In-App Purchase プログラミングガイド

Developer

アプリケーションの起動時に、トランザクションキューのオブザーバを登録します(リスト 4-1を参照)。オブザーバは、キューにトランザクションを追加した直後だけでなく、いつでもトランザクションを処理できることを確認します。たとえば、トンネルに入る直前にユーザがアプリケーションで何かを購入する場合について考慮します。ネットワーク接続がないため、アプリケーションは購入されたコンテンツを配信できません。アプリケーションが次回起動されたときに、Store Kitはトランザクションキューのオブザーバを再度呼び出して、購入された項目をその時点で配信します。同様に、アプリケーションがトランザクションに終了のマークを付けられなかった場合、トランザクションが終了したと適切にマークされるまで、Store Kitはアプリケーションが起動されるたびに毎回オブザーバを呼び出します。

リスト 4-1　トランザクションキューのオブザーバの登録

```
- (BOOL)application:(UIApplication *)application
 didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    /* ... */

    [[SKPaymentQueue defaultQueue] addTransactionObserver:observer];
}
```

トランザクションキューのオブザーバに`paymentQueue:updatedTransactions:`を実装します。トランザクションの状態が、たとえば支払い要求が処理されたために変化すると、Store Kitはこのメソッドを呼び出します。トランザクションの状態は、表 4-1およびリスト 4-2に示すように、アプリケーションで実行する必要があるアクションがどれであるのかを通知します。キューに入っているトランザクションの状態が変わる順序は不定です。アプリケーションはいつでもアクティブなトランザクションを処理できるようにしておかなければなりません。

表 4-1　トランザクションの状態と対応するアクション

状態	アプリケーションで実行するアクション
`SKPaymentTransaction-StatePurchasing`	進捗している状態を反映するようにUIを更新し、再び呼び出されるのを待機します。
`SKPaymentTransaction-StateDeferred`	遅延している状態を反映するようにUIを更新し、再び呼び出されるのを待機します。
`SKPaymentTransaction-StateFailed`	`error`プロパティの値を使用して、ユーザーにメッセージを表示します。エラーを表す定数については、『*Store Kit Constants Reference*』の`SKErrorDomain`に関する項に一覧が載っています。

状態	アプリケーションで実行するアクション
SKPaymentTransaction-StatePurchased	購入した機能を提供します。
SKPaymentTransaction-StateRestored	以前に購入した機能を復元します。

リスト 4-2　トランザクションの状態への応答

```
- (void)paymentQueue:(SKPaymentQueue *)queue
 updatedTransactions:(NSArray *)transactions
{
    for (SKPaymentTransaction *transaction in transactions) {
        switch (transaction.transactionState) {
            // Call the appropriate custom method for the transaction state.
            case SKPaymentTransactionStatePurchasing:
                [self showTransactionAsInProgress:transaction deferred:NO];
                break;
            case SKPaymentTransactionStateDeferred:
                [self showTransactionAsInProgress:transaction deferred:YES];
                break;
            case SKPaymentTransactionStateFailed:
                [self failedTransaction:transaction];
                break;
            case SKPaymentTransactionStatePurchased:
                [self completeTransaction:transaction];
                break;
            case SKPaymentTransactionStateRestored:
                [self restoreTransaction:transaction];
                break;
            default:
                // For debugging
                NSLog(@"Unexpected transaction state %@",
@(transaction.transactionState));
                break;
        }
    }
```

```
}
```

待機中にユーザインターフェイスを最新の状態に保つには、トランザクションキューのオブザーバは`SKPaymentTransactionObserver`プロトコルから次のようにオプションメソッドを実装できます。`paymentQueue:removedTransactions:`メソッドは、トランザクションがキューから削除されると呼び出されます。このメソッドの実装では、対応する項目をアプリケーションのUIから削除します。`paymentQueueRestoreCompletedTransactionsFinished:`メソッドまたは`paymentQueue:restoreCompletedTransactionsFailedWithError:`メソッドは、Store Kitがトランザクションの復元を終了するとエラーの有無に応じて呼び出されます。これらのメソッドの実装では、アプリケーションのUIが処理の成功またはエラーを反映するように更新します。

購入の持続

プロダクトの提供を開始した後、アプリケーションは購入の持続的な記録を作成する必要があります。アプリケーションは起動時にその持続的な記録を使用して、プロダクトの提供を継続します。また、購入の復元にも記録を使用します(“購入したプロダクトの復元” （40 ページ）を参照)。アプリケーションを持続させる方法は、販売するプロダクトの種類とiOSのバージョンによって異なります。

- iOS 7以降の非消耗型プロダクトと自動更新購読では、アプリケーションのレシートを持続的な記録として使用します。
- iOS 7より前のバージョンの非消耗型プロダクトと自動更新購読では、User DefaultsシステムまたはiCloudを使用して持続的な記録を管理します。
- 非更新購読では、iCloudまたは独自のサーバを使用して持続的な記録を管理します。
- 消耗型プロダクトの場合は、購入を反映するようにアプリケーションの内部状態をアプリケーションで更新しますが、消耗型プロダクトは復元やデバイス間での同期を行わないため、持続的な記録を保持する必要はありません。更新された状態が、状態の保存をサポートするオブジェクト(iOS)、またはアプリケーションの起動時にわたって手動で状態を保存するオブジェクトの一部である(iOSまたはOS X)ことを確認します。状態の保存の詳細については、“State Preservation and Restoration” in *iOS App Programming Guide* を参照してください。

User DefaultsシステムまたはiCloudを使用している場合、アプリケーションは数値やブール値などの値、またはトランザクションレシートのコピーを格納できます。OS Xでは、`defaults`コマンドを使用してUser Defaultsシステムを編集できます。レシートを格納するには追加のアプリケーションロジックが必要ですが、持続的な記録の改ざんを防止できます。

iCloudを使用して持続する場合、アプリケーションの持続的な記録はデバイス間で同期されますが、ほかのデバイスに関連コンテンツをダウンロードするのはアプリケーションです。

