iOSテクノロジーの概要

2011-10-12

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

はじめに

iOSは、iPhone、iPod touch、およびiPadデバイスの中核となるオペレーティングシステムです。

iOSプラットフォームは、Mac OS Xの開発で培われた知識を使用して開発されました。このため、iOSプラットフォームでの開発に使われるツールおよびテクノロジーの多くは、Mac OS Xをルーツにしています。iOSは、Mac OS Xと似ている点がありますが、Mac OS Xアプリケーションの開発経験は必須ではありません。iOSアプリケーションの開発を始めるために必要なすべてのものは、**iOS SDK (Software Development Kit)**によって提供されます。

対象読者

『*iOSテクノロジーの概要*』は、iOSプラットフォームの初心者向けの入門書です。開発プロセスに効果をもたらすテクノロジーとツールの概要を説明し、関連文書などの情報源へのリンクを紹介します。本書は、次の目的のために参照してください。

- iOSプラットフォームを理解する。
- iOSソフトウェアの各テクノロジーについて、使用する理由やどのような場合に使用するかを学ぶ。
- iOSプラットフォーム用の開発のチャンスについて学ぶ。
- ほかのプラットフォームからiOSへの移植方法についてのヒントおよびガイドラインを学ぶ。
- 関心を持ったテクノロジーに関連する主要な文書を見つける。

この文書には、ユーザレベルのシステム機能についての情報や、ソフトウェア開発プロセスに関係のない機能についての情報は含まれていません。

経験の浅いデベロッパの場合は、iOSに慣れるためにこの文書が役立ちます。熟練デベロッパの場合は、特定のテクノロジーや開発手法を探究するためのロードマップとしてこの文書を使用できます。

この書類の構成

この文書は次の章と付録で構成されています。

- 「iOSの開発」（13 ページ）では、iOSおよびiOS SDKの概要について説明します。
- 「Cocoa Touchレイヤ」（19 ページ）では、アプリケーションの上位レベルの動作を提供するCocoa Touchレイヤのテクノロジーについて説明します。
- 「Mediaレイヤ」（29 ページ）では、ビジュアルコンテンツやオーディオコンテンツの表示を処理するMediaレイヤで使用されるテクノロジーについて説明します。
- 「Core Servicesレイヤ」（39 ページ）では、ハードウェア固有およびシステム固有のさまざまなタスクを処理するCore Servicesレイヤで使用されるテクノロジーについて説明します。
- 「Core OSレイヤ」（49 ページ）では、アプリケーションの下位レベルの基本要素を提供するCore OSレイヤのテクノロジーについて説明します。
- 「Cocoaからの移行」（53 ページ）では、既存のCocoaアプリケーションをiOSに移植する際の初歩的なアドバイスを提供します。
- 「iOSのフレームワーク」（67 ページ）では、各フレームワークがiOSに導入された時期など、システムフレームワークの要約とシステムフレームワークに関する一般情報を提供します。
- 「iOSのデベロッパツール」（63 ページ）では、iOS用のソフトウェアを作成するために使用するアプリケーションの概要を説明します。

iOS SDKの入手方法

iOS SDKには、iOS用ソフトウェアを設計、作成、デバッグ、および最適化するために必要なツールが含まれています。また、iOSプラットフォームのテクノロジー向けのヘッダファイル、サンプルコード、およびドキュメントも含まれています。iOS SDKは、iOS Dev Centerの会員サイトからウンロードできます。URLは次のとおりです。

http://developer.apple.com/devcenter/ios

MacOS Xおよびそのテクノロジーで作業をするために利用できるツールの詳細については、「iOSのデベロッパツール」（63 ページ）を参照してください。

フィードバックの提供

ドキュメントに関するフィードバックは、各ページの一番下にある組み込みのフィードバックフォームを使って送ってください。

Appleのソフトウェアやドキュメントのバグを発見した場合は、Appleにお知らせください。また、製品やドキュメントの今後の改訂版に期待する機能についての機能拡張リクエストを提出することもできます。バグ報告や機能拡張リクエストを提出するには、以下の AppleのデベロッパWebサイトのBug Reportingページにアクセスしてください。

http://developer.apple.com/jp/bugreporter/

バグ報告を提出するには、Appleデベロッパとして登録する必要があります。ログイン名は、Apple Developer Registrationページに説明されている手順に従って無料で入手できます。

関連項目

iOSの開発が初めての方は、この文書が扱っているのはiOSの概要だけである点に注意してください。iOSアプリケーションを開発する方法を詳しく知るには、次の文書をお読みください。

- 『*Cocoa Fundamentals Guide*』では、iOSアプリケーションの開発に使用するデザインパターンとデザインプラクティスに関する基本的な情報を提供しています。
- 『*iOS App Programming Guide*』では、iOSアプリケーションのアーキテクチャを詳しく説明するとともに、マルチタスクなど、主要なテクノロジーがアプリケーションの全体的なデザインに与える影響について解説しています。
- 『*iOS Human Interface Guidelines*』では、iOSアプリケーションのユーザインターフェイスの設計方法についてのガイダンスと重要な情報を提供しています。
- 『*iOS App Development Workflow Guide*』では、iOSの開発プロセスをツールの観点から説明しています。この文書は、ソフトウェアをビルド、実行、およびテストするための、デバイスの構成やプロビジョンとXcode（およびその他のツール）の使用についても言及しています。
- 『*The Objective-C Programming Language*』では、Objective-Cプログラミング言語およびObjective-Cランタイムを紹介しています。

iOSの開発

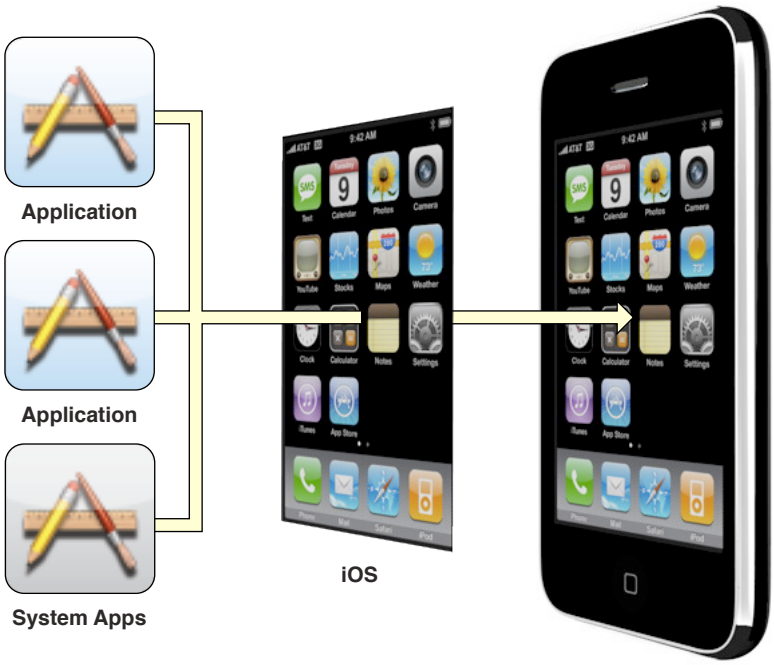
iOSは、iPhone、iPod touchデバイス、iPadデバイス上で稼働するオペレーティングシステムであり、デバイスハードウェアを管理するとともに、ネイティブアプリケーションの実装に必要な各種テクノロジーを提供します。また、iOSには「電話(Phone)」、「メール(Mail)」、「Safari」などの標準的なシステムサービスをユーザに提供するさまざまなシステムアプリケーションが付属しています。

iOS SDKには、ネイティブアプリケーションの開発、インストール、実行、およびテストを行うために必要なツールとインターフェイスが含まれています。ネイティブアプリケーションは、iOSのシステムフレームワークとObjective-C言語を使用して構築され、iOS上で直接実行されます。Webアプリケーションとは違い、ネイティブアプリケーションはデバイス上に物理的にインストールされるため、デバイスが機内モードであっても、ユーザは常にネイティブアプリケーションを利用できます。ネイティブアプリケーションはその他のシステムアプリケーションと並んで配置されます。アプリケーションとすべてのユーザデータは、iTunesを介してユーザのコンピュータと同期されます。

iOSのアーキテクチャ

iOSのアーキテクチャは、Mac OS Xの基本のアーキテクチャに似ています。最上位レベルでは、iOSはそれが稼働しているハードウェアと画面に表示されるアプリケーションの間の仲介役を果たします。その様子を図 1-1に示します。デベロッパが作成するアプリケーションは、ハードウェアと直接やり取りすることはめったにありません。代わりにアプリケーションは、ハードウェアの変更からアプリケーションを保護する、明確に定義された一連のシステムインターフェイスを介してハードウェアとやり取りします。このような抽象化によって、ハードウェアの能力が異なるデバイス上でも一貫性を保って動作するアプリケーションを容易に作成できます。

図 1-1 iOSの最上位層にあるアプリケーション

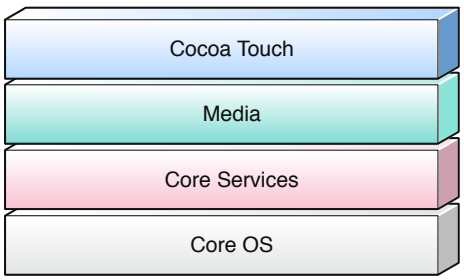


注： アプリケーションは、一般にハードウェアの変更から保護されますが、それでもデバイス間の違いをコード内では考慮する必要があります。たとえば、デバイスによっては、カメラを内蔵しているものと内蔵していないものがあります。作成するアプリケーションが、当該機能の有無に関係なく動作する場合、該当するフレームワークが提供するインターフェイスを使用して、その機能の有無を判別してください。

特定のハードウェア機能がないと動作しないアプリケーションは、その要件を情報プロパティリスト(`Info.plist`)ファイルを使用して指定する必要があります。必須のハードウェアの指定に関する詳細については、『*iOS App Programming Guide*』の「Advanced App Tricks」を参照してください。

iOSテクノロジーの実装は、図1-2に示すように、1つのレイヤセットで表現できます。システムの下位の各レイヤには、すべてのアプリケーションが依存する基本的なサービスとテクノロジーがあります。一方、上位レベルの各レイヤには、より高度なサービスとテクノロジーが含まれます。

図 1-2 iOSのレイヤ



コードを書くときには、下位レベルのフレームワークよりも、できる限り上位レベルのフレームワークを選ぶようにします。上位レベルのフレームワークは、下位レベルの構成体のオブジェクト指向の抽象化を提供するために用意されています。一般にこれらの抽象化により、記述するコード量が減り、ソケットやスレッドなどの複雑になりがちな機能がカプセル化されるため、コードの記述が非常に楽になります。下位レベルのテクノロジーは抽象化されますが、隠されるわけではありません。下位レベルのフレームワークや、上位層では公開されていない下位レベルのフレームワークの側面を使用したい場合は、依然としてこれらを利用できます。

各レイヤのテクノロジーとフレームワークについては、この後の章で説明します。

iOS SDKの構成要素

iOS SDKには、IntelベースのMacintoshコンピュータを利用してiOSアプリケーションを開発するために必要なすべてのインターフェイス、ツール、およびリソースが含まれています。

Appleでは、フレームワークと呼ばれる特殊なパッケージとしてほとんどのシステムインターフェイスを提供しています。**フレームワーク**は、共有ダイナミックライブラリとそのライブラリをサポートするために必要なリソース（ヘッダファイル、画像、ヘルパアプリケーションなど）を含む1つのディレクトリです。フレームワークを使用するには、その他の共有ライブラリと同様にそれらをアプリケーションプロジェクトにリンクします。フレームワークをプロジェクトにリンクするとそのフレームワークの機能にアクセスできるようになります。また、開発ツールにヘッダファイルとその他のフレームワークリソースの場所を知らせることもできます。

フレームワークのほかにAppleでは、標準の共有ライブラリという形でいくつかのテクノロジーを提供しています。iOSはUNIXをベースにしているため、オペレーティングシステムの下位レベルを構成する多くのテクノロジーはオープンソースのテクノロジーから派生しています。このため、これらのテクノロジーのインターフェイスは標準のライブラリおよびインターフェイスのディレクトリで利用できます。

ｉOS SDKのその他の主要なコンポーネントには、以下のものがあります。

- Xcodeツール - iOSアプリケーションの開発を支援するツール群で、以下の重要なアプリケーションが含まれています。
    - **Xcode** - アプリケーションのプロジェクトを管理し、コードの編集、コンパイル、実行、およびデバッグが可能な統合開発環境。Xcodeはその他の多くのツールと統合されており、開発の際に使用するメインアプリケーションです。
    - **Instruments** - 実行時のパフォーマンス分析およびデバッグのツール。Instrumentsを使用すると、アプリケーションの実行時の動作についての情報を収集し、潜在的な問題を識別できます。
- **iOSシミュレータ** - iOSテクノロジーのスタックをシミュレートするMac OS Xアプリケーション。シミュレータを使用して、iOSアプリケーションをIntelベースのMacintoshコンピュータ上でローカルにテストできます。
- **iOS Developer Library** — iOSテクノロジーおよびアプリケーション開発プロセスに関するすべてをデベロッパに伝える、参照情報と概念を記した文書。詳細については、「Developer Libraryの使い方」（16 ページ）を参照してください。

iOSシミュレータでアプリケーションを実行することはできますが、XcodeとInstrumentsを利用すると、接続されたデバイス上で直接アプリケーションを実行したり、デバッグしたりできます。シミュレータは、アプリケーションのビルドとテストを行う上で理想的ですが、実際のデバイスで行うテストの代わりにはなりません。実際のデバイス上で開発するには、Appleの有料のiOSデベロッパプログラムにサインアップし、デバイスを開発用に構成する必要があります。詳細については、iOS Dev CenterのiOS Developer Programを参照してください。

iOS SDKのインストール方法、およびiOSアプリケーションを開発するためのiOS SDKの使用方法については、『*iOS App Development Workflow Guide*』を参照してください。

作成可能なアプリケーション

iOSは、以下に示す2種類のアプリケーションの開発を支援します。

- ネイティブアプリケーション
- Webアプリケーション

iOS SDKでは、デバイスのホーム（Home）画面にのみ表示されるネイティブアプリケーションを作成できます。その他の種類のコード（ドライバ、フレームワーク、またはダイナミックライブラリなど）は作成できません。フレームワークのコードやダイナミックライブラリをアプリケーションに組み込むには、プロジェクトをビルドする際にそのコードを静的にアプリケーションの実行可能ファイルにリンクする必要があります。

Webアプリケーションは、HTML、CSS(Cascading Style Sheets)およびJavaScriptコードを組み合わせて対話型アプリケーションを実現します。Webアプリケーションは、Webサーバ上に配置され、ネットワーク経由で転送されてSafariのWebブラウザ内で実行されます。一方、ネイティブアプリケーションはデバイスに直接インストールされ、ネットワーク接続がなくても実行できます。

