Local NotificationおよびPush Notificationプログラミングガイド

iPhone > User Experience

2011-08-09

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/

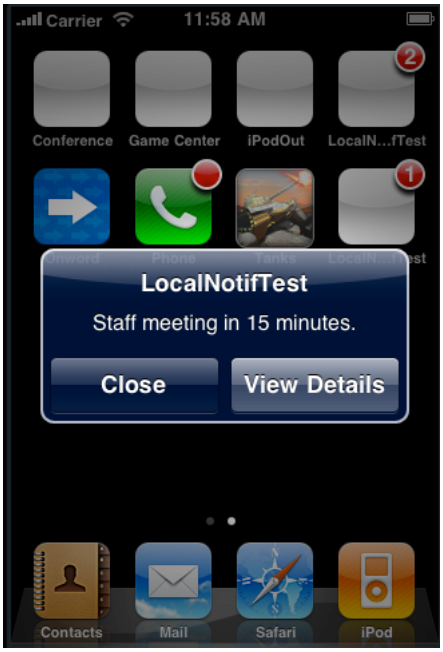
目次

図、表、リスト

Local NotificationおよびPush Notificationの概要

Local NotificationおよびPush Notificationは、フォアグラウンドで実行されていないアプリケーションが、ユーザに対して情報を知らせるための通知です。その情報は、メッセージ、間近のカレンダーイベント予定、またはリモートサーバ上の新しいデータの存在などの可能性があります。オペレーティングシステムによって表示される場合、Local NotificationおよびPush Notificationの見た目やサウンドは同じです。それらの通知は、警告メッセージまたはアプリケーションアイコンのバッジを表示できます。警告や数字付きバッジを表示するときに、サウンドを再生することもできます。

Push NotificationはiOS 3.0およびMac OS X v7.0で導入されました。Local NotificationsはiOS 4.0で導入されました。Mac OS Xでは使えません。



アプリケーションからのメッセージ、イベント、あるいはそのほかのデータがあることがユーザに通知されると、ユーザはアプリケーションを起動してその詳細を表示できます。また、通知を無視することも選べますが、その場合、アプリケーションはアクティブになりません。

注： Push NotificationおよびLocal Notificationは、通知のブロードキャスト（`NSNotificationCenter`）やキー値監視通知との関連は*ありません*。

At a Glance

Local NotificationとPush Notificationにはいくつか、知っておくべき重要な問題があります。

Local NotificationおよびPush Notificationが解決する問題

フォアグラウンドでは、1度に1つのアプリケーションしかアクティブになれません。多くのアプリケーションは、時間ベースまたは相互接続された環境で動作しているため、アプリケーションがフォアグラウンドにないときに、ユーザが関心を持っているイベントが発生する可能性があります。Local NotificationおよびPush Notificationにより、アプリケーションは、これらのイベントが発生したことをユーザに通知できるようになります。

関連する章： 「Local NotificationおよびPush Notificationの詳細」 （11 ページ）

Local NotificationとPush Notificationは発生源が異なる

Local NotificationとPush Notificationの設計上のニーズは異なります。Local Notificationは、iPhone、iPad、iPod touch上のアプリケーションによって手元で発生します。Push Notification（*Remote Notification*とも呼ばれる）は、デバイスの外部から届きます。Push Notificationは、表示すべきメッセージやダウンロードすべきデータがあるときに、リモートサーバ（アプリケーションのプロバイダ）上で発生して、デバイス上のアプリケーションに（Apple Push Notificationサービス経由で）配信（プッシュ）されます。

関連する章： 「Local NotificationおよびPush Notificationの詳細」 （11 ページ）

Local Notificationのスケジューリング、Push Notificationの登録、および両方の通知の処理

iOSから後でLocal Notificationが配信されるようにするには、アプリケーションは`UILocalNotification`オブジェクトの作成、配信日時の割り当て、表示の詳細の指定、そしてスケジューリングを行います。Push Notificationを受け取るには、アプリケーションは通知を受け取るための登録を行い、オペレーティングシステムから取得したデバイストークンをプロバイダに渡す必要があります。

オペレーティングシステムからLocal Notification（iOSのみ）またはPush Notification（iOSまたはMac OS X）が配信され、対象アプリケーションがフォアグラウンドで実行されていない場合、オペレーティングシステムは通知（警告、数字付きバッジアイコン、サウンド）を提示します。通知警告が提示され、ユーザがアクションボタンをタップまたはクリックすると（またはアクションスライダを動かすと）、アプリケーションが起動し、メソッドが呼び出され、Local Notificationオブジェクトまたは Remote Notificationペイロードが渡されます。通知が配信されたときにアプリケーションがフォアグラウンドで実行中であれば、アプリケーションデリゲートはLocal notificationまたはPush Notificationを受け取ります。

関連する章： 「通知のスケジューリング、登録、処理」（17 ページ）

Apple Push NotificationサービスはPush Notificationのゲートウェイ

Apple Push Notificationサービス（APNs）により、これらの通知を受け取るための登録を行ったアプリケーションを持つデバイスに、Push Notificationが伝達されます。各デバイスは、このサービスとの間で認証済みの暗号化されたIP接続を確立し、この永続的な接続を介して通知を受信します。プロバイダは、対象となるクライアントアプリケーション宛ての受信データを監視している間、永続的でセキュアなチャンネルを介してAPNsに接続します。アプリケーション宛ての新しいデータを受信すると、プロバイダは通知を作成してこのチャンネルを介してAPNsに送信します。APNsは、その通知をターゲットデバイスに配信（プッシュ）します。

関連する章： 「Apple Push Notificationサービス」（27 ページ）

Push Notificationのセキュリティ認証情報の取得

Push Notificationを行うためにアプリケーションのプロバイダ側を開発してデプロイするには、適切なDev CenterからSSL証明書を取得する必要があります。証明書は、バンドルIDで識別される1つのアプリケーションに対して1つと限定されています。また、証明書はサンドボックス（開発とテスト用）および製品の2つの環境のいずれか1つに限定されています。これらの環境は固有のIPアドレスを割り当てられており、固有の証明書を必要とします。また、これらの環境のそれぞれにプロビジョニングプロファイルを取得する必要もあります。

関連する章： 「プロビジョニングおよび開発」（41 ページ）

プロバイダによるバイナリインターフェイス経由でのAPNsとの通信

バイナリインターフェイスは非同期であり、Push Notificationをバイナリコンテンツとして APNsに送信するために、TCPソケットを介するストリーミング設計を用います。サンドボックス環境と製品環境は別々のインターフェイスがあり、それぞれ固有のアドレスとポートを持ちます。インターフェイスごとに、TLS（またはSSL）およびSSL証明書を使用して、セキュアな通信チャンネルを確立する必要があります。プロバイダは、それぞれの送信通知を作成し、このチャンネル経由でAPNsに送信します。

APNsには、配信に失敗したデバイス（つまり、APNsがデバイス上のアプリケーションへPush Notificationを配信できなかった対象デバイス）のアプリケーションごとのリストを管理するフィードバックサービスがあります。プロバイダは、定期的にフィードバックサービスに接続し、配信が失敗したデバイスへのPush Notificationの送信を停止できるように、失敗が続いているデバイスを確認するべきです。

関連する章： 「Apple Push Notificationサービス」 （27 ページ）、「プロバイダとApple Push Notificationサービスの通信」 （45 ページ）

Prerequisites

Local Notification、およびPush Notificationのクライアント側を実装するには、iOSのアプリケーション開発に精通していることが前提になります。プロバイダ側の実装には、TLS/SSLおよびストリーミングソケットに関する知識が役立ちます。

関連項目

次の情報は、Local NotificationおよびPush Notificationの理解および実装の参考になります。

- `UILocalNotification`、`UIApplication`、および`UIApplicationDelegate`のリファレンス文書では、iOSで動作するクライアントアプリケーション用のLocal NotificationおよびPush NotificationのAPIについて説明しています。
- `NSApplication`および`NSApplicationDelegate Protocol`のリファレンス文書では、Mac OS X上で動作するクライアントアプリケーション用のPush Notification APIについて説明しています。
- 『*Security Overview*』では、iOSおよびMac OS Xの両プラットフォームで使われているセキュリティのテクノロジーと手法について説明しています。
- RFC 5246はTLSプロトコルの標準です。

  データプロバイダとApple Push Notificationサービス間のセキュアな通信には、Transport Layer Security（TLS）、またはその前身のSecure Sockets Layer（SSL）に関する知識が必要です。詳細については、これらの暗号プロトコルのオンラインまたは印刷版解説書を参照してください。

Local NotificationおよびPush Notificationの詳細

Local NotificationおよびPush Notificationの目的の本質は、アプリケーションがフォアグラウンドで実行中でないときに、ユーザに対して何らかの情報（メッセージ、次の予約など）があることの通知をアプリケーションが行えるようにすることです。Local NotificationとPush Notificationの間の本質的な相違はシンプルです。

- Local Notificationは、アプリケーションによってスケジューリングされ、同じデバイス上のiOSによって配信されます。

  Local NotificationはiOSでのみ利用できます。

- Push Notification（Remote Notificationとも呼ばれる）は、アプリケーションのリモートサーバ（アプリケーションのプロバイダ）によってApple Push Notificationサービスに送信され、そこからアプリケーションがインストールされているデバイスに通知が配信（プッシュ）されます。

  Push Notificationは、iOSのほか、v10.7（Lion）以降のMac OS Xでも利用できます。

以降の各セクションでは、Local NotificationとPush Notificationの共通点と相違点について説明します。

注： Push NotificationおよびLocal Notificationを使用する際のガイドラインについては、『*iOS Human Interface Guidelines*』の「Enabling Push Notifications」 in *iOS Human Interface Guidelines*を参照してください。

ユーザからは同じに見えるPush NotificationとLocal Notification

ユーザの視点からは、Push NotificationとLocal Notificationはまったく同じように見えます。しかしそれは、ユーザが関心を持っている情報があることを（現在フォアグラウンドで実行されていない可能性のある）アプリケーションのユーザに通知するという目的が同じだからです。

たとえば、iPhoneを使用している様子（電話をかけたり、インターネットサーフィンをしたり、音楽を聴いたりするなど）を想像してください。iPhoneにはチェスのアプリケーションがインストールされており、あなたは遠隔地でプレイしている友人とゲームを始めることにします。あなたは先手を打ちます（これは、ゲームのプロバイダによって正しく認識されます）。次に、電子メールを読むためにこのクライアントアプリケーションを終了します。その間に、友人があなたの手に反撃します。チェスアプリケーションのプロバイダはこの手を認識した後、あなたのデバイス上のチェスアプリケーションが接続されていないことを検知して、Apple Push Notificationサービス（APNs）にPush Notificationを送信します。

ほとんどすぐに、あなたのデバイス（正確には、デバイス上で実行中のオペレーティングシステム）は、APNsからWi-Fi接続または携帯電話接続経由でこの通知を受信します。チェスアプリケーションは現在実行されていないため、iOSは図 1-1に示すような警告を表示します。このメッセージは、アプリケーション名、短いメッセージ、および（この場合は）2つのボタン（「Close」と

「View」）で構成されています。右側のボタンはアクションボタンと呼ばれ、デフォルトのタイトルは「View」です。アプリケーションは、アクションボタンのタイトルをカスタマイズして、ボタンタイトルとメッセージを国際化し、ユーザが設定している言語で表示させることができます。

図 1-1　通知警告

「View」ボタンをタップすると、チェスアプリケーションが起動してプロバイダに接続し、新しいデータをダウンロードします。そして、チェス盤のユーザインターフェイスを調整して友人の手を表示します（「Close」をタップすると、この警告が消えます）。

Mac OS Xにおける注意事項： 現在、Mac OS X上で、動作していないアプリケーションに対して使えるPush Notificationのタイプは、アイコンバッジだけです。すなわち、Dock内のアプリケーションアイコンがバッジになるのは、当該アプリケーションが動作していない場合のみです。まだDockにアイコンを置いていなければ、システムがアイコンをDockに置いて、バッジできるようにします（その後、アプリケーションが停止した時点で削除）。動作中のアプリケーションは、ほかのタイプの通知（警告表示や警告音）の通知ペイロードを調べ、適切に処理できます。

要求の異なる別のタイプのアプリケーションを考えてみましょう。このアプリケーションはToDoリストを管理し、リストの各項目には完了しなければならない日時があります。ユーザは、この期日に達するまで、特定の間隔で通知することをアプリケーションに要求できます。これを達成するため、アプリケーションはその日時にLocal Notificationをスケジューリングします。警告メッセージを指定する代わりに、今度はアプリケーションは数字（1）付きバッジを指定することを選びます。予定の時間になると、iOSによって、アプリケーションのアイコンの右上隅に数字付きバッジが表示されます。その様子を図 1-2に示します。

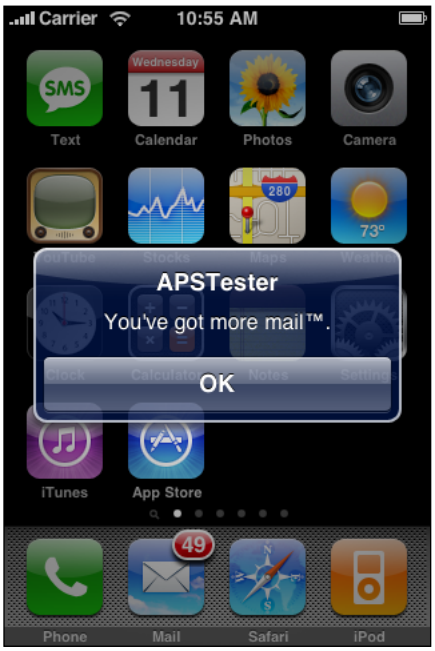
Local Notification、Push Notificationとも、バッジの数字はアプリケーション固有のもので、どんな事柄の数を表してもかまいません（たとえば、間近なカレンダーイベントの数、ダウンロード可能なデータ項目の数、ダウンロード済みだけれど未読の電子メールメッセージの数など）。ユーザがバッジを見て、アプリケーションアイコンをタップする（Mac OS Xの場合はドックないのアイコンをクリックする）と、アプリケーションが起動してToDo項目を表示するなd、ユーザにとって望ましい動作をします。