Appレシートを使用した持続

Appレシートには、ユーザの購入記録とAppleによって暗号化された署名が含まれています。詳細については、『*Receipt Validation Programming Guide*』を参照してください。

消耗型プロダクトと非更新購読の情報は、支払いが行われるとシートに追加され、トランザクションを終了するまでレシート上に残ります。トランザクションの終了後、この情報はレシートが次に更新されるとき、たとえばユーザが次に購入を行ったときに削除されます。

これ以外の種類のプロダクト購入の情報は、支払いが行われるとレシートに追加され、レシートに残り続けます。

User DefaultsまたはiCloudを使用した値の持続

User DefaultsまたはiCloudに情報を保存するには、キーの値を設定します。

```
#if USE_ICLOUD_STORAGE
NSUbiquitousKeyValueStore *storage = [NSUbiquitousKeyValueStore defaultStore];
#else
NSUserDefaults *storage = [NSUserDefaults standardUserDefaults];
#endif

[storage setBool:YES forKey:@"enable_rocket_car"];
[storage setObject:@15 forKey:@"highest_unlocked_level"];

[storage synchronize];
```

User DefaultsまたはiCloudを使用したレシートの持続

User DefaultsまたはiCloudにトランザクションのレシートを保存するには、レシートのデータのキーの値を設定します。

```
#if USE_ICLOUD_STORAGE
NSUbiquitousKeyValueStore *storage = [NSUbiquitousKeyValueStore defaultStore];
#else
NSUserDefaults *storage = [NSUserDefaults standardUserDefaults];
#endif

NSData *newReceipt = transaction.transactionReceipt;
```

```
NSArray *savedReceipts = [storage arrayForKey:@"receipts"];
if (!savedReceipts) {
    // Storing the first receipt
    [storage setObject:@[newReceipt] forKey:@"receipts"];
} else {
    // Adding another receipt
    NSArray *updatedReceipts = [savedReceipts arrayByAddingObject:newReceipt];
    [storage setObject:updatedReceipts forKey:@"receipts"];
}

[storage synchronize];
```

独自のサーバを使用した持続

どのレシートがどのユーザに属しているのかを追跡できるように、ある種の証明書またはIDとともにレシートのコピーをサーバに送信します。たとえば、電子メールまたはユーザ名とパスワードを使って、サーバに対してユーザが本人であることを証明します。`UIDevice`の`identifierForVendor`プロパティは使用しないでください。デバイスが異なると、このプロパティの値も異なるため、同じユーザの別のデバイスで行われた購入の識別と復元ができないので、このプロパティを使用することはできません。

アプリケーションの機能のアンロック

プロダクトによってアプリケーションの機能を有効にする場合、コードパスを有効にするためのブール値を設定し、必要に応じてユーザインタフェースを更新します。案ロックする機能を決定するには、トランザクション発生時にアプリケーションによって作成された持続的な記録を確認してください。購入の完了時とアプリケーションの起動時、アプリケーションはこのブール値を更新する必要があります。

たとえば、アプリケーションのレシートを使用する場合、コードは次のようになります。

```
NSURL *receiptURL = [[NSBundle mainBundle] appStoreReceiptURL];
NSData *receiptData = [NSData dataWithContentsOfURL:receiptURL];

// Custom method to work with receipts
BOOL rocketCarEnabled = [self receipt:receiptData
```

```
        includesProductID:@"com.example.rocketCar"];
```

また、User Defaultsシステムを使用する場合は次のようになります。

```
NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
BOOL rocketCarEnabled = [defaults boolForKey:@"enable_rocket_car"];
```

その後、その情報を使用して、アプリケーションで適切なコードパスを有効にします。

```
if (rocketCarEnabled) {
    // Use the rocket car.
} else {
    // Use the regular car.
}
```

関連コンテンツの配信

プロダクトに関連付けられたコンテンツがある場合は、そのコンテンツもアプリケーションでユーザに配信する必要があります。たとえば、ゲームのレベルを購入するときにレベルを定義したファイルを配信する必要がある場合、また音楽アプリケーションで追加の楽器を購入するときに、ユーザがその楽器を演奏するために必要なサウンドファイルを配信する必要がある場合があります。

そのコンテンツをアプリケーションバンドルに埋め込むか、必要に応じてダウンロードできます。いずれの方法にもそれぞれの利点と欠点があります。アプリケーションバンドルに含めるコンテンツが少なすぎると、購入したものが小さなものであってもダウンロードされる間待つ必要があります。アプリケーションバンドルに多くを詰め込み過ぎると、アプリケーションの初期ダウンロード時間が長くなり、関連プロダクトの購入を望まないユーザの空き容量を無駄にすることになります。また、アプリケーションのサイズが大きすぎると、携帯電話のネットワークではアプリケーションをダウンロードできなくなります。

特に、ユーザの大半がプロダクトを購入することが予想される場合は、アプリケーションにはサイズの小さいファイル(数メガバイト程度まで)を埋め込みます。アプリケーションバンドルのコンテンツは、ユーザが購入した時点ですぐに使用できます。ただし、アプリケーションバンドルのコンテンツの更新や追加を行う場合は、更新されたバージョンのアプリケーションを登録する必要があります。

必要に応じて、サイズの大きいファイルをダウンロードしてください。アプリケーションバンドルとコンテンツを分離しておくと、アプリケーションの初期ダウンロードが小さくて済みます。たとえば、ゲームの最初のレベルはアプリケーションにバンドルしておいて、残りのレベルはユーザに購入

