア可能でなければなりません。これらのプロダクトをリストアするには、購入された際にデベロッパ独自のサーバにトランザクションを記録し、ユーザのデバイスにこれらのトランザクションをリストアする独自メカニズムを提供しなければなりません。

アプリケーションへのStoreの追加

この章では、アプリケーションにStoreを追加する手順を説明します。

段階的手順

プロジェクトをセットアップするときは、必ず`StoreKit.framework`にリンクするようにしてください。その後、次のステップに従ってStoreを追加できます。

1. アプリケーションで配布するプロダクトを決めます。

   提供できる機能のタイプには制限があります。Store Kitでは、アプリケーション自身にパッチを適用したり、追加のコードをダウンロードしたりできません。プロダクトは、アプリケーション内の既存のコードで操作するか、リモートサーバから配布されるデータファイルを使って実装する必要があります。ソースコードの変更が必要な機能を追加する場合は、アプリケーションの更新されたバージョンを出荷する必要があります。

2. iTunes Connectでプロダクトごとにプロダクト情報を登録します。

   アプリケーションのStoreに新しいプロダクトを追加するたびに、再度このステップに戻ります。すべてのプロダクトには一意のプロダクトID文字列が必要です。App Storeはこの文字列を使用して、プロダクトの情報検索およびペイメント処理を行います。プロダクトIDは、iTunes Connectアカウントに固有のもので、アプリケーションを登録したときと同じような手順でiTunes Connectを使用して登録します。

   プロダクト情報の作成および登録処理については、『*iTunes Connect Developer Guide*』で説明しています。

3. ペイメント処理を進めてよいか判断します。

   ユーザはアプリケーション内で購入機能を無効にできます。アプリケーションでは、新しいペイメントリクエストをキューに入れる前に、ペイメントによる購入が可能かを確認する必要があります。この確認は、Storeをユーザに表示する前に行うのが良いでしょう（以下に示します）。あるいは、ユーザが実際にアイテムを購入しようとするときに行うこともできます。後者の場合、ユーザは、ペイメントを有効化させたときに、購入できるアイテムを確認できます。

```
if ([SKPaymentQueue canMakePayments]) {
    // ユーザにStoreを表示する。
```

```
} else {
    // 購入できないことをユーザに警告する。
}
```

4. プロダクトに関する情報を取得します。

   アプリケーションはSKProductsRequestオブジェクトを作成して、販売するアイテムの一連のプロダクトIDでそれを初期化し、リクエストにデリゲートを添付してリクエストを開始します。レスポンスには、すべての有効なプロダクトID用にローカライズされたプロダクト情報が保持されます。

   レスポンスから取得した、有効なプロダクトのリストを保持しておきます。ユーザが購入するプロダクトを選択する際には、ペイメントリクエストを作成するためのプロダクトオブジェクトが必要になります。

```
- (void) requestProductData
{
    SKProductsRequest *request= [[SKProductsRequest alloc]
initWithProductIdentifiers:
            [NSSet setWithObject: kMyFeatureIdentifier]];
    request.delegate = self;
    [request start];
}
- (void)productsRequest:(SKProductsRequest *)request
didReceiveResponse:(SKProductsResponse *)response
{
    NSArray *myProducts = response.products;
    // プロダクトリストをもとにUIを構築する。
    // プロダクトリストの参照を保存する。
}
```

5. ユーザに表示するプロダクトのユーザインターフェイスを追加します。

   Store Kitはユーザインターフェイスクラスを提供していません。プロダクトを顧客にどのように提供するかのルックアンドフィールを決めるのはデベロッパです。

6. トランザクションオブザーバをペイメントキューに登録します。

   アプリケーションでトランザクションオブザーバをインスタンス化し、ペイメントキューのオブザーバとして追加します。

```
MyStoreObserver *observer = [[MyStoreObserver alloc] init];
[[SKPaymentQueue defaultQueue] addTransactionObserver:observer];
```

アプリケーションでは、アプリケーションの起動時にオブザーバを追加する必要があります。App Storeはアプリケーションがすべてのトランザクションの完了前に終了しても、キューに入れられたトランザクションを記憶しています。初期化中にオブザーバを追加することで、以前キューに入れられたすべてのトランザクションをアプリケーションで確実に認識することができます。