図 1-2　数字付きバッジが表示されたアプリケーションアイコン（iOS）

iOSでは、アプリケーションは、警告メッセージや数字付きバッジと一緒に、サウンドファイルを指定できます。サウンドファイルには、短く、特徴的なサウンドを含めるようにします。iOSは警告の表示またはアイコンへのバッジの付加と同時に、通知の受信をユーザに知らせるために警告音を再生します。

通知警告は、2つではなく1つのボタンを持つことができます。ボタンを1つにした場合は、図 1-3に示すように、アクションボタンが表示されません。この種類の警告に対してユーザができることは、閉じることだけです。

図 1-3 アクションボタンが非表示にされた通知警告メッセージ



オペレーティングシステムは、アプリケーションが実行されているかどうかに関わらずアプリケーションにLocal NotificationまたはPush Notificationを配信します。通知の到着時にアプリケーションが実行中であれば、デバイスの画面がロックされていても（iOSの場合）、警告の表示、アイコンのバッジ付加、サウンドの再生は行われません。代わりに、アプリケーションデリゲートが通知の知らせを受け、直接処理することができます(（「通知のスケジューリング、登録、処理」 （17 ページ）で、さまざまな配信シナリオの詳細を説明します）。

iPhone、iPadおよびiPod touchデバイスのユーザは、デバイス、またはデバイスにインストールされているアプリケーションがPush Notificationを受信するかどうかを制御できます。アプリケーションごとにPush Notificationのタイプ（アイコンのバッジ、警告メッセージ、警告音）を選択的に有効にしたり無効にしたりもできます。これらの制限は、「設定(Settings)」アプリケーションの「通知(Notifications)」環境設定で設定します。UIKitフレームワークでは、特定のアプリケーションのユーザ環境設定を検出するプログラミングインターフェイスを提供しています。

Local Notificationに関する追加情報

Local Notification（iOSのみ）は、シンプルなカレンダーアプリケーションやToDoリストアプリケーションを含む、時間ベースで動作するアプリケーションに最適です。iOSで許されている短い時間の間にバックグラウンドで動作するアプリケーションも、Local Notificationが役立つかもしれません。たとえば、メッセージやデータをサーバに依存するアプリケーションは、バックグラウンドでの動作中に、受信する項目をサーバにポーリングできます。表示すべきメッセージやダウンロードすべき更新があれば、サーバはすぐにLocal Notificationを提示してユーザにそのことを通知できます。

Local Notificationは、次の3つの一般的な種類のプロパティを持つ、`UILocalNotification`のインスタンスです。

- **スケジューリング**。オペレーティングシステムから通知を配信する日時を指定する必要があります。これは、*配信日（fire date）*とも呼ばれます。ユーザが旅をするときなどにシステムが調整できるように、配信日を特定のタイムゾーンで修飾することができます。また、ある一定の間隔（毎週、毎月など）で、通知をスケジュールしなおすようにオペレーティングシステムに要求することもできます。
- **通知のタイプ**。このカテゴリには、警告メッセージ、アクションボタンのタイトル、アプリケーションアイコンのバッジの数字、および再生するサウンドが含まれます。
- **カスタムデータ**。Local Notificationには、カスタムデータの辞書を含めることができます。

「Local Notificationのスケジューリング」（18 ページ）では、プログラミングの面からこれらのプロパティの詳細を説明します。アプリケーションは、Local Notificationオブジェクトを作成したら、オペレーティングシステムに対してスケジューリングするか、すぐに提示するかのいずれかを行えます。

デバイス上の各アプリケーションは、最も近いものから64件までのLocal Notificationをスケジューリングできます。オペレーティングシステムは、この制限を超える通知は破棄します。また、オペレーティングシステムは、繰り返しの通知は1つの通知と見なします。

Push Notificationに関する追加情報

iOSまたはMac OS X上のアプリケーションは、多くの場合、クライアント/サーバモデルに基づくより大きなアプリケーションの一部に過ぎません。クライアント側のアプリケーションはデバイスまたはコンピュータにインストールされ、サーバ側のアプリケーションはたくさんのクライアントアプリケーションにデータを提供することが主な役割です（このため、サーバ側のアプリケーションは*プロバイダ*と呼ばれます）。クライアントアプリケーションは時々プロバイダに接続して、そのアプリケーション用の更新データをダウンロードします。電子メールアプリケーションやソーシャルネットワーキングアプリケーションは、このようなクライアントサーバモデルの例です。

しかし、ダウンロード可能な更新データがプロバイダにあるときに、アプリケーションがプロバイダに接続されていなかったり、デバイスまたはコンピュータ上で実行すらされていない場合はどうなるでしょうか？アプリケーションはどのようにして更新データのことを知るのでしょうか？Push Notificationがこの悩みを解決します。Push Notificationはプロバイダがデバイスまたはコンピュータのオペレーティングシステムに配信した短いメッセージです。オペレーティングシステムはこれを受信すると、ダウンロード可能なデータ、表示可能なメッセージなどが存在することをクライアントアプリケーションのユーザに通知します。ユーザが（iOSで）この機能を有効にしていて、アプリケーションが正しく登録されていれば、この通知はオペレーティングシステム（アプリケーションの場合もある）に配信されます。Apple Push Notificationサービス（略してAPNs）はPush Notification機能の基本テクノロジーです。

Push Notificationは、デスクトップシステム上のバックグラウンドアプリケーションと同じ働きをしますが、オーバーヘッドが加わることはありません。現在フォアグラウンドで実行されていない（iOSの場合は動作していない）アプリケーションに対しては、間接的に通知が行われます。オペレーティングシステムは、アプリケーションの代わりにPush Notificationを受け取って、ユーザに警告します。警告を受け、ユーザがアプリケーションの起動を選択すると、アプリケーションはプロバイダからデータをダウンロードします。通知を受信したときにアプリケーションが実行されている場合は、アプリケーションはその通知を直接処理することを選ぶこともできます。

iOSにおける注意事項： iOS 4.0からは、アプリケーションをバックグラウンドで実行できますが、実行可能な時間は限られています。フォアグラウンドでは、1度に1つのアプリケーションしか実行できません。

Apple Push Notificationサービス（APNs）は、その名のとおり、デバイスやコンピュータに通知を配信するためにプッシュ型の設計を使用しています。プッシュ型設計がその反対のプル型設計と異なるのは、通知の受け手（この場合はオペレーティングシステム）が、即座に更新情報をポーリングするのではなく、受動的に更新情報を検知する点です。プッシュ型設計によって、情報を広範囲かつタイムリーに配信することが可能になり、プル型設計につきもののスケーラビリティの問題もほとんどありません。APNsでは、Push Notificationを実現するために永続的なIP接続を使用します。

Push Notificationのほとんどは、ペイロード（ユーザへの通知方法を指定するAPNs定義のプロパティを含むプロパティリスト）で構成されています。パフォーマンスの理由から、ペイロードは意図的に小さくしています。ペイロードにカスタムプロパティを定義することもできますが、Push Notificationによる配信は保証されていないため、データ送信の目的でRemote Notificationメカニズムを使用してはいけません。ペイロードの詳細については「通知ペイロード」（34 ページ）を参照してください。

APNsは、デバイスまたはコンピュータ上のアプリケーション宛てにプロバイダから受信した最後の通知を保持しています。したがって、デバイスがオンラインになったときに、その通知をまだ受信していない場合、APNsは保存されている通知をデバイスに配信します。iOSが動作するデバイスは、Wi-Fi接続、携帯電話接続の両方でPush Notificationを受け取ります。Mac OS Xが動作するコンピュータは、Wi-Fi接続、Ethernet接続の両方でPush Notificationを受け取ります。

重要： iOSで、Wi-Fi接続がPush Notificationに使用されるのは、携帯電話接続が存在しない場合、またはデバイスがiPod touchの場合だけです。デバイスによっては、Wi-Fi経由で通知を受信するには、デバイスのディスプレイがオンになっている（つまり、スリープ状態でない）か、デバイスがプラグに接続されていなければなりません。これに対して、iPadでは、スリープ状態でもWi-Fiアクセスポイントと結ばれているので、Push Notificationの配信が可能です。Wi-Fi無線により、すべての受信トラフィックに対して、ホストプロセッサのスリープが解除されます。

アプリケーションにRemote Notification機能を追加するには、iOS、Mac OS Xとも、Dev Centerから適切な証明書を取得し、クライアント側とプロバイダ側の各アプリケーションに必要なコードを記述する必要があります。「プロビジョニングおよび開発」（41 ページ）では、プロビジョニングとセットアップの手順について説明します。また、「プロバイダとApple Push Notificationサービスの通信」（45 ページ）および「通知のスケジューリング、登録、処理」（17 ページ）では、実装の詳細について説明します。

Apple Push Notificationサービスは、不正な動作がないかどうか継続的にプロバイダを監視し、急な活動増加、すばやい接続／切断の繰り返し、およびそれに類似の活動を探します。このような動作を検出した場合、Appleはプロバイダへの通知を試みます。それでも、その動作が続く場合は、そのプロバイダの証明書を失効リストに登録し、それ以降の接続を拒否します。不正または問題となる動作が続くと、プロバイダのAPNsへのアクセスが停止される場合もあります。

通知のスケジューリング、登録、処理

この章では、Local Notificationのスケジューリング、Remote Notificationの登録、Local NotificationとRemote Notificationの両方の処理を行うために、iPhone、iPad、またはiPod touchアプリケーションで行わなければならない（または行う可能性のある）作業について説明します。Push Notificationのクライアント側のAPIでは、Push Notificationは*Remote Notification*と呼ばれるため、この章ではこの用語を使用します。

カスタム警告音の準備

iOSにおけるRemote Notificationに関してデベロッパは、iOSがアプリケーションにLocal NotificationまたはRemote Notificationを提示したときに再生するカスタム警告音を指定できます。このサウンドファイルは、クライアントアプリケーションのメインバンドル内になければなりません。

カスタム警告音はiOSのシステムサウンド機能によって再生されるため、次のいずれかのオーディオデータフォーマットでなければなりません。

- リニアPCM
- MA4（IMA/ADPCM）
- μLaw
- aLaw

このオーディオデータは`aiff`、`wav`、または`caf`ファイルとして保存できます。次に、Xcodeで、このサウンドファイルをアプリケーションバンドルの非ローカライズリソースとしてプロジェクトに追加します。

`afconvert`ツールを使用してサウンドを変換することもできます。たとえば、16ビットリニアPCMのシステムサウンドの`Submarine.aiff`をIMA4オーディオのCAFファイルに変換するには、ターミナルアプリケーションで次のコマンドを使用します。

```
afconvert /System/Library/Sounds/Submarine.aiff ~/Desktop/sub.caf -d ima4 -f
caff -v
```

QuickTime Playerでサウンドを開き、「ウインドウ」メニューから「ムービーインスペクタを表示」を選ぶと、そのサウンドのデータフォーマットを判別できます。

カスタムのサウンドの再生時間は30秒未満でなければなりません。カスタムのサウンドがこの制限を超える場合、代わりにデフォルトのシステムサウンドが再生されます。

Local Notificationのスケジューリング

iOSでLocal Notificationの作成とスケジューリングを行うには、次に示すように、いくつかの簡単な手順が必要です。

1. `UILocalNotification`オブジェクトを割り当て、初期化します。
2. オペレーティングシステムが通知を配信しなければならない日時を設定します。これは、`fireDate`プロパティです。

   `timeZone`プロパティを現在のロケールの`NSTimeZone`オブジェクトに設定した場合、システムは、デバイスが異なるタイムゾーンにわたって移動したときに、自動的に配信日を調整します（タイムゾーンは、指定されたカレンダーと日付の値をシステムが計算する、日付コンポーネント（日、月、時、年、分）の値に影響します）。通知の配信が繰り返し行われるように（毎日、毎週、毎月など）スケジューリングすることもできます。
3. 通知の実体（警告、アイコンの数字付きバッジ、サウンド）を設定します。
   - 警告には、メッセージのプロパティ（`alertBody`プロパティ）と、アクションボタンまたはスライダのタイトルのプロパティ（`alertAction`プロパティ）があります。これらの文字列値はともに、ユーザの現在の言語環境のために国際化することができます。
   - アプリケーションアイコン上に表示する数字付きバッジは、`applicationIconBadgeNumber`プロパティを通して設定します。
   - アプリケーションのメインバンドル内にあるローカライズされていないカスタムサウンドのファイル名は、`soundName`プロパティに割り当てることができます。デフォルトのシステムサウンドを取得するには、`UILocalNotificationDefaultSoundName`プロパティを割り当てます。サウンドは、必ず警告メッセージまたはアイコンのバッジに付随するべきです。そうでない場合は、再生すべきではありません。
4. 必要に応じて、`userInfo`プロパティを通して、通知にカスタムデータをアタッチできます。

   `userInfo`辞書のキーと値は、プロパティリストオブジェクトでなければなりません。
5. Local Notificationの配信をスケジューリングします。

   Local Notificationのスケジュール設定は、`UIApplication`の`scheduleLocalNotification:`メソッドを呼び出すことによって行います。アプリケーションは、`UILocalNotification`オブジェクトに指定されている配信日（fire date）を配信のタイミングとして使用します。あるいは、`presentLocalNotificationNow:`メソッドを呼び出して、すぐに通知を提示することもできます。

リスト 2-1の方法では、架空のToDoリストアプリケーションに設定されているToDo項目の期日が間近であることをユーザに知らせるために、通知の作成とスケジューリングを行っています。これについて、いくつかの注意すべき点を示します。`alertBody`プロパティおよび`alertAction`プロパティに関しては、ユーザによって設定されている言語環境に対応して、メインバンドル（`NSLocalizedString`マクロを介して）からローカライズされた文字列がフェッチされます。また、`userInfo`プロパティに割り当てられている辞書に、関連するToDo項目の名前も追加されます。