してもらうことができます。アプリケーションバンドルにハードコーディングされているのではなく、アプリケーションが独自のサーバからプロダクトリストを取得する場合、コンテンツの追加やアプリケーションによってダウンロードされたコンテンツの更新のためにアプリケーションを再登録する必要はありません。

iOS 6以降では、ほとんどのアプリケーションにおいて、ダウンロード済みファイルにAppleによってホストされたコンテンツを使用する必要があります。Appleによってホストされたコンテンツバンドルを作成するには、XcodeでIn-App Purchase Contentターゲットを使用してiTunes Connectに登録します。Appleのサーバでコンテンツをホストする場合、サーバを提供する必要はありません。アプリケーションのコンテンツは、App Storeなどの大規模運用を支えているAppleのインフラストラクチャに保存されます。また、Appleによってホストされたコンテンツは、アプリケーションが実行されていない場合であっても、バックグラウンドで自動的にダウンロードされます。

独自のサーバインフラストラクチャがあり、旧バージョンのiOSをサポートする必要がある場合や、複数のプラットフォームでサーバインフラストラクチャを共有する場合、独自のコンテンツをホストすることも可能です。

> **注意:** アプリケーションのバイナリにパッチを当てたり、実行コードをダウンロードしたりすることはできません。アプリケーションの登録時には、アプリケーションのすべての機能をサポートするために必要なすべての実行コードが含まれている必要があります。新しいプロダクトによりコードの変更が必要になった場合は、アプリケーションの更新されたバージョンを登録する必要があります。

ローカルコンテンツのロード

ロケットのロードには、アプリケーションバンドルから他のリソースをロードするときと同様に、`NSBundle`クラスを使用します。

```
NSURL *url = [[NSBundle mainBundle] URLForResource:@"rocketCar"
                                     withExtension:@"plist"];
[self loadVehicleAtURL:url];
```

ホストしたコンテンツのAppleのサーバからのダウンロード

Appleによってホストされたコンテンツと関連付けられたプロダクトをユーザが購入した場合、トランザクションキューのオブザーバに渡されたトランザクションにも関連コンテンツをダウンロードする`SKDownload`のインスタンスが含まれます。

そのコンテンツをダウンロードするには、downloadsのstartDownloads:メソッドを呼び出すことによって、ダウンロードオブジェクトをトランザクションのSKPaymentQueueプロパティからトランザクションキューに追加します。downloadsプロパティの値がnilの場合は、このトランザクションにはAppleによってホストされたコンテンツがありません。アプリケーションのダウンロードとは異なり、コンテンツのダウンロードでは一定のサイズよりも大きなコンテンツであってもWi-Fi接続を自動的に要求しません。ユーザからの明示的なアクションがない限り、大きなファイルのダウンロードでは携帯電話ネットワークの使用を避けてください。

進行状況をUIで更新するなど、ダウンロード状態の変更に応答するには、トランザクションキューのオブザーバにpaymentQueue:updatedDownloads:メソッドを実装します。ダウンロードが失敗した場合、errorプロパティの情報を使用してユーザにエラーを通知します。

アプリケーションでエラーが正常に処理されることを確認します。たとえば、ダウンロード中にディスク容量が不足した場合、不完全なダウンロードを破棄するか、素ペースを確保してからダウンロードを再開するかをユーザが選択できるようにします。

コンテンツのダウンロード状況をユーザインターフェイスで更新するには、progressプロパティとtimeRemainingプロパティの値を使用します。ユーザにダウンロード進行状況をコントロールしてもらうには、SKPaymentQueueのpauseDownloads:メソッド、resumeDownloads:メソッド、およびcancelDownloads:メソッドを使用できます。ダウンロードが完了したかどうかを判別するには、downloadStateプロパティを使用します。状態をチェックするために、ダウンロードオブジェクトのprogressまたはtimeRemainingプロパティを使用しないでください。これらのプロパティはUIの更新用です。

> **注意:** トランザクションが終了する前に、Appleによってホストされたすべてのコンテンツをダウンロードします。トランザクションの終了後は、トランザクションオブジェクトを使用できなくなります。

iOSでは、ダウンロードしたファイルをアプリケーションで管理できます。StoreKitフレームワークによって保存されたファイルは、Caches辞書でバックアップフラグが設定されていない状態になっています。ダウンロード完了後は、アプリケーションが適切な場所に移動します。デバイスのディスク容量が不足した場合に削除可能なコンテンツ(後でアプリケーションによって再ダウンロード可能)については、ファイルをCachesディレクトリに残します。その他の場合、ファイルをDocumentsフォルダに移動、ユーザのバックアップから除外するフラグを設定します。

リスト 4-3 ダウンロード済みコンテンツのバックアップからの除外

```
NSError *error;
BOOL success = [URL setResourceValue:[NSNumber numberWithBool:YES]
                               forKey:NSURLIsExcludedFromBackupKey
```

```
                                error:&error];
if (!success) { /* エラーを処理... */ }
```

OS Xでは、ダウンロードされたコンテンツはシステムによって管理されます。アプリケーションで直接これらのファイルを移動したり削除したりすることはできません。ダウンロード後にコンテンツのある位置を特定するには、ダウンロードオブジェクトの`contentURL`プロパティを使用します。引き続いて起こる起動中にファイルの位置を特定するには、`SKDownload`の`contentURLForProductID:`クラスメソッドを使用します。ファイルを削除するには、`deleteContentForProductID:`クラスメソッドを使用します。アプリケーションのレシートからプロダクトIDを読み込む方法の詳細については、『*Receipt Validation Programming Guide*』を参照してください。

独自サーバからのコンテンツのダウンロード

アプリケーションとサーバとの連携に関する詳細と、独自のサーバからコンテンツをダウンロードするプロセスの詳細およびメカニズムは、すべてユーザに委ねられています。最小限、通信は次の手順を経ます。