7. `MyStoreObserver`に`paymentQueue:updatedTransactions:`メソッドを実装します。

   新しいトランザクションが作成されたり更新されたりするたびに、オブザーバの`paymentQueue:updatedTransactions:`メソッドが呼び出されます。

```
- (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;
        }
    }
}
```

8. オブザーバは、ユーザが正常にアイテムを購入したときにプロダクトを提供します。

```
- (void) completeTransaction: (SKPaymentTransaction *)transaction
```

```
{
    // アプリケーションではこれらの2つのメソッドを実装する必要がある
    [self recordTransaction:transaction];
    [self provideContent:transaction.payment.productIdentifier];

    // ペイメントキューからトランザクションを削除する
    [[SKPaymentQueue defaultQueue] finishTransaction: transaction];
}
```

正常なトランザクションには`transactionIdentifier`プロパティ、および処理済みのペイメントの詳細が記録された`transactionReceipt`プロパティが含まれます。この情報に対してアプリケーションがしなければならないことは特にありません。トランザクションの追跡記録を作成するためにこの情報を記録しておく必要があるかも知れません。アプリケーションがサーバを使用してコンテンツを配布する場合、レシートをサーバに送信してApp Storeで有効性を確認することができます。

ユーザが購入したプロダクトを提供するために必要なすべての手段を、アプリケーションは確実に実行しなければなりません。ペイメントをすでに支払っているため、ユーザは新しく購入したプロダクトを受け取ることを期待します。この実装方法の推奨事項については、“機能の配布” （10 ページ）を参照してください。

プロダクトを配布したら、アプリケーションは`finishTransaction:`を呼び出してトランザクションを完了する必要があります。`finishTransaction:`を呼び出すと、トランザクションはペイメントキューから削除されます。プロダクトをなくさないように、アプリケーションでは`finishTransaction:`を呼び出す前にプロダクトを配布する必要があります。

9. リストアされた購入のトランザクションを終了させます。

```
- (void) restoreTransaction: (SKPaymentTransaction *)transaction
{
    [self recordTransaction: transaction];
    [self provideContent:
transaction.originalTransaction.payment.productIdentifier];
    [[SKPaymentQueue defaultQueue] finishTransaction: transaction];
}
```

このルーチンは購入されたアイテムに対するルーチンと似ています。リストアされた購入は、異なるトランザクションIDとレシートなど、新規のトランザクション情報を提供します。必要に応じて任意の追跡記録の一部としてこの情報を別に保存できます。ただし、トランザクションを完了する時点では、実際のペイメントオブジェクトを保持しているオリジナルのトランザクションを回復し、そのプロダクトIDを使用する方がよいかも知れません。

10. 失敗した購入のトランザクションを終了させます。

```
- (void) failedTransaction: (SKPaymentTransaction *)transaction
{
    if (transaction.error.code != SKErrorPaymentCancelled) {
        // 必要に応じてここでエラーを表示する
    }
    [[SKPaymentQueue defaultQueue] finishTransaction: transaction];
}
```

通常、トランザクションが失敗するのはユーザがアイテムの購入を取り止めたからです。アプリケーションは、失敗したトランザクションの`error`フィールドを読み取って、なぜトランザクションが失敗したのかを正確に把握できます。

失敗した購入に対する唯一の要件は、アプリケーションでそれをキューから削除することです。アプリケーションでユーザにエラーを表示するダイアログを出す場合、ユーザが購入をキャンセルした場合にエラーを表示しないようにする必要があります。

11. すべてのインフラストラクチャを設定したら、ユーザインターフェイスを終了できます。ユーザがStore内のアイテムを選択したら、ペイメントオブジェクトを作成し、ペイメントキューに追加します。

```
SKProduct *selectedProduct = <#from the products response list#>;
SKPayment *payment = [SKPayment paymentWithProduct:selectedProduct];
[[SKPaymentQueue defaultQueue] addPayment:payment];
```

Storeで2つ以上のプロダクトを購入できるようにする場合は、ペイメントを1つ作成し、数量プロパティを設定することができます。

```
SKProduct *selectedProduct = <#from the products response list#>;
SKPayment *payment = [SKPayment paymentWithProduct:selectedProduct];
payment.quantity = 3;
[[SKPaymentQueue defaultQueue] addPayment:payment];
```

次に学ぶこと

これらの手順で示したコードは、アプリケーションに組み込むプロダクトモデルでの使用に適しています。サーバを使用してコンテンツを配布するアプリケーションの場合、アプリケーションとサーバ間のやり取りに使用するプロトコルの設計と実装の責任はデベロッパにあります。また、サーバではアプリケーションにプロダクトを配布する前にレシートを確認する必要もあります。

Storeレシートの確認

アプリケーションでは、Store Kitから受領したレシートがAppleからきたものであることを確認する追加ステップを実行する必要があります。これは、アプリケーションが定期購読、サービス、またはダウンロード可能なコンテンツの提供を別々のサーバに頼っている場合は特に重要です。自分のサーバ上でレシートを確認することで、アプリケーションからのリクエストが有効なものであることを確認できます。

Important iOSでは、Storeレシートの内容およびフォーマットは非公開であり、変更される場合があります。アプリケーションで直接レシートのデータを分析しようとしないでください。ここで説明している仕組みを使用して、レシートの有効性を確認し、そこに保存されている情報を取得するようにしてください。

Mac OS Xでは、ストアレシートの内容や書式は『*Validating Mac App Store Receipts*』に記述されています。Mac OS Xは、この章で説明するサーバ検証メソッドと、『*Validating Mac App Store Receipts*』に記載されているローカル検証メソッドの、両方に対応しています。

App Storeを使用したレシートの確認

Store Kitが、ペイメントキューオブザーバに対して完了済みの購入を返した場合、トランザクションの`transactionReceipt`プロパティにはトランザクションにとって重要な情報がすべて記録された署名済みのレシートが含まれています。サーバからApp Storeにこのレシートを送信し、レシートが有効なものであり、かつ改ざんされていないことを確認できます。App Storeに直接送信されたクエリは、RFC 4627に定義されているように、JSON辞書として送受信されます。

レシートを確認するには、次の手順を実行します。

1. トランザクションの`transactionReceipt`プロパティ（iOSの場合）、またはアプリケーションバンドル内のレシートファイル（Mac OS Xの場合）からレシートデータを取得し、base64エンコードを使用してそれをエンコードします。
2. `receipt-data`という名前の1つのキーと、手順1で作成した文字列でJSONオブジェクトを作成します。JSONコードは次のようになります。

```
{
    "receipt-data" : "(receipt bytes here)"
```

```
}
```

3. HTTPのPOSTリクエストを使用してJSONオブジェクトをApp Storeに送信します。StoreのURLは`https://buy.itunes.apple.com/verifyReceipt`です。
4. App Storeから受信するレスポンスは、`status`と`receipt`の2つのキーからなるJSONオブジェクトです。次のようになるはずです。

```
{
    "status" : 0,
    "receipt" : { (receipt here) }
}
```

`status`キーの値が`0`であれば、レシートは有効です。値が`0`以外であれば、レシートは無効です。

Storeレシート

App Storeに送信したレシートは、トランザクションの情報をエンコードします。App Storeがレシートの有効性を確認する際、レシートデータに格納されているデータはデコードされて、レスポンスの`receipt`キーに返されます。レシートレスポンスは、アプリケーションの`SKPaymentTransaction`に返されるすべての情報が含まれているJSON辞書です。サーバはこのフィールドを使用して購入に関する詳細を取得できます。Appleは、レシートデータのみをサーバに送信し、購入情報の詳細はレシートの有効性確認によって取得することをお勧めします。App Storeがレシートのデータが改ざんされていないことを確認するため、サーバにレシートデータとトランザクションデータの両方を送信するより、レスポンスから情報を取得するほうがより安全です。

表 5-1に、購入に関する情報を取得するために使用するキーのリストを示します。これらのキーの多くは`SKPaymentTransaction`クラスのプロパティと一致します。表 5-1に指定されていないキーは、すべてAppleにより予約されています。

注意 キーのいくつかは、アプリケーションがApp Storeに接続しているのか、サンドボックステスト環境に接続しているのかによって変化します。サンドボックスの詳細については、"Storeのテスト"（29 ページ）を参照してください。