リスト 2-1　Local Notificationの作成、設定、スケジューリング

```
- (void)scheduleNotificationWithItem:(ToDoItem *)item interval:(int)minutesBefore
 {
    NSCalendar *calendar = [NSCalendar autoupdatingCurrentCalendar];
    NSDateComponents *dateComps = [[NSDateComponents alloc] init];
    [dateComps setDay:item.day];
    [dateComps setMonth:item.month];
    [dateComps setYear:item.year];
    [dateComps setHour:item.hour];
    [dateComps setMinute:item.minute];
    NSDate *itemDate = [calendar dateFromComponents:dateComps];
    [dateComps release];

    UILocalNotification *localNotif = [[UILocalNotification alloc] init];
    if (localNotif == nil)
        return;
    localNotif.fireDate = [itemDate addTimeInterval:-(minutesBefore*60)];
    localNotif.timeZone = [NSTimeZone defaultTimeZone];

   localNotif.alertBody = [NSString stringWithFormat:NSLocalizedString(@"%@ in
 %i minutes.", nil),
         item.eventName, minutesBefore];
    localNotif.alertAction = NSLocalizedString(@"View Details", nil);

    localNotif.soundName = UILocalNotificationDefaultSoundName;
    localNotif.applicationIconBadgeNumber = 1;

    NSDictionary *infoDict = [NSDictionary dictionaryWithObject:item.eventName
 forKey:ToDoItemKey];
    localNotif.userInfo = infoDict;

    [[UIApplication sharedApplication] scheduleLocalNotification:localNotif];
    [localNotif release];
}
```

スケジューリングされた特定の通知をキャンセルするには、アプリケーションオブジェクトの`cancelLocalNotification:`を呼び出します。また、スケジューリングされたすべての通知をキャンセルするには、`cancelAllLocalNotifications`を呼び出します。これらのメソッドはともに、現在表示されている通知警告もプログラム上で閉じます。

Local Notificationが有用なケースとしては、アプリケーションがバックグラウンドで実行されているときに、ユーザが関心を持つようなメッセージ、データ、そのほかの項目を受信した場合が考えられます。その場合は、`UIApplication`の`presentLocalNotificationNow:`メソッドを使用して、すぐに通知を提示すべきです（iOSでは、アプリケーションがバックグラウンドで動作できる時間は限られています）。リスト 2-2に、これを行う方法を示します。

リスト 2-2　バックグラウンドで実行中にすぐにLocal Notificationを提示する

```
- (void)applicationDidEnterBackground:(UIApplication *)application {
    NSLog(@"Application entered background state.");
    // bgTaskはインスタンス変数
    NSAssert(self->bgTask == UIInvalidBackgroundTask, nil);

    bgTask = [application beginBackgroundTaskWithExpirationHandler: ^{
        dispatch_async(dispatch_get_main_queue(), ^{
            [application endBackgroundTask:self->bgTask];
```

```
            self->bgTask = UIInvalidBackgroundTask;
        });
    }];

    dispatch_async(dispatch_get_main_queue(), ^{
        while ([application backgroundTimeRemaining] > 1.0) {
            NSString *friend = [self checkForIncomingChat];
            if (friend) {
                UILocalNotification *localNotif = [[UILocalNotification alloc]
 init];
                if (localNotif) {
                    localNotif.alertBody = [NSString stringWithFormat:
                        NSLocalizedString(@"%@ has a message for you.", nil),
friend];
                    localNotif.alertAction = NSLocalizedString(@"Read Message",
 nil);
                    localNotif.soundName = @"alarmsound.caf";
                    localNotif.applicationIconBadgeNumber = 1;
                    [application presentLocalNotificationNow:localNotif];
                    [localNotif release];
                    friend = nil;
                    break;
                }
            }
        }
        [application endBackgroundTask:self->bgTask];
        self->bgTask = UIInvalidBackgroundTask;
    });

}
```

Remote Notificationのための登録

アプリケーションのプロバイダから送信されたRemote Notificationを受信するには、アプリケーションはデバイスやコンピュータ上のオペレーティングシステムのApple Push Notificationサービスに登録する必要があります。登録には次の3つのステージがあります。

1. アプリケーションが`registerForRemoteNotificationTypes:`メソッドを呼び出します。
2. デリゲートはデバイストークンを受信するために`application:didRegisterForRemoteNotificationsWithDeviceToken:`メソッドを実装します。
3. アプリケーションが、デバイストークンを非オブジェクトのバイナリ値としてプロバイダに渡します。

注： 特に注記しない限り、この節で取り上げたメソッドはすべて、UIApplicationとNSApplicationで、およびデリゲートに関しては、NSApplicationDelegate ProtocolとUIApplicationDelegateで、同じシグニチャで宣言されています。

この一連の処理の間に、アプリケーション、デバイス、Apple Push Notificationサービス、およびプロバイダの間で何が行われるかについては、「トークンの生成と共有」（31 ページ）の図 3-3に示しています。

アプリケーションは、起動のたびに登録を行って、現在のトークンをプロバイダに渡す必要があります。アプリケーションは、登録プロセスを開始するためにregisterForRemoteNotificationTypes:メソッドを呼び出します。このメソッドのパラメータは、アプリケーション側で受信したい初期の通知の種類（たとえば、アイコンバッジと警告音、ただし警告メッセージはなしなど）を指定するUIRemoteNotificationType（Mac OS Xの場合はNSRemoteNotificationType）ビットマスクです。iOSでは、ユーザはその後、有効になった通知タイプを、「Settings」アプリケーションの「Notifications」環境設定で修正できます。iOS、Mac OS Xともに、現在有効な通知タイプは、enabledRemoteNotificationTypesメソッドを呼び出すことにより調べることができます。これらの通知の種類が有効になっていない場合は、たとえ通知ペイロードで指定されていても、オペレーティングシステムは、バッジアイコン、警告メッセージの表示、または警告音の再生を行いません。

Mac OS Xにおける注意事項： 動作していないアプリケーションに対してサポートされる通知タイプはアイコンバッジだけなので、単にNSRemoteNotificationTypeBadgeを、registerForRemoteNotificationTypes:のパラメータとして渡してください。

登録に成功すると、APNsはデバイスにデバイストークンを返します。iOSは、application:didRegisterForRemoteNotificationsWithDeviceToken:メソッド内でこのトークンをアプリケーションデリゲートに渡します。アプリケーションは、プロバイダに接続して、このデバイストークンをバイナリ形式にエンコードして渡さなければなりません。トークンを取得する際に問題が発生した場合、オペレーティングシステムはapplication:didFailToRegisterForRemoteNotificationsWithError:メソッドを呼び出してデリゲートに知らせます。このメソッドに渡されるNSErrorオブジェクトには、エラーの原因が明確に記述されています。たとえば、エラーは、プロビジョニングプロファイルのaps-environment値の誤りであるかもしれませんエラーは一時的な状態として見るだけにし、パースしようとはしないでください（詳細については、「プロビジョニングプロファイルの作成とインストール」（43 ページ）を参照してください）。

iOSにおける注意事項： 携帯電話接続またはWi-Fi接続が使用できない場合、application:didRegisterForRemoteNotificationsWithDeviceToken:メソッドおよびapplication:didFailToRegisterForRemoteNotificationsWithError:メソッドはどちらも呼び出されません。Wi-Fi接続では、これは、デバイスがポート5223経由で接続できない場合に発生することがあります。これが発生した場合、ユーザはこのポートがブロックされていないほかのWi-Fiネットワークに移動します。iPhoneまたはiPadでは、携帯電話データサービスが使用可能になるまで待機します。どちらの場合も、接続が成功すると、デリゲーションメソッドの1つが呼び出されます。

アプリケーションが起動するたびに、デバイストークンを要求してそれをプロバイダに渡すことで、プロバイダが最新のデバイストークンを持つことを保証できます。バックアップの作成元となったものとは異なるデバイスやコンピュータにバックアップを復元した場合は（たとえば、新しいデバイスやコンピュータにデータをインポートした場合）、再び通知を受信するために少なくとも一度はアプリケーションを起動する必要があります。バックアップデータを新しいデバイスやコ

ンピュータに復元した場合や、オペレーティングシステムを再インストールした場合は、デバイストークンが変更されます。さらに、デバイストークンをキャッシュしてプロバイダに渡してはいけません。常に、デバイストークンは必要になったときにその都度システムから取得します。アプリケーションが以前に登録されていれば、`registerForRemoteNotificationTypes:`を呼び出すと、オペレーティングシステムは、余分なオーバヘッドを生じさせることなく、ただちにデバイストークンをデリゲートに渡します。

リスト 2-3は、iOSアプリケーションにおけるRemote Notificationの登録方法を示した簡単な例です Mac OS Xアプリケーションでもほぼ同じです。（`SendProviderDeviceToken`は、プロバイダに接続してデバイストークンを渡すためにクライアントで定義されている仮定のメソッドです）。

リスト 2-3　Remote Notificationのための登録

```
- (void)applicationDidFinishLaunching:(UIApplication *)app {
   // その他のセットアップ作業をここで行う
    [[UIApplication sharedApplication]
registerForRemoteNotificationTypes:(UIRemoteNotificationTypeBadge |
UIRemoteNotificationTypeSound)];
}

// デリゲーションメソッド
- (void)application:(UIApplication *)app
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)devToken {
    const void *devTokenBytes = [devToken bytes];
    self.registered = YES;
    [self sendProviderDeviceToken:devTokenBytes]; // カスタムメソッド
}

- (void)application:(UIApplication *)app
didFailToRegisterForRemoteNotificationsWithError:(NSError *)err {
    NSLog(@"Error in registration. Error: %@", err);
}
```

Local NotificationおよびRemote Notificationの処理

オペレーティングシステムがアプリケーションにLocal NotificationまたはRemote Notificationを配信するときに考えられるシナリオについて見てみましょう。

- 通知が、アプリケーションがフォアグラウンドで動作中でないときに配信される。

  この場合、システムは、警告の表示、アイコンのバッジの表示、また、サウンドの再生を行うことで通知を提示します。

- 通知が提示された結果、ユーザが警告のアクションボタンまたはアプリケーションアイコンをタップ（またはクリック）する。

  （iOSが稼働するデバイス上で）アクションボタンがタップされると、システムはアプリケーションを起動します。そのアプリケーションはデリゲートの`application:didFinishLaunchingWithOptions:`メソッド（実装されている場合）を呼び出し、通知のペイロード（Remote Notificationの場合）またはLocal Notificationオブジェクト（Local Notificationの場合）を渡します。

iOSが稼働するデバイス上で、アプリケーションアイコンがタップされた場合、アプリケーションは同じメソッドを呼び出しますが、通知に関する情報は渡しません。. Mac OS Xが稼働するコンピュータ上でアプリケーションアイコンがクリックされた場合、アプリケーションはデリゲートの`applicationDidFinishLaunching:`メソッドを呼び出し、デリゲートはこのメソッド内でRemote Notificationペイロードを取得できます。

iOSにおける注意事項： アプリケーションデリゲートは、`application:didFinishLaunchingWithOptions:`の代わりに`applicationDidFinishLaunching:`を実装することも可能ですが、これはお勧めできません。後者の方法では、アプリケーションは、通知以外の情報を含め、起動された理由に関する情報を受け取れます。

- 通知が、アプリケーションがフォアグラウンドで動作中に配信される。

  アプリケーションは、そのデリゲートの`application:didReceiveRemoteNotification:`メソッド（Remote Notificationの場合）または`application:didReceiveLocalNotification:`メソッド（Local Notificationの場合）を呼び出して、通知のペイロードまたはLocal Notificationオブジェクトを渡します。

注： この節で取り上げたデリゲートメソッドで、名前に「RemoteNotification」とついているものは、`NSApplicationDelegate Protocol`と`UIApplicationDelegate`のどちらでも、同じシグニチャで宣言されています。

アプリケーションは、渡されたRemote Notificationペイロード、または、iOSでは`UILocalNotification`オブジェクトを利用して、通知に関連する項目を処理するコンテキストを設定できます。理論上、デリゲートは各プラットフォームで、あらゆる状況でRemote NotificationやLocal Notificationを配信できるよう、次のような処理をします。

- Mac OS Xの場合、`NSApplicationDelegate Protocol`プロトコルに準拠し、`applicationDidFinishLaunching:`メソッドおよび`application:didReceiveRemoteNotification:`メソッドを実装しなければなりません。
- iOSの場合、`UIApplicationDelegate`プロトコルに準拠し、`application:didFinishLaunchingWithOptions:`メソッドと、`application:didReceiveRemoteNotification:`または`application:didReceiveLocalNotification:`メソッドの、両方を実装しなければなりません。

iOSにおける注意事項： iOSでは、ユーザがアクションボタンをタップしたからアプリケーションが起動されたのか、アプリケーション状態を確認したことによって動作中のアプリケーションに通知が配信されたのかどうかを特定できます。デリゲートの`application:didReceiveRemoteNotification:`メソッドまたは`application:didReceiveLocalNotification:`メソッドの実装で、`applicationState`プロパティの値を取得し、その値を評価します。値が`UIApplicationStateInactive`の場合は、ユーザがアクションボタンをタップしました。値が`UIApplicationStateActive`の場合は、通知を受け取ったときにそのアプリケーションが最前面にありました。

リスト 2-4では、iOSアプリケーション用のデリゲートに`application:didFinishLaunchingWithOptions:`メソッドを実装して、Local Notificationを処理します。`UIApplicationLaunchOptionsLocalNotificationKey`キーを使用して、起動オプション辞

書から、関連するUILocalNotificationオブジェクトを取得します。UILocalNotificationオブジェクトのuserInfo辞書から、通知の理由となるToDo項目にアクセスして、アプリケーションの初期コンテキストの設定に使用します。この例に示すように、通知を処理する一環として、アプリケーションアイコンのバッジの数字を適切にリセットすべきです（未処理の項目がなければバッジを消去します）。

リスト 2-4　アプリケーションが起動されたときのLocal Notificationの処理

```
- (BOOL)application:(UIApplication *)app
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    UILocalNotification *localNotif =
        [launchOptions
objectForKey:UIApplicationLaunchOptionsLocalNotificationKey];
    if (localNotif) {
        NSString *itemName = [localNotif.userInfo objectForKey:ToDoItemKey];
        [viewController displayItem:itemName];  // カスタムメソッド
        application.applicationIconBadgeNumber =
localNotif.applicationIconBadgeNumber-1;
    }
    [window addSubview:viewController.view];
    [window makeKeyAndVisible];
    return YES;
}
```

Remote Notificationの場合の実装も、各プラットフォームで特に宣言された定数を通知ペイロードにアクセスするキーとして使用することを除けば、これと似ています。