1. アプリケーションからサーバにレシートを送信し、コンテンツを要求します。
2. 『*Receipt Validation Programming Guide*』の説明に従って、サーバはレシートを検証しプロダクトが購入済みであることを確立します。
3. レシートが有効である場合は、サーバはアプリケーションに対してコンテンツで応答します。

アプリケーションでエラーが正常に処理されることを確認します。たとえば、ダウンロード中にディスク容量が不足した場合、不完全なダウンロードを破棄するか、素ペースを確保してからダウンロードを再開するかをユーザが選択できるようにします。

コンテンツをホストする方法、およびアプリケーションがサーバと通信する方法に関しては、セキュリティを実装することを検討してください。詳細については、『*Security Overview*』を参照してください。

トランザクションの終了

トランザクションが終了すると、購入で必要なすべての処理が完了したことがStore Kitに通知されます。終了していないトランザクションは終了するまではキューに残されたままになります。アプリケーションが起動されるたびにアプリケーションによって未完のトランザクションを終了できるように、トランザクションキューのオブザーバが呼び出されます。アプリケーションは、処理結果（成功か否か）にかかわらず、トランザクションをすべて終了させる必要があります。

トランザクションを終了する前には、次のアクションをすべて完了してください。

- 購入の持続
- 関連コンテンツのダウンロード
- ユーザがプロダクトにアクセスできるためのアプリケーションUIの更新

トランザクションを終了するには、支払いキューで`finishTransaction:`メソッドを呼び出します。

```
SKPaymentTransaction *transaction = <# The current payment #>;
[[SKPaymentQueue defaultQueue] finishTransaction:transaction];
```

トランザクションの終了後には、終了したトランザクションでアクションを実行したり、プロダクトを配信したりする動作を一切行わないでください。未完了の動作がある場合は、トランザクションを終了する準備がアプリケーションでできていないことになります。

> **注意:** 終了していないトランザクションをアプリケーションで追跡するための他のメカニズムを使用するため、トランザクションが実際に完了する前に`finishTransaction:`メソッドを呼び出そうとしないでください。Store Kitは、このような方法で使用されるようにはデザインされていません。これを行うと、Appleによってホストされたコンテンツがアプリケーションでダウンロードできなくなり、さらに他の問題にもつながることがあります。

推奨されるテスト手順

正しく実装されていることを検証するために、コードの各部分をテストします。

支払い要求のテスト

既にテスト済みの有効なプロダクトIDを使用して`SKPayment`のインスタンスを作成します。ブレークポイントを設定して、支払い要求を調査します。トランザクションキューに支払い要求を追加し、ブレークポイントを設定してオブザーバの`paymentQueue:updatedTransactions:`メソッドが呼び出されることを確認します。

テスト中は、コンテンツを提供せずに直ちにトランザクションを終了しても構いません。ただしテスト中であっても、終了していないトランザクションがキュー内に残っていると、後で行うテストと干渉するため、トランザクションの終了に失敗すると問題が発生することがあります。

オブザーバコードの検証

トランザクションオブザーバの`SKPaymentTransactionObserver`プロトコルの実装を確認します。アプリケーションのストアUIが現在表示されておらず、また最近購入を行なっていない場合であっても、オブザーバがトランザクションを処理できることを確認します。

コードで`SKPaymentQueue`の`addTransactionObserver:`メソッドが呼び出されている場所を探します。アプリケーションの起動時にこのメソッドが呼び出されていることを検証します。

成功したトランザクションのテスト

テスト用ユーザアカウントでApp Storeにサインインして、アプリケーションで購入を行います。トランザクションキューのオブザーバの`paymentQueue:updatedTransactions:`メソッドの実装にブレークポイントを設定し、トランザクションを調査して、その状態が`SKPaymentTransactionStatePurchased`であることを確認します。

コードで購入を存続している場所にブレークポイントを設定し、購入が成功したときにこのコードが呼び出されることを確認します。User DefaultsまたはiCloudキー値ストアを調査して、正しい情報が記録されていることを確認します。

中断したトランザクションのテスト

トランザクションキューのオブザーバの`paymentQueue:updatedTransactions:`メソッドにブレークポイントを設定し、プロダクトを配信するかどうかを制御できるようにします。続いて、テスト環境で通常どおり購入をし、ブレークポイントを使用してトランザクションを維一時的に無視します。たとえば、LLDBの`thread return`コマンドを使用して、メソッドから直ちに復帰します。

終了してアプリケーションを再起動します。起動後すぐに、Store Kitが`paymentQueue:updatedTransactions:`メソッドを再び呼び出します。このときはアプリケーションで正常に応答します。アプリケーションでプロダクトが正しく配信され、トランザクションが完了したことを確認します。

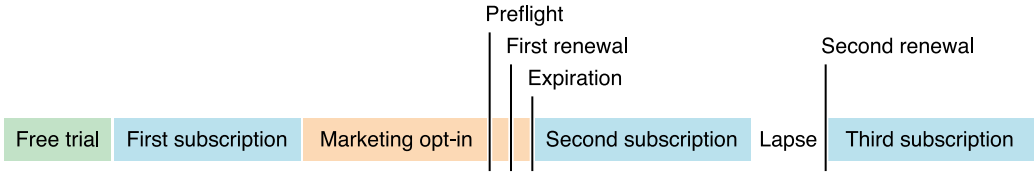
トランザクションが終了したことの確認

アプリケーションで`finishTransaction:`メソッドが呼び出されている場所を特定します。このメソッドが呼び出される前に、トランザクションに関係するすべての作業が完了していることを確認し、成功か否かにかかわらず、このメソッドが呼び出されることを確認します。