Developer Libraryの使い方

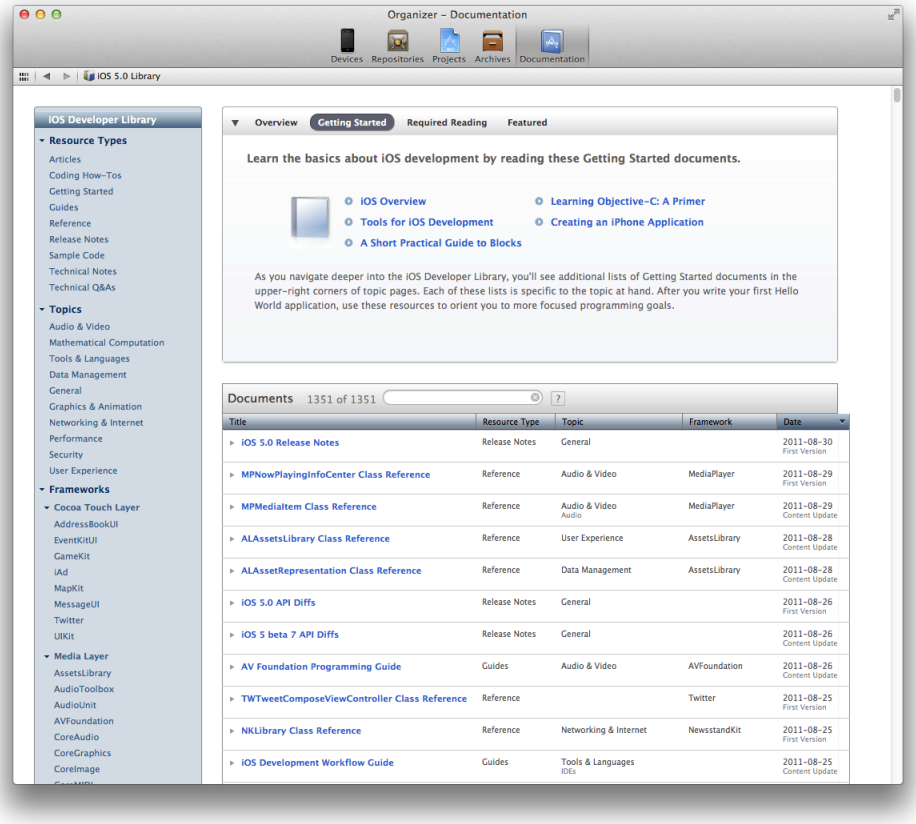
iOS Developer Libraryには、文書、サンプルコード、チュートリアル、およびiOSアプリケーション作成に必要なその他の情報が含まれています。Developer Libraryには、概念的な入門書から下位レベルのAPIリファレンス文書まで数千ページにもおよぶ文書が含まれています。このため、情報の検索方法を理解することは開発プロセスにおける重要なステップです。Developer Libraryでは、内容を参照しやすくするためにいくつかの技法を使用しています。

iOS Developer Libraryには、Apple Developer Webサイトから、またはXcodeからアクセスできます。Xcodeで、「ヘルプ(Help)」＞「Developer Documentation」を選ぶと、Xcodeの「ドキュメント(Documentation)」ウインドウが表示されます。これは、iOS向け開発についての情報にアクセスするための中心的なリソースです。このウインドウを使用して、ドキュメントの閲覧や検索の実行、および後で参照したい文書にブックマークを付けることができます。

iOS SDKをインストールすると、Xcodeによって自動的にiOS Developer Libraryが利用できるようになります。（Xcodeもアップデートを自動的にダウンロードします。これは、環境設定で変更できます）。iOS Developer Libraryにはたくさんの情報が含まれているため、少なくともそのレイアウトに慣れておくことが重要です。図 1-3は、Xcodeの「ドキュメント(documentation)ウインドウ」に表示されたDeveloper Libraryのメインページを示しています。ページの最上部にあるツールバーには、文

書の参照に使用する検索フィールドとボタンが含まれています。ライブラリは、探しているトピック、フレームワーク、またはリソースタイプごとに参照できます。文書のリストの上にある絞り込みのフィールドを使用して、表示される文書を絞り込むこともできます。

図 1-3 iOS Developer Library



重要： iOS Developer Libraryの内容は定期的に更新されますが、iOS Dev Centerからの最新ドキュメント、リリースノート、Tech Notes、テクニカルQ&A、およびサンプルコードにアクセスすることもできます。すべての文書はHTML形式で参照できます。また、そのほとんどはPDF形式でも参照できます。

Developer Libraryには膨大な量の情報が含まれているため、コードを記述しようとしている最中にそれらすべての情報をソートするのは面倒です。特定の情報をすばやく検索できるよう、Xcodeには、メインプロジェクトウインドウの「Utilities」セクションに、「Quick Help」ペインがあります。このペインには、指定されたシンボルについての情報（構文、説明、利用可／不可の情報を含む）が表示されます。また、関連するドキュメントやサンプルコードも表示されます。このペイン内のリンクをクリックすると、Developer Library内の対応するリソースが表示されます。

「ドキュメント（Documentation）」ウインドウと「Quick Help」ウインドウの使い方の詳細については、『*Xcode 4 User Guide*』を参照してください。

Cocoa Touchレイヤ

Cocoa Touchには、iOSアプリケーションを作成するための主要なフレームワークが含まれています。このレイヤは、基本的なアプリケーションインフラストラクチャを定義し、マルチタスク、タッチベースの入力、Push Notification、上位レベルの多くのシステムサービスなど、主要なテクノロジーをサポートしています。アプリケーションをデザインするときは、最初にこのレイヤのテクノロジーを調査して、要件を満たしているかを検討する必要があります。

上位レベルの機能

以下のセクションでは、Cocoa Touchレイヤで利用できる主要なテクノロジーのいくつかについて解説します。

ストーリーボード

iOS 5で導入されたストーリーボードは、アプリケーションのユーザインターフェイスを設計する方法として、nibファイルに代わるものを推奨しています。nibファイルと違い、ユーザインターフェイス全体を1箇所で設計できるので、ビューやビューコントローラがすべて連携して動作する状況を確認できます。ストーリーボードで重要なのは、あるビューコントローラから別のビューコントローラへの滑らかな遷移（セグエ）を定義できることです。アプリケーションはこの遷移を、Xcodeで視覚的に定義し、あるいはプログラム的に起こすことができます。この遷移を見れば、ユーザインターフェイスの中身だけでなく、流れも把握できます。

ひとつのストーリーボードファイルに、アプリケーションのビューコントローラやビューをすべて格納することも、あるいは複数のビューストーリーボードを使い、部分ごとに作成したインターフェイスを組み合わせることも可能です。Xcodeは構築時に、ストーリーボードファイルの中身を分割します。必要になった時点で個別にロードし、性能を改善できるようにするのが目的です。ただし、アプリケーションがこの分割について意識する必要はありません。UIKitフレームワークは、コードからストーリーボードの内容にアクセスするための、コンビニエンスクラスを提供しています。

ストーリーボードを使ってインターフェイスを設計する手順については、『*Xcode 4 User Guide*』を参照してください。コードからストーリーボードにアクセスする方法については、『*UIStoryboard Class Reference*』を参照してください。

ドキュメントのサポート

iOS 5で導入されたUIKitフレームワークには`UIDocument`クラスが組み込まれており、ユーザ文書に関連づけられたデータを管理できます。このクラスを使えば、文書ベースアプリケーション、特に文書をiCloudに格納するアプリケーションを、より簡単に実装できます。`UIDocument`クラスは、文書に関係するデータをすべて収容する入れ物を提供するだけでなく、ファイルデータを非同期的に読み書きする、データを安全に保存する、データを自動的に保存する、iCloudにおける版の食い違い

を検出する、フラットファイルまたはパッケージファイルの形でデータを扱う、などの機能が組み込まれています。データモデルとしてCore Dataを使っているアプリケーションは、`UIManagedDocument`のサブクラスを使ってデータストアを管理できます。

`UIDocument`クラスについては『*UIDocument Class Reference*』を参照してください。`UIManagedDocument`クラスについては『*UIManagedDocument Class Reference*』を参照してください。

マルチタスク

iOS SDK 4.0以降を使用して作成し、iOS 4.0以降で実行するアプリケーションは、ユーザがホーム(Home)ボタンを押しても終了せず、バックグラウンドでの実行に移ります。UIKitで定義されているマルチタスク対応により、アプリケーションのバックグラウンド状態との間の切り替えをスムーズに行えます。

バッテリーの持続時間を長くするため、ほとんどのアプリケーションはバックグラウンド状態になるとすぐに、システムによって一時停止されます。一時停止中のアプリケーションはメモリ内に保持されますが、コードは一切実行しません。こうした処理によってアプリケーションは、一時停止中はバッテリーを消費することなく、再び起動したときにすばやく再開することができます。ただし、以下の理由から、アプリケーションはバックグラウンドで実行し続けることができる場合があります。

- アプリケーションは、何らかの重要なタスクを完了できるよう、一定の時間を要求できます。
- アプリケーションは、定期的にバックグラウンドで実行する必要のある特定のサービスをサポートすることを、自身で宣言できます。
- アプリケーションは、実行中かどうかに関係なく、指定した時間にユーザにアラートを出すLocal Notificationを使用できます。

アプリケーションが一時停止中か、またはバックグラウンドで実行中かを問わず、マルチタスクに対応するにはデベロッパ側で追加の作業が必要です。システムは、バックグラウンドとの間で状態の遷移が生じると、アプリケーションに通知します。この通知ポイントは、ユーザデータの保存など、アプリケーションの重要なタスクを実行するための合図です。

マルチタスクの遷移を処理する方法およびバックグラウンドでの実行時間を要求する方法の詳細については、『*iOS App Programming Guide*』を参照してください。

印刷

UIKitでの印刷機能のサポートはiOS 4.2で導入され、アプリケーションは近くのプリンタにワイヤレスでコンテンツを送信することができます。ほとんどの場合、印刷に関連する負担の大きい作業のすべてをUIKitが行います。UIKitは、印刷インターフェイスの管理、印刷可能なコンテンツの描画におけるアプリケーションとの連携、およびプリンタにおける印刷ジョブのスケジュール管理と実行の処理を行います。

アプリケーションから送信される印刷ジョブは、印刷システムへと渡され、印刷システムが実際の印刷プロセスを管理します。あるデバイス上のすべてのアプリケーションから送られる印刷ジョブはキューに格納され、先に格納されたものから順に印刷されます。ユーザは、Printer Centerアプリケーションから印刷ジョブの状況を取得でき、このアプリケーションを使用して印刷ジョブをキャンセルすることもできます。印刷関連のその他の側面はすべて、ユーザに代わってシステムが自動的に処理します。

注： ワイヤレス印刷は、マルチタスクに対応したデバイス上でのみ利用できます。`UIPrintInteractionController`オブジェクトを使用して、アプリケーションから印刷機能を利用できるかを検出できます。

アプリケーションに印刷機能を組み込む方法については、『*Drawing and Printing Guide for iOS*』の「Printing」を参照してください。

データ保護

iOS 4.0で導入されたデータ保護により、機密性の高いユーザデータを扱うアプリケーションは、一部のデバイスで利用可能な内蔵の暗号化機能を利用できるようになりました。特定のファイルを保護対象として指定すると、システムはそのファイルを暗号化された形式でディスク上に格納します。デバイスがロックされている間、アプリケーションからも侵入者からもファイルの内容にはアクセスできません。一方、ユーザがデバイスをロック解除すると、暗号鍵が作成されてアプリケーションからファイルにアクセスできるようになります。

iOS 5以降、データ保護の機能に、保護対象ファイルのセキュリティレベルが追加されました。このレベルでは、ユーザがデバイスをロックしていても、開いているファイルにアクセスできるようになります。また、デバイスをブートして最初にロック解除した後、再びユーザがデバイスをロックしたとしても、やはりアクセス可能です。

データ保護を実装するには、保護対象のデータの作成方法と管理方法を理解しておく必要があります。アプリケーションは、データの作成時にデータのセキュリティを確保し、ユーザがデバイスをロックまたはロック解除したときのアクセスの変更に備えるよう設計されなければなりません。

アプリケーションのファイルにデータ保護を追加する方法の詳細については、『*iOS App Programming Guide*』を参照してください。

Apple Push Notificationサービス

iOS 3.0以降では、Apple Push Notificationサービスが、アプリケーションが実際には実行中でないときでも新着情報があることをユーザに通知する手段を提供します。このサービスを使用すると、ユーザデバイス上でいつでも、文字による通知のプッシュ、アプリケーションアイコンへのバッジの追加、音声による警告のトリガを行うことができます。これらのメッセージは、ユーザに、アプリケーションを開いて関連情報を受け取る必要があることを知らせます。

設計の観点から見ると、iOSアプリケーションのPush Notificationを機能させるには2つの部分が必要です。まず、アプリケーションは通知の配信を要求し、通知が配信されたら通知データを処理する必要があります。2つ目に、まず初めの段階に通知を生成するために、サーバ側のプロセスを提供する必要があります。このプロセスは、独自のローカルサーバ上に置き、Apple Push Notificationサービスと一緒に機能させて通知をトリガします。

アプリケーションを設定してRemote Notificationを使用する方法については、『*Local and Push Notification Programming Guide*』を参照してください。

Local Notification

ローカル通知はiOS 4.0で導入され、外部サーバに依存せずにローカルで通知を生成する手段をアプリケーションに提供することによって、既存のプッシュ通知メカニズムを補完します。バックグラウンドで実行中のアプリケーションは、重要なイベントが発生すると、ユーザの注意を引きつける手段としてローカル通知を使用できます。たとえば、バックグラウンドで実行中の道案内アプリケーションであれば、道を曲がるタイミングをユーザに知らせるためにローカル通知を使用できます。アプリケーションは、将来の日時を指定してローカル通知の配信をスケジューリングしたり、スケジューリングされた通知をアプリケーションが実行されていないときでも配信させることができます。

ローカル通知の利点は、通知がアプリケーションに依存しない点です。通知がスケジューリングされると、システムが通知の配信を管理します。アプリケーションは、通知が配信されるときに実行中である必要もありません。

ローカル通知の使用方法については、『*Local and Push Notification Programming Guide*』を参照してください。

Gesture Recognizer

iOS 3.2で導入されたGesture Recognizerは、ビューにアタッチするオブジェクトで、スワイプやピンチなど、一般的なタイプのジェスチャを検出するために使用します。Gesture Recognizerをビューにアタッチした後、ジェスチャが発生した時に実行するアクションをGesture Recognizerに伝えます。Gesture Recognizerオブジェクトは、未処理のタッチイベントを追跡し、所定のジェスチャに合わせてシステム定義の類推法を適用します。Gesture Recognizerがなければ、非常に複雑となる可能性のあるこうした作業をすべてデベロッパ自身で行わなければなりません。

UIKitには、すべてのGesture Recognizerのために基本動作を定義する`UIGestureRecognizer`クラスが含まれています。独自のカスタムGesture Recognizerサブクラスを定義することも、UIKit付属のサブクラスのうちの1つを使用して、以下の標準ジェスチャのいずれかを処理することもできます。

- タップ（タップ数は任意）
- ピンチインとピンチアウト（拡大縮小用）
- パニング（ドラッグ）
- スワイプ（任意の方向に対応）
- 回転（互いに反対の方向への指の移動）
- 長押し

使用可能なGesture Recognizerの詳細については、『*Event Handling Guide for iOS*』を参照してください。

ファイル共有のサポート

ファイル共有はiOS 3.2で導入されました。ファイル共有によりアプリケーションは、iTunes 9.1以上を通じてユーザのデータファイルを利用できます。ファイル共有のサポートを宣言したアプリケーションでは、ユーザは`/Documents`ディレクトリの内容を利用できます。ユーザは必要に応じて、iTunesからこのディレクトリとの間でファイルを移動することができます。この機能では、アプリケーションは、同一デバイス上のほかのアプリケーションとの間でファイルを共有することはできません。この動作には、ペーストボードまたはDocument Interaction Controllerオブジェクトが必要です。

アプリケーションにファイル共有機能を持たせるには、次の手順を実行します。

1. アプリケーションの`Info.plist`ファイルに`UIFileSharingEnabled`キーを追加し、キーの値を`YES`に設定します。
2. 共有するファイルはすべて、アプリケーションの`Documents`ディレクトリに置きます。
3. デバイスがユーザのコンピュータに接続されていると、iTunesは選択されたデバイスの「Apps」タブに「File Sharing」セクションを表示します。
4. ユーザはこのディレクトリにファイルを追加したり、デスクトップにファイルを移動したりできます。

ファイル共有に対応したアプリケーションは、`Documents`ディレクトリにファイルが追加されたことを認識し、適切に対応できなければなりません。たとえば、アプリケーションはそのインターフェイスを通じて、新しいファイルの内容を利用できるようにすることが考えられます。このディレクトリのファイルの一覧をユーザに表示して、それらのファイルに対する操作を決めるようにユーザに求めたりしてはなりません。

`UIFileSharingEnabled`キーの詳細については、『*Information Property List Key Reference*』を参照してください。

ピアツーピアサービス

Game KitフレームワークはiOS 3.0で導入され、Bluetooth経由でピアツーピア接続を提供します。ピアツーピア接続を使用することで、近くにあるデバイスとの間で通信セッションを開始したり、マルチプレーヤーゲームで見られる機能の多くを実装したりできます。ピアツーピア接続は、主にゲームで使用されますが、その他のタイプのアプリケーションで使用することもできます。

アプリケーションでピアツーピア接続機能を使用する方法の詳細については、『*Game Kit Programming Guide*』を参照してください。Game Kitフレームワークの詳細については、「Game Kit Framework」（25 ページ）を参照してください。