- iOSの場合、デリゲートはapplication:didFinishLaunchingWithOptions:メソッドの実装において、UIApplicationLaunchOptionsRemoteNotificationKeyキーを、起動オプション辞書からペイロードにアクセスするためのキーとして使用します。
- Mac OS Xの場合、デリゲートはapplicationDidFinishLaunching:メソッドの実装において、NSApplicationLaunchRemoteNotificationKeyキーを、メソッドに渡されたNSNotificationオブジェクトのuserInfo辞書からペイロード辞書にアクセスするためのキーとして使用します。

ペイロード自体は、通知の要素（警告メッセージ、バッジの数字、サウンドなど）を含む、NSDictionaryオブジェクトです。ペイロードには、アプリケーションが当初のユーザインターフェイスをセットアップするときにそのコンテキストの設定に使用できるカスタムデータも含めることができます。Remote Notificationペイロードの詳細については、「通知ペイロード」（34 ページ）を参照してください。

重要： 顧客データやほかの機密データを送信することを目的として通知ペイロードにカスタムプロパティを定義してはいけません。Remote Notificationの配信は保証されていません。カスタムペイロードプロパティの適切な使い方の一例は、メッセージがクライアントにダウンロード済みの電子メールアカウントを表す文字列です。アプリケーションは、この文字列をダウンロードインターフェイスに組み込むことができます。カスタムペイロードプロパティのもう1つの例は、プロバイダが最初に通知を送信した日時を表すタイムスタンプです。クライアントアプリケーションは、この値を使用してその通知がどのくらい古いかを測定できます。

`application:didFinishLaunchingWithOptions:`または`applicationDidFinishLaunching:`でRemote Notificationを処理する場合、アプリケーションデリゲートは大きなタスクをもう1つ実行する場合があります。アプリケーションの起動直後、デリゲートはそのプロバイダに接続して、待機中のデータをフェッチする必要があります。リスト 2-5は、この手順を簡単に示したものです。

リスト 2-5 プロバイダからのデータのダウンロード

```
- (void)application:(UIApplication *)app
didFinishLaunchingWithOptions:(NSDictionary *)opts {
    // 通知ペイロードおよびカスタムデータのlaunchOptionsを確認して、UIコンテキストを設定
    [self startDownloadingDataFromProvider];  // カスタムメソッド
    app.applicationIconBadgeNumber = 0;
    // その他のセットアップ作業をここで行う
}
```

注： クライアントアプリケーションは、非同期または二次スレッドで常にプロバイダと通信する必要があります。

リスト 2-6のコードに、アプリケーションがフォアグラウンドで動作中の場合に呼び出される、`application:didReceiveLocalNotification:`メソッドの実装を示します。ここでのアプリケーションデリゲートは、リスト 2-4で行ったのと同じ作業を行います。オブジェクトがメソッドの引数であるため、ここでは`UILocalNotification`オブジェクトに直接アクセスできます。

リスト 2-6 アプリケーションがすでに動作中の場合のLocal Notificationの処理

```
- (void)application:(UIApplication *)app
didReceiveLocalNotification:(UILocalNotification *)notif {
    NSString *itemName = [notif.userInfo objectForKey:ToDoItemKey]
    [viewController displayItem:itemName];  // カスタムメソッド
    application.applicationIconBadgeNumber =
notification.applicationIconBadgeNumber-1;
}
```

アプリケーションがフォアグラウンドで動作しているときにシステムから配信されたRemote Notificationをキャッチさせたい場合には、アプリケーションデリゲートは`application:didReceiveRemoteNotification:`メソッドを実装する必要があります。デリゲートは、待機中のデータ、メッセージ、またはほかの項目をダウンロードする手順を開始し、終了後に、アプリケーションアイコンからバッジを消去すべきです（アプリケーションが頻繁にプロバイダに新しいデータがないかどうかをチェックしている場合は、このメソッドを実装する必要はないかもしれません）。このメソッドの2番目のパラメータで渡された辞書は通知ペイロードです。そこに含まれているカスタムプロパティをアプリケーションの現在のコンテキストを変更するために使用すべきではありません。

Mac OS Xで、動作していないアプリケーションに対してサポートされる通知タイプはアイコンバッジだけですが、デリゲートは`application:didReceiveRemoteNotification:`を実装することにより、ほかのタイプの通知のペイロードを調べ、適切に処理する（警告を表示する、警告音を鳴らすなど）ことができます。

iOSにおける注意事項： Remote Notification警告の表示直後にユーザがデバイスをロック解除すると、オペレーティングシステムは警告に関連付けられたアクションを自動的にトリガします（この動作は、SMSおよびカレンダーの警告と同じです）。これにより、Remote Notificationに関連するアクションが害を及ぼす結果にならないことが、さらに重要になります。データが格納されるアプリケーションに関して、データを破棄する結果となる判断は、必ずユーザが行わなければなりません。

プロバイダに現在の言語設定を渡す（Remote Notification）

アプリケーションが、ローカライズされた警告メッセージをクライアント側で取得するために、`aps`辞書の`loc-key`プロパティと`loc-args`プロパティを使用しない場合は、プロバイダが、通知ペイロードに含める警告メッセージのテキストをローカライズする必要があります。ただしそれには、デバイスのユーザが設定言語として選択している言語を検出する必要があります（ユーザはこれを、「設定(Settings)」アプリケーションの「一般(General)」＞「言語環境(International)」＞「言語(Language)」ビューで設定します）。クライアントアプリケーションは設定言語の識別子をプロバイダに送信する必要があります。これは、標準化されているIETF BCP 47の言語識別子（「en」、「fr」など）です。

注： `loc-key`プロパティと`loc-args`プロパティ、およびクライアント側のメッセージローカライズの詳細については、「通知ペイロード」（34 ページ）を参照してください。

リスト 2-7は、現在選択されている言語を取得して、それをプロバイダに送信する手法を示しています。iOSでは、`NSLocale`の`preferredLanguages`で返される配列に1つのオブジェクト（選択されている言語を識別する言語コードをカプセル化した`NSString`オブジェクト）が含まれています。`UTF8String`は、この文字列オブジェクトをUTF8エンコードのC文字列に変換します。

リスト 2-7　現在サポートされている言語を取得してプロバイダに送信する

```
NSString *preferredLang = [[NSLocale preferredLanguages] objectAtIndex:0];
const char *langStr = [preferredLang UTF8String];
[self sendProviderCurrentLanguage:langStr]; // カスタムメソッド
}
```

アプリケーションは、ユーザが現在のロケールを何か変更するたびに設定言語をプロバイダに送信することもできます。それには、`NSCurrentLocaleDidChangeNotification`という名前の通知を検知し、通知処理メソッド内で、設定言語の識別コードを取得してそれをプロバイダに送信します。

設定言語がアプリケーションでサポートしていない言語の場合、プロバイダは広く普及している代替言語（英語、スペイン語など）にメッセージテキストをローカライズしなければなりません。

Apple Push Notificationサービス

Apple Push Notificationサービス（略してAPNs）はPush Notification機能の中核です。これは、iPhone、iPad、およびiPod touchなどのデバイスに情報を配信するための堅牢で非常に効率的なサービスです。各デバイスは、このサービスとの間で認証済みの暗号化されたIP接続を確立し、この永続的な接続を介して通知を受信します。アプリケーションが実行されていないときに通知が届いた場合、デバイスはそのアプリケーションに更新データがあることをユーザに警告します。

ソフトウェアのデベロッパ（「プロバイダ」）は、サーバソフトウェア内で通知を作成します。プロバイダは、対象となるクライアントアプリケーション宛ての受信データを監視している間、永続的でセキュアなチャンネルを介してAPNsに接続します。アプリケーション宛ての新しいデータを受信すると、プロバイダは通知を作成してこのチャンネルを介してAPNsに送信します。APNsは、その通知をターゲットデバイスに配信（プッシュ）します。

APNsは単純でありながら効率的で高い処理能力を持つ配信サービスです。さらに、APNsには蓄積交換機能を提供するデフォルトのQuality-of-Serviceコンポーネントが含まれています。詳細については、「Quality of Service」（28 ページ）を参照してください。

「プロバイダとApple Push Notificationサービスの通信」（45 ページ）および「通知のスケジューリング、登録、処理」（17 ページ）では、プロバイダとiOSアプリケーションに固有の実装要件について、それぞれ説明しています。

Push Notificationとその経路

Apple Push Notificationサービスは、特定のプロバイダから特定のデバイスに通知をルーティングします。通知は、デバイストークンとペイロードという2つの主要なデータ部分から構成された短いメッセージです。デバイストークンは電話番号にたとえることができます。そこには、クライアントアプリケーションがインストールされているデバイスをAPNsが見つけるために必要な情報が含まれています。また、APNsはこれを使用して通知のルーティングを認証します。ペイロードは、デバイス上のアプリケーションのユーザに警告する方法を指定するための、JSONで定義されたプロパティリストです。

注： デバイストークンの詳細については、「セキュリティアーキテクチャ」（29 ページ）を参照してください。通知ペイロードの詳細については、「通知ペイロード」（34 ページ）を参照してください。

Remote Notificationのデータの流れは単方向です。プロバイダは、クライアントアプリケーションのデバイストークンとペイロードを含む通知パッケージを作成します。プロバイダは、その通知をAPNsに送信します。そして、APNsがその通知をデバイスに配信（プッシュ）します。

APNsへの認証に成功すると、プロバイダはデータの提供先となるアプリケーションを識別するトピックをAPNsに提供します。このトピックは、現在のところiOSデバイス上のターゲットアプリケーションのバンドル識別子です。

図 3-1 プロバイダからクライアントアプリケーションへのPush Notification

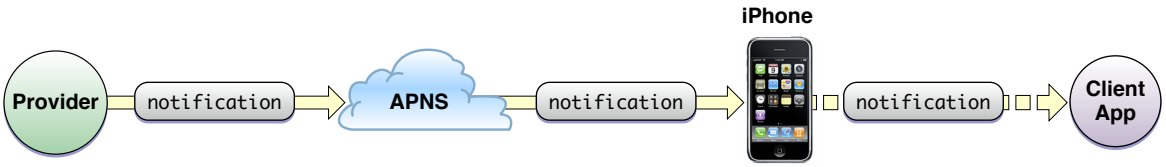
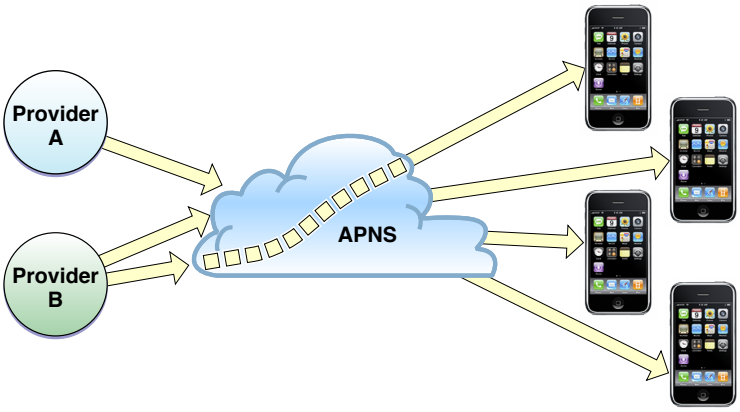


図 3-1は、APNsによってプロバイダとデバイスの間に実現される仮想ネットワークを非常に単純化した図です。ASPsのデバイス側とプロバイダ側には、共に複数の接続ポイントがあります。そして、プロバイダ側の接続ポイントをゲートウェイと呼びます。通常は、複数のプロバイダが存在し、それぞれがこれらのゲートウェイを介して、APNsとの間に1つ以上の永続的でセキュアな接続を確立します。そして、これらのプロバイダは、APNsを経由してクライアントアプリケーションがインストールされている複数のデバイスに通知を送信します。図 3-2は、上よりも少し現実的な図です。

図 3-2 複数のプロバイダから複数のデバイスへのPush Notification



フィードバックサービス

時には、APNsがデバイス上のアプリケーションに通知を配信しようとしても、ターゲットアプリケーションが存在しないためにデバイスに何度も配信を拒否されることがあります。このような状況は、ユーザがそのアプリケーションをアンインストールしている場合にしばしば発生します。このような場合、APNsは、プロバイダが接続しているフィードバックサービスを通してプロバイダに知らせます。フィードバックサービスは、アプリケーションごとに、最近通知の配信に何度も失敗しているデバイスのリストを管理しています。プロバイダは、このデバイスリストを取得してそのデバイスへの通知の送信を停止する必要があります。フィードバックサービスの詳細については「フィードバックサービス」 （50 ページ）を参照してください。

Quality of Service

Apple Push Notificationサービスには、蓄積交換機能を実行するデフォルトのQuality of Service（QoS）コンポーネントが含まれています。APNsが通知を配信しようとしたときにデバイスがオフラインだった場合は、QoSが通知を保存します。QoSが保持するのは、デバイス上のアプリケーションごと

に1つの通知（そのアプリケーション用にプロバイダから受信した最後の通知）だけです。オフラインのデバイスが後で再接続してきたときに、QoSは保存していた通知をデバイスに転送します。QoSは一定期間通知を保存した後、その通知を削除します。

セキュリティアーキテクチャ

プロバイダとデバイス間の通信を可能にするには、Apple Push Notificationサービスが、それらに対して特定のエントリポイントを公開しなければなりません。ただしセキュリティを保証するために、これらのエントリポイントへのアクセスを規制する必要もあります。この目的のためにAPNsは、プロバイダ、デバイス、およびそれらの通信に対して、2つの異なる信頼レベルを要求します。これらは、接続信頼とトークンによる信頼と呼ばれます。

接続信頼は、プロバイダ側のAPNs接続が、通知を配信することをAppleから許可されている認定済みのプロバイダとの接続であることを保証します。デバイス側の接続では、APNsはその接続が正当なデバイスとの接続であることを検証しなければなりません。

エントリポイントでの信頼を確立したら、APNsは正当なエンドポイントだけに通知を配信することを保証しなければなりません。それには、この接続を通してメッセージが送信されるルートを検証して、意図した通知先であるデバイスだけが通知を受信するようにしなければなりません。

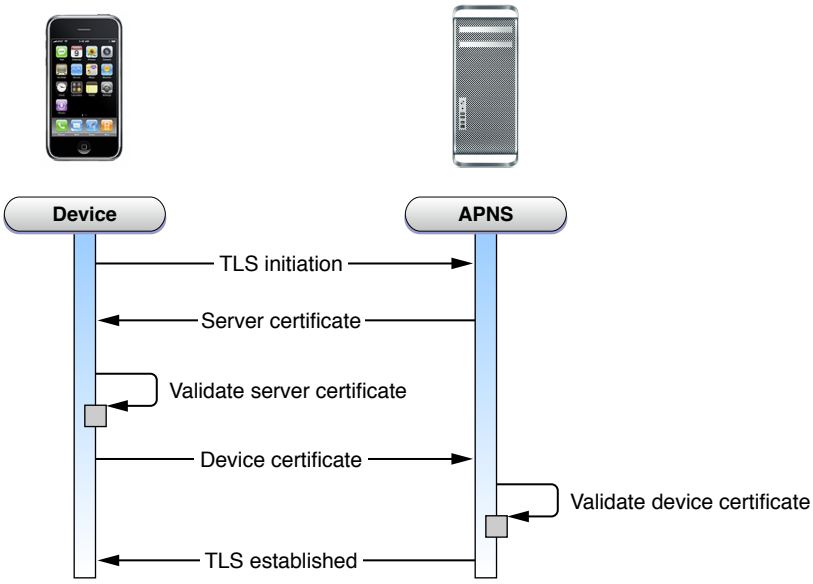
APNsでは、デバイストークンを利用して正確なメッセージルーティングを保証すること（つまり、**トークンによる信頼**）が可能になっています。デバイストークンは、APNsが初めてデバイスに接続したときに、APNsからそのデバイスに渡される不透過なデバイス識別子です。デバイスは、このデバイストークンをプロバイダと共有します。その後は、このトークンがプロバイダからのすべての通知に添付されます。これが、特定の通知のルーティングが正当であることを保証するための基礎になります（たとえて言えば、デバイストークンは、通信先を識別するための電話番号と同じ機能を果たします）。

> **注：** デバイストークンは、`UIDevice`の`uniqueIdentifier`プロパティによって返されるデバイスのUDIDとは別物です。

この後のセクションでは、接続信頼とトークンによる信頼に必要なコンポーネントと、信頼を確立するための4つの手続きについて説明します。

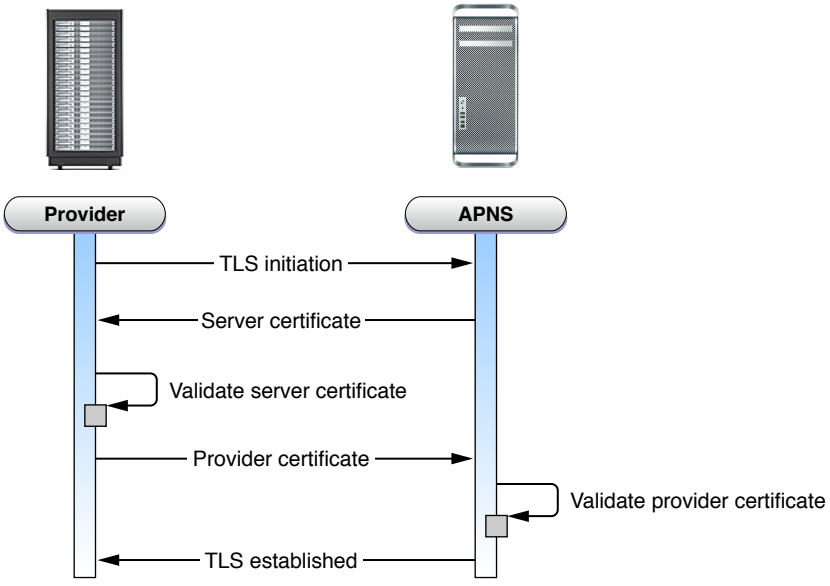
サービスーデバイス間の接続信頼

APNsは、TLSのピアツーピア認証を利用して、接続してきたデバイスの認証を行います（接続信頼のこのステージはiOSによって処理されるので、デベロッパが自身で実装する必要はありません）。この手続きの中で、デバイスはAPNsとのTLS接続を開始し、APNsはサーバ証明書を返します。デバイスはこのサーバ証明書を検証して、APNsにデバイス証明書を送信します。APNsはそのデバイス証明書を検証します。



プロバイダーサービス間の接続信頼

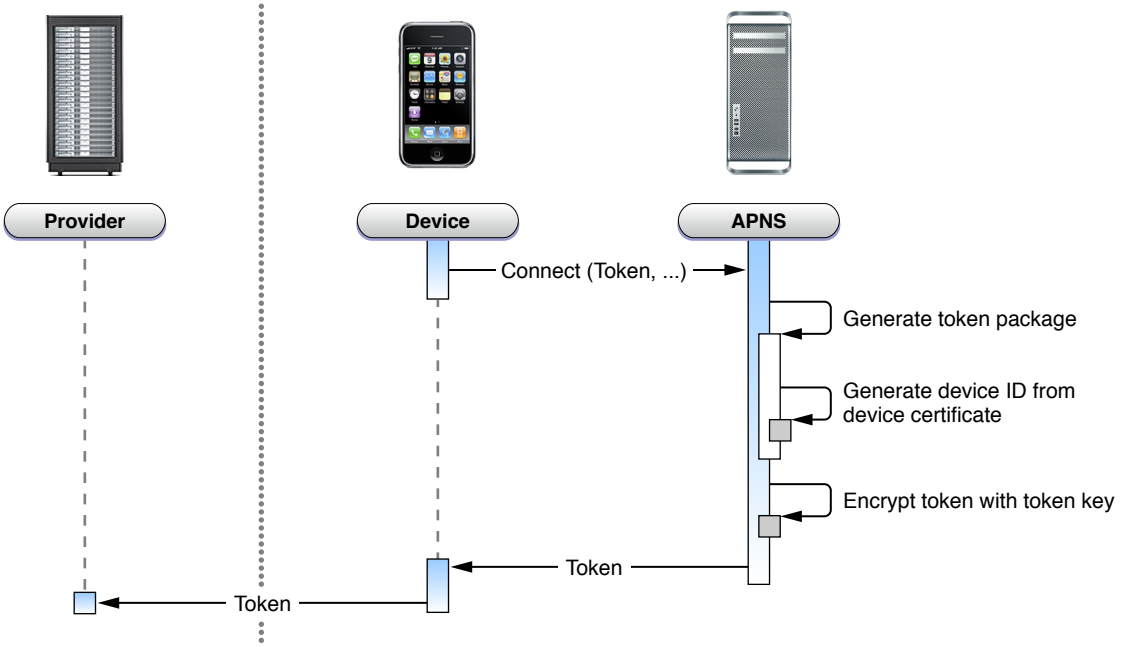
プロバイダとAPNs間の接続信頼も、TLSのピアツーピア認証を利用して確立されます。手続きも「サービスーデバイス間の接続信頼」（29 ページ）で説明したものと同様です。プロバイダはTLS接続を開始し、APNsからサーバ証明書を取得します。そしてそのサーバ証明書を検証します。次に、プロバイダはAPNsにプロバイダ証明書を送信します。今度は、APNsがそのプロバイダ証明書を検証します。この手続きが完了したら、セキュアなTLS接続が確立されます。これでAPNsは、この接続が正当なプロバイダによって確立したことを保証できます。



このプロバイダ接続は、証明書に規定されたトピック（バンドルID）によって識別される、ある特定のアプリケーションへの配信に対してのみ有効です。また、APNsは証明書失効リストを管理しています。プロバイダの証明書がこのリストに存在する場合、APNsはプロバイダの信頼を無効にする（つまり、接続を拒否する）こともあります。

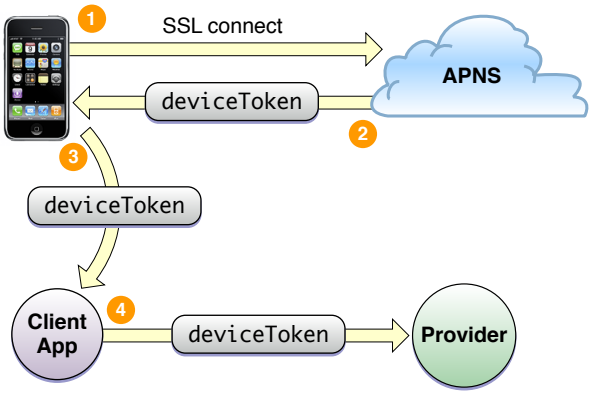
トークンの生成と共有

iOSベースのアプリケーションは、Push Notificationを受信するための**登録**をしなければなりません。通常、これは、アプリケーションがデバイスにインストールされた直後に行われます（この手続きについては「通知のスケジューリング、登録、処理」（17 ページ）で説明していデバイストークンを暗号化してデバイスに返しますます）。iOSは、アプリケーションからの登録要求を受け取ると、APNsに接続してその要求を転送します。APNsは、一意のデバイス証明書に含まれている情報を使用してデバイストークンを生成します。このデバイストークンには、デバイスの識別子が含まれています。次に、トークンキーを利用してデバイストークンを暗号化してデバイスに返します。



デバイスは、このデバイストークンの要求を出してきたアプリケーションに`NSData`オブジェクトとして返します。次に、アプリケーションはこのデバイストークンをバイナリ形式または16進形式でプロバイダに渡さなければなりません。図 3-3は、トークンの生成と共有の順番を示しています。さらに、プロバイダにデバイストークンを供給する際のクライアントアプリケーションの役割も示しています。

図 3-3 デバイストークンの共有

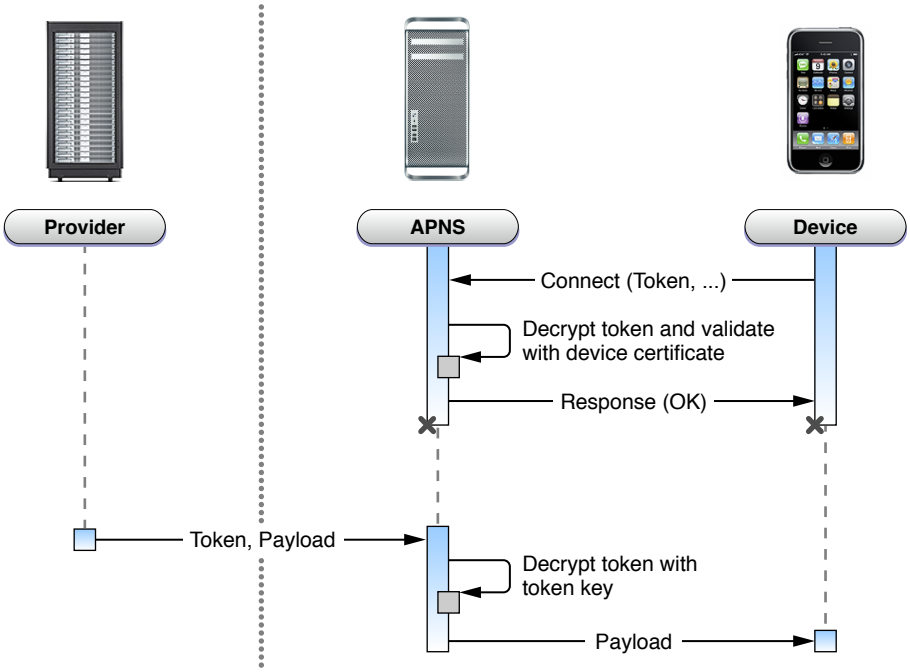


このような形態のトークンによる信頼フェーズによって、後で認証に使われるトークンを生成するのはAPNsだけであることが保証されます。また、APNsは、デバイスから渡されたトークンが、その特定のデバイスのために（そのデバイスのためだけに）以前用意したものと同じトークンであることを自ら確認できます。

トークンによる信頼（通知）

iOSは、APNsからデバイストークンを取得した後は（「トークンの生成と共有」（31 ページ）で説明）、APNsに接続するたびにそのトークンをAPNsに渡さなければなりません。APNsはデバイストークンを解読し、そのトークンが接続してきたデバイス用に生成されたものかどうかを検証します。検証のために、APNsは、トークンに含まれているデバイス識別子と、デバイス証明書のデバイス識別子が一致するかどうかを確認します。

デバイスへの配信のためにプロバイダがAPNsに送信するすべての通知には、そのデバイス上のアプリケーションから取得したデバイストークンが添付されていなければなりません。APNsはトークンキーを使用してトークンを解読し、通知が正当であることを確認します。次に、デバイストークンに含まれているデバイスIDを使用して、通知の配信先となるデバイスを判別します。



信頼に関連するコンポーネント

APNsのセキュリティモデルをサポートするために、プロバイダとデバイスは、特定の証明書、認証局（CA）証明書、またはトークンを所有していなければなりません。

- **プロバイダ**：各プロバイダには、一意のプロバイダ証明書と、APNsとの接続を検証するために使用する秘密暗号鍵が必要になります。この証明書はAppleから提供されますが、プロバイダが発行する特定のトピック（クライアントアプリケーションのバンドルID）を識別できなければなりません。通知のたびに、プロバイダはターゲットデバイスを識別するためのデバイストークンをAPNsに渡さなければなりません。プロバイダは、APNsサーバから提供された公開サーバ証明書を使用して、接続先のサービスを検証することもできます。

- **デバイス**：iOSは、APNsから渡された公開サーバ証明書を使用して、接続先のサービスを認証します。iOSは、一意の秘密鍵と証明書を持っており、これを使用してサービスへの認証を行いTLS接続を確立します。iOSは、デバイスの起動中にこの秘密鍵とデバイス証明書を取得して、キーチェーンに格納します。また、iOSは、特定のデバイストークンも保持しています。これは、サービス接続の処理中に受け取ります。登録済みの各クライアントアプリケーションは、このトークンをコンテンツプロバイダに送信する役割を担っています。

APNsサーバも、プロバイダとデバイスの接続とIDを検証するために、必要な証明書、CA証明書、および暗号鍵（秘密鍵と公開鍵）を持っています。

通知ペイロード

各Push Notificationと一緒にペイロードが送信されます。ペイロードは、クライアントアプリケーションにダウンロード可能なデータが存在することをユーザに警告する方法を指定します。通知ペイロードに許される最大サイズは256バイトです。Apple Push Notificationサービスでは、この上限を超える通知は拒否されます。通知の配信は「ベストエフォート型」のため、保証されないことを忘れないでください。

通知ごとに、プロバイダはRFC 4627に厳密に準拠したJSON辞書オブジェクトを作成しなければなりません。この辞書には、キー`aps`で識別されるもう1つの辞書が含まれている必要があります。`aps`辞書には、次のアクションを指定する1つ以上のプロパティが含まれています。

- ユーザに表示する警告メッセージ
- アプリケーションアイコンに付けるバッジの数字
- 警告音

注： 1つの通知で、警告メッセージ、アイコンバッジ、および警告音を組み合わせることができますが、Push Notificationがヒューマンインターフェイスに与える影響を考慮すべきです。たとえば、警告音付きの警告メッセージが頻繁に表示されると、ユーザはそれを便利ではなく邪魔だと感じるかもしれません。ダウンロードすべきデータが重要ではない場合は、特にそう感じるでしょう。

通知が届いたときにターゲットアプリケーションが実行されていない場合は、警告メッセージ、警告音、またはバッジが再生または表示されます。アプリケーションが実行されている場合は、iOSがその通知を`NSDictionary`オブジェクトとしてアプリケーションデリゲートに渡します。この辞書には、それに対応するCocoaプロパティリストオブジェクト（`NSNull`も含む）が含まれています。

プロバイダは、Appleによって予約されている`aps`名前空間の外部に、カスタムペイロード値を定義することもできます。カスタム値にはJSONの構造体および基本型（辞書（オブジェクト）、配列、文字列、数字、Boolean）を使用しなければなりません。カスタムペイロードデータとして顧客情報を含めるべきではありません。代わりに、（ユーザインターフェイスの）コンテキストや内部基準の設定などの目的にカスタムペイロードデータを使用します。たとえば、カスタムペイロード値を、インスタントメッセージのクライアントアプリケーションで使用する会話識別子にしたり、プロバイダが通知を送信した日時を識別するタイムスタンプにできます。警告メッセージに関連付けられているアクションは破壊的（たとえば、デバイス上のデータを削除するなど）であるべきではありません。

重要： 通知の配信は保証されないため、ペイロードを使って重要なデータをアプリケーションに送信するために、Remote Notification機能を利用すべきではありません。また、機密データをペイロードに含めてはいけません。Remote Notification機能は、新しいデータが利用できることをユーザに*通知する*ためだけに利用すべきです。

表 3-1は、`aps`ペイロードのキーと値のリストです。

表 3-1 aps辞書のキーと値

キー	値の型	説明
alert	文字列または 辞書	このプロパティが含まれていると、iOSは標準的な警告を表示します。alertの値として、文字列または辞書を指定できます。文字列を指定すると、その文字列は2つのボタン（「Close」と「View」）が付いた警告のメッセージテキストになります。ユーザが「View」をタップすると、アプリケーションが起動します。 alertの値として辞書を指定することもできます。この辞書のキーの説明は、表 3-2 （35 ページ）を参照してください。
badge	数値	アプリケーションアイコンのバッジとして表示する数字です。このプロパティが存在しない場合は、現在表示されているバッジの数字はすべて削除されます。
sound	文字列	アプリケーションバンドル内のサウンドファイルの名前です。このファイル内のサウンドが、警告音として再生されます。サウンドファイルが存在しないか、値としてdefaultが指定されている場合は、デフォルトの警告音が再生されます。このオーディオは、システムサウンドと互換性のあるオーディオデータフォーマットのいずれかでなければなりません。詳細については「カスタム警告音の準備」 （17 ページ）を参照してください。

表 3-2 alertプロパティの子プロパティ

キー	値の型	説明
body	文字列	警告メッセージのテキストです。
action-loc-key	文字列またはnull	文字列が指定されている場合は、2つのボタン付きの警告が表示されます。これらのボタンの動作については表 3-1で説明しています。ただし、iOSは、この文字列をキーとして使用して現在のローカライズからローカライズ文字列を取得し、「View」の代わりに右ボタンのタイトルとして使用します。この値がnullの場合は、システムは「OK」ボタンだけが付いた警告を表示します。このボタンをタップすると単純に警告が消えます。詳細については「ローカライズ形式の文字列」 （36 ページ）を参照してください。
loc-key	文字列	現在のローカライズ（ユーザの言語環境によって設定される）に対応するLocalizable.stringsファイル内の警告メッセージ文字列のキーです。このキー文字列を%@指定子や%*n*$@指定子を使用して書式化すると、loc-argsで指定した変数に置き換えることができます。詳細については「ローカライズ形式の文字列」 （36 ページ）を参照してください。
loc-args	文字列の配列	loc-key内の書式指定子の代わりに表示する変数文字列値です。詳細については「ローカライズ形式の文字列」 （36 ページ）を参照してください。

キー	値の型	説明
launch-image	文字列	アプリケーションバンドル内の画像ファイルのファイル名です。拡張子を含んでも省略してもかまいません。画像は、ユーザがアクションボタンをタップするか、アクションスライダを動かしたときの起動画像として使われます。このプロパティが指定されていない場合、システムは以前のスナップショットを使用するか、アプリケーションのInfo.plistファイルのUILaunchImageFileキーで指定された画像を使用するか、Default.pngにフォールバックします。 このプロパティはiOS 4.0で追加されました。

注： iPhone、iPad、またはiPod touchデバイスで、「Close」ボタンと「View」ボタンの両方を持つ警告内にメッセージテキストをそのまま表示する場合は、直接alertの値として文字列を指定します。辞書がbodyプロパティしか持たない場合は、alertの値として辞書を指定しないでください。

ローカライズ形式の文字列

ローカライズされた警告メッセージを表示するには2つの方法があります。1つは、通知を作成するサーバでテキストをローカライズする方法です。それには、デバイスで現在選択されている言語設定をサーバが検出しなければなりません（「プロバイダに現在の言語設定を渡す（Remote Notification）」（26 ページ）を参照）。もう1つは、クライアントアプリケーションが、サポート対象の各ローカライズ用に翻訳した警告メッセージ文字列をバンドルに格納しておく方法です。プロバイダは、通知ペイロードのaps辞書で、loc-keyプロパティとloc-argsプロパティを指定します。（アプリケーションが実行されていない場合）デバイスは通知を受信すると、これらのaps辞書プロパティを使用して、現在の言語にローカライズされた文字列を検索し書式化します。そして、それをユーザに表示します。

ここでは、2番目の方法が動作する仕組みをもう少し詳しく説明します。

iOSアプリケーションは、画像、サウンド、テキストなどのリソースをサポート対象の各言語用に国際化できます。国際化によって、これらのリソースは集められて、2つの部分（言語コードと.lprojという拡張子）から構成される名前（たとえば、fr.lproj）のバンドルのサブディレクトリに配置されます。プログラムで表示されるローカライズ文字列は、Localizable.stringsというファイルに保存されています。このファイルの各エントリにはキーとローカライズ文字列値が含まれています。この文字列には、変数値に置き換え可能な書式指定子を含めることができます。アプリケーションが特定のリソース（たとえば、ローカライズ文字列）を要求すると、ユーザによって現在選択されている言語にローカライズされたリソースを取得します。たとえば、設定言語がフランス語の場合、警告メッセージ用の文字列の値はアプリケーションバンドルのfr.lprojディレクトリ内のLocalizable.stringsから取得されます（iOSは、NSLocalizedStringマクロを利用してこの要求を出します）。

注： action-loc-keyプロパティの値が文字列の場合も、この一般的なパターンに従います。この文字列は、現在選択されている言語のローカライズディレクトリ内のLocalizable.stringsへのキーです。iOSは、このキーを使用して警告メッセージの右側のボタン（“アクション”ボタン）のタイトルを取得します。

わかりやすいように例を考えてみましょう。プロバイダは、警告プロパティの値として次のような辞書を指定しています。

```
"alert" : { "loc-key" : "GAME_PLAY_REQUEST_FORMAT", "loc-args" : [ "Jenna",
"Frank"] },
```

デバイスはこの通知を受信すると、"GAME_PLAY_REQUEST_FORMAT"をキーとして使用し、現在の言語の.lprojディレクトリ内のLocalizable.stringsファイルから、それに関連付けられた文字列値を検索します。現在のローカライズに次のようなLocalizable.stringsエントリがあるとします。

```
"GAME_PLAY_REQUEST_FORMAT" = "%@ and %@ have invited you to play Monopoly";
```

デバイスは、「Jenna and Frank have invited you to play Monopoly.」というメッセージを含む警告を表示します。

書式指定子%@のほかに、%*n*$@書式指定子を使用して、位置を指定して文字列変数を置き換えることができます。*n*は、loc-args内の置換対象の配列値のインデックス（1から始まる）です（パーセント記号（%）を表す%%指定子もあります）。たとえば、Localizable.stringsのエントリが次のようになっているとします。

```
"GAME_PLAY_REQUEST_FORMAT" = "%2$@ and %1$@ have invited you to play Monopoly";
```

デバイスは、「Frank and Jenna have invited you to play Monopoly.」というメッセージを含む警告を表示します。

loc-keyプロパティとloc-argプロパティを使用した通知ペイロードの完全な例については、「JSONペイロードの例」の後半の例を参照してください。iOSの国際化の詳細については、『*iOS App Programming Guide*』の「Build-Time Configuration Details」を参照してください。国際化についての一般的な情報は、『*Internationalization Programming Topics*』を参照してください。文字列の書式化については、『*String Programming Guide*』の「Formatting String Objects」で解説されています。

注： loc-keyプロパティとloc-argsプロパティ（および、たいていのalert辞書）は、本当に必要な場合にだけ使用してください。これらのプロパティの値は、特に長い文字列の場合、パフォーマンスが向上するどころか帯域幅を使い果たすおそれがあります。ほとんどとは言わないまでも多くのアプリケーションでは、これらのプロパティは必要ありません。メッセージ文字列はユーザから与えられるため、暗黙のうちに「ローカライズされている」からです。

JSONペイロードの例

通知のペイロード部分に関する次の例は、表 3-1で示したプロパティの実際の使い方を示しています。キー名に「acme」を含むプロパティは、カスタムペイロードデータの例です。この例には、読みやすくするために空白文字や改行文字が含まれていますが、パフォーマンス向上のためには、プロバイダは空白文字や改行文字を省くべきです。

例1：次のペイロードには、デフォルトの警告ボタン（「Close」と「View」）付きの警告メッセージに対応した簡単でお勧めの形式のaps辞書が含まれています。この例では、alertの値として、辞書ではなく文字列を使用しています。このペイロードにはカスタム配列プロパティも含まれています。

```
{
    "aps" : { "alert" : "Message received from Bob" },
    "acme2" : [ "bang",  "whiz" ]
}
```

例2：この例のペイロードでは、aps辞書を使用して、左側には「Close」ボタン、右側の"アクション"ボタンにはローカライズされたタイトルが表示される警告メッセージをデバイスに要求しています。ここでは、「Play」と同じ意味のローカライズ文字列を取得するために、現在選択されている言語のLocalizable.stringsファイルのキーとして、「PLAY」を使用しています。また、このaps辞書は、アプリケーションアイコンに付けるバッジに5と表示するように要求しています。

```
{      "aps" : {          "alert" : { "body" : "Bob wants to play poker",
"action-loc-key" : "PLAY" },          "badge" : 5,      },      "acme1" : "bar",
      "acme2" : [ "bang",  "whiz" ]  }
```

例3：この例のペイロードでは、デバイスが「Close」と「View」の両方のボタンが付いた警告メッセージを表示するよう指定しています。また、アプリケーションアイコンに9という数字のバッジを付け、通知の配信時にバンドルされている警告音を鳴らすことも要求しています。

```
{
    "aps" : {
        "alert" : "You got your emails.",
        "badge" : 9,
        "sound" : "bingbong.aiff"
    },
    "acme1" : "bar",
    "acme2" : 42
}
```

例4：この例のペイロードのおもしろい点は、アプリケーションバンドルからローカライズ形式の文字列を取得し、適切な場所にある変数文字列値（loc-args）に置き換えるために、alert辞書の子プロパティloc-keyとloc-argsを使用していることです。また、カスタムサウンドを指定していますし、カスタムプロパティも含まれています。

```
{
    "aps" : {
        "alert" : { "loc-key" : "GAME_PLAY_REQUEST_FORMAT", "loc-args" : [
"Jenna", "Frank"] },
        "sound" : "chime",
    },
    "acme" : "foo"
}
```

例5：次の例は、空のaps辞書を示しています。badgeプロパティがないため、アプリケーションアイコン上のバッジに付けられた数字は削除されます。acme2カスタムプロパティは2つの整数の配列です。

```
{
    "aps" : {
    },
    "acme2" : [ 5,  8 ]
}
```

パフォーマンスを向上させるために、ペイロードを通知に含める前に、すべての空白文字と改行文字をペイロードから削除することを忘れないでください。

プロビジョニングおよび開発

サンドボックス環境と製品環境

クライアントサーバアプリケーションのプロバイダ側を開発してデプロイするには、適切なDev CenterからSSL証明書を取得する必要があります。証明書は、バンドルIDで識別される1つのアプリケーションに対して1つと限定されています。各証明書は、それぞれIPアドレスが割り当てられた、2つの開発環境のいずれか1つに限定されています。

- **サンドボックス**：サンドボックス環境は、プロバイダアプリケーションの初版の開発とテストに使用されます。サーバユニットの数は少ないものの、製品環境と同じサービスセットが提供されます。サンドボックス環境は、シミュレートされたエンドツーエンドテストを可能にする仮想デバイスとしての役割も果たします。

  サンドボックス環境には、`gateway.sandbox.push.apple.com`のアウトバウンドTCPポート2195からアクセスできます。

- **製品**：製品版のプロバイダアプリケーションをビルドするときは製品環境を使用します。製品環境を使用するアプリケーションは、Appleの信頼性要件を満たしている必要があります。

  製品環境には、`gateway.push.apple.com`のアウトバウンドTCPポート2195からアクセスできます。

サンドボックス（開発）環境用と製品環境用に、別々の証明書を取得する必要があります。この証明書はPush Notificationを受信するアプリケーションの識別子に関連付けられます。この識別子にはアプリケーションのバンドルIDが含まれています。これらの環境のいずれかのプロビジョニングプロファイルを作成すると、必要な資格が自動的にこのプロファイルに追加されます。これには、Push Notificationに固有の資格（`<aps-environment>`）も含まれます。この2つのプロビジョニングプロファイルは、DevelopmentとDistributionと呼ばれます。Distributionプロビジョニングプロファイルは、アプリケーションをApp Storeに投稿するための必要条件です。

Mac OS Xにおける注意事項： Mac OS Xプロビジョニングプロファイルの資格は`com.apple.developer.aps-environment`で、プラットフォームに範囲を限定。

どの環境で作業をしているかは、Xcodeでコード署名IDを選択すると判別できます。「iPhone Developer: *Firstname Lastname*」という証明書／プロビジョニングプロファイルのペアが表示された場合は、サンドボックス環境にいます。「iPhone Distribution: *Companyname*」という証明書／プロビジョニングプロファイルのペアが表示された場合は、製品環境にいます。この2つの環境を区別しやすくするために、XcodeでDistributionリリース構成を作成するとよいでしょう。

SSL証明書がプロビジョニングプロファイルに含まれていなくても、この証明書と特定のアプリケーションIDが関連付けられているため、`<aps-environment>`がプロファイルに追加されます。その結果、この資格がアプリケーションに組み込まれて、Push Notificationの受信が可能になります。

プロビジョニングの手順

iOSデベロッパプログラムでは、開発チームの各メンバは、Team Agent、Team Admin、およびTeam Memberの3つの役割のいずれか1つを持ちます。これらの役割の違いは、iPhone開発証明書とプロビジョニングプロファイルに関する部分です。Team Agentは、Development（サンドボックス）SSL証明書とDistribution（製品）SSL証明書を作成できる、チーム内で唯一の人物です。Team AdminとTeam Agentは、DevelopmentとDistributionの両方のプロビジョニングプロファイルを作成できます。Team Memberは、証明書とプロビジョニングプロファイルのダウンロードとインストールのみができます。この後のセクションで説明する手順では、これらの役割を参照します。

注： iOSプロビジョニングポータルでは、すべてのiOSデベロッパプログラムメンバに、ユーザガイドと、証明書の作成とプロビジョニングのすべての側面を説明した一連のビデオを公開しています。この後のセクションでは、APNs固有の手順に焦点を当て、その他の側面については簡単に説明します。ポータルにアクセスするには、iOSデベロッパプログラムのメンバは、iOS Dev Center（http://developer.apple.com/devcenter/ios）にアクセスしてログインし、「iOS Provisioning Portal」ボタン（右上のリンク）をクリックでこのページに移ります。

SSL証明書と鍵の作成

iOS Dev Centerのプロビジョニングポータルで、Team AgentはAPNs用のアプリケーションIDを選択します。また、Team Agentは次の手順を実行してSSL証明書を作成します。

1. ウインドウの左側のサイドバーにある「App IDs」をクリックします。

   次のページに、有効なアプリケーションIDが表示されます。アプリケーションIDは、アプリケーションのバンドルIDの前に、Appleが作成した10文字のコードが付加された構成になっています。Team Adminは、バンドルIDを入力しなければなりません。証明のためには固有のバンドルIDを入力する必要があります。“ワイルドカード”付きのアプリケーションIDは使用できません。

2. サンドボックスSSL証明書（Developmentプロビジョニングプロファイルに関連付けられている）の場合はアプリケーションIDを指定して、「Configure」をクリックします。

   このアプリケーションIDに対する証明書を構成するには、Apple Push Notificationサービスの列の下に「Available」と表示されていなければなりません。

3. 「Configure App ID」ページで、「Enable for Push Notification Services」ボックスにチェックし、「Configure」ボタンをクリックします。

   このボタンをクリックするとAPNs Assistantが起動します。このガイドに従って次の一連の手順を実行します。

4. 最初の手順では、キーチェーンアクセスアプリケーションを起動して、CSR（証明書署名要求：Certificate Signing Request）を生成する必要があります。

   アシスタントの指示に従います。CSRの生成が終了したら、証明書アシスタントの「完了」をクリックしてAPNs Assistantに戻ります。

   CSRの生成時に、キーチェーンアクセスは秘密暗号鍵と公開暗号鍵のペアを生成します。秘密鍵はデフォルトでログインキーチェーンに組み込まれます。公開鍵は、CA（認証局）に送信されるCSRに含められます。CAから証明書が戻ってくると、その証明書の項目の1つが公開鍵になっています。

5. 「Submit Certificate Signing Request」ペインで「ファイルを選択」をクリックします。前の手順で作成したCSRファイルに移動し、それを選択します。

6. 「Generate」ボタンをクリックします。

   「Generate Your Certificate」ペインが表示されている間に、APNs Assistantはクライアント SSL証明書を構成して生成します。これが正常に終了すると、「Your APNs Certificate has been generated.」というメッセージが表示されます。「Continue」をクリックして次の手順に進みます。

7. 次のペインで、「Download Now」ボタンをクリックして証明書ファイルをダウンロード場所にダウンロードします。その場所に移動し、その証明書ファイル（`cer`という拡張子を持つ）をダブルクリックしてキーチェーンにインストールします。終了したら、APNs Assistantの「Done」をクリックします。

   このファイルをダブルクリックすると、キーチェーンアクセスが起動します。プロバイダの開発に使用しているコンピュータ上のログインキーチェーンに証明書がインストールされていることを確認します。キーチェーンアクセスでは、証明書のユーザIDがアプリケーションのバンドルIDと一致していることを確認します。APNs SSL証明書は通知サーバ上にインストールする必要があります。

これら3つの手順が終了すると、iOS Dev Centerのポータルの「Configure App ID」ページに戻ります。証明書には緑色の丸と「Enabled」というラベルが付いているはずです。

製品環境用の証明書を作成するには同じ手順を繰り返します。ただし、製品環境の証明書用のアプリケーションIDを選択します。

プロビジョニングプロファイルの作成とインストール

次に、Team AdminまたはTeam Agentは、サーバ側のRemote Notification開発で使用するプロビジョニングプロファイル（DevelopmentまたはDistribution）を作成しなければなりません。このプロビジョニングプロファイルは、アプリケーションのデベロッパとそのデバイスを、承認済みの開発チームと関連付けて、これらのデバイスをテストに使用できるようにするための資産を集めたものです。このプロファイルには、証明書、デバイス識別子、アプリケーションのバンドルID、およびすべての資格（`<aps-environment>`を含む）が含まれています。すべてのTeam Memberは、アプリケーションの実行とテストを行うデバイスにこのプロビジョニングプロファイルをインストールする必要があります。

注： プロビジョニングプロファイル作成の詳細については、プログラムユーザガイドを参照してください。

プロビジョニングプロファイルをダウンロードしてインストールするために、Team Memberは次の手順を実行する必要があります。

1. iOS Dev Centerのプロビジョニングポータルに移動します。

2. APNsに登録したApp IDを含む新規のプロビジョニングプロファイルを作成します。

3. 新しいプロビジョニングプロファイルをダウンロードする前に*既存*のプロファイルに修正を加えます。

ポータルに新しいプロビジョニングプロファイルを生成させるには、プロファイルに少し変更を加える必要があります（たとえば、オプションを切り替えるなど）。プロファイルの変更がそれほど大きくなければ、プッシュ権限なしでオリジナルのプロファイルが提供されます。

4. このプロファイルファイル（`mobileprovision`という拡張子を持つ）をダウンロード場所からXcodeまたはiTunesのアプリケーションアイコンにドラッグします。

   または、プロファイルファイルを`~/ライブラリ/MobileDevice/Provisioning Profiles`に移動します。このディレクトリが存在しない場合は、ディレクトリを作成します。

5. プロビジョニングプロファイルファイル内の権限が正しいことを確認します。そのためには、テキストエディタで`.mobileprovision`ファイルを開きます。ファイル内容はXML形式で構造化されています。Entitlements辞書の中で`aps-environment`キーを探します。開発用のプロビジョニングプロファイルではこのキーの文字列の値は`development`となっているはずです。配布用のプロビジョニングプロファイルではこの文字列の値は`production`となっているはずです。

6. Xcodeの「オーガナイザ(Organizer)」ウインドウで、「Provisioning Profiles」セクションに移動し、このプロファイルをデバイスにインストールします。

プロジェクトをビルドすると、バイナリには秘密鍵を使用して証明書が付けられます。

SSL証明書と鍵のサーバへのインストール

事前に取得しておいたSSL Distribution証明書と秘密暗号鍵を、プロバイダのコードを実行して、サンドボックス版または製品版のAPNsに接続するサーバコンピュータにインストールする必要があります。それには、次の手順を実行します。

1. キーチェーンアクセスユーティリティを開き、左側のペインの「自分の証明書」カテゴリをクリックします。

2. インストールする証明書を検索し、中身を表示します。

   すると証明書と秘密鍵の両方が表示されます。

3. 証明書と鍵の両方を選択して、「ファイル」メニューの「書き出す」を選択します。そして、それらを「個人情報交換（.p12）」ファイルとして書き出します。

4. RubyやPerlなどの言語で実装されたサーバは、多くの場合、個人情報交換（Personal Information Exchange：PIE）形式の証明書のほうが取り扱いやすくなっています。証明書をこの形式に変換するためには、次の手順を実行します。

   a. キーチェーンアクセスで、証明書を選び、「ファイル」メニューから「書き出す」を選択します。「個人情報交換（.p12）」オプションを選び、保存する場所を選択して、「保存」をクリックします。

   b. ターミナルアプリケーションを起動してプロンプトの後に次のコマンドを入力します。

      ```
      openssl pkcs12 -in 証明書名.p12 -out 証明書名.pem -nodes
      ```

5. `.pem`証明書を新しいコンピュータにコピーして適切な場所にインストールします。

プロバイダとApple Push Notificationサービスの通信

この章では、Apple Push Notificationサービス（APNs）との通信のためにプロバイダが使用するインターフェイスについて説明し、プロバイダに期待されるいくつかの機能について解説します。

プロバイダの一般的な要件

プロバイダは、バイナリインターフェイスを介してApple Push Notificationサービスと通信します。このインターフェイスは、プロバイダ用の高速で大容量のインターフェイスです。このインターフェイスは、TCPソケットを介したストリーミング設計をバイナリコンテンツと組み合わせて使用しています。このバイナリインターフェイスは非同期です。

製品環境のバイナリインターフェイスは`gateway.push.apple.com`のポート2195から利用できます。サンドボックス（開発）環境のバイナリインターフェイスは`gateway.sandbox.push.apple.com`のポート2195から利用できます。同じゲートウェイまたは複数のゲートウェイインスタンスに、複数の接続を並行して確立することもできます。

インターフェイスごとに、TLS（またはSSL）を使用してセキュアな通信チャンネルを確立する必要があります。これらの接続に必要なSSL証明書は、iOSプロビジョニングのポータルから提供されます（詳細については、「プロビジョニングおよび開発」（41 ページ）を参照してください）。認証済みのプロバイダであることを証明するには、接続時にピアツーピア認証を使用してこの証明書をAPNsに渡さなければなりません。

注： APNsとTLSセッションを確立するには、Entrust Secure CAルート証明書をプロバイダのサーバ上にインストールする必要があります。サーバがMac OS Xを実行している場合、このルート証明書はすでにキーチェーンに存在します。その他のシステム上では、証明書が存在しない可能性があります。この証明書はEntrust SSL Certificateのウェブサイトからダウンロードできます。

また、複数の通知にわたってAPNsとの接続を維持しなければなりません。APNsは、接続と切断がすばやく何度も繰り返される場合、それをDoS（サービス運用妨害：denial-of-service）攻撃と見なします。エラーが生じると、APNsはエラーが発生した場所の接続を閉じます。

プロバイダは、Push Notificationの次の側面を担当します。

- 通知ペイロードを作成する必要があります（「通知ペイロード」（34 ページ）を参照）。
- アプリケーションアイコンに表示するバッジの数字を提供する必要があります。
- 定期的にフィードバック用のWebサーバに接続して、何度も配信失敗が報告されているデバイスの最新リストを取得すべきです。そして、これらのアプリケーションに関連付けられているデバイスへの通知の送信を停止する必要があります。詳細については、「フィードバックサービス」（50 ページ）を参照してください。

複数の言語で通知メッセージをサポートする場合に、ローカライズされた警告文字列をクライアント側で取得するためにapsペイロード辞書のloc-keyプロパティとloc-argsプロパティを使用しないときは、サーバ側で警告メッセージのテキストをローカライズする必要があります。それには、クライアントアプリケーションから現在の言語環境を検出する必要があります。「通知のスケジューリング、登録、処理」（17 ページ）では、この情報を取得するためのアプローチを示します。loc-keyプロパティとloc-argsプロパティの詳細については、「通知ペイロード」（34 ページ）を参照してください。

バイナリインターフェイスの形式と通知形式

バイナリインターフェイスは、本質的にストリーミング方式のバイナリコンテンツにはプレーンなTCPソケットを採用します。パフォーマンスを最適化するために、TCP/IPのNagleアルゴリズムを使用するか、またはインターフェイスを通して明示的に複数の通知を1回の転送で一括送信する必要があります。

インターフェイスは、単純形式と、単純形式に存在するいくつかの問題に対処した拡張形式の2つの形式の通知パケットをサポートします。

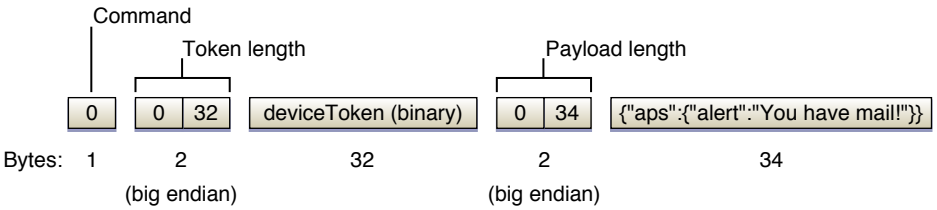
- **通知期限**。APNsには、デバイス上のアプリケーションに送信された最新の通知を保持する蓄積交換機能があります。配信時にデバイスがオフラインの場合、APNsはデバイスが次にオンラインになった時に通知を配信します。単純形式では、その通知が適切かどうかに関わらず通知が配信されます。つまり、時間の経過とともに通知は“古く”なっている場合があります。拡張形式では、通知が有効な期間を示す期限の値が含まれます。APNsは、この期間を経過すると蓄積交換内の通知を破棄します。