購読の取り扱い

購読を使用するアプリケーションについては、追加の動作と考慮事項があります。購読には時間の要素が組み込まれているため、アプリケーションには購読が現在有効であるかどうかと、サブスクリプションが有効だったのが過去のどの期間であったのかを判断する適切なロジックが備わっている必要があります。アプリケーションは新規の購読と購読の更新にも対応し、また期限切れの購読も適切に処理する必要があります。図 5-1に、アプリケーションで処理する必要のある複雑な部分も含めた、購読のタイムラインの例を示します。

図 5-1 購読タイムラインの例



購読の有効期間の計算

アプリケーションでは、購読が有効な期間に基づいて、ユーザがアクセスできるコンテンツを決定する必要があります。たとえば、表 5-1に示すタイムラインに従い、毎月初日に最新号が出る雑誌を購読する、という状況を考えてみましょう。

表 5-1 購読例のタイムライン

日付	イベント
2月1日	2月号発売。まだ購読を始めていないので提供しない（購読開始後に提供）。
2月20日	購読を開始（期間は1か月）。最新である2月号をすぐに提供する。
3月1日	3月号発売。購読中なのですぐに提供する。
3月20日	購読の自動更新（1か月）。
4月1日	4月号発売。購読中なのですぐに提供する。
4月20日	ユーザーが購読の更新を取りやめ、購読期間が終了。
5月1日	5月号発売。購読期間が終了しているので提供しない。

日付	イベント
6月1日	6月号発売。購読期間が終了しているので、この時点では提供しない（再び購読を開始した後に提供）。
6月17日	ユーザーが再び購読を開始。6月号をすぐに提供する。
7月1日	7月号発売。購読中なのですぐに提供する。

このロジックをアプリケーションに実装するには、各コンテンツが発行された日付を記録しておきます。各レシートエントリの「Original Purchase Date」および「Subscription Expiration Date」フィールドを確認し、購読の開始日と終了日を特定します。（レシートについては『*Receipt Validation Programming Guide*を参照）。ユーザーは、購読を購入した時点で最初にアンロックされたコンテンツに加えて、購読期間の各開始日と各終了日の間に発行されたすべてのコンテンツにアクセスすることができます。購読が失効した場合、購読がアクティブな期間が複数あることになり、購読期間の開始時にコンテンツがアンロックされます。

> **注意:** 購入日に購読期間を加えて購読期間を計算しないでください。このやり方では、無料トライアル期間、マーケティング目的の期間、およびユーザーが購読を購入した直後にコンテンツが利用可能になる点が考慮されていません。
>
> たとえば、購読を開始すればロックが解除されるので、月刊誌を1か月購読するだけで、2号分を読めることになってしまいます。開始した時点で前月号のロックが解除され、続いて今月号の発売時点でこちらも解除されるからです。

表 5-1に示した例を続けましょう。レシートには開始日と終了日が次のように記載されます。

2月20日～3月20日

3月20日～4月20日

（4月20日から6月17日についてはレシートに記載なし）

6月17日～7月17日

ユーザーは、購読を開始/再開した時点でロックが解除される、2月号と6月号を読めます。

さらに、3月号、4月号、7月号も読めます。それぞれの発売日時点で購読中だからです。

期限切れと更新

更新プロセスは、期限切れの10日前から開始される「事前」チェックから始まります。この10日の間に、たとえば、顧客に有効な支払い方法があるかどうか、ユーザがこの購読を購入してからプロダクトの価格が上昇していないかどうか、またはプロダクトが利用できなくなっていないかどうかなど、購読の自動更新が遅れたり更新できなくなる可能性のある問題がApp Storeによってチェックされます。問題がある場合はApp Storeによってユーザに通知されるため、購読の更新が必要になる前に問題を解決することができ、購読が中断されないようにします。

> **注意:** 購読の価格が上昇しても、この10日以内に購読が期限切れとなる顧客を除いて、自動更新が自動的に無効になることはありません。この価格の改定が間違いであった場合に、元の価格に戻しても、影響を受けるユーザはいません。この変更が意図的なものであり、新しいより高い価格が維持される場合は、残りのユーザが10日間の更新期間に入っていくにつれ、自動更新は順番に無効になっていきます。

購読が期限切れになる前の24時間の期間中に、App Storeは購読の自動更新を開始します。App Storeは一定期間、数回にわたって購読の自動更新を試みますが、失敗する回数が多いと最終的に更新を停止します。

App Storeは購読期間に隙間が生じないように、期限切れになる少し前に更新を行います。ただし、これでも隙間が生じる可能性があります。たとえば、ユーザの支払情報が有効でなかった場合、最初の更新は失敗します。購読が期限切れになるまでにユーザが支払情報を更新しなかった場合は、購読が期限切れになった日付と、これに続いて自動更新が成功した日付との間に短い隙間が生じます。ユーザは自動更新を無効にして、購読を意図的に期限切れにしてから、後日、購読を更新して、購読期間に長い間隔を空けることができます。アプリケーションの購読ロジックでは、いろいろな長さの購読期間の間隔を正しく扱えるようにしてください。

購読の更新に成功すると、Store Kitはトランザクションキューに更新用のトランザクションを追加します。アプリケーションは起動時にトランザクションキューをチェックして、他のトランザクションと同じ方法で更新の処理を行います。購読の更新時にアプリケーションが既に実行中であった場合は、トランザクションのオブザーバは*呼び出されません*。アプリケーションは次回起動されたときに更新を認識します。

キャンセル

購読の購入時には全額が支払われ、Appleカスタマーサービスにご連絡されることによってのみ払い戻されます。たとえば、ユーザが間違って違うプロダクトを購入した場合、カスタマーサポートはこの購読をキャンセルして払い戻しを行うことができます。購読期間の途中で気が変わって、残りの期間について支払わないということはできません。