表 5-1 購入情報のキー

キー	解説
quantity	購入したアイテム数。この値は、トランザクションのquantityプロパティに保存されているSKPaymentオブジェクトのpaymentプロパティに対応します。
product_id	購入したアイテムのプロダクトID。この値は、トランザクションのproductIdentifierプロパティに保存されているSKPaymentオブジェクトのpaymentプロパティに対応します。
transaction_id	購入したアイテムのトランザクションID。この値は、トランザクションのtransactionIdentifierプロパティに対応します。
purchase_date	トランザクションの発生した日時。この値は、トランザクションのtransactionDateプロパティに対応します。
original_transaction_id	以前のトランザクションをリストアするトランザクションについては、これが元のトランザクションIDを保持します。
original_purchase_date	以前のトランザクションをリストアするトランザクションについては、これが元の購入日を保持します。
app_item_id	ペイメントトランザクションを作成したアプリケーションを一意に識別するために、App Storeが使用する文字列。サーバが複数のアプリケーションをサポートする場合、この値を使用してそれらを区別できます。サンドボックス内で実行しているアプリケーションはまだapp-item-idを割り当てられていないため、サンドボックスで作成されるレシートにはこのキーは存在しません。
version_external_identifier	アプリケーションの改訂を一意に識別する任意の番号。サンドボックスで作成されたレシートにはこのキーは存在しません。
bid	アプリケーション用のバンドルID。
bvrs	アプリケーションのバージョン番号。

Storeのテスト

開発中、アプリケーションをテストして購入が正しく動作することを確認する必要があります。しかし、アプリケーションのテスト中に、ユーザに請求したくはありません。Appleが提供するサンドボックス環境を利用すると、会計処理を発生させることなくアプリケーションのテストが行えます。

> **注意** Store KitはiOSシミュレータ上では動作しません。iOSシミュレータ上でアプリケーションを実行していると、Store Kitは、アプリケーションがペイメントキューを取り出そうとすると警告をログに記録します。Storeのテストは実際のデバイス上で行う必要があります。

サンドボックス環境

Xcodeからアプリケーションを起動すると、Store KitはApp Storeに接続しません。代わりに、サンドボックスの特別なStore環境に接続します。サンドボックス環境はApp Storeのインフラストラクチャを使用しますが、実際のペイメントは処理しません。サンドボックス環境は、ペイメントが正常に処理されたかのようにトランザクションを返します。サンドボックス環境は、In-App Purchaseのテストに制限されたiTunes Connectの特別なアカウントを使用します。サンドボックス内でのStoreのテストに通常のiTunes Connectアカウントを使用することはできません。

アプリケーションをテストするには、iTunes Connectに特別なテストアカウントを1つ以上作成します。アプリケーションがローカライズされる地域ごとに、最低1つのテストアカウントを作成する必要があります。テストアカウントの作成の詳細については、『*iTunes Connect Developer Guide*』を参照してください。

サンドボックスでのテスト

アプリケーションをサンドボックスでテストするには、次の手順に従います。

1. テスト用デバイスで自身のiTunesアカウントからログアウトします。

   アプリケーションをテストする前に、まず通常のiTunesアカウントからログアウトする必要があります。iOS 3.0の「設定(Settings)」アプリケーションには、「Store」カテゴリがあります。iTunesアカウントからログアウトするには、アプリケーションを終了し、「設定(Settings)」アプリケーションを起動して「Store」アイコンをクリックします。現在のアクティブなアカウントからサインアウトしてください。

> **Important** 「設定(Settings)」アプリケーションにテスト用アカウントでサインインしないでください。

2. アプリケーションを起動します。

   自身のアカウントからサインアウトしたら、「設定(Settings)」を終了し、アプリケーションを起動します。アプリケーションのStoreから購入を行うと、Store Kitはトランザクションを認証するように促します。テストアカウントを使用してログインし、ペイメントを承認します。会計処理は発生しませんが、ペイメントが行われたかのようにトランザクションが完了します。

サンドボックスでのレシートの有効性確認

サンドボックス環境で作成されたレシートの有効性を確認することもできます。サンドボックスから受信したレシートの有効性を確認するコードは実際のApp Storeのものと同一ですが、サーバがサンドボックスのURLにリクエストを指示しなければならないところが違います。

```
NSURL *sandboxStoreURL = [[NSURL alloc] initWithString:
@"https://sandbox.itunes.apple.com/verifyReceipt"];
```

自動更新購読（Auto-Renewable Subscriptions）

注意　購読が可能なのはiOS上に限ります。

In-App Purchaseを使用すると、標準的な方法で自動更新購読を実装できます。自動更新購読には、いくつか重要な特徴があります。

- iTunes Connectで自動更新購読を設定したときに、購読の期間その他のマーケティングオプションも設定します。
- 自動更新購読は、非消耗型プロダクトのリストアで使用したStore Kit関数を使用して、自動的にリストアされます。それぞれの更新に対するトランザクションだけでなく、オリジナルの購入トランザクションもアプリケーションに送付されます。“トランザクションのリストア”（19 ページ）を参照してください。
- サーバがApp Storeを使用してレシートを確認し、定期購読がアクティブで、かつApp Storeにより更新されていたときに、App Storeはアプリケーションに更新済みのレシートを返します。

自動更新購読のStoreへの追加

自動更新購読を実装するには、次の手順に従います。

- iTunes Connectに接続し、その接続を使用して新たに共有秘密鍵を作成します。共有秘密鍵は、自動更新購読のレシートを検証する場合にサーバが提供する必要があるパスワードです。この共有秘密鍵は、App Storeとの取引に対し、高いレベルのセキュリティを提供します。詳細については、『*iTunes Connect Developer Guide*』を参照してください。
- 新しい自動更新購読タイプを使用して、iTunes Connectで新しいプロダクトを設定します。
- サーバでレシートを確認するためのコードを修正して、App Storeに送信するJSONデータに共有秘密鍵を追加するようにします。サーバ検証コードでは、応答を解析して、購読が期限切れか否かを判断しなければなりません。ユーザが購読を更新済みであれば、最新のレシートがサーバに返されます。

クライアントアプリケーションの設計

多くの場合、クライアントアプリケーションは最小限の変更だけで、自動更新購読に対応できます。実際、非消耗型プロダクトを回復する場合と同じコードを使用して自動更新購読を回復できるため、クライアントアプリケーションはよりシンプルに作成されています。アプリケーションは、定期購読が更新されたときに定期的に個別にトランザクションを受信します。アプリケーションはそれぞれのレシートを個別に検証する必要があります。

自動更新購読のレシートの確認方法

自動更新購読用のレシートの確認方法は、“Storeレシートの確認”（26 ページ）で説明された手順とほとんど同じです。アプリケーションはJSONオブジェクトを作成して、App Storeに送信します。自動更新購読用レシートのJSONオブジェクトには、2番目のパラメータがあります。iTunes Connectで以前作成した共有秘密鍵です。

```
{
    "receipt-data" : "(receipt bytes here)",
    "password"     : "(shared secret bytes here)"
}
```

応答にはレシートが正常に検証されたか否かを表すステータスフィールドがあります。

```
{
    "status" : 0,
    "receipt" : { (receipt here) },
    "latest_receipt" : "(base-64 encoded receipt here)",
    "latest_receipt_info" : { (latest receipt info here) }
}
```

ユーザのレシートが有効で、定期購読がアクティブならば、ステータスフィールドは0を保持し、receiptフィールドにはデコード済みのレシートデータが設定されます。サーバが0以外のステータス値を受信した場合、表 7-1を使用して0以外のステータスコードを解釈します。

表 7-1　自動更新購読のステータスコード

ステータスコード	解説
21000	App Storeは、提供したJSONオブジェクトを読むことができません。
21002	receipt-dataプロパティのデータは形式が不正です。
21003	レシートを認証できません。
21004	この共有秘密鍵は、アカウントのファイルに保存された共有秘密鍵と一致しません。
21005	レシートサーバは現在利用できません。
21006	このレシートは有効ですが、定期購読の期限が切れています。ステータスコードがサーバに返される際、レシートデータもデコードされ、応答の一部として返されます。
21007	このレシートはサンドボックスレシートですが、検証のためにプロダクションサービスに送られました。
21008	このレシートはプロダクションレシートですが、検証のためにサンドボックスに送られました。

Important　この0以外のステータスコードは、自動更新購読についての情報を回復したときだけに適応できます。ほかの種類のプロダクトの応答をテストする場合は、これらのステータスコードを使用しないでください。

JSONオブジェクトのreceiptフィールドには、レシートから解析された情報が保持されています。自動更新購読のレシートデータには、1つの追加キーが含まれています。表 5-1 （28 ページ）で以前説明したほかのキーは、定期購読用に少し変更されています。新規のキーと変更されたキーの詳細については、表 7-2を参照してください。

表 7-2　自動更新購読情報のキー

キー	解説
`expires_date`	定期購読レシートの有効期限は、1970年1月1日00:00:00 GMTからミリ秒単位で表示されています。このキーは、リストアされたトランザクションには含まれていません。
`original_transaction_id`	これには、最初に購入したときのトランザクション識別子が含まれます。この定期購入のすべての更新情報とリストアされたトランザクションは、この識別子を共有します。

キー	解説
original_purchase_date	最初に購入したときの購入日を保持しており、定期購入の開始日を表します。
purchase_date	このトランザクションが発生した請求日を保持しています。更新可能な定期購読のトランザクションに対しては、定期購読が更新された日付になります。App Storeで解析されるレシートがこの定期購読の最新のレシートである場合、このフィールドには更新された最新の定期購読の日付を保持します。

応答にはreceipt_dataフィールドのほか、2つの新しいフィールドが入っているかも知れません。ユーザの定期購読がアクティブで、サーバがApp Storeに送ったレシートに置き換わるトランザクションで更新が行われた場合、latest_receiptフィールドには、この定期購読の最新の更新に対応する新しいbase64でエンコードされたレシートが含まれます。この新しいレシートのデコード済みデータも、latest_expired_receipt_infoフィールドに与えられます。サーバはこの新しいレシートを使用して、最新の更新レコードを維持できます。

自動更新購読のリストア

App Storeは、定期購読を更新するたびに別のトランザクションを作成します。アプリケーションが以前の購入情報をリストアするときに、Store Kitはそれぞれのトランザクションを送付します。この購入の内容をリストアするには、アプリケーションで各トランザクションの購入日と定期購読の長さを組み合わせる必要があります。定期購読の長さは、SKPaymentTransactionオブジェクトやApp Storeレシート検証ソフトから送付されるレシートデータには含まれていません。各定期購読の長さについては、異なるプロダクト識別子を使用する必要があります。そうすればアプリケーションはプロダクト識別子を定期購読の長さに変換できます。この長さと購入日付を使用して間隔を決定し、このレシートが有効かどうかを判断します。

書類の改訂履歴

この表は「*In App Purchase プログラミングガイド*」の改訂履歴です。

日付	メモ
2012-02-16	プラットフォームをまたがる使い方を反映するよう図版を作り直しました。コードから非推奨になったメソッドを削除しました。 "アプリケーションへのStoreの追加"（20ページ）で、非推奨になったメソッド`paymentWithProductIdentifier:`を`paymentWithProduct:`に置き換えました。
2012-01-09	Mac OS Xにおけるこの技術の活用に関して細かな改訂を行いました。
2011-10-12	概要の項に新しいタイプの購入方法に関する情報を追加しました。
2011-06-06	本書のMac OS X向けの初版。
2011-05-26	自動更新購読に関して、最新のサーバの振る舞いに合わせて修正しました。
2011-03-08	更新可能な定期購読の章でキーのリストを訂正しました。
2011-02-15	アプリケーションは購入情報の提出前に、必ず製品情報を取得する必要があり、これによって製品が販売状態にあることを確認できる旨を追加しました。更新可能な定期購読についての情報を追加しました。
2010-09-01	細かな編集を行いました。

日付	メモ
2010-06-14	SKRequestについての説明をわかりやすくしました。
2010-01-20	JSONレシートオブジェクトの誤字を訂正しました。
2009-11-12	レシートデータはbase64でエンコードしてから検証サーバに渡す必要があることを追加しました。その他の細かな更新を行いました。
2009-09-09	「はじめに」の章を改訂しました。レシートデータの使い方を明確にしました。プロダクトメタデータの作成について推奨する主要な情報源として、『iTunes Connect Developer Guide』を記載しました。ドキュメント名を『Store Kit Programming Guide』から『In App Purchase Programming Guide』に変更しました。
2009-06-12	Apple App Storeからのプロダクトの価格情報の取得に関する説明、およびStoreでのレシートの確認に関する説明を追加しました。
2009-03-13	アプリケーションにStoreを実装するためのStoreKit APIの使用方法について説明した新規ドキュメント。

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
U.S.A.

アップルジャパン株式会社
〒163-1450 東京都新宿区西新宿
3 丁目20 番2 号
東京オペラシティタワー
http://www.apple.com/jp/