標準のシステムView Controller

Cocoa Touchレイヤのフレームワークの多くには、標準のシステムインターフェイスを表示するためのView Controllerが含まれています。一貫したユーザ体験を提供するために、これらのView Controllerをアプリケーションで使用することが推奨されます。以下のいずれかのタスクを実行する必要のあるときは、対応するフレームワークに含まれるView Controllerを使用してください。

- **連絡先情報の表示または編集** — Address Book UIフレームワークのView Controllerを使用します。
- **カレンダーイベントの作成または編集** — Event Kit UIフレームワークのView Controllerを使用します。
- **電子メールまたはSMSメッセージの作成** — Message UIフレームワークのView Controllerを使用します。
- **ファイルの内容を開く、またはプレビューする** — UIKitフレームワークの`UIDocumentInteractionController`クラスを使用します。
- **写真を撮る、またはユーザのフォトライブラリから写真を選ぶ** — UIKitフレームワークの`UIImagePickerController`クラスを使用します。
- **ビデオクリップの撮影** — UIKitフレームワークの`UIImagePickerController`クラスを使用します。

ビューコントローラを表示し、消去する方法の詳細については、『*View Controller Programming Guide for iOS*』を参照してください。特定のView Controllerによって表示されるインターフェイスの詳細については、該当するフレームワークのリファレンスを参照してください。

外部ディスプレイのサポート

外部ディスプレイのサポートは、iOS 3.2で導入されました。これにより一部のiOSデバイスを、サポート対象のケーブル経由で外部ディスプレイに接続できます。接続すると、アプリケーションは関連付けられた画面を使用してコンテンツを表示できます。画面に関する情報（サポートされている解像度など）は、UIKitフレームワークのインターフェイスを通じて取得できます。このフレームワークを使用して、アプリケーションのウインドウを1つの画面、またはさらにもう1つの画面に関連付けることができます。

外部ディスプレイに接続する方法および外部ディスプレイにコンテンツを表示する方法の詳細については、『*View Programming Guide for iOS*』を参照してください。

Cocoa Touchフレームワーク

以下のセクションでは、Cocoa Touchレイヤのフレームワーク、およびこれらフレームワークが提供するサービスについて説明します。

Address Book UIフレームワーク

Address Book UIフレームワーク(`AddressBookUI.framework`)は、新規の連絡先を作成したり既存の連絡先を編集または選択したりするための、標準的なシステムインターフェイスの表示に使われるObjective-Cのプログラミングインターフェイスです。このフレームワークを利用するとアプリケーションに連絡先情報を表示するために必要な作業を容易に行えます。また、ほかのアプリケーションと同じインターフェイスを使用できるため、プラットフォーム全体の一貫性を保証できます。

Address Book UIフレームワークのクラスとその使い方については、『*Address Book Programming Guide for iOS*』および『*Address Book UI Framework Reference for iOS*』を参照してください。

Event Kit UIフレームワーク

Event Kit UIフレームワーク(`EventKitUI.framework`)は、iOS 4.0で導入され、カレンダー関連イベントの閲覧および編集のための標準システムインターフェイスを表示するView Controllerを提供します。このフレームワークは、Event Kitフレームワーク（「Event Kit Framework」（46 ページ）参照）のイベント関連データに基づいて構築されています。

このフレームワークのクラスやメソッドについては、『*Event Kit UI Framework Reference*』を参照してください。

Game Kitフレームワーク

Game Kitフレームワーク(`GameKit.framework`)は、iOS 3.0で導入され、このフレームワークを使用してアプリケーションにピアツーピアネットワーク機能を追加できます。特にこのフレームワークは、ピアツーピア接続機能とゲーム内ボイス機能を提供します。これらの機能はマルチプレーヤーネットワークゲームで最もよく使用されていますが、ゲーム以外のアプリケーションに組み込むこともできます。このフレームワークは、Bonjourの上に構築されているシンプルかつ強力な一連のクラスを通じてネットワーク機能を提供します。これらのクラスはネットワークの詳細の多くを抽象化します。このフレームワークを使用することで、ネットワークプログラミングの経験が浅いデベロッパでも、アプリケーションにネットワーク機能を組み込むことができます。

iOS 4.1で導入されたGame Centerは、Game Kitフレームワークの拡張機能で、以下の機能を提供します。

- ニックネーム ― ユーザがオンライン上で個人を識別するものを作成できます。ユーザはGame Centerにログインし、ニックネームを通じてほかのプレーヤーと匿名でやり取りします。プレーヤーは、特定の人を友だちとして登録できるだけでなく、ステータスメッセージを設定できます。
- Leaderboard ― アプリケーションがユーザのスコアをGame Centerに送信し、後でスコアを取得できます。この機能は、アプリケーションのすべてのユーザの中でのベストスコアを表示するのに使用できます。
- マッチメーク ― Game Centerにログインしているプレーヤーと接続することにより、アプリケーションでマルチプレーヤーゲームを実現できます。各プレーヤーは、マルチプレーヤーゲームに参加するために物理的にそばにいる必要はありません。
- アチーブメント（Achievement、成績） ― ゲームにおけるプレーヤーの達成状況を記録することができます。

iOS 5以降、`GKTurnBasedMatch`クラスを使って、ターン制のゲームを実装できるようになりました。状態がiCloudに格納されるので、中断してもまた続きを始めることができます。このゲームでは、対戦の状態情報を管理しており、どのプレーヤーが対戦を先に進めるか、を判断できます。

Game Kitフレームワークの使用方法の詳細については、『*Game Kit Programming Guide*』および『*Game Kit Framework Reference*』を参照してください。

iAdフレームワーク

iOS 4.0で導入されたiAdフレームワーク(`iAd.framework`)を使用すると、アプリケーションからバナー広告を配信できます。広告は、ユーザインターフェイスに統合し、必要なときに表示する標準ビューに組み込まれます。標準ビュー自体はAppleの広告サービスとともに機能し、広告コンテンツのロードと表示、および広告内でのタップ操作への応答に関連するすべての作業を自動的に処理します。

アプリケーションでiADを使う方法については、『*iAd Programming Guide*』を参照してください。

Map Kitフレームワーク

iOS 3.0で導入されたMap Kitフレームワーク(`MapKit.framework`)は、既存のビュー階層に統合できる、スクロール可能なマップインターフェイスを提供します。このマップを使用して、方向を示したり、関心のある地点を強調したりできます。アプリケーションでは、プログラミングによって地図の属性を設定したり、ユーザが地図に沿って移動できるようにしたりできます。地図にはカスタム画像またはコンテンツを注釈として付けることもできます。

iOS 4.0では、基本のMap Viewで、ドラッグ可能な注釈とカスタムオーバーレイを使用できるようになりました。ドラッグ可能な注釈を使用することで、注釈が地図上に配置された後、プログラミングまたはユーザとの対話によって注釈を再配置できます。オーバーレイは、複数の点で構成される複雑な注釈を作成する手段となります。たとえば、オーバーレイを使用して、バスのルート、選挙マップ、公園の境界線、天気予報（レーダーデータなど）といった情報を地図上に重ねることができます。

Map Kitフレームワークのクラスの使用法については、『*Location Awareness Programming Guide*』を参照してください。

Message UIフレームワーク

iOS 3.0で導入されたMessage UIフレームワーク(`MessageUI.framework`)は、ユーザの送信ボックス内での電子メールメッセージ作成とキューイングができるよう支援します。メッセージ作成の機能は、アプリケーションで表示するView Controllerインターフェイスで構成されます。このView Controllerのフィールドをあらかじめ埋めて、メッセージに含める宛先、件名、本文の内容、および任意の添付を設定することができます。View Controllerを表示した後、ユーザは送信前にメッセージを編集することもできます。

iOS 4.0以降では、このフレームワークはSMSメッセージ作成画面を表示するためのView Controllerを提供します。このView Controllerを使用することで、アプリケーションを離れることなく、SMSメッセージの作成と編集を行うことができます。メール作成インターフェイスと同様に、このインターフェイスでは、ユーザが送信前にメッセージを編集することができます。

Message UIフレームワークのクラスの詳細については、『*Message UI Framework Reference*』を参照してください。

Twitterフレームワーク

iOS 5で導入されたTwitterフレームワーク（`Twitter.framework`）には、ユーザに代わってTwitter要求を送信する機能、ツイートを組み立てて送信する機能があります。フレームワークは、要求に応じてそのユーザ認証部分を処理し、また、要求のHTTP部分を生成するためのテンプレートを提供し

ます（要求の内容を埋める処理についてはTwitter APIを参照）。ツイートの組み立てには`TWTweetComposeViewController`クラスを使います。このクラスは、指定されたツイート内容を送信するためのビューコントローラです。このクラスはユーザに、送信前にツイートを編集、修正する機会を与えるようになっています。

ユーザは、アプリケーションが自分の代わりにTwitterと通信することを許可するかどうか、「Settings」画面で制御できます。TwitterフレームワークをAccountsフレームワーク（`Accounts.framework`）と併用すれば、ユーザのアカウントにアクセスすることもできます。

Twitterフレームワークのクラスについては、『*Twitter Framework Reference*』を参照してください。Accountsフレームワークについては、「Accounts Framework」（43 ページ）を参照してください。

UIKitフレームワーク

UIKitフレームワーク(`UIKit.framework`)は、iOSにグラフィカルなイベント駆動型アプリケーションを実装するための主要なインフラストラクチャを提供します。iOSアプリケーションはこのフレームワークを使用して、以下に示す主要な機能を実装できます。

- アプリケーションの管理
- ユーザインターフェイスの管理（ストーリーボードやnibファイルの処理を含む）
- グラフィックス処理、ウインドウ管理（マルチディスプレイを含む）
- マルチタスク対応（「マルチタスク」（20 ページ）参照）
- 印刷機能（「印刷」（20 ページ）参照）
- 標準的なUIKitコントロールの外観のカスタマイズ（iOS 5以降）
- ほかのビューコントローラの内容を取り込むビューコントローラの実装
- タッチベースイベントおよびモーションベースイベントの処理
- 標準的なシステムビューとコントロールを表すオブジェクト
- テキストおよびWebコンテンツのサポート
- カット、コピー、ペースト
- ユーザインターフェイスコンテンツのアニメーション化
- URLスキームによるシステム上のほかのアプリケーションとの統合
- Apple Push Notificationサービス（「Apple Push Notificationサービス」（21 ページ）参照）
- 体の不自由なユーザのためのアクセシビリティのサポート
- Local Notificationのスケジューリングと配信（「Local Notification」（22 ページ）参照）
- PDF作成
- システムキーボードのように動作するカスタム入力ビューの使用
- システムキーボードとやり取りを行うカスタムテキストビューの作成

アプリケーション開発の基本的なコードを提供するほかに、UIKitは以下のようなデバイス固有機能にも対応します。

- 加速度センサー
- 内蔵カメラ（ある場合）
- ユーザのフォトライブラリ
- デバイス名およびモデル情報
- バッテリー状態情報
- 近接センサー情報
- 接続されたヘッドセットからのリモートコントロール情報

UIKitフレームワークのクラスの詳細については、『*UIKit Framework Reference*』を参照してください。

Mediaレイヤ

Mediaレイヤには、モバイルデバイス上で利用できる最高のマルチメディア体験を演出することを目的とする、グラフィックス、オーディオ、およびビデオの各テクノロジーが含まれています。このレイヤのテクノロジーは見た目もサウンドもすばらしいアプリケーションを簡単に開発できるように設計されています。

グラフィックステクノロジー

高品質のグラフィックスは、iOSアプリケーションの重要な部分です。アプリケーションを作成する、最も簡単で効率の良い方法は、あらかじめレンダリングされた画像とともに、UIKitフレームワークの標準的なビューとコントロールを使用して、システムに描画させることです。ただし、単純なグラフィックスでは不十分な状況があります。そのような場合は、アプリケーションのグラフィカルコンテンツを管理する以下のテクノロジーを使用できます。

- Core Graphics（別名Quartz）は、ベクトルベースおよび画像ベースのネイティブ2Dレンダリングを処理します。
- Core Animation（Quartz Coreフレームワークの一部）は、ビューやその他のコンテンツをアニメーション化するための高度なサポートを提供します。
- Core Imageはビデオ画像や静止画像を操作するための高度なサポートを提供します。
- OpenGL ESおよびGLKitは、ハードウェアによって加速化されたインターフェイスを使用して、2Dおよび3Dのレンダリングを支援します。
- Core Textは、テキストのレイアウトとレンダリングを処理する高度なエンジンを提供しています。
- Image I/Oには、ほとんどの画像形式の読み取り、書き込みを行うためのインターフェイスが用意されています。
- Assets Libraryフレームワークは、ユーザのフォトライブラリの写真やビデオにアクセスする手段を提供します。

ほとんどの場合、Retinaディスプレイを搭載したデバイス上で実行するアプリケーションは、ほとんどまたはまったく変更なしで動作します。描画するコンテンツは、高解像度画面に対応するために必要な拡大縮小が自動的に行われます。ベクトルベースの描画コードの場合、システムフレームワークが自動的にピクセルを追加してコンテンツの鮮明度を上げます。アプリケーション内で画像を使用する場合、UIKitが既存の画像の高解像度のバリエーションを自動的に読み込む機能を提供します。高解像度画面への対応に必要な要件の詳細については、『*iOS App Programming Guide*』の「Document Revision History」を参照してください。

グラフィックス関連のフレームワークに関する詳細については、「Mediaレイヤのフレームワーク」（32 ページ）の該当する部分を参照してください。

オーディオテクノロジー

iOSで利用可能なオーディオテクノロジーは、ユーザに豊かなオーディオ体験を提供することを意図して設計されています。この体験には、高品質オーディオの再生や録音の機能、そして特定のデバイス上におけるバイブレーションのトリガ機能が含まれています。

システムは、オーディオコンテンツの再生や録音を行ういくつかの方法を提供しています。以下に示すフレームワークは、上位レベルから下位レベルの順に並んでおり、Media Playerフレームワークが、使用可能な最上位レベルのインターフェイスを提供します。オーディオテクノロジーを選ぶときは、上位レベルのフレームワークの方がより使いやすく、一般的に推奨されることを覚えておいてください。下位レベルのフレームワークは、より柔軟性が高く制御しやすい一方、必要な作業が多くなります。

- Media Playerフレームワークは、ユーザのiTunesライブラリへの簡単なアクセス、および曲やプレイリストの再生を支援します。
- AV Foundationフレームワークは、オーディオの再生と録音を管理する、使いやすいObjective-Cの一連のインターフェイスを提供します。
- OpenALは、定位オーディオを提供するためのクロスプラットフォーム対応の一連のインターフェイスを提供します。
- Core Audioフレームワークは、オーディオコンテンツの再生と録音を行うための、シンプルかつ洗練されたインターフェイスを提供します。これらのインターフェイスを使用して、システムの警告音の再生、デバイスのバイブレーション機能のトリガ、ローカルまたはストリーミングのマルチチャンネルオーディオコンテンツのバッファリングと再生の管理を行うことができます。

iOSのオーディオテクノロジーは、以下のオーディオフォーマットに対応しています。

- AAC
- Apple Lossless (ALAC)
- A-law
- IMA/ADPCM (IMA4)
- リニアPCM
- µ-law
- DVI/Intel IMA ADPCM
- Microsoft GSM 6.10
- AES3-2003

オーディオ関連の各フレームワークの詳細については、「Mediaレイヤのフレームワーク」（32ページ）の該当する部分を参照してください。

ビデオテクノロジー

アプリケーションからムービーファイルを再生する場合、またはネットワークからのストリーミングを再生する場合のどちらであっても、iOSはビデオベースのコンテンツを再生するためのテクノロジーをいくつか用意しています。適切なビデオハードウェアを搭載したデバイス上では、これらのテクノロジーを使用してビデオをキャプチャしたり、アプリケーションにビデオを組み込んだりすることもできます。

また、ビデオコンテンツの再生と録音・録画を行う方法を複数提供しており、デベロッパは要件に応じて選ぶことができます。ビデオテクノロジーを選ぶときは、より上位レベルのフレームワークを使用したほうが、必要な機能に対応するための手間が軽減されるため、一般に推奨されます。以下に示すフレームワークは、上位レベルから下位レベルの順に並んでおり、Media Playerフレームワークが、使用可能な最上位レベルのインターフェイスを提供します。

- UIKitの`UIImagePickerController`クラスは、サポートされているカメラを搭載したデバイス上でビデオを録画するための標準インターフェイスを提供します。
- Media Playerフレームワークは、アプリケーションから全画面または画面の一部でムービーを表示するための、簡単に使える一連のインターフェイスを提供します。
- AV Foundationフレームワークは、ムービーのキャプチャと再生を管理する、Objective-Cの一連のインターフェイスを提供します。
- Core Mediaは、上位レベルのフレームワークによって使用される下位レベルのデータ型を記述し、メディアを操作するための下位レベルのインターフェイスを提供します。

iOSのビデオテクノロジーは、ファイル名拡張子が`.mov`、`.mp4`、`.m4v`、`.3gp`で、以下の圧縮規格を使用したムービーファイルの再生を支援します。

- H.264ビデオ：最高1.5Mbps、640×480ピクセル、毎秒30フレーム、H.264バージョンのLow-Complexityベースラインプロファイル（最高160KbpsのAAC-LCオーディオ）、48 kHz、`.m4v`、`.mp4`、`.mov`ファイルフォーマットのステレオオーディオ
- H.264ビデオ：最高768Kbps、320×240ピクセル、毎秒30フレーム、最高レベル1.3のベースラインプロファイル（最高160KbpsのAAC-LCオーディオ）、48 kHz、`.m4v`、`.mp4`、`.mov`ファイルフォーマットのステレオオーディオ
- MPEG-4ビデオ：最高2.5Mbps、640×480ピクセル、毎秒30フレーム、シンプルプロファイル（最高160KbpsのAAC-LCオーディオ）、48 kHz、`.m4v`、`.mp4`、`.mov`ファイルフォーマットのステレオオーディオ
- 複数のオーディオフォーマット（「オーディオテクノロジー」（30 ページ）で示したフォーマットを含む）

Mediaレイヤのビデオ関連の各フレームワークの詳細については、「Mediaレイヤのフレームワーク」（32 ページ）の該当する部分を参照してください。`UIImagePickerController`クラスの使い方については、『*Camera Programming Topics for iOS*』を参照してください。

AirPlay

AirPlayは、アプリケーションからApple TVや他社製のAirPlayスピーカー／レシーバーに対して、オーディオをストリーミング配信するためのテクノロジーです。AirPlayのサポートは、AV Foundationフレームワークおよび Core Audioファミリのフレームワークに含まれています。これらのフレームワークを使用して再生するオーディオコンテンツは、自動的にAirPlay配布対応となります。ユーザがAirPlayを使用したオーディオ再生を選ぶと、システムが自動的に経路を選択します。

iOS 5ではどのアプリケーションでも、AirPlayを使って、iPad 2の内容をそのままApple TV 2に表示できるようになりました。（同一内容ではなく）別の表示がしたい場合は、新しいウインドウオブジェクトを`UIScreen`オブジェクトに割り当て、これをAirPlay経由でiPad 2に接続します。iOS 5では、コンテンツをAirPlay経由で配信する方法として、AV Foundationフレームワークの`AVPlayer`クラス、UIKitフレームワークの`UIWebView`クラスも使えます。Media Playerフレームワークにはさらに、「Now Playing」情報をいくつかの箇所に表示する（AirPlay経由で配信するコンテンツへの埋め込み表示を含む）機能も加わりました。

AirPlayの活用方法については、『*AirPlay Overview*』を参照してください。

Mediaレイヤのフレームワーク

以下のセクションでは、Mediaレイヤのフレームワーク、およびこれらフレームワークが提供するサービスについて説明します。

Assets Libraryフレームワーク

iOS 4.0で導入されたAssets Libraryフレームワーク(`AssetsLibrary.framework`)は、ユーザのデバイスから写真やビデオを取得する、クエリベースのインターフェイスを提供します。このフレームワークを使用して、ユーザが保存したフォトアルバムに含まれているアイテムや、デバイスにインポートされた写真やビデオなど、通常「写真（Photos）」アプリケーションが管理するものと同じアセットにアクセスできます。新しい写真やビデオをユーザが保存したフォトアルバムに保存することもできます。

このフレームワークのクラスやメソッドについては、『*Assets Library Framework Reference*』を参照してください。

AV Foundationフレームワーク

iOS 2.2で導入されたAV Foundationフレームワーク(`AVFoundation.framework`)には、オーディオコンテンツを再生するためのObjective-Cクラスが含まれています。これらのクラスを使用して、ファイルベースまたはメモリベースの任意の長さのサウンドを再生できます。複数のサウンドを同時に再生したり、各サウンドのさまざまな再生の側面を制御できます。iOS 3.0から、このフレームワークにはオーディオの録音およびオーディオセッション情報の管理を行うためのサポートも追加されています。

さらにiOS 4.0からは、次の機能に対応するようにサービスが拡張されました。

- メディアアセットの管理

- メディアの編集
- ムービーのキャプチャ
- ムービーの再生
- トラックの管理
- メディアアイテムのメタデータ管理
- ステレオパンニング
- サウンド間の正確な同期
- データフォーマット、サンプルレート、チャンネル数など、サウンドファイルに関する詳細を決定するObjective-Cインターフェイス

iOS 5では、AV Foundationフレームワークの`AVPlayer`クラスに、AirPlay経由でオーディオ/ビデオを配信する機能が加わりました。AirPlayのサポートはデフォルトで有効になっていますが、必要ならばアプリケーション側で無効にすることができます。

AV Foundationフレームワークは、iOSにおいてオーディオとビデオの録音・録画と再生を行うためのシングルソースです。メディアアイテムの操作と管理を、より上位レベルのフレームワークよりもはるかに洗練された方法で支援します。

AV Foundationフレームワークのクラスの詳細については、『*AV Foundation Framework Reference*』を参照してください。

Core Audio

オーディオのネイティブサポートが、表3-1に示すCore Audioファミリのフレームワークによって提供されています。**Core Audio**は、ステレオベースのオーディオの操作を支援するC言語ベースのインターフェイスです。iOSのCore Audioを使用すると、アプリケーション内でオーディオの生成、録音、ミキシング、再生を行うことができます。また、バイブレーション機能に対応したデバイスでは、Core Audioを使用してデバイス上でバイブレーション機能をトリガできます。

表 3-1 Core Audioフレームワーク

フレームワーク	サービス
`CoreAudio.framework`	Core Audio全体で使われるオーディオデータ型を定義します。
`AudioToolbox.framework`	オーディオファイルおよびストリームに対して再生および録音サービスを提供します。このフレームワークは、オーディオファイルの管理、システムの警告音の再生、および対応デバイスにおけるバイブレーション機能のトリガを可能にします。
`AudioUnit.framework`	オーディオ処理モジュールである内蔵Audio Unitを使用するためのサービスを提供します。

Core Audioの詳細については、『*Core Audio Overview*』を参照してください。Audio Toolboxフレームワークでサウンドを再生する方法については、『*Audio Queue Services Programming Guide*』および『*Audio Toolbox Framework Reference*』を参照してください。

Core Graphicsフレームワーク

Core Graphicsフレームワーク(`CoreGraphics.framework`)には、Quartz 2D描画API用のインターフェイスが含まれています。**Quartz**は、Mac OS Xで使われているものと同じ、高度なベクトルベースの描画エンジンです。パスベースの描画、アンチエイリアス化されたレンダリング、グラデーション、画像、色、座標空間の変換、およびPDF文書の作成、表示、解析をサポートします。APIはC言語ベースですが、オブジェクトベースの抽象化を使用して基礎的な描画オブジェクトを表します。これにより、グラフィックスコンテンツの保存と再利用が簡単に行えます。

Quartzを使って図形を描画する方法については、『*Quartz 2D Programming Guide*』および『*Core Graphics Framework Reference*』を参照してください。

Core Imageフレームワーク

iOS 5で導入されたCore Imageフレームワーク（`CoreImage.framework`）には、ビデオや静止画像を操作する、強力なフィルタ群が組み込まれています。この組み込みフィルタを使えば、簡単な処理（写真のレタッチや修正）から高度な操作（顔面認識、特徴認識など）まで、さまざまなことが可能です。このフィルタには、もとの画像を直接変更せず、非破壊的に加工するようになっています。さらにCore Imageには、CPUやGPUの処理能力を最大限活かして、高速かつ効率よく処理できるという利点もあります。

`CIImage`クラスから、写真の画質を向上する、標準的なフィルタ群を利用できます。標準以外のフィルタを作成したい場合は、フィルタタイプに応じた`CIFilter`オブジェクトを作成、設定することになります。

Core Imageフレームワークのクラスやフィルタについては、『*Core Image Reference Collection*』を参照してください。

Core MIDIフレームワーク

iOS 4.2で導入されたCore MIDIフレームワーク(`CoreMIDI.framework`)は、ハードウェアキーボードやシンセサイザなどのMIDIデバイスと通信する標準的な手段を提供します。このフレームワークを使用して、MIDIメッセージの送受信を行ったり、Dockコネクタやネットワークを使用してiOSベースのデバイスに接続されたMIDI機器とやり取りを行ったりします。

このフレームワークの使い方については、『*Core MIDI Framework Reference*』を参照してください。

Core Textフレームワーク

iOS 3.2で導入されたCore Textフレームワーク(`CoreText.framework`)には、テキストのレイアウトとフォントの処理を行う、シンプルで高性能なCベースの一連のインターフェイスが含まれています。Core Textフレームワークは、画面上のテキストの配置の管理に使用できる、総合的なテキストレイアウトエンジンを提供します。管理するテキストには、さまざまなフォントやレンダリング属性を使用してスタイルを設定することもできます。

このフレームワークは、ワードプロセッサなど、高度なテキスト処理能力を必要とするアプリケーションでの使用を意図しています。単純な文字入力と表示だけで十分な場合は、UIKitフレームワークの既存のテキストクラスを引き続き使用してください。

Core Textインターフェイスの詳細については、『*Core Text Programming Guide*』および『*Core Text Reference Collection*』を参照してください。

Core Videoフレームワーク

iOS 4.0で導入されたCore Videoフレームワーク(`CoreVideo.framework`)は、Core Mediaフレームワークに対してバッファおよびバッファプールのサポートを提供します。ほとんどのアプリケーションは、このフレームワークを直接使用する必要はありません。

Image I/Oフレームワーク

iOS 4.0で導入されたImage I/Oフレームワーク(`ImageIO.framework`)は、画像データおよび画像メタデータのインポートとエクスポートのためのインターフェイスを提供します。このフレームワークは、Core Graphicsデータ型およびCore Graphics関数を使用し、iOSで利用できる標準的なあらゆる種類の画像に対応しています。

このフレームワークの関数やデータ型については、『*Image I/O Reference Collection*』を参照してください。

GLKitフレームワーク

iOS 5で導入されたGLKitフレームワーク（`GLKit.framework`）には、OpenGL ES 2.0アプリケーションの開発を容易にする、Objective-Cベースの一連のユーティリティクラスが組み込まれています。GLKitは、アプリケーション開発における、次の4つの重要な分野で支援機能を提供しています。

- `GLKView`クラス、`GLKViewController`クラスは、OpenGL ESに対応したビューやこれに関連するレンダリングループの、標準的な実装を提供します。ビューはアプリケーションに代わって、基盤となるフレームバッファオブジェクトを管理するので、アプリケーションはここに描画するだけで済みます。

- `GLKTextureLoader`クラスは、画像変換や画像読み込みのルーチンを提供します。アプリケーションは、テクスチャ画像を自動的にコンテキストに読み込むことができます。テクスチャは、同期読み込みと非同期読み込みの両方が可能です。非同期読み込みの場合、アプリケーションは完了ハンドラブロックに、コンテキストへの読み込みが完了したときの処理を記述します。

- GLKitフレームワークには、ベクトル、行列、四元数や、行列スタックの演算が実装されており、これはOpenGL ES 1.1と同等の機能です。

- `GLKBaseEffect`、`GLKSkyboxEffect`、`GLKReflectionMapEffect`の各クラスは、よく使われるグラフィックス演算を実装した、適応性の高いグラフィックスシェイダを提供します。特に`GLKBaseEffect`クラスは、OpenGL ES 1.1の仕様で規定されている光源モデル、素材モデルが実装されており、アプリケーションをOpenGL ES 1.1からOpenGL ES 2.0に移行する作業も容易です。

GLKitフレームワークのクラスについては、『*GLKit Framework Reference*』を参照してください。

Media Playerフレームワーク

Media Playerフレームワーク(`MediaPlayer.framework`)は、アプリケーションからオーディオコンテンツおよびビデオコンテンツを再生するための高レベルのサポートを提供します。このフレームワークを使用することで、標準のシステムインターフェイスを使用してビデオを再生できます。

iOS 3.0では、ユーザのiTunes音楽ライブラリにアクセスするためのサポートが追加され、楽曲トラックやプレイリストの再生、曲の検索、ユーザに対するメディアピッカーインターフェイスの表示を行えるようになりました。

iOS 3.2では、サイズ変更可能なビューからビデオを再生するための変更が加えられました（以前は全画面でのみ可能でした）。さらに、ムービー再生の設定と管理を支援する多数のインターフェイスが追加されました。

iOS 5では、ロック画面やマルチタスキングコントロールに、「Now Playing」情報を表示する機能が加わりました。この情報をAirPlay経由で配信するコンテンツに埋め込んで、Apple TVに表示することもできます。また、ビデオがAirPlay経由で配信されたものかどうか、を検出するインターフェイスもあります。

Media Playerフレームワークのクラスの詳細については、『*Media Player Framework Reference*』を参照してください。これらのクラスを使用してiTunesライブラリにアクセスする方法の詳細については、『*iPod Library Access Programming Guide*』を参照してください。

OpenALフレームワーク

Open Audio Library(OpenAL)インターフェイスは、アプリケーションで定位のオーディオを提供するためのクロスプラットフォームの標準です。これを使用すると、定位のオーディオ出力が必要なゲームやその他のプログラムに高性能で高品質なオーディオを実現できます。OpenALはクロスプラットフォームの標準であるため、OpenALを使用して記述されたiOS上のコードモジュールは、ほかの多くのプラットフォームに簡単に移植できます。

OpenALおよびその使い方の詳細については、http://www.openal.orgを参照してください。

OpenGL ESフレームワーク

OpenGL ESフレームワーク(`OpenGLES.framework`)には、2Dコンテンツや3Dコンテンツを描画するための各種ツールが用意されています。これはC言語ベースのフレームワークで、デバイスハードウェアと密接に連携して動作し、全画面のゲームアプリケーションに高いフレームレートを提供します。

OpenGLフレームワークは常にEAGLインターフェイスと組み合わせて使用します。これらのインターフェイスはOpenGL ESフレームワークの一部で、OpenGL ESの描画コードとUIKitによって定義されるネイティブウインドウオブジェクトとの間のインターフェイスとなります。

iOS 3.0以降では、OpenGL ESフレームワークは、OpenGL ES 2.0およびOpenGL ES 1.1のどちらのインターフェイス仕様にも対応しています。2.0の仕様では、フラグメントシェーダおよび頂点シェーダの使用が可能で、iOS 3.0以降を実行する特定のiOSデバイスでのみ利用できます。OpenGL ES 1.1のサポートは、iOSベースのすべてのデバイスおよびすべてのバージョンのiOSで利用できます。

アプリケーションでOpenGL ESを利用する方法の詳細については、『*OpenGL ES Programming Guide for iOS*』を参照してください。参考情報については、『*OpenGL ES Framework Reference*』を参照してください。

Quartz Coreフレームワーク

Quartz Coreフレームワーク(`QuartzCore.framework`)には、Core Animationインターフェイスが含まれています。**Core Animation**は、最適化されたレンダリングパスを使用して複雑なアニメーションや視覚効果を実装する、アニメーションと合成の高度なテクノロジーです。このテクノロジーは、アニメーションや効果を設定し、パフォーマンスを発揮できるようにハードウェアを使用してレンダリングされる、高水準のObjective-Cインターフェイスを提供します。Core Animationは、システムの標準的な動作の多くをアニメーション化する`UIView`などのUIKitクラスをはじめ、iOSの多くの部分に組み込まれています。また、このフレームワークのOvjective-Cインターフェイスを使用してカスタムアニメーションを作成することもできます。

アプリケーションでCore Animationを利用する方法の詳細については、『*Core Animation Programming Guide*』および『*Core Animation Reference Collection*』を参照してください。

Core Servicesレイヤ

Core Servicesレイヤには、すべてのアプリケーションで使用する基本的なサービスが含まれています。これらのサービスを直接使わないとしても、システム内のほとんどの部分がこれらの上に構築されています。

上位レベルの機能

以下のセクションでは、Core Servicesレイヤで利用できる主要なテクノロジーの一部を説明します。

iCloudストレージ

iOS 5で導入されたiCloudストレージを使えば、アプリケーションはユーザ文書やデータを、ある中央の集中保存ストレージに書き込むようになります。これにはほかのコンピュータやiOSデバイスからもアクセスできます。iCloudを利用して、ユーザ文書をどこからでもアクセスできるようにすれば、意識的に同期や転送の操作をすることなく、どのデバイスからでも文書を表示し、編集できるようになります。文書をユーザのiCloudアカウントに格納すれば、安全性も確保されます。ユーザがデバイスを紛失しても、文書はiCloudストレージ上にあるので、失うことはありません。

アプリケーションがiCloudストレージを活用する方法は2つあり、それぞれ異なる用途を想定しています。

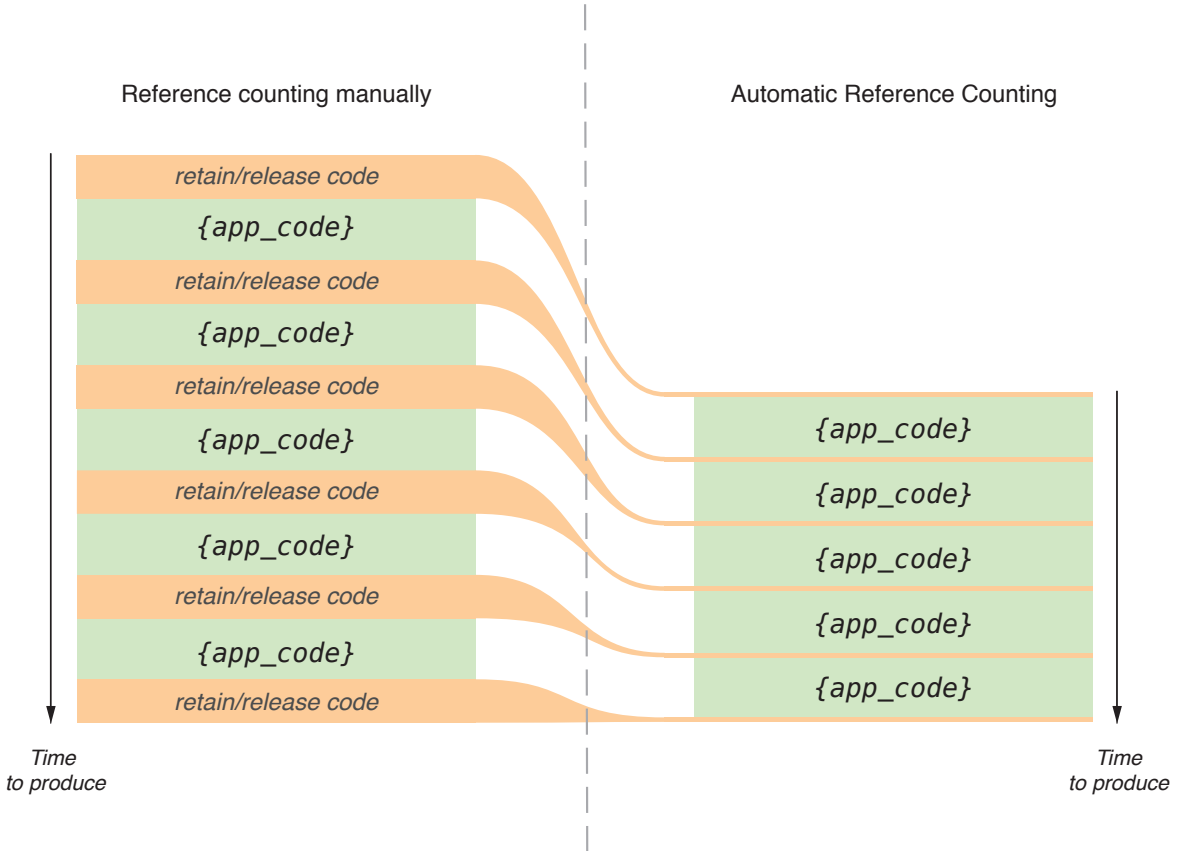
- **iCloud文書ストレージ** — ユーザ文書やデータを、ユーザのiCloudアカウントに格納します。
- **iCloudキー値データストレージ** — 少量のデータを、複数のアプリケーション間で共有します。

アプリケーションの多くはiCloud文書ストレージを使い、ユーザのiCloudアカウントに置かれている文書を共有します。iCloudストレージと聞いて普通思い浮かべる機能はこれでしょう。ユーザは、文書がデバイス間で共有されているかどうか、だけを意識していればよく、どのデバイスからでも文書を表示、管理できます。一方、iCloudキー値データストレージは、ユーザの目に触れるものではありません。アプリケーションがごく少量（数十キロバイト程度）のデータを、ほかで動作している同じアプリケーションと共有するためのものです。この機能を使って保存するのは、重要なデータではなく、環境設定など付随的なアプリケーションデータです。

iCloudの操作機能をアプリケーションに組み込む方法の概要については、『*iOS App Programming Guide*』を参照してください。

自動参照カウント

iOS 5で導入された**自動参照カウント（ARC、Automatic Reference Counting）**は、Objective-Cのオブジェクトのライフサイクル管理処理を容易にする、コンパイラレベルの機能です。オブジェクトをいつ獲得、解放するか、開発者が意識して管理する代わりに、ARCがオブジェクトのライフサイクル管理に必要な事項を評価し、コンパイル時に、自動的に適切なメソッド呼び出しを挿入します。



ARCの導入により、従来のiOSで使われていた、管理メモリモデル型のプログラミングスタイルは不要になりました。今後作成する新しいプロジェクトでは、自動的にARCを使うようになります。また、既存のプロジェクトも、Xcodeの移行ツールで、ARCを使うよう変換できます。移行の手順については、『*Xcode New Features User Guide*』を参照してください。また、ARCそのものについては、『*Programming With ARC Release Notes*』を参照してください。

ブロックオブジェクト

iOS 4.0で導入されたブロックオブジェクトは、CおよびObjective-Cのコードに組み込むことのできる、Cレベル言語の構成体です。ブロックオブジェクトは、基本的には匿名関数およびその関数に伴うデータであり、ほかの言語では*クロージャ*やラムダと呼ばれているものです。ブロックは特に、コールバックとして使用したり、または実行するコードと関連するデータを簡単に組み合わせる手段が必要な場合に使用できます。

iOSでは、ブロックは一般的に以下のシナリオで使用されます。

- デリゲートおよびデリゲートメソッドの代替として
- コールバック関数の代替として
- 1回限りの操作の終了ハンドラを実装するため
- コレクション内のすべての要素に対するタスクの実行を容易にするため

- ディスパッチキューとともに非同期タスクを実行するため

ブロックオブジェクトの概要および使い方については、『*A Short Practical Guide to Blocks*』を参照してください。ブロックについては、『*Blocks Programming Topics*』を参照してください。

Grand Central Dispatch

iOS 4.0で導入されたGrand Central Dispatch (GCD)はBSDレベルのテクノロジーで、アプリケーション内のタスクの実行の管理に使用します。GCDは非同期プログラミングモデルと高度に最適化されたコアを一体化させることで、スレッドに代わる簡便で効率的な手段を提供します。また、ファイル記述子の読み取りと書き込み、タイマーの実装、シグナルイベントおよびプロセスイベントの監視など、さまざまな下位レベルタスクに対する便利な代替手段も提供します。

アプリケーションでGCDを使う方法については、『*Concurrency Programming Guide*』を参照してください。個々のGCD関数の詳細については、『*Grand Central Dispatch (GCD) Reference*』を参照してください。

In-App Purchase

iOS 3.0で導入されたIn-App Purchaseにより、アプリケーション内部からコンテンツおよびサービスを提供することができます。In-App Purchaseは、ユーザのiTunesアカウントを使用した会計処理に必要なインフラストラクチャを提供するStore Kitフレームワークを使用して実装されています。アプリケーションは、全体的なユーザ体験、および購入可能なコンテンツやサービスの表示を処理します。

In-App Purchaseのサポートについて詳しくは、『*In-App Purchase Programming Guide*』を参照してください。Store Kitフレームワークの詳細については、「Store Kit Framework」（47 ページ）を参照してください。

SQLite

SQLiteライブラリにより、軽量なSQLデータベースをアプリケーションに組み込むことができます。これにより、リモートで別にデータベースサーバプロセスを実行する必要がありません。アプリケーションからローカルデータベースファイルを作成し、ファイル内のテーブルやレコードを管理できます。ライブラリは汎用の目的のために設計されていますが、データベースのレコードにすばやくアクセスできるようにさらに最適化されています。

SQLiteライブラリのアクセス用ヘッダファイルは<*iOS_SDK*>`/usr/include/sqlite3.h`にあります。<*iOS_SDK*>は、Xcodeのインストールディレクトリ内のターゲットSDKへのパスです。SQLiteの使用の詳細については、http://www.sqlite.orgを参照してください。

XMLサポート

Foundationフレームワークは、XMLドキュメントから要素を取得するための`NSXMLParser`クラスを提供しています。XMLコンテンツの操作の追加サポートは、libXML2ライブラリによって提供されています。このオープンソースライブラリを利用すると任意のXMLデータをすばやく解析したり記述したりできます。また、XMLコンテンツをHTMLに変換することもできます。

libXML2ライブラリのアクセス用ヘッダファイルは*<iOS_SDK>*/usr/include/libxml2/ディレクトリにあります。*<iOS_SDK>*は、Xcodeのインストールディレクトリ内のターゲットSDKへのパスです。libXML2の使用に関する詳細については、http://xmlsoft.org/index.htmlを参照してください。

Core Servicesフレームワーク

以下のセクションでは、Core Servicesレイヤのフレームワーク、およびこれらフレームワークが提供するサービスについて説明します。

Accountsフレームワーク

iOS 5で導入されたAccountsフレームワーク（Accounts.framework）は、ユーザアカウントにログインするための「シングルサインオン」モデルを提供します。「シングルサインオン」が利用できると、アカウントに関係するログイン情報が必要な場合でも、アプリケーションがその都度ユーザに問い合わせることがなくなるので、使い勝手がよくなります。また、アプリケーションはアカウント認証処理を委ねることができるので、開発モデルも簡明になります。iOS 5.0では、このフレームワークをTwitterフレームワークと併用することにより、ユーザのTwitterアカウントにアクセスできます。

Accountsフレームワークのクラスについては、『*Accounts Framework Reference*』を参照してください。

Address Bookフレームワーク

Address Bookフレームワーク(AddressBook.framework)は、ユーザのデバイス上に格納されている連絡先情報にプログラムからアクセスするための手段を提供します。アプリケーションが連絡先情報を使用する場合は、このフレームワークを使用してユーザの連絡先データベース内のレコードにアクセスしたり修正したりできます。たとえば、チャットプログラムでこのフレームワークを使用して、チャットセッションを開始可能な連絡先リストを取得したりその連絡先をカスタムビューに表示したりすることができます。

Address Bookフレームワークの関数の詳細については、『*Address Book Framework Reference for iOS*』を参照してください。

CFNetworkフレームワーク

CFNetworkフレームワーク(CFNetwork.framework)は、ネットワークプロトコルを処理するためにオブジェクト指向の抽象化を使用する、高性能なC言語ベースのインターフェイスセットです。これらの抽象化によって、プロトコルスタックに対して細かな制御が可能になり、BSDソケットなどの下位レベルの構成体を使うのが簡単になります。このフレームワークを使用することで、FTPサーバおよびHTTPサーバとの通信やDNSホスト解決などの作業が簡単に行えます。CFNetworkフレームワークを使用することで、以下のことが可能になります。

- BSDソケットの使用
- SSLまたはTLSを使用した暗号化接続の作成

- DNSホストの解決
- HTTPサーバ、認証HTTPサーバ、およびHTTPSサーバの使用
- FTPサーバの使用
- Bonjourサービスの公開、解決、ブラウズ

CFNetworkは、物理的にも論理的にもBSDソケットを基にしています。CFNetworkの使い方については、『*CFNetwork Programming Guide*』および『*CFNetwork Framework Reference*』を参照してください。

Core Dataフレームワーク

iOS 3.0で導入されたCore Dataフレームワーク(`CoreData.framework`)は、Model-View-Controllerアプリケーションのデータモデルを管理するためのテクノロジーです。Core Dataは、すでにデータモデルが高度に構造化されているアプリケーションでの使用を目的としています。プログラムでデータ構造を定義する代わりに、Xcodeのグラフィカルなツールを使ってデータモデルを表すスキーマを作成できます。実行時には、Core Dataフレームワークを通してデータモデルエンティティのインスタンスの作成、管理、使用が行われます。

アプリケーションのデータモデルを管理することにより、Core Dataはアプリケーションの作成に必要なコード量を大幅に削減します。Core Dataは以下の機能も提供します。

- パフォーマンスの最適化のためのSQLiteデータベース内のオブジェクトデータ格納域
- Table Viewの結果を管理するための`NSFetchedResultsController`クラス
- 基本的なテキスト編集以外の取り消し(Undo)／やり直し(Redo)の管理
- プロパティ値の検証のサポート
- 変更の伝達のサポート、およびオブジェクト間の関係の一貫性の保証
- メモリ内のデータの、グループ化、フィルタ処理、編成のサポート

新しいアプリケーションの開発や、既存のアプリケーションに対して重要な更新を計画している場合は、Core Dataの使用を検討してください。iOSアプリケーションでのCore Dataの使用方法の例については、『*Core Data Tutorial for iOS*』を参照してください。Core Dataフレームワークのクラスの詳細については、『*Core Data Framework Reference*』を参照してください。

Core Foundationフレームワーク

Core Fundationフレームワーク(`CoreFoundation.framework`)は、iOSアプリケーションの基本的なデータ管理およびサービス機能を提供する、C言語ベースのインターフェイスセットです。このフレームワークは、以下をサポートしています。

- コレクションデータ型（配列、集合など）
- バンドル
- 文字列管理

- 日付と時刻の管理
- 未加工データブロック管理
- 環境設定管理
- URLおよびストリーム操作
- スレッドおよび実行ループ
- ポートおよびソケット通信

Core Foundationフレームワークは、Foundationフレームワークと密接に関係しており、同じような基本機能にObjective-Cインターフェイスを提供します。FoundationのオブジェクトとCore Foundationの型を混在させる必要がある場合は、この2つのフレームワーク間にある「toll-free bridging」（犠牲を伴わない橋渡し）を利用できます。**toll-free bridging**とは、いくつかのCore Foundation型とFoundation型は、どちらのフレームワークのメソッドや関数でも同じように使用できることを意味します。このサポートは、コレクションデータ型や文字列データ型など多くのデータ型に利用できます。それぞれのフレームワークのクラスと型の説明には、オブジェクトがtoll-free bridgingに対応しているかどうかが示されています。また、対応している場合には対応先のオブジェクトが示されています。

このフレームワークの詳細については、『*Core Foundation Framework Reference*』を参照してください。

Core Locationフレームワーク

Core Locationフレームワーク(`CoreLocation.framework`)は、位置と方角の情報をアプリケーションに提供します。位置情報について、このフレームワークは、オンボードGPS、携帯電話、またはWi-Fiの無線を使用して、ユーザの現在の経度と緯度を把握します。このテクノロジーをアプリケーションに組み込んで、ユーザに位置情報を提供することができます。たとえば、近隣のレストラン、店舗、または施設を検索するようなサービスでは、ユーザの現在位置に基づいて検索ができます。

iOS 3.0では、磁力センサーが搭載されたiOSベースのデバイスにおける、コンパスベースの方角情報へのアクセスのサポートが追加されました。

iOS 4.0では、ユーザの位置の変化を追跡するため携帯電話の基地局を使用する、低消費電力の位置監視サービスのサポートが導入されました。

位置情報および方角情報の収集にCore Locationを使用する方法の詳細については、『*Location Awareness Programming Guide*』を参照してください。

Core Mediaフレームワーク

iOS 4.0で導入されたCore Mediaフレームワーク(`CoreMedia.framework`)は、AV Foundationフレームワークが使用する下位レベルのメディアタイプを提供します。ほとんどのアプリケーションはこのフレームワークを使用する必要はありませんが、オーディオコンテンツとビデオコンテンツの作成と表示において、より正確な制御を必要とするデベロッパのためにこのフレームワークが提供されています。

このフレームワークの関数やデータ型については、『*Core Media Framework Reference*』を参照してください。

Core Telephonyフレームワーク

iOS 4.0で導入されたCore Telephonyフレームワーク(`CoreTelephony.framework`)は、携帯電話用無線機能が備わったデバイス上の電話ベースの情報をやり取りするためのインターフェイスを提供します。アプリケーションは、このフレームワークを使用して、ユーザの携帯電話サービスプロバイダに関する情報を取得することができます。携帯電話の呼び出しイベント（VoIPアプリケーションなど）が関係するアプリケーションは、呼び出しイベントの発生時に通知を受けることもできます。

このフレームワークのクラスやメソッドの使い方については、『*Core Telephony Framework Reference*』を参照してください。

Event Kitフレームワーク

iOS 4.0で導入されたEvent Kitフレームワーク(`EventKit.framework`)は、ユーザデバイス上のカレンダーイベントにアクセスするためのインターフェイスを提供します。このフレームワークを使用することで、既存のイベントを取得したり、ユーザのカレンダーに新規のイベントを追加したりできます。カレンダーイベントには、配信する必要がある日時の規則を設定できるアラームを含めることができます。

このフレームワークのクラスやメソッドについては、『*Event Kit Framework Reference*』を参照してください。さらには、「Event Kit UIフレームワーク」（25 ページ）も参照してください。

Foundationフレームワーク

Foundationフレームワーク(`Foundation.framework`)は、Core Foundationフレームワーク（「Core Foundationフレームワーク」（44 ページ）を参照）の機能の多くにObjective-Cラッパーを提供します。Foundationフレームワークは、以下の機能をサポートします。

- コレクションデータ型（配列、集合など）
- バンドル
- 文字列管理
- 日付と時刻の管理
- 未加工データブロック管理
- 環境設定管理
- URLおよびストリーム操作
- スレッドおよび実行ループ
- Bonjour
- 通信ポート管理

- 国際化
- 正規表現の一致
- キャッシュのサポート

Foundationフレームワークのクラスの詳細については、『*Foundation Framework Reference*』を参照してください。

Mobile Core Servicesフレームワーク

iOS 3.0で導入されたMobile Core Servicesフレームワーク(`MobileCoreServices.framework`)は、Uniform Type Identifiers (UTI)で使用される下位レベルのタイプを定義します。

このフレームワークで定義されているタイプについては、『*Uniform Type Identifiers Reference*』を参照してください。

Newsstand Kitフレームワーク

iOS 5で導入されたNewsstandは、雑誌や新聞の購読者向けに、集中管理の場を提供します。Newsstandを介して雑誌や新聞を配布しようとする出版者は、独自のiOSアプリケーションを、Newsstand Kitフレームワーク（`NewsstandKit.framework`）を使って開発できます。最新号の雑誌や新聞を、バックグラウンドでダウンロードする機能があります。ダウンロードを始めると、その後の処理はシステムが行い、最新号が読めるようになるとアプリケーションに通知します。

Newsstandによるダウンロード処理に用いるクラスについては、『*Newsstand Kit Framework Reference*』を参照してください。アプリケーションに通知を送る手順については、『*Local and Push Notification Programming Guide*』を参照してください。

Quick Lookフレームワーク

iOS 4.0で導入されたQuick Lookフレームワーク(`QuickLook.framework`)は、アプリケーションが直接サポートしていないファイルの内容をプレビューするための直接的なインターフェイスを提供します。このフレームワークは主に、ネットワークからファイルをダウンロードするアプリケーション、または未知のソースから取得したファイルを扱うアプリケーションを対象にしています。ファイルを取得した後、このフレームワークが提供するView Controllerを使用してユーザインターフェイス内に直接そのファイルの内容を表示します。

このフレームワークのクラスとメソッドについては、『*Quick Look Framework Reference*』を参照してください。

Store Kitフレームワーク

iOS 3.0で導入されたStore Kitフレームワーク(`StoreKit.framework`)は、iOSアプリケーション内からコンテンツやサービスを購入できる機能を提供します。たとえば、この機能を使用すると、ユーザは追加のアプリケーション機能のロック解除ができるようになります。また、ゲームデベロッパで

あれば、この機能を使用して追加のゲームレベルを提供することもできます。どちらの場合も、会計面の処理、ユーザのiTunes Storeアカウントを通じてのペイメントリクエストの処理、購入に関する情報のアプリケーションへの提供は、Store Kitフレームワークによって扱われます。

Store Kitは、会計面のトランザクションに焦点をしぼり、トランザクションが安全かつ正確に行われることを保証します。アプリケーションでは、購入インターフェイスの表示や適切なコンテンツのダウンロード（またはロック解除）など、会計面以外のトランザクションを処理します。この役割分担により、デベロッパはコンテンツ購入のユーザ体験を制御できます。デベロッパは、ユーザにどんな種類の購入インターフェイスを表示するか、またそれをいつ行うかを決定します。また、アプリケーションに最も適した配布メカニズムを決定します。

Store Kitフレームワークの使い方については、『*In-App Purchase Programming Guide*』および『*Store Kit Framework Reference*』を参照してください。

System Configurationフレームワーク

System Configurationフレームワーク(`SystemConfiguration.framework`)は、デバイスのネットワーク構成を特定するために使用できる、到達性のためのインターフェイスを提供します。このフレームワークは、Wi-Fi接続または携帯電話接続が使用中かどうか、そして特定のホストサーバがアクセス可能かどうかを特定するために使用できます。

このフレームワークのインターフェイスについては、『*System Configuration Framework Reference*』を参照してください。このフレームワークを使用してネットワーク情報を取得する方法の例については、『*Reachability*』サンプルコードプロジェクトを参照してください。

Core OSレイヤ

Core OSレイヤには、ほかのほとんどのテクノロジーの土台となる下位レベルの機能が含まれています。これらのテクノロジーは、アプリケーションで直接使用されていない場合でも、たいていは他のフレームワークによって使用されています。セキュリティや外部のハードウェアアクセサリとの通信を明示的に扱う必要のある状況では、このレイヤのフレームワークを使用する必要が生じます。

Accelerateフレームワーク

iOS 4.0で導入されたAccelerateフレームワーク（`Accelerate.framework`）には、DSP、線形代数、画像処理の計算を実行するためのインターフェイスが含まれています。これらのインターフェイスは、独自に作成することもできますが、このフレームワークを使用したほうがiOSベースデバイスのあらゆるハードウェア構成に対して最適化されているという利点があります。そのため、一度コードを記述すれば、すべてのデバイス上で効率良くコードを実行できます。

Accelerateフレームワークの機能については、『*Accelerate Framework Reference*』を参照してください。

Core Bluetooth

Core Bluetoothフレームワーク（`CoreBluetooth.framework`）を導入すれば、開発者は特に、Bluetooth Low-Energy（「LE」）アクセサリを活用できます。このフレームワークのObjective-Cインターフェイスを利用すれば、どのようなLEアクセサリがあるか調べ、いずれかのアクセサリを接続/切断し、サービス内で属性を読み書きし、サービスや属性の変更通知を受け取るよう登録するなど、さまざまな処理ができます。

Core Bluetoothフレームワークのインターフェイスについては、ヘッダファイルを参照してください。

External Accessoryフレームワーク

iOS 3.0で導入されたExternal Accessoryフレームワーク(`ExternalAccessory.framework`)は、iOSベースのデバイスに接続されているハードウェアアクセサリとの通信のためのサポートを提供します。アクセサリは、デバイスの30ピンDockコネクタを使用して接続するか、Bluetoothを使用してワイヤレスに接続できます。External Accessoryフレームワークは、利用可能な個々のアクセサリについての情報を取得し、通信セッションを初期化する方法を提供します。その後、サポートされているコマンドのいずれかを使用して、アクセサリを直接自由に操作できます。

このフレームワークの使用方法の詳細については、『*External Accessory Programming Topics*』を参照してください。iOSデバイス向けアクセサリの開発については、http://developer.apple.com/jpを参照してください。

Generic Security Servicesフレームワーク

iOS 5で導入されたGeneric Security Servicesフレームワーク（`GSS.framework`）を使えば、セキュリティ関係の標準的なサービス群をiOSアプリケーションに組み込むことができます。このフレームワークの基本的なインターフェイス仕様は、IETF RFC 2743およびRFC 4401に記述されています。iOSには、標準的なインターフェイスに加え、規格では指定されていないけれども多くのアプリケーションで必要な、証明書を管理するための機能がいくつか追加されています。

GSSフレームワークのインターフェイスについては、ヘッダファイルを参照してください。

Securityフレームワーク

内蔵のセキュリティ機能に加えてiOSでは、アプリケーションで管理するデータのセキュリティを保証できる、明示的なSecurityフレームワーク(`Security.framework`)も提供します。このフレームワークは、証明書、公開鍵と非公開鍵、および信用ポリシーを管理するインターフェイスを提供します。暗号技術的にセキュアな擬似乱数の生成をサポートします。また、キーチェーンへの証明書や暗号鍵の保存もサポートします。キーチェーンは、機密性の高いユーザデータのためのセキュリティ保護されたリポジトリです。

CommonCryptoライブラリは、対称暗号化、HMAC、およびダイジェストをサポートします。ダイジェスト機能は、OpenSSLライブラリに標準的に含まれる機能と本質的に互換性のある関数を提供します。ただし、OpenSSLライブラリはiOSでは利用できません。

iOS 3.0以降では、作成した複数のアプリケーション間でキーチェーンアイテムを共有できます。アイテムを共有すると、同じスイートのアプリケーションをよりスムーズに相互運用できます。たとえば、この機能を使用して、複数のユーザパスワードやその他の要素（つまり、共有しなければ、個々のアプリケーションから別々にユーザへの指示を要求される可能性のあるような要素）を共有できます。アプリケーション間でデータを共有するには、各アプリケーションのXcodeプロジェクトを正しいエンタイトルメントで設定する必要があります。

Securityフレームワークに関連した関数や機能の詳細については、『*Security Framework Reference*』を参照してください。キーチェーンにアクセスする方法については、『*Keychain Services Programming Guide*』を参照してください。Xcodeプロジェクトでのエンタイトルメントの設定の詳細については、『*iOS App Development Workflow Guide*』を参照してください。設定できるエンタイトルメントの詳細については、『*Keychain Services Reference*』の`SecItemAdd`関数の説明を参照してください。

System

システムレベルには、カーネル環境、ドライバ、オペレーティングシステムの下位レベルUNIXインターフェイスが含まれています。カーネル自体はMachを基にしており、オペレーティングシステムのすべての側面に関与しています。また、仮想メモリシステム、スレッド、ファイルシステム、ネットワーク、およびプロセス間通信を管理します。このレイヤのドライバは、利用可能なハード

ウェアとシステムフレームワークとの間のインターフェイスも提供します。セキュリティを確保する目的で、カーネルおよびドライバへのアクセスは、一部のシステムフレームワークとアプリケーションに限定されています。

iOSは、オペレーティングシステムの下位レベルの多くの機能にアクセスするためのインターフェイスを提供します。アプリケーションは、`LibSystem`ライブラリを通してこれらの機能にアクセスします。これらのインターフェイスはC言語ベースであり、以下をサポートしています。

- スレッド処理（POSIXスレッド）
- ネットワーク（BSDソケット）
- ファイルシステムアクセス
- 標準I/O
- BonjourサービスおよびDNSサービス
- ロケール情報
- メモリ割り当て
- 数学的演算処理

多くのCore OSテクノロジー用のヘッダファイルは*<iOS_SDK>*`/usr/include/`ディレクトリにあります。*<iOS_SDK>*は、Xcodeのインストールディレクトリ内のターゲットSDKへのパスです。この技術に関連する関数については、『*iOS Manual Pages*』を参照してください。

Cocoaからの移行

Cocoaデベロッパの方であれば、iOSで利用可能なフレームワークの多くはすでに馴染みがあるでしょう。iOSの基本的なテクノロジースタックは、多くの点でMac OS Xのものと似ています。しかし類似性がある一方、iOSのフレームワークは、Mac OS Xの対応するフレームワークとは完全に同じではありません。この章では、iOSアプリケーションを作成する際に直面する違いについて説明します。また、より大きな違いに対処する方法についても説明します。

注： この章は、Cocoaの用語とプログラミング技法をすでによく理解しているデベロッパを対象としています。Cocoaアプリケーション（およびiOSアプリケーション）で使われている基本的なデザインパターンの詳細については、『*Cocoa Fundamentals Guide*』を参照してください。

移行の際の一般的な注意

Model-View-Controllerデザインパターンに基づいて設計されているCocoaアプリケーションの場合は、アプリケーションの主要部分をiOSに移植するのは比較的簡単です。

データモデルの移行

FoundationフレームワークおよびCore Foundationフレームワークのクラスに基づくデータモデルを持つCocoaアプリケーションは、ほとんど（またはまったく）変更せずにiOSに移植できます。どちらのフレームワークもiOSでサポートされており、Mac OS Xのものと実質的には同じです。その違いのほとんどは、あまり重要でないか、iOS版のアプリケーションではいずれにしても削除しなければならない機能に関連するものです。たとえば、iOSアプリケーションはAppleScriptをサポートしません。違いの詳細なリストについては、「Foundationフレームワークの違い」（58 ページ）を参照してください。

CocoaアプリケーションがCore Dataの上に構築されている場合、iOS 3.0以降のiOSアプリケーションへデータモデルを移行できます。Core Dataは、以前のバージョンのiOSではサポートされません。iOSのCore Dataフレームワークは、バイナリおよびSQLiteデータストア（XMLデータストアではない）をサポートし、既存のCocoaアプリケーションからの移行をサポートします。サポートされるデータストアでは、Core Dataリソースファイルを iOSアプリケーションプロジェクトにコピーして、それを使用することができます。XcodeプロジェクトでのCore Dataの使い方の詳細については、『*Core Data Programming Guide*』を参照してください。

画面上にたくさんのデータを表示するCocoaアプリケーションの場合は、iOSに移植する際にデータモデルを単純化することを考えます。iOSでもたくさんのデータを使用する充実したアプリケーションを作成することはできますが、それがユーザのニーズを満たさない場合もあることを心に留めておいてください。モバイルユーザは一般に、最小限の時間で最も重要な情報だけを必要とします。画面のスペースが限られているため、一度にあまりに多くのデータをユーザに提供するのは現実的ではありません。また、そのデータをロードするための余分な処理によってアプリケーションが遅くなる可能性もあります。iOSで優れたパフォーマンスとユーザ体験を提供するには、Cocoaアプリケーションのデータ構造をリファクタリングする価値はあります。

ユーザインターフェイスの移行

iOSのユーザインターフェイスの構造と実装は、Cocoaアプリケーションのそれとは大きく異なります。たとえば、Cocoaのビューとウインドウを表すオブジェクトを例に見てみましょう。iOSとCocoaはともにビューとウインドウを表すオブジェクトを持っていますが、オブジェクトの動作はプラットフォームごとに少し異なります。さらに、iOSでは画面サイズが限られているためビューに表示するものを厳選する必要があります。また、タッチイベントを処理するビューには、ユーザが指で差すのに十分なスペースがなければなりません。

ビューオブジェクト自体の違いだけでなく、実行時のビューの表示方法にも大きな違いがあります。たとえば、Cocoaアプリケーションでたくさんのデータを表示する場合は、ウインドウのサイズを大きくしたり、複数のウインドウを使用したり、データを管理するタブを使用したりすることができます。iOSアプリケーションには固定サイズのウインドウが1つしかありません。このためアプリケーションは、情報を適当なサイズに分割してそれらを複数のビューで表示しなければなりません。ひとまとまりの新しい情報を表示したい場合は、画面上に新しいビューセットをプッシュして、既存のセットに置き換えます。これによってインターフェイスの設計が多少複雑になりますが、これは情報を表示するための非常に重要な手段であるため、iOSではこの種の編成のためにかなりのサポートを提供しています。

iOSのView Controllerは、ユーザインターフェイスの管理において重要な部分を占めます。View Controllerは、ビジュアルコンテンツの構築、画面上へのコンテンツの表示、および向きの変化などデバイス固有の動作の処理に使用します。View Controllerは、ビューの管理に加え、適切なタイミングでのビューのロードとアンロードをシステムとともに処理します。したがって、View Controllerの役割とアプリケーションでのそれらの使い方を理解することは、ユーザインターフェイスを設計するために非常に重要です。

View Controller、およびそれを使用したユーザインターフェイスの整理と管理の方法の詳細については、『*View Controller Programming Guide for iOS*』を参照してください。iOSでのユーザインターフェイスの設計方針についての一般的な情報は、『*iOS Human Interface Guidelines*』を参照してください。インターフェイスの作成に使用するウインドウとビュー、およびそれらを支える基盤アーキテクチャの追加情報については、『*View Programming Guide for iOS*』を参照してください。

メモリ管理

iOSでは常に、メモリ管理されたモデルを使ってオブジェクトの保持、解放、自動解放を行います。ガベージコレクションは、iOSではサポートされません。

iOSベースのデバイスでは、メモリはMacintoshコンピュータよりもさらに厳しく制限されるため、自動解放プールの使用を調整して、自動解放されたオブジェクトの蓄積を防ぐ必要があります。可能な場合は常に、オブジェクトを自動解放するのではなく、直接的に解放する必要があります。たくさんのオブジェクトを短いループ内で割り当てる場合は、これらのオブジェクトを直接開放するか、またはメモリ内に使用されないオブジェクトが作成されないよう、より頻繁に自動解放プールを空にする必要があります。ループの終了まで待機すると、メモリ不足の警告やアプリケーションの強制終了につながることがあります。

フレームワークの違い

iOSのフレームワークのほとんどはMac OS Xにも存在しますが、これらのフレームワークの実装と使用方法にはプラットフォームによる違いがあります。以降の各セクションでは、既存のCocoaデベロッパがiOSアプリケーションを開発する際に気付く重要な違いをいくつか示します。

UIKitとAppKit

iOSでは、UIKitフレームワークが、グラフィカルアプリケーションを作成するためのインフラストラクチャを提供し、イベントループの管理や、その他のインターフェイス関連のタスクを実行します。ただし、UIKitフレームワークはAppKitフレームワークとはまったく別物です。したがって、iOSアプリケーションを設計する際には別物として扱わなければなりません。このため、CocoaアプリケーションをiOSに移植する場合は、かなりの数のインターフェイス関連のクラスおよびロジックを置き換えなければなりません。表 6-1は、iOSアプリケーションに必須なものは何かを理解するのに役立つ、フレームワーク間の明確な違いのリストです。

表 6-1　インターフェイステクノロジーの違い

相違	説明
ビュークラス	UIKitは、デベロッパ向けに厳選されたカスタムビューおよびコントロールのセットを提供しています。AppKitにあるビューとコントロールの多くは、iOSベースのデバイスではうまく動作しない可能性があります。その他のビューにもiOS固有の代替手段があります。たとえば、`NSBrowser`クラスの代わりに、iOSではまったく異なるパラダイム（Navigation Controller）を使って階層情報の表示を管理します。 iOSで利用できるビューとコントロールと、その使い方の詳細については、『*iOS Human Interface Guidelines*』を参照してください。
ビューの座標系	UIKitビューの描画モデルは、1つの例外を除き、AppKitのモデルとほぼ同じです。AppKitビューは、ウインドウとビューの原点はデフォルトでは左下隅にあり、軸は右上に伸びます。UIKitでは、原点は左上隅にあり、それぞれの軸は下方向と右方向に伸びます。AppKitでは、この座標系は*変形座標系*として知られていますが、UIKitでは、デフォルトの座標系です。 ビューの座標系については、『*View Programming Guide for iOS*』を参照してください。

相違	説明
ビューとしてのウインドウ	概念上は、UIKitのウインドウとビューは、AppKitの場合と同様に同じ構成体を表しています。しかし、実装の観点では、この2つのプラットフォームはウインドウとビューをまったく異なった方法で実装しています。Mac OS Xでは、`NSWindow`クラスは`NSResponder`のサブクラスですが、iOSでは、`UIWindow`クラスは実際は`UIView`のサブクラスです。継承に関するこの変更は、UIKitのウインドウがCore Animationレイヤの背後にあり、ビューが実行するものとほとんど同じタスクを実行可能であることを意味します。 UIKitのすべてでウインドウオブジェクトを持つ理由は、オペレーティングシステム内でウインドウのレイヤ化をサポートするためです。たとえば、システムは、アプリケーションのウインドウの前景に表示される独立したウインドウにステータスバーを表示します。 iOSとMac OS Xのもう1つの違いは、ウインドウの使用方法です。Mac OS Xアプリケーションにはいくつでもウインドウを持たせることができますが、ほとんどのiOSアプリケーションではウインドウが1つしかありません。アプリケーションが表示するコンテンツを変更したい場合は、新規のウインドウを作成するのではなく、ウインドウのビューを切り替えます。
イベント処理	UIKitのイベント処理モデルは、AppKitのイベント処理モデルとは大きく異なります。マウスやキーボードのイベントを送付する代わりに、UIKitは、タッチイベントとモーションイベントをビューに送付します。これらのイベントには、さまざまなメソッドセットを実装する必要があります。また、イベント処理コード全体にいくつかの変更を加える必要もあります。たとえば、ローカルな追跡ループから待機中のイベントを抽出してタッチイベントを追跡してはいけません。 iOSアプリケーションのイベント処理については、『*Event Handling Guide for iOS*』を参照してください。
ターゲット／アクションモデル	UIKitは、3形態のアクションメソッドをサポートします。これは、1つしかサポートしないAppKitとは対照的です。UIKitのコントロールは、対話のさまざまなフェーズでアクションを起動できます。1つの対話に複数のターゲットを割り当てることもできます。このように、UIKitでは、1つのコントロールが1つの対話サイクルの間に複数のターゲットに複数の異なるアクションを送付することができます。 iOSアプリケーションのターゲット／アクションモデルについては、『*Event Handling Guide for iOS*』を参照してください。
描画と印刷サポート	UIKitの描画機能は、UIKitクラスのレンダリング要件をサポートするように拡張されています。このサポートには、画像のロードと表示、文字列の表示、カラー管理、フォント管理、および矩形の描画とグラフィックスコンテキストの取得を行ういくつかの関数が含まれています。UIKitには、汎用の描画クラスは含まれていません。いくつかほかの選択肢（QuartzとOpenGL ES）が、iOSにすでにあるためです。 iOS 4.2以降では、アプリケーションはUIKitの印刷サポートを使用して、近くのプリンタにワイヤレスでデータを送付できます。 グラフィックスや描画の詳細については、『*Drawing and Printing Guide for iOS*』を参照してください。

相違	説明
テキストサポート	iOSのテキストサポートは、主に電子メールとメモの作成を対象としています。UIKitクラスを使ってアプリケーションは、簡単な文字列と、多少複雑なHTMLコンテンツの表示や編集を行うことができます。 iOS 3.2以降では、Core TextフレームワークおよびUIKitフレームワークを通して、より高度なテキスト処理機能が提供されます。これらのフレームワークを使用して、高度なテキスト編集や表示ビューを実装したり、これらビュー用にカスタム入力方式をサポートしたりできます。 テキスト処理支援については、『*Text, Web, and Editing Programming Guide for iOS*』を参照してください。
アクセサメソッドとプロパティの使用	UIKitでは、クラス宣言によってプロパティを広範囲に利用しています。プロパティはMac OS Xバージョン10.5から導入されました。つまり、AppKitフレームワークで多くのクラスが作成された後に導入されたものです。UIKitでは、AppKitの同じgetterメソッドとsetterメソッドを単にまねるのではなく、クラスインターフェイスを簡素化する方法としてプロパティが使われます。 プロパティの使い方については、『*The Objective-C Programming Language*』の「Declared Properties」を参照してください。
コントロールとセル	UIKitのコントロールは、セルを使いません。セルは、AppKitではビューに代わる軽量な手段として使われます。UIKitでは、ビュー自体が非常に軽量なオブジェクトであるため、セルは必要ありません。命名規則に反して、`UITableView`クラスで使うように設計されたセルが、実際には`UIView`クラスに基づいています。
Table View	iOSの`UITableView`クラスは、UIKitフレームワークの`NSTableView`クラスと`NSOutlineView`クラスの中間と考えることができます。この2つのAppKitクラスからの機能を使って、小さな画面でのデータの表示により適したツールを作成しています。`UITableView`クラスは一度に1つのカラムを表示し、関連する行をセクションにグループ化できます。情報の階層的なリストを表示したり編集するための手段でもあります。 テーブルビューを作成、使用する方法については、『*Table View Programming Guide for iOS*』を参照してください。
メニュー	iOS向けに記述されたほぼすべてのアプリケーションは、それに相当するMac OS Xアプリケーションよりもコマンドセットがはるかに小さくなります。このため、iOSではメニューバーはサポートされず、また一般にメニューは必要ありません。必要なコマンドがわずかな場合は、一般に、ツールバーまたはボタンセットの方が適しています。データに基づくメニューに対しては、多くの場合はピッカーまたはナビゲーションコントローラインターフェイスが適しています。iOSのコンテキスト依存型コマンドについては、「Cut（カット）」、「Copy（コピー）」、「Paste（ペースト）」に加え（またはこれらの代わりに）編集メニューにコマンドを表示することができます。

相違	説明
Core Animationレイヤ	iOSでは、すべての描画サーフェスの背後にCore Animationレイヤがあり、多くのビュー関連プロパティに対して暗黙的なアニメーションサポートがすでに提供されています。アニメーションサポートが組み込まれているため、通常はコード内で明示的にCore Animationレイヤを使う必要はありません。ほとんどのアニメーションは、単に（そしてより直接的に）、対象となるビューのプロパティを変更するだけで実行できます。レイヤを直接使う必要があるのは、レイヤツリーを厳密に制御する必要があるときや、ビューレベルでは公開されていない機能を使う必要があるときだけです。 Core Animationレイヤが iOSの描画モデルにどのように組み込まれているかの詳細については、『*View Programming Guide for iOS*』を参照してください。

UIKitのクラスについては、『*UIKit Framework Reference*』を参照してください。

Foundationフレームワークの違い

同じバージョンのFoundationフレームワークを、Mac OS XとiOSの両方で利用できます。また、ほとんどのクラスが両方で利用できます。どちらのフレームワークも、値、文字列、コレクション、スレッド、およびその他の一般的なデータ型の管理をサポートしています。ただし、iOSに含まれないテクノロジーも存在します。iOSに含まれないテクノロジーは、関連クラスが利用できない理由とともに、表 6-2に示されています。この表には、できる限り代わりに利用できる代替テクノロジーを掲載しました。

表 6-2 iOSでは利用できないFoundationテクノロジー

技法	メモ
メタデータと述語の管理	iOS 5以降、ユーザのiCloudストレージに格納されたファイルのみが対象ですが、メタデータクエリが使えるようになりました。iOS 5までは、メタデータクエリはまったく使えませんでした。
分散オブジェクトとポート名サーバ管理	分散オブジェクトテクノロジーは利用できませんが、`NSPort`ファミリのクラスを使ってポートおよびソケットとやり取りできます。また、Core FoundationフレームワークとCFNetworkフレームワークを使ってネットワーク要件に対応することもできます。
Cocoaバインディング	Cocoaバインディングは iOSではサポートされません。代わりに、iOSでは、ターゲット／アクションモデルを若干変更したバージョンを採用しており、コード中のアクションの処理方法に柔軟性が加わっています。
Objective-Cのガベージコレクション	ガベージコレクションは、iOSではサポートされません。代わりに、メモリ管理されたモデルを使う必要があります。このモデルでは、オブジェクトを保持して所有権を要求し、不要になったらオブジェクトを解放します。
AppleScriptサポート	AppleScriptは、iOSではサポートされません。

Foundationフレームワークは、NSXMLParserクラスによってXML解析をサポートします。ただし、その他のXML解析クラス（NSXMLDocument、NSXMLNode、NSXMLElementを含む）は、iOSでは利用できません。NSXMLParserクラスのほかに、C言語ベースのXML解析インターフェイスを提供するlibXML2ライブラリを使用することもできます。

Mac OS Xで利用でき、iOSでは利用できない特定のクラスの一覧については、『*Foundation Framework Reference*』の「The Foundation Framework」にあるクラス階層図を参照してください。

その他のフレームワークの変更

表 6-3は、iOSのその他のフレームワークでの主な違いを示しています。

表 6-3　iOSとMac OS Xに共通のフレームワークにおける違い

フレームワーク	相違
AddressBook.framework	このフレームワークには、ユーザの連絡先情報にアクセスするためのインターフェイスが含まれています。名前は同じですが、iOS版のこのフレームワークはMac OS X版のものとは大きく異なります。 iOSでは、連絡先データにアクセスするためのCレベルのインターフェイスに加え、Address Book UIフレームワークのクラスを使用して、標準のピッカーや連絡先の編集インターフェイスを表示することもできます。 詳細については、『*Address Book Framework Reference for iOS*』を参照してください。
AudioToolbox.framework AudioUnit.framework CoreAudio.framework	これらのフレームワークのiOSバージョンは、主に、シングルまたはマルチチャンネルのオーディオコンテンツの録音、再生、ミキシングをサポートします。高度なオーディオ処理機能や、カスタムオーディオユニットプラグインはサポートされていません。ただし、iOSでは、該当するハードウェアを搭載したiOSベースのデバイス向けにバイブレートオプションをトリガする機能が追加されています オーディオサポートの使用方法の詳細については、『*Multimedia Programming Guide*』を参照してください。
CFNetwork.framework	このフレームワークには、Core Foundation Networkインターフェイスが含まれています。iOSでは、CFNetworkフレームワークはサブフレームワークではなく、最上位のフレームワークです。ただし、実際のインターフェイスのほとんどは変わっていません。 詳細については、『*CFNetwork Framework Reference*』を参照してください。

フレームワーク	相違
`CoreGraphics.framework`	このフレームワークにはQuartzインターフェイスが含まれています。iOSでは、Core Graphicsフレームワークはサブフレームワークではなく、最上位のフレームワークです。Quartzを使ってMac OS Xの場合とまったく同じ方法で、パス、グラデーション、シェーディング、パターン、色、イメージ、ビットマップを作成できます。ただし、PostScriptサポート、イメージのソースとデスティネーション、Quartz Display Servicesサポート、Quartz Event Servicesサポートなど、いくつかのQuartz機能はiOSには含まれていません。 詳細については、『*Core Graphics Framework Reference*』を参照してください。
`OpenGLES.framework`	OpenGL ESは、組み込みシステム向けに特別に設計されたOpenGLのバージョンです。OpenGLデベロッパなら、OpenGL ESインターフェイスには馴染みがあるはずです。ただし、OpenGL ESインターフェイスにもいくつか重要な相違点があります。1つは、使用可能なグラフィックスハードウェアを使って効率よく実行できる機能のみをサポートしているため、非常にコンパクトなインターフェイスである点です。もう1つは、デスクトップOpenGLで標準的に使用する拡張子の多くが、OpenGL ESでは使用できない可能性がある点です。このような相違がありますが、デスクトップ上で通常実行される操作のほとんどは、同じように実行できます。ただし、既存のOpenGLコードを移植する場合、iOSで異なる描画技術を使用するように、コードの一部を書き直す必要が生じるかもしれません。 iOSにおけるOpenGL ESのサポートについては、『*OpenGL ES Programming Guide for iOS*』を参照してください。
`QuartzCore.framework`	このフレームワークにはCore Animationインターフェイスが含まれています。Core Animationインターフェイスのほとんどは、iOSとMac OS Xとで同じです。ただし、iOSでは、レイアウトの制約を管理するためのクラスは利用できず、Core Imageフィルタの使用もサポートされていません。また、Core ImageとCore Video用のインターフェイス（これもMac OS Xバージョンのフレームワークに含まれています）はありません。 詳細については、『*Quartz Core Framework Reference*』を参照してください。
`Security.framework`	このフレームワークにはセキュリティインターフェイスが含まれています。iOSでは、このフレームワークは、暗号化と復号、擬似乱数生成、Keychainのサポートを提供することにより、アプリケーションデータのセキュリティ確保にフォーカスしています。このフレームワークには認証または承認のインターフェイスは含まれていません。また証明書の内容の表示もサポートされていません。さらに、Keychainインターフェイスは、Mac OS Xで使われているKeychainインターフェイスの簡略版です。 セキュリティ支援機能については、『*iOS App Programming Guide*』を参照してください。

フレームワーク	相違
`System-Configuration.framework`	このフレームワークにはネットワーク関連のインターフェイスが含まれています。iOSでは、このフレームワークには到達性のためのインターフェイスのみ含まれています。これらのインターフェイスを使って、デバイスがどのような方法でネットワークに接続されているか（たとえば、EDGE、GPRS、またはWi-Fiが使われているなど）を特定できます。

iOSデベロッパツール

iOS向けのアプリケーションを開発するには、IntelベースのMacintoshコンピュータとXcodeツールが必要です。Xcodeは、プロジェクト管理、コード編集、実行可能ファイルのビルド、ソースレベルのデバッグ、ソースコードのリポジトリ管理、パフォーマンスチューニング、その他多数の機能をサポートする、Appleの開発ツールスイートです。この開発ツールスイートの中心にあるのは、基本的なソースコード開発環境を提供する、Xcodeアプリケーション自身です。Xcodeは単なるツールではありません。以降の各セクションでは、iOS向けのソフトウェアの開発に使用する主要なアプリケーションを紹介します。

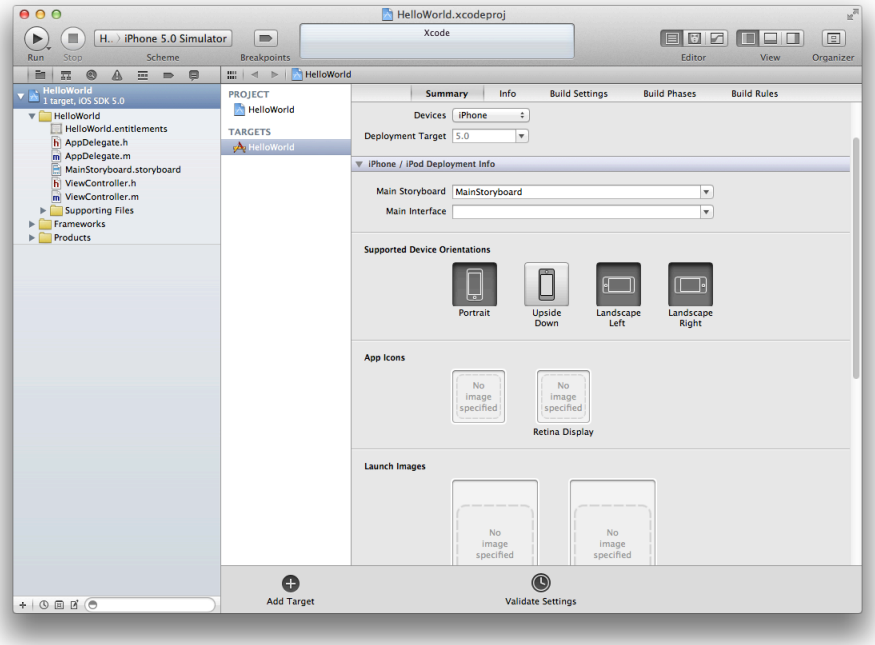
Xcode

開発作業の中心は、Xcodeアプリケーションです。Xcodeは、iOSプロジェクトとソースファイルの作成と管理、ユーザインターフェイスのアセンブル、実行可能ファイルへのコードのビルド、iOSシミュレータまたはデバイス上でのソースコードの実行とデバッグに必要なあらゆるツールを備えた統合開発環境（IDE）です。Xcodeには、iOSアプリケーションの開発を支援する、以下のようなさまざまな機能が組み込まれています。

- ソフトウェア製品を定義するためのプロジェクト管理システム
- 構文の色分け、コード補完、シンボルのインデックス化などの機能を備えたコード編集環境
- ストーリーボードやnibファイルを作成する統合エディタ
- Appleドキュメントを表示したり検索するための高度なドキュメントビューア
- 選択中のコードシンボルについての情報を表示するコンテキスト依存のインスペクタ
- 依存関係のチェック機能およびビルド規則の評価機能を備えた高度なビルドシステム
- C、C++、Objective-C向けのLLVMおよびClangのサポート
- C言語、C++言語、Objective-C言語、Objective-C++言語、その他の言語をサポートするGCCコンパイラ
- アプリケーションの振る舞いを検証し、問題が起こり得る箇所を指摘するスタティックアナライザ。
- GDBを使用した統合化されたソースレベルでのデバッグ
- 統合化されたソースコード管理のサポート
- DWARFおよびStabsの各デバッグ情報のサポート（DWARFデバッグ情報は、デフォルトですべての新規プロジェクトに生成されます）
- iOS開発デバイスの管理支援

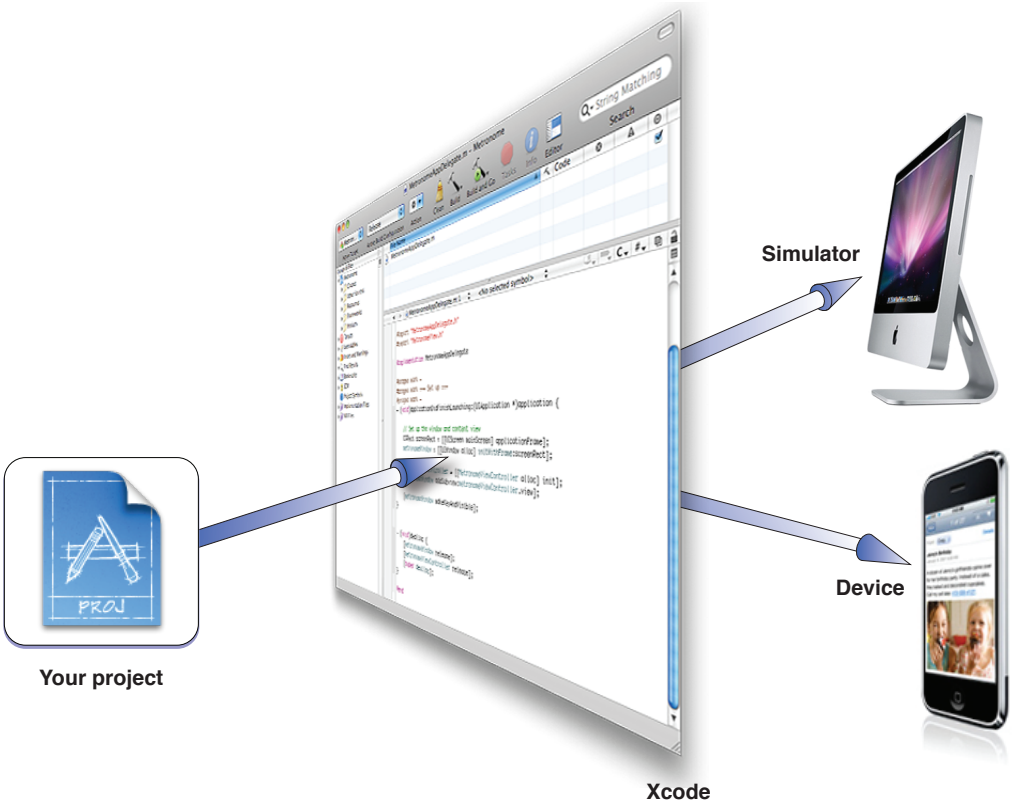
新しいiOSアプリケーションを作成するには、まず、Xcodeで新規プロジェクトを作成します。1つのプロジェクトで、ソースファイル、ビルド設定、それらの項目をすべてまとめるために必要なルールなど、作成するアプリケーションに関するすべての情報を管理します。すべてのXcodeプロジェクトの中心にあるのが、図 A-1に示すプロジェクトウインドウです。このウインドウから、作成中のアプリケーションの主要な要素すべてに簡単にアクセスできるようになっています。「グループとファイル（Groups & Files）」リストでは、各種ソースファイルおよびソースファイルから作成されたビルドターゲットなど、プロジェクト内のファイルを管理します。ツールバーからは、よく使うツールやコマンドにアクセスできます。次いで、編集に必要なペインを表示し、プロジェクトの中身をナビゲートし、デバッグし、各項目の追加情報を取得するよう、ワークスペースを設定できます。

図 A-1　Xcodeのプロジェクトウインドウ



Xcodeでアプリケーションをビルドするときには、iOSシミュレータ向けにビルドするか、またはあるデバイス向けにビルドするかを選択できます。シミュレータは、アプリケーションが基本的に要求どおりに動作するかをテストするためのローカル環境を提供します。アプリケーションの基本動作に問題がなければ、Xcodeでデバイス用にアプリケーションをビルドし、コンピュータに接続したiOSベースのデバイス上で実行できます。デバイス上でアプリケーションを実行する場合は最高のテスト環境が提供され、Xcodeを使ってそこで実行するコードに組み込みデバッガをアタッチできます。

図 A-2 Xcodeからのプロジェクトの実行

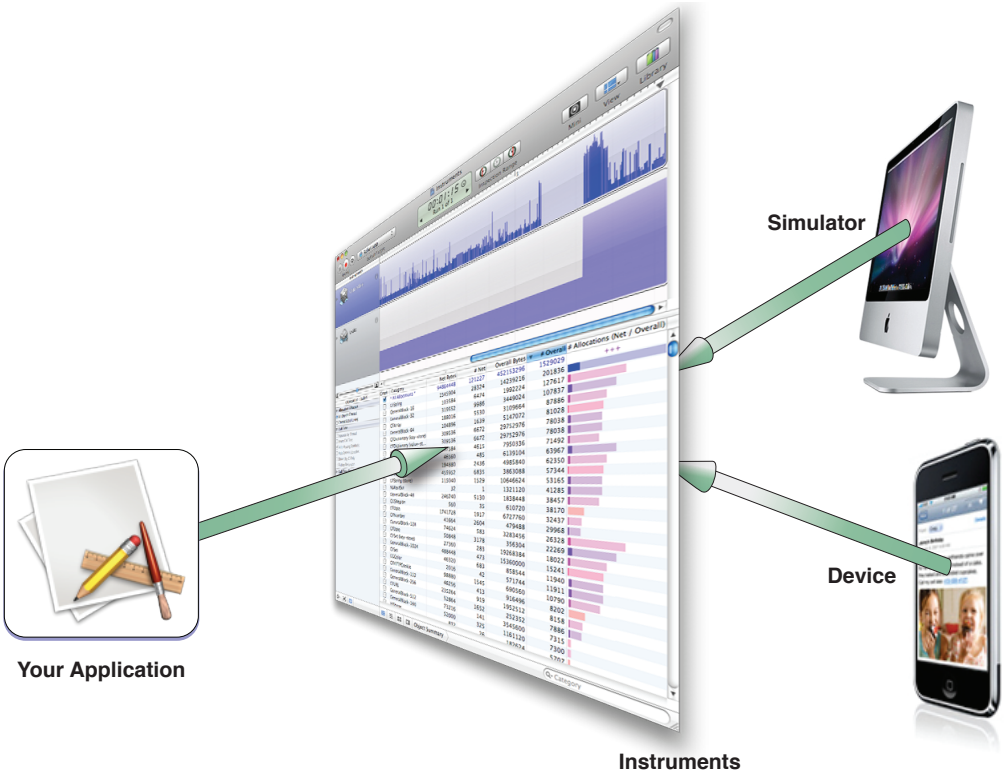


iOSでのプロジェクトのビルドおよび実行の方法の詳細については、『*iOS App Development Workflow Guide*』を参照してください。Xcode環境の詳細については、『*Xcode 4 User Guide*』を参照してください。

Instruments

自分のソフトウェアに最高のユーザ体験をもたらすことができるように、Instruments環境では、iOSアプリケーションをシミュレータまたはデバイス上で実行しながら、そのパフォーマンスを分析できます。Instrumentsは、実行中のアプリケーションからデータを収集し、タイムラインビューと呼ばれるグラフ表示でそのデータを表現します。アプリケーションのメモリ使用量、ディスクアクティビティ、ネットワークアクティビティ、グラフィックスのパフォーマンスについてのデータを収集できます。タイムラインビューには、あらゆる種類の情報が並んで表示されるため、特定の領域の動作だけでなく、アプリケーション全体の動作を相関させることができます。さらに詳細な情報を得るには、Instrumentsが収集する詳細なサンプルを表示することもできます。

図 A-3 Instrumentsを使ったアプリケーションのチューニング



タイムラインビューのほかに、Instrumentsは時間の経過に伴うアプリケーションの動作の分析に役立つツールを提供します。たとえば、「Instruments」ウインドウを使って、複数の実行結果からのデータを格納し、アプリケーションの動作が実際に向上しているか、あるいはまだ作業が必要かを見ることができます。これらの実行結果からのデータをInstruments文書に保存し、それをいつでも開くことができます。

iOSアプリケーションに対するInstrumentsの使用方法の詳細については、『*iOS App Development Workflow Guide*』を参照してください。Instrumentsの使い方全般については、『*Instruments User Guide*』を参照してください。

iOSのフレームワーク

この付録では、iOSの各種フレームワークを紹介します。これらのフレームワークは、iOSプラットフォーム用にソフトウェアを記述するために必要なインターフェイスを提供します。該当する場合は、フレームワークのクラス、メソッド、関数、型、または定数で使われる主要なプレフィックスを掲載しました。独自のシンボル名には、ここに指定されているプレフィックスの使用は避ける必要があります。

デバイスのフレームワーク

表 B-1に、iOSベースのデバイスで利用可能なフレームワークを示します。これらのフレームワークは、
<Xcode>`/Platforms/iPhoneOS.platform/Developer/SDKs/`*<iOS_SDK>*`/System/Library/Frameworks`
ディレクトリにあります。*<Xcode>*は、Xcodeのインストールディレクトリ、*<iOS_SDK>*は、ターゲットになっている特定のSDKバージョンを表します。「最初のリリース」の欄は、このフレームワークが最初に導入されたiOSリリースです。

表 B-1 デバイスのフレームワーク

Name	最初のリリース	プレフィックス	説明
`Accelerate.framework`	4.0	`cblas`, `vDSP`	高速化された数学関数およびDSP関数が含まれています詳細については、『*Accelerate Framework Reference*』を参照してください。
`Accounts.framework`	5.0	`AC`	ユーザのシステムアカウントへのアクセスを管理するためのインターフェイスが含まれています詳細については、『*Accounts Framework Reference*』を参照してください。
`AddressBook.framework`	2.0	`AB`	ユーザの連絡先データベースに直接アクセスするための関数が含まれています詳細については、『*Address Book Framework Reference for iOS*』を参照してください。
`AddressBookUI.framework`	2.0	`AB`	システム定義のPeopleピッカーと編集インターフェイスを表示するためのクラスが含まれています詳細については、『*Address Book UI Framework Reference for iOS*』を参照してください。

Name	最初のリリース	プレフィックス	説明
AssetsLibrary.framework	4.0	AL	ユーザの写真やビデオにアクセスするためのクラスが含まれています詳細については、『*Assets Library Framework Reference*』を参照してください。
AudioToolbox.framework	2.0	AU, Audio	オーディオストリームデータを扱ったり、オーディオの再生と録音のためのインターフェイスが含まれています詳細については、『*Audio Toolbox Framework Reference*』を参照してください。
AudioUnit.framework	2.0	AU, Audio	オーディオユニットのロードと使用のためのインターフェイスが含まれています詳細については、『*Audio Unit Framework Reference*』を参照してください。
AVFoundation.framework	2.2	AV	オーディオとビデオの再生と録音のためのObjective-Cインターフェイスが含まれています詳細については、『*AV Foundation Framework Reference*』を参照してください。
CFNetwork.framework	2.0	CF	Wi-Fi無線と携帯電話無線によってネットワークにアクセスするためのインターフェイスが含まれています詳細については、『*CFNetwork Framework Reference*』を参照してください。
CