
In App Purchase プログラミングガイド



2010-06-14



Apple Inc.
© 2010 Apple Inc.
All rights reserved.

本書の一部あるいは全部を Apple Inc. から書面による事前の許諾を得ることなく複製（コピー）することを禁じます。また、製品に付属のソフトウェアは同梱のソフトウェア使用許諾契約書に記載の条件のもとでお使いください。書類を個人で使用する場合に限り1台のコンピュータに保管すること、またその書類にアップルの著作権表示が含まれる限り、個人的な利用を目的に書類を複製することを認めます。

Apple ロゴは、米国その他の国で登録された Apple Inc. の商標です。

キーボードから入力可能な Apple ロゴについても、これを Apple Inc. からの書面による事前の許諾なしに商業的な目的で使用すると、連邦および州の商標法および不正競争防止法違反となる場合があります。

本書に記載されているテクノロジーに関しては、明示または黙示を問わず、使用を許諾しません。本書に記載されているテクノロジーに関するすべての知的財産権は、Apple Inc. が保有しています。本書は、Apple ブランドのコンピュータ用のアプリケーション開発に使用を限定します。

本書には正確な情報を記載するように努めました。ただし、誤植や制作上の誤記がないことを保証するものではありません。

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/>

App Store is a service mark of Apple Inc.

Apple, the Apple logo, iPhone, iTunes, and Xcode are trademarks of Apple Inc., registered in the United States and other countries.

Apple Inc. は本書の内容を確認しておりますが、本書に関して、明示的であるか黙示的であるかを問わず、その品質、正確さ、市場性、または特定の目的に対する適合性に関して何らかの保証または表明を行うものではありません。その結果、本書は「現状有姿のまま」提供され、本書の品質または正確さに関連して発生するすべての損害は、購入者であるお客様が負うものとしします。

いかなる場合も、Apple Inc. は、本書の内容に含まれる瑕疵または不正確さによって生じる直接

的、間接的、特殊的、偶発的、または結果的損害に対する賠償請求には一切応じません。そのような損害の可能性があらかじめ指摘されている場合においても同様です。

上記の損害に対する保証および救済は、口頭や書面によるか、または明示的や黙示的であるかを問わず、唯一のものであり、その他一切の保証にかわるものです。Apple Inc. の販売店、代理店、または従業員には、この保証に関する規定に何らかの変更、拡張、または追加を加える権限は与えられていません。

一部の国や地域では、黙示あるいは偶発的または結果的損害に対する賠償の免責または制限が認められていないため、上記の制限や免責がお客様に適用されない場合があります。この保証はお客様に特定の法的権利を与え、地域によってはその他の権利がお客様に与えられる場合もあります。

目次

序章 はじめに 7

- 対象読者 7
- この書類の構成 7
- 関連項目 8

第1章 In App Purchaseの概要 9

- プロダクト 9
- App Storeへのプロダクトの登録 10
- 機能の配布 11
 - 組み込みプロダクトモデル 11
 - サーバプロダクトモデル 12

第2章 プロダクト情報の取得 15

- App Storeへのリクエストの送信 15
 - SKRequest 15
 - SKRequestDelegate 15
- プロダクトに関する情報のリクエスト 16
 - SKProductsRequest 16
 - SKProductsRequestDelegate 16
 - SKProductsResponse 17
 - SKProduct 17

第3章 購入を処理する 19

- 支払いの受け取り 19
 - SKPayment 20
 - SKPaymentQueue 20
 - SKPaymentTransaction 20
 - SKPaymentTransactionObserver 20
- トランザクションのリストア 21

第4章 アプリケーションへのStoreの追加 23

- 段階的手順 23
- 次に学ぶこと 26

第5章 Storeレシートの確認 27

- App Storeを使用したレシートの確認 27

Storeレシート 28

第6章

Storeのテスト 31

サンドボックス環境 31

サンドボックスでのテスト 31

サンドボックスでのレシートの有効性確認 32

改訂履歴

書類の改訂履歴 33

図、表

第1章 In App Purchaseの概要 9

- 図 1-1 App Storeモデル 9
- 図 1-2 組み込みプロダクトの配布 12
- 図 1-3 サーバプロダクトの配布 13

第2章 プロダクト情報の取得 15

- 図 2-1 Store Kitリクエストモデル 15
- 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 16

第3章 購入を処理する 19

- 図 3-1 ペイメントリクエストのキューへの追加 19

第5章 Storeレシートの確認 27

- 表 5-1 購入情報のキー 28

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iPhone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、ユーザからの支払いを安全に処理します。Store Kitは、ユーザに支払いの承認を求めた後、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支払い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

重要： In App Purchaseが行うのは支払い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。この文書では、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り扱い規定を確認する必要があります。

対象読者

この文書は、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

この文書は、次の章で構成されています。

- 「[In App Purchaseの概要](#)」（9 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](#)」（15 ページ）では、iPhone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](#)」（19 ページ）では、App Storeからの支払い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。

序章

はじめに

- 「[アプリケーションへのStoreの追加](#)」（23 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](#)」（27 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](#)」（31 ページ）では、サンドボックス環境を使用してiPhoneアプリケーションをテスト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。

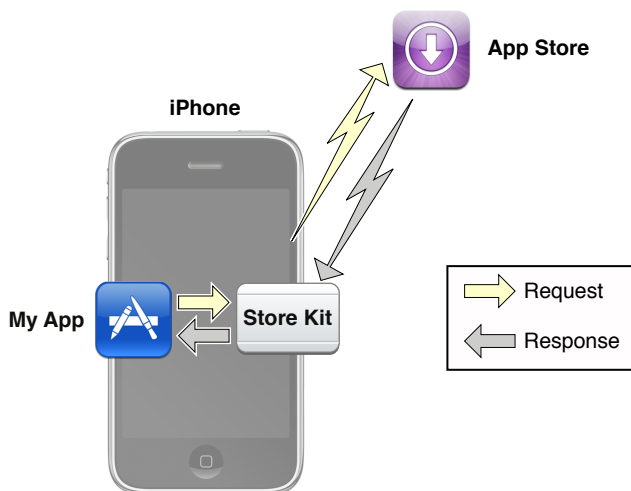
『[iTunes Connect開発者用ガイド](#)』では、App Storeでのプロダクトおよびアカウントテストの設定方法について説明しています。

[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。デベロッパは、適度な更新頻度を定めて、わずらわしいメッセージでユーザを困らせないようにする必要があります。定期購読の期限の管理と更新時の請求は、どちらもデベロッパが管理する必要があります。App Storeが購読期間を監視したり、自動請求の仕組みを提供したりするわけではありません。

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ゆだねています。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。

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一意に識別します。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、前述のさまざまなプロダクトを次の3つの基本的なタイプに大別しています。

- **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。
- **Subscriptions (購読型)** プロダクト。消耗型プロダクトと非消耗型プロダクトの両方の性質を合わせ持ちます。消耗型プロダクトと同様に、購読型プロダクトは何度も購入される可能性があるため、デベロッパはアプリケーションに独自の更新メカニズムを実装できるようになっています。ただし、購読型は1ユーザに関連付けられたすべてのデバイスに提供される必要があります。In App Purchaseでは、購読型プロダクトについてはデベロッパが提供する外部サーバを通じて提供されるものと想定しています。複数デバイスに購読型プロダクトを提供するには、インフラも提供しなければなりません。

App Storeへのプロダクトの登録方法の詳細については、『[iTunes Connect開発者用ガイド](#)』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最もよく使われます。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。

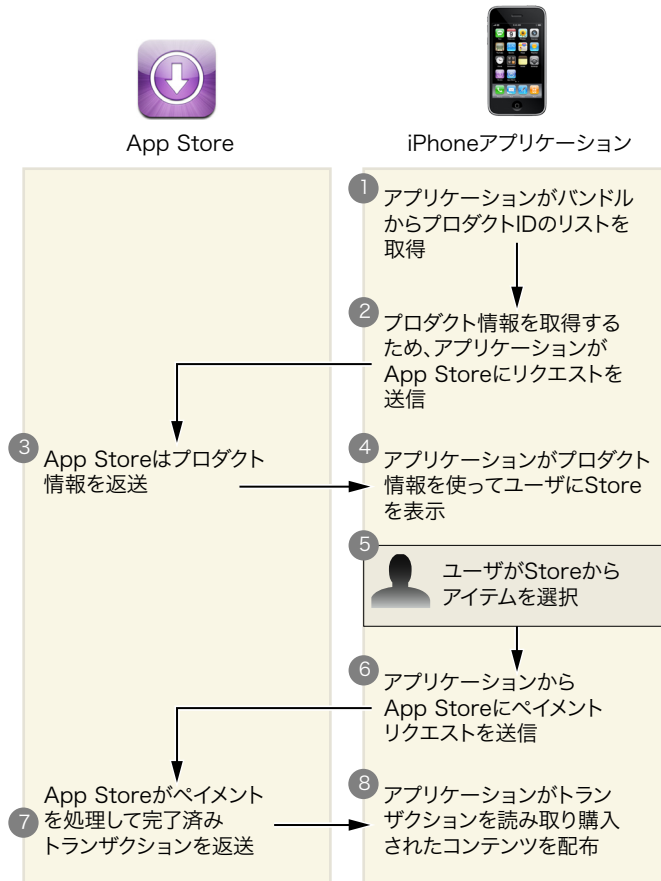
重要： In App Purchaseは、問題なく購入された後のアプリケーションにパッチを適用する機能を提供しません。プロダクトでアプリケーションのバンドルを変更する必要がある場合、App Storeにアプリケーションの更新版を配布しなければなりません。

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最も簡単な方法は、アプリケーションの環境設定を変更することです。アプリケーションの環境設定は、ユーザがiPhone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iPhone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

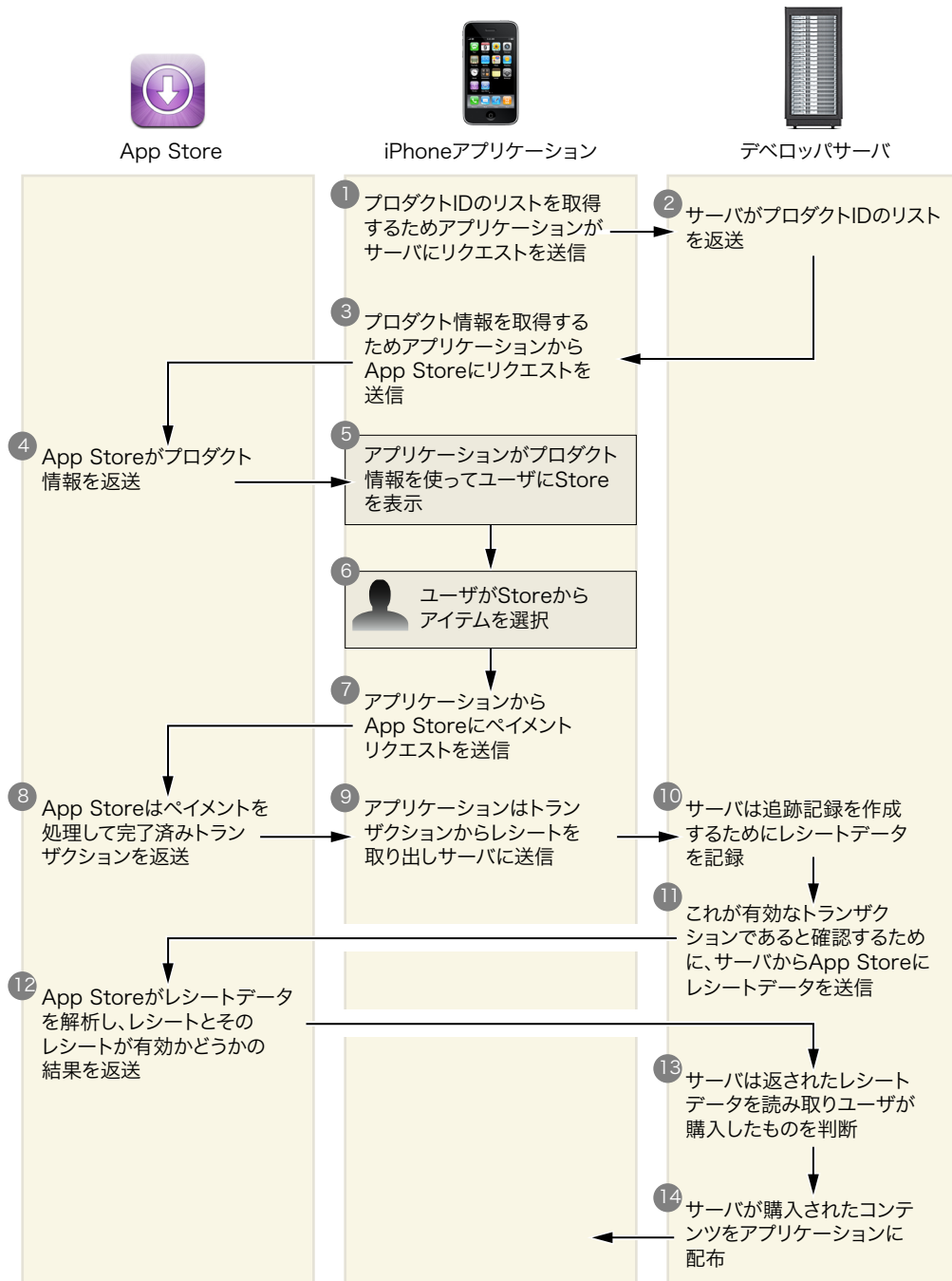


サーバプロダクトモデル

サーバプロダクトモデルでは、iPhone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ーが別途用意します。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。StoreKitはサーバの設計や、iPhone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。iPhoneアプリケーションとデベロッパー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ーにあります。加えて、StoreKitは特定のユーザー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、iPhoneユーザーを識別するメカニズムをデベロッパー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、特定のユーザー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、デベロッパー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](#)」(27 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復元できますが、購読型はサーバからリストアしなければなりません。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。

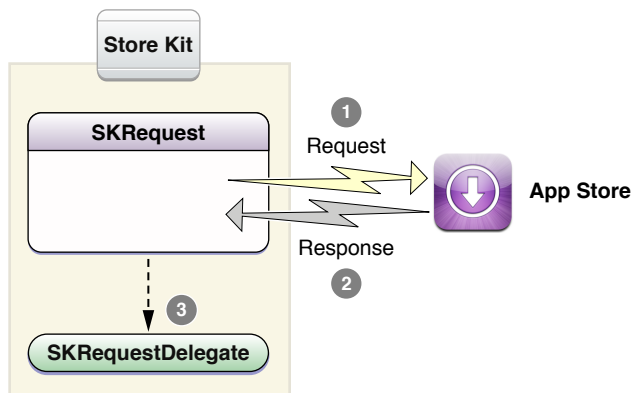
プロダクト情報の取得

アプリケーションでユーザにStoreを表示する準備ができれば、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

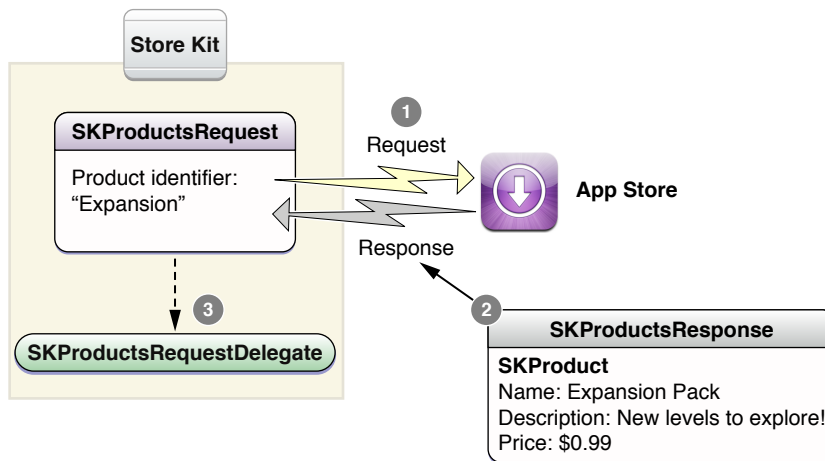
SKRequestDelegateは、正常に終了したリクエストと、エラーによって失敗したリクエストを処理するためにアプリケーションが実装するプロトコルです。

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、開発者が事前にiTunes Connectに入力したローカライズされた情報を返します。開発者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



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。

第2章

プロダクト情報の取得

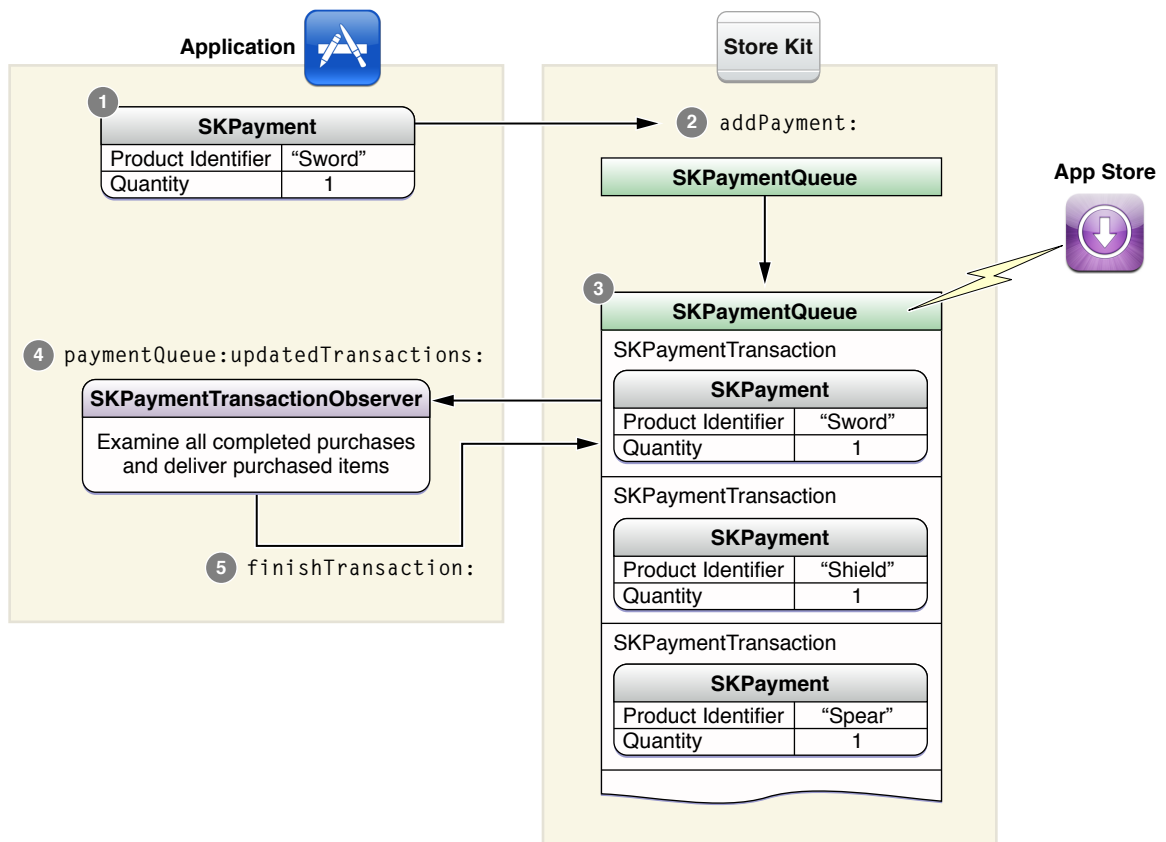
購入を処理する

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に支払いの受け取りを依頼します。アプリケーションから支払い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支払い処理を続けます。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。

支払いの受け取り

支払いを受け取るには、図 3-1に示すように、アプリケーションで支払いオブジェクトを作成し、オブジェクトを支払いキューに入れます。

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。非消耗型プロダクトまたは購読型プロダクトをサポートしている場合、アプリケーションではこれらの購入したプロダクトを追加のデバイス上でリストアできる機能をユーザに提供しなければなりません。

Store Kitは、非消耗型プロダクト用に、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。以前完了した非消耗型プロダクトのそれぞれの購入については、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にはオリジナルのトランザクションのコピーが含まれます。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すると、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。

すでに購入済みの非消耗型アイテムをユーザが購入しようとする時、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。

購読型プロダクト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。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。

第3章

購入を処理する

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。StoreKit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支払い処理を行います。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開発者用ガイド](#)』で説明しています。

3. 支払い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、新しい支払いリクエストをキューに入れる前に、支払い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、支払い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。

```
- (void) requestProductData
{
    SKProductsRequest *request= [[SKProductsRequest alloc]
initWithProductIdentifiers:[NSSet setWithObject:kMyFeatureIdentifier]];
    request.delegate = self;
    [request start];
}
- (void)productsRequest:(SKProductsRequest *)request
didReceiveResponse:(SKProductsResponse *)response
{
    NSArray *myProduct = response.products;
    // UIを埋める
    [request autorelease];
}
```

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。

StoreKit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しれません。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](#)」 (11 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しれません。

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。

```

SKPayment *payment = [SKPayment
paymentWithProductIdentifier:kMyFeatureIdentifier];
[[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。

```

SKMutablePayment *payment = [SKMutablePayment
paymentWithProductIdentifier:kMyFeatureIdentifier];
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、iPhone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、iPhone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。

重要： 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。

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" : "(actual 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

```
{
  "status" : 0,
  "receipt" : { ... }
}
```

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： 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](#)」 (31 ページ) を参照してください。

表 5-1 購入情報のキー

キー	説明
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iPhoneアプリケーションを一意的に識別するために、App Storeが使用する文字列。サーバが複数のiPhone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。

キー	説明
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	iPhoneアプリケーション用のバンドルID。
bvrs	iPhone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： Store KitはiPhoneシミュレータ上では動作しません。iPhone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開発者用ガイド](#)』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iPhone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。iPhone OS 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。

重要： 「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、StoreKit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"];
```


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
2010-06-14	SKRequestについての説明を分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。その他若干の変更をしました。
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開発者ガイド』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の使用方法を記述した新規文書。

改訂履歴

書類の改訂履歴