- **エラー応答**。単純形式では、何らかの誤った形式の（たとえば指定の限度を超えたペイロードなど）通知パケットを送った場合、APNsは接続を切断することで応答します。なぜ通知が拒否されたか、理由は一切表示されません。拡張形式では、任意の識別子をつけてプロバイダに通知を区別させます。エラーがあれば、APNsはエラーコードと識別子を結びつけるパケットを返します。この応答により、プロバイダは誤った形式の通知を特定し、修正することができます。

ほとんどのプロバイダには拡張形式がお勧めです。

単純形式の通知は拡張形式と形式の多くの部分が共通なため、まずは単純形式の通知から見てみましょう。図 5-1にこの形式を示します。

図 5-1　単純形式の通知



単純形式の最初のバイトはコマンド値で0（ゼロ）です。デバイストークンとペイロードの長さは、ネットワークのバイト順序（つまり、ビッグエンディアン）になっていなければなりません。さらに、デバイストークンをバイナリ形式にエンコードする必要があります。ペイロードは256バイトを超えたり、ヌルで終了したりしては*いけません*。

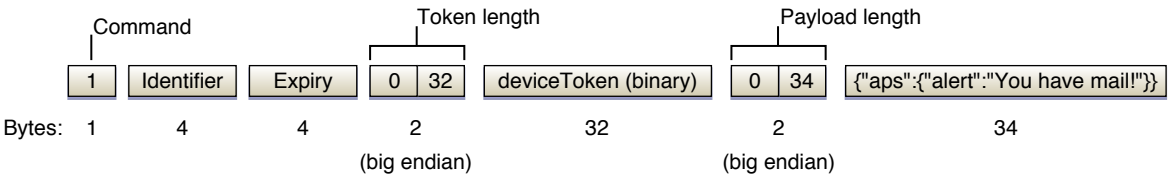
リスト 5-1に、単純形式の通知を使い、バイナリインターフェイスを介してAPNsへPush Notificationを送信する関数の例を示します。この例は、事前にgateway.push.apple.com（またはgateway.sandbox.push.apple.com）にSSL接続してピア交換認証を行っていることを前提としています。

リスト 5-1　バイナリインターフェイスを通しての単純形式の通知の送信

```
static bool sendPayload(SSL *sslPtr, char *deviceTokenBinary, char *payloadBuff,
 size_t payloadLength)
{
    bool rtn = false;
    if (sslPtr && deviceTokenBinary && payloadBuff && payloadLength)
    {
        uint8_t command = 0; /* コマンド番号 */
        char binaryMessageBuff[sizeof(uint8_t) + sizeof(uint16_t) +
            DEVICE_BINARY_SIZE + sizeof(uint16_t) + MAXPAYLOAD_SIZE];
        /* メッセージ形式は |COMMAND|TOKENLEN|TOKEN|PAYLOADLEN|PAYLOAD| */
        char *binaryMessagePt = binaryMessageBuff;
        uint16_t networkOrderTokenLength = htons(DEVICE_BINARY_SIZE);
        uint16_t networkOrderPayloadLength = htons(payloadLength);

        /* コマンド */
        *binaryMessagePt++ = command;

       /* トークン長（ネットワークバイト順序） */
        memcpy(binaryMessagePt, &networkOrderTokenLength, sizeof(uint16_t));
        binaryMessagePt += sizeof(uint16_t);

        /* デバイストークン */
        memcpy(binaryMessagePt, deviceTokenBinary, DEVICE_BINARY_SIZE);
        binaryMessagePt += DEVICE_BINARY_SIZE;

        /* ペイロード長（ネットワークバイト順序） */
        memcpy(binaryMessagePt, &networkOrderPayloadLength, sizeof(uint16_t));
        binaryMessagePt += sizeof(uint16_t);

        /* ペイロード */
        memcpy(binaryMessagePt, payloadBuff, payloadLength);
        binaryMessagePt += payloadLength;
        if (SSL_write(sslPtr, binaryMessageBuff, (binaryMessagePt -
binaryMessageBuff)) > 0)
            rtn = true;
    }
    return rtn;
}
```

図 5-2に拡張形式の通知パケットを示します。この形式では、APNsは理解できないコマンドに遭遇すると、切断する前にエラー応答を返します。

図 5-2　拡張形式の通知

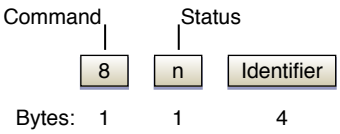


拡張形式の最初のバイトはコマンド値で1です。この形式の新しい2つのフィールドは識別子用と期限の値用です（その他はすべて単純形式の通知と同じです）。

- **識別子**。この通知を識別する任意の値です。APNsが通知を解釈できない場合、同じ識別子がエラー応答パケット内に返されます。
- **期限**。秒（UTC）で表わされた固定のUNIXエポック日です。通知が無効になり破棄できるようになるタイミングを示します。期限の値はネットワークバイト順序（ビッグエンディアン）である必要があります。期限の値が正であれば、APNsは少なくとも一度は通知を配信しようとします。値には、APNsに通知をまったく保持させないようにゼロまたはゼロより小さい値を指定することができます。

通知を送信した結果、APNsがその通知を誤った形式、あるいはその他の理解できない形式であると認識すると、APNsは切断の前にエラー応答パケットを返します（エラーがない場合はAPNsは何も返しません）。図 5-3にエラー応答パケットの形式を示します。

図 5-3 エラー応答パケットの形式



パケットのコマンド値は8であり、その後に1バイトのステータスコードと、プロバイダが通知の作成時に指定したのと同じ通知識別子が続きます。表 5-1に可能なステータスコードとその意味を示します。

表 5-1 エラー応答パケット内のコード

ステータスコード	説明
0	遭遇したエラーはなし
1	処理エラー
2	デバイストークン欠如
3	トピック欠如
4	ペイロード欠如
5	無効なトークンサイズ
6	無効なトピックサイズ
7	無効なペイロードサイズ
8	無効なトークン
255	なし（不明）

リスト 5-2では、APNsに送信する前に拡張形式のPush Notificationを作成するためにリスト 5-1 （47 ページ）のコードを修正しています。前述の例と同様に、これは事前にgateway.push.apple.com（またはgateway.sandbox.push.apple.com）にSSL接続してピア交換認証を行っていることを前提としています。

リスト 5-2　バイナリインターフェイスを通しての拡張形式の通知の送信

```
static bool sendPayload(SSL *sslPtr, char *deviceTokenBinary, char *payloadBuff,
 size_t payloadLength)
{
  bool rtn = false;
  if (sslPtr && deviceTokenBinary && payloadBuff && payloadLength)
  {
      uint8_t command = 1; /* コマンド番号 */
     char binaryMessageBuff[sizeof(uint8_t) + sizeof(uint32_t) + sizeof(uint32_t)
 + sizeof(uint16_t) +
          DEVICE_BINARY_SIZE + sizeof(uint16_t) + MAXPAYLOAD_SIZE];
     /* メッセージ形式は |COMMAND|ID|EXPIRY|TOKENLEN|TOKEN|PAYLOADLEN|PAYLOAD| */
      char *binaryMessagePt = binaryMessageBuff;
      uint32_t whicheverOrderIWantToGetBackInAErrorResponse_ID = 1234;
      uint32_t networkOrderExpiryEpochUTC = htonl(time(NULL)+86400); // 1日で配
信されなければメッセージは期限切れ
      uint16_t networkOrderTokenLength = htons(DEVICE_BINARY_SIZE);
      uint16_t networkOrderPayloadLength = htons(payloadLength);

      /* コマンド */
      *binaryMessagePt++ = command;

     /* プロバイダ指定の順序のID */
     memcpy(binaryMessagePt, &whicheverOrderIWantToGetBackInAErrorResponse_ID,
 sizeof(uint32_t));
     binaryMessagePt += sizeof(uint32_t);

     /* 期限日（ネットワークバイト順序） */
     memcpy(binaryMessagePt, &networkOrderExpiryEpochUTC, sizeof(uint32_t));
     binaryMessagePt += sizeof(uint32_t);

     /* トークン長（ネットワークバイト順序） */
      memcpy(binaryMessagePt, &networkOrderTokenLength, sizeof(uint16_t));
      binaryMessagePt += sizeof(uint16_t);

      /* デバイストークン */
      memcpy(binaryMessagePt, deviceTokenBinary, DEVICE_BINARY_SIZE);
      binaryMessagePt += DEVICE_BINARY_SIZE;

      /* ペイロード長（ネットワークバイト順序） */
      memcpy(binaryMessagePt, &networkOrderPayloadLength, sizeof(uint16_t));
      binaryMessagePt += sizeof(uint16_t);

      /* ペイロード */
      memcpy(binaryMessagePt, payloadBuff, payloadLength);
      binaryMessagePt += payloadLength;
      if (SSL_write(sslPtr, binaryMessageBuff, (binaryMessagePt -
binaryMessageBuff)) > 0)
          rtn = true;
  }
  return rtn;
}
```

本番環境のデバイストークンと開発（サンドボックス）環境のデバイストークンは同じ値ではないことに注意してください。

フィードバックサービス

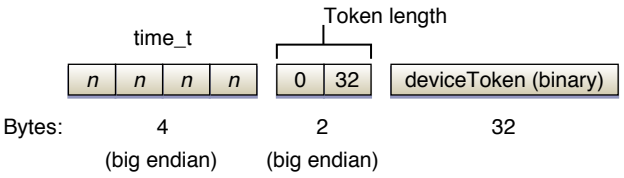
プロバイダがアプリケーションにPush Notificationを配信しようとしたときに、デバイス上にアプリケーションがもう存在しない場合、デバイスはその事実をApple Push Notificationサービスに報告します。このような状況は、ユーザがそのアプリケーションをアンインストールしている場合にしばしば発生します。アプリケーションへの配信に失敗したことがデバイスから報告された場合は、プロバイダがそのデバイスへの通知の送信を停止できるように、APNsは何らかの方法でプロバイダに知らせる必要があります。それによって、不必要なメッセージによるオーバーヘッドが減少し、システム全体のパフォーマンスが向上します。

その目的で、Apple Push Notificationサービスには、配信に失敗したデバイスの、アプリケーションごとのリストを継続的に更新するフィードバックサービスが含まれています。デバイスはバイナリ形式でエンコードされたデバイストークンによって識別されます。プロバイダは、定期的にこのフィードバックサービスに問い合わせて、対象のアプリケーション（トピックによって識別される）に対応するデバイストークンのリストを取得しなければなりません。そして、そのアプリケーションが識別されたデバイスに最近再登録されていないことを確認したら、プロバイダはそのデバイスへの通知の送信を停止すべきです。

フィードバックサービスへのアクセスは、Push Notificationの送信に使われるものと同様のバイナリインターフェイスを介して行われます。製品環境のフィードバックサービスには`feedback.push.apple.com`のポート2196からアクセスします。サンドボックス環境のフィードバックサービスには`feedback.sandbox.push.apple.com`のポート2196からアクセスします。Push Notificationのバイナリインターフェイスと同様に、TLS（またはSSL）を使用してセキュアな通信チャンネルを確立する必要があります。これらの接続に必要なSSL証明書は、通知を送信するために提供されたものと同じです。認証済みのプロバイダであることを証明するには、接続時にピアツーピア認証を使用してこの証明書をAPNsに渡さなければなりません。

接続すると、ただちに転送が始まります。APNsには一切コマンドを送信する必要はありません。読み込むべきデータがなくなるまで、フィードバックサービスによって書き込まれたストリームを読み込みます。受信したデータは次の形式を持つタプルです。

図 5-4　フィードバックタプルのバイナリ形式



Timestamp	デバイス上にアプリケーションがもう存在しないとAPNsが判断した日時を表すタイムスタンプ（4バイトの`time_t`値）。この値（ネットワークのバイト順序になっている）は、1970年からの秒数（UTCに基づく）を表します。 このタイムスタンプを使用して、デバイス上のアプリケーションが、デバイストークンがフィードバックサービスの記録された後に再登録されたかどうかを判断します。再登録されていない場合は、そのデバイスへのPush Notificationの送信を停止する必要があります。
トークン長	デバイストークンの長さ（ネットワークのバイト順序による2バイトの整数値）。
デバイストークン	バイナリ形式のデバイストークン。

注： APNsは、プロバイダがフィードバックサービスをチェックして、デバイス上に存在しないアプリケーションへのPush Notificationの送信を停止する作業をこまめに行っているかどうかを監視します。

書類の改訂履歴

この表は「*Local NotificationおよびPush Notification プログラミングガイド*」の改訂履歴です。

日付	メモ
2011-08-09	Mac OS Xデスクトップクライアントでの Push Notificationの実装に関する情報を追加しました。iOS向けおよびMac OS X向けのガイドを統合しました。
2010-08-03	ユーザが通知警告のアクションボタンをタップすることでアプリケーションを起動したかどうかを判別する方法について説明しました。
2010-07-08	「iPhone OS」という記述を「iOS」に変更しました。
2010-05-27	iOS 4.0で導入されたLocal Notification機能についての記述を更新、再編しました。APNsに送信するPush Notificationの新しい形式についても説明しています。
2010-01-28	細かな訂正を数多く行いました。
2009-08-14	細かな訂正を行い、Cocoaの概念に関する短い記事へのリンクを作成しました。
2009-05-22	Wi-Fiおよび登録頻度についての注記を追加し、サンドボックス用のゲートウェイアドレスを追加しました。いくつかの説明を明確にし、拡充しました。
2009-03-15	プロバイダがApple Push Notificationサービスを使用してクライアントアプリケーションにPush Notificationを送信する仕組みについて説明した文書の初版。

書類の改訂履歴