購入がキャンセルされたかどうかをチェックするには、レシートの「Cancellation Date」フィールドを確認してください。フィールドに日付が入力されていた場合は、購読が期限切れになる日付にかかわらず、購入はキャンセルされています。キャンセルされたレシートは、購入が行われなかったものとして扱われます。

プロダクトのタイプに応じて、現在有効な購読のみをチェックできたり、過去のすべての購読をチェックする必要があったりします。たとえば、雑誌のアプリケーションでは、ユーザがアクセスできたのがどの号であったのかを判断するために、過去のすべての購読についてチェックする必要があります。

プラットフォームをまたがる場合の考慮事項

プロダクトIDは、1つのアプリケーションに対して関連付けられています。アプリケーションにiOSバージョンとOS Xバージョンの両方がある場合は、各プラットフォームに対して、独立したプロダクトに独立したプロダクトIDを与えます。iOSバージョンのアプリケーションで購読を購入しているユーザがOS Xバージョンのアプリケーションから(またはこの逆から)コンテンツにアクセスできるようにする機能の実装は、アプリケーションの開発者に委ねられています。これには、非更新購読を使用するアプリケーションに実装されるものと同様の、ユーザを識別してユーザが購読したコンテンツの記録を取るための何らかのシステムが必要になります。

ユーザーによる購読管理

アプリケーションに購読管理用のUIを実装せず、次のURLを開く形にしても構いません。

```
https://buy.itunes.apple.com/WebObjects/MZFinance.woa/wa/manageSubscriptions
```

このURLを開くとiTunesまたはiTunes Storeが起動し、「Manage Subscription」ページが現れます。

テスト環境

テストに関しては、自動更新購読の本番環境とテスト環境の間には動作の点で多少の相違があります。

更新が発生する頻度が速くなり、自動更新購読が1日で最大6回更新されます。これにより、購読の更新、購読の失効、購読期間に間隔がある購読履歴がアプリケーションでどのように処理されるかをテストできます。

更新と期限切れの頻度が増しているために、購読期間に短い間隔を残したまま、システムが購読の更新を実行しようとする前に、購読が期限切れになる場合があります。本番環境でも、このような間隔が生じるのには多くの理由があります。アプリケーションでこれらを正しく処理できることを確認してください。

購入したプロダクトの復元

ユーザは購入済みのコンテンツに引き続きアクセスするためにトランザクションを復元します。たとえば、新しい携帯電話にアップグレードする場合、古い携帯電話で購入したアイテムはいずれも失われません。「Restore Purchases」ボタンなど、ユーザが購入を復元できる仕組みをアプリケーションに組み込んでください。購入の復元では、ユーザのApp Storeクレデンシャルが求められ、アプリケーションのフローが中断されます。そのため、(特にアプリケーションが起動されるたびに)購入が自動的に復元されるようにはしないでください。

ほとんどの場合、アプリケーションで必要な処理は、レシートの更新とレシートに含まれるプロダクトの配信のみです。更新されたレシートには、このデバイスまたはほかのデバイス上においてユーザがこのアプリケーションで行った購入記録が含まれています。ただし、アプリケーションによっては、次のような理由から別の方法を取る必要があります。

- Appleによってホストされているコンテンツを使用している場合、完了したトランザクションを復元すると、コンテンツをダウンロードするためのトランザクションオブジェクトがアプリケーションに与えられます。
- アプリケーションのレシートが使用できない、iOS 7より前のiOSバージョンをサポートする必要がある場合、完了したトランザクションを復元してください。
- アプリケーションが非更新購読を使用する場合、アプリケーションで復元プロセスを処理します。

レシートを更新すると、App Storeにレシートの最新コピーが要求されます。レシートを更新しても、新しいトランザクションは作成されません。何度も続けて更新することは避けるべきですが、この操作は1回だけ更新するのと同じ結果をもたらします。

完了したトランザクションを復元すると、ユーザが完了したトランザクションごとに新しいトランザクションが作成され、トランザクションキューのオブザーバの履歴が再現されます。トランザクション復元中、アプリケーションは完了したトランザクションを復元している理由と処理方法を把握するために状態を維持します。何度も復元すると、完了したトランザクションごとに復元したトランザクションが複数作成されます。

> **注意:** ユーザーがアプリケーションの復元インタフェースを使用する代わりに購入済みのプロダクトを購入しようとすると、App Storeによって復元トランザクションではなく正常なトランザクションが作成されます。そのプロダクトについて、ユーザが二重請求されることはありません。これらのトランザクションは、元のトランザクションとまったく同じように扱ってください。

再ダウンロードできるコンテンツについて、適切なレベルのコントロールをユーザに提供します。たとえば、日刊の新聞を3年分、数百メガバイト分のゲームレベルをまとめてダウンロードすることは避けてください。

Appレシートの更新

レシート更新要求を作成し、デリゲートを設定して、要求を開始します。要求では、期限切れのレシートなど、テスト中のさまざまな状態のレシートの取得のためのオプションのプロパティがサポートされます。詳細については、`SKReceiptRefreshRequest`の`initWithReceiptProperties:`メソッドの値を参照してください。

```
request = [[SKReceiptRefreshRequest alloc] init];
request.delegate = self;
[request start];
```

レシートの更新後、そのレシートを確認し、追加されているプロダクトを配信します。

完了したトランザクションの復元

アプリケーションは`SKPaymentQueue`の`restoreCompletedTransactions`メソッドを呼び出すことでプロセスを開始します。アプリケーションのすべての完了したトランザクションを復元するため、App Storeに要求を送信します。アプリケーションが支払い要求の`applicationUsername`プロパティに値を設定している場合(“異常アクティビティの検知” （21 ページ）を参照)、`restoreCompletedTransactionsWithApplicationUsername:`メソッドを使用してトランザクション復元時と同じ情報を提供します。

以前に完了したトランザクションごとに、App Storeが新しいトランザクションを生成します。復元されたトランザクションには、元のトランザクションへの参照が含まれています。`SKPaymentTransaction`のインスタンスには`originalTransaction`プロパティがあり、レシートのエントリには「Original Transaction Identifier」フィールドがあります。

> **注意:** 復元された購入では、日付フィールドの意味が多少異なります。詳細については、『*Receipt Validation Programming Guide*』の「Purchase Date」および「Original Purchase Date」フィールドに関する項目を参照してください。

トランザクションキューのオブザーバは復元された各トランザクションの`SKPaymentTransactionStateRestored`状態で呼び出されます。これについては、"App Storeがトランザクションを処理するのを待機する"（23 ページ）で説明します。この時点での対応は、アプリケーションの設計によって異なります。

- アプリケーションがアプリケーションレシートを使用し、Appleによってホストされたコンテンツがない場合、アプリケーションは完了したトランザクションを復元しないため、このコードは不要です。復元されたトランザクションをすぐに終了してください。
- アプリケーションがアプリケーションレシートを使用し、Appleによってホストされたコンテンツがある場合、復元プロセスを開始する前にユーザが復元するプロダクトを選択できるようにします。復元中、ユーザが選択したコンテンツを再ダウンロードし、ほかのトランザクションはすぐに終了します。

```
NSMutableArray *productIDsToRestore = <# From the user #>;
SKPaymentTransaction *transaction = <# Current transaction #>;


if ([productIDsToRestore containsObject:transaction.transactionIdentifier])
 {
    // Re-download the Apple-hosted content, then finish the transaction
    // and remove the product identifier from the array of product IDs.
} else {
    [[SKPaymentQueue defaultQueue] finishTransaction:transaction];
}
```

- アプリケーションがアプリケーションレシートを使用しない場合、復元時にすべての完了したトランザクションが調査されます。元の購入ロジックと同様のコードパスを使用してプロダクトを使用可能な状態にし、トランザクションを終了します。

  多数のプロダクト(特に関連付けられたコンテンツがあるプロダクト)があるアプリケーションでは、すべてをまとめて復元するのではなく、ユーザが復元するプロダクトを選択できるようにします。このようなアプリケーションは、復元時に処理が必要な完了したトランザクションと、すぐに終了して無視できるトランザクションを把握します。

アプリケーション審査の準備

テストが終了すると、アプリケーションを審査のために登録する準備が整ったことになります。この章では、審査プロセスについてのいくつかのヒントに焦点を当てます。

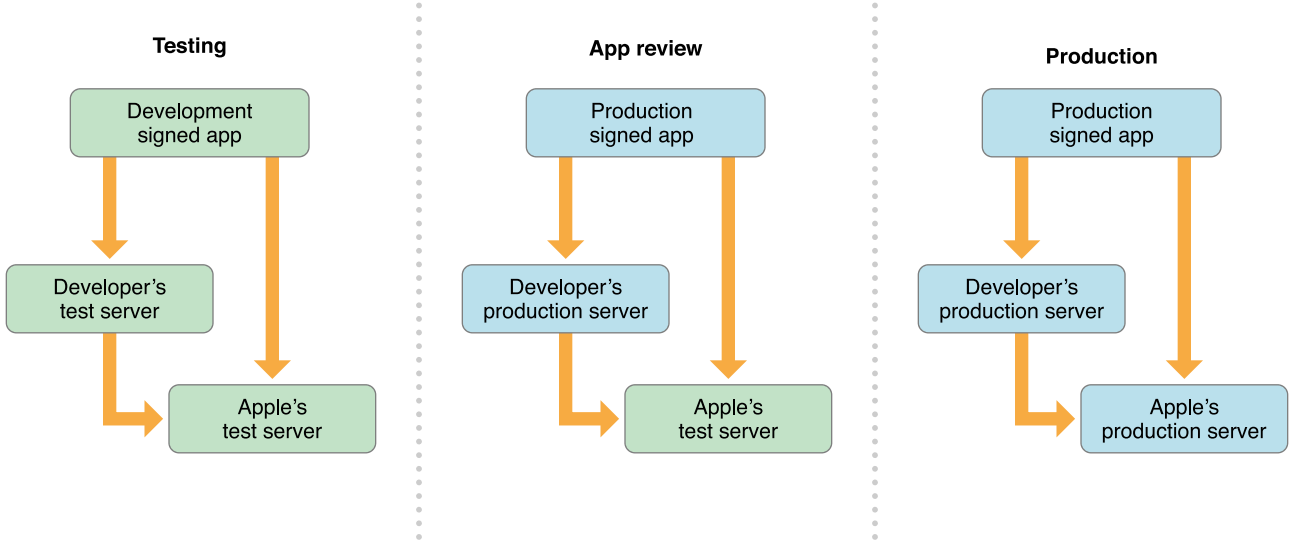
プロダクトを審査用に登録する

アプリケーションを初めて審査のために登録する場合は、同時に審査を受けるためにアプリケーション内プロダクトも登録する必要があります。最初の登録後は、審査のためのアプリケーションおよびプロダクトに対する更新の登録をそれぞれ独自に行うことができます。詳細については、『*In-App Purchase Configuration Guide for iTunes Connect*』を参照してください。

テスト環境のレシート

図 7-1に示すように、アプリケーションは、開発、審査、および本番の各段階において異なる環境で実行されます。

図 7-1 開発、審査、本番の各環境



開発の段階では、ユーザは開発用として署名されたバージョンのアプリケーションを実行します。このアプリケーションはユーザのテスト用サーバとApp Storeのテスト用環境に接続します。本番では、ユーザは本番用として署名されたバージョンのアプリケーションを実行します。このアプリケーションはユーザの本番用サーバと本番のApp Storeに接続されます。ただしアプリケーションの審査期間中は、アプリケーションは本番用の環境とテスト用の環境が混ざった状態で実行されます。つまりアプリケーションは本番用として署名され、本番用のサーバに接続されますが、App Storeのテスト用の環境に接続されます。

ユーザのサーバでレシートを検証する場合、サーバでは本番用として署名されたアプリケーションを処理して、そのレシートをAppleのテスト用環境から受け取ることができる必要があります。本番用のサーバでお勧めできるアプローチは、最初から本番のApp Storeで常にレシートの検証を開始するというものです。検証が「Sandbox receipt used in production（サンドボック用のレシートが本番で使用）」というエラーコードで失敗したら、本番用の代わりにテスト用の環境に対する検証を行います。

実装におけるチェックリスト

審査を受けるためにアプリケーションを登録する前に、必要な動作がすべて実装されていることを確認してください。アプリケーション内での購入のための次のコア動作(一般的な開発プロセスの順に記載)が実装されていることを確認します。

- iTunes Connectでプロダクトを作成し、設定します。

  プロダクトの変更はプロセス全体で可能ですが、コードをテストするには1つ以上のプロダクトが設定されている必要があります。

- アプリケーションバンドルまたは独自のサーバからプロダクトIDのリストを取得します。`SKProductsRequest`のインスタンスを使用して、そのリストをApp Storeに送信します。

- App Storeから返された`SKProduct`のインスタンスを使用して、App Store向けのユーザインタフェースを実装します。最初はテーブルビューや少数のボタンなどの単純なインタフェースから開発を始めましょう。

  開発プロセスの適切な段階で、App Store向けの最終ユーザインタフェースを実装します。

- `SKPaymentQueue`の`addPayment:`メソッドを使用して`SKPayment`のインスタンスをトランザクションキューに追加することで、支払いを要求します。

- `paymentQueue:updatedTransactions:`メソッドからトランザクションキューのオブザーバを実装します。

  開発プロセスの適切な段階で、`SKPaymentTransactionObserver`プロトコルの他のメソッドを実装します。

- 購入したプロダクトを配信するには、アプリケーションの今後の起動に備えて購入記録を持続し、関連するコンテンツをダウンロードした後、`SKPaymentQueue`の`finishTransaction:`メソッドを呼び出します。

  開発では、最初にこのコードの簡易版(たとえば、画面に「プロダクト配信済み」と表示するもの)を実装できます。その後、開発プロセスの適切な段階で実際のコードを実装します。

アプリケーションで非消耗型プロダクトや自動更新購読、非更新購読を販売する場合、次の復元ロジックが実装されていることを確認してください。

- 復元プロセスを開始するためのUIを提供します。
- `SKReceiptRefreshRequest`クラスを使用してAppレシートを更新するか、`SKPaymentQueue`クラスの`restoreCompletedTransactions`メソッドを使用して完了したトランザクションを復元することで、過去の購入情報を取得します。
- ユーザがコンテンツを再ダウンロードするようにします。

  Appleによってホストされたコンテンツを使用している場合、完了したトランザクションを復元し、トラン策ザクションの`downloads`プロパティを使用して`SKDownload`のインスタンスを取得します。

  独自のコンテンツを使用している場合、独自のサーバに対して適切な呼び出しを実行します。

アプリケーションで自動更新購読または非更新購読を販売する場合、次の購読ロジックが実装されていることを確認してください。

- 最新の発行済みコンテンツ(雑誌の最新号など)を配信することで、新しく購入した購読を処理します。
- 新規コンテンツが発行された場合、ユーザが参照できるようにします。
- 購読の有効期限が切れた場合、ユーザが更新するようにします。

  アプリケーションで自動更新購読を販売する場合、App Storeがこのプロセスを処理します。アプリケーションで処理しないようにしてください。

  アプリケーションで非更新購読を販売する場合、アプリケーションでこのプロセスを処理します。
- 購読が無効になった時点で、新規コンテンツの提供を停止します。購読を再度有効にして購入するオプションがユーザに表示されるよう、インタフェースを更新します。
- コンテンツが発行された場合に追跡するためのシステムを実装します。購入を復元する際には、このシステムを使用して、購読がアクティブだった期間に基づき、ユーザが支払い済みのコンテンツにアクセスできるようにします。

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
2015-10-21	SKProductRequestのインスタンスを保持することに関する情報を追加しました。
2013-10-22	プロダクトの配信に関する説明を拡張しました。復元に関する章を追加しました。全体を通して細かな変更を行いました。 "支払いの要求"（20 ページ）に`applicationUsername`プロパティの説明を追加しました。"プロダクトの配信"（23 ページ）で、購入の持続およびコンテンツのダウンロードに関する説明を拡張しました。"購入したプロダクトの復元"（40 ページ）の章を追加しました。"アプリケーション審査の準備"（43 ページ）に実装チェックリストを追加しました。
2013-09-18	全体的に内容を拡張しわかりやすくしました。 レシートの検証に関する情報は、『*Receipt Validation Programming Guide*』に移動されました。
2012-09-19	expires_dateキーは復元されたトランザクションには存在しないという注意が削除されました。 復元されたトランザクションのexpires_dateきーについては、自動更新購読のベストプラクティスで簡単に触れられています。自動更新行動の復元についてのセクションを削除しました。
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。

日付	メモ
	「アプリケーションへのStoreの追加」で、非推奨になった `paymentWithProductIdentifier:` メソッドを `paymentWithProduct:` メソッドで置き換えました。
2012-01-09	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 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。
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。その他の細かな更新を行いました。

日付	メモ
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.

Apple Japan
〒106-6140 東京都港区六本木 6
丁目10番1号 六本木ヒルズ
http://www.apple.com/jp

Offline copy. Trademarks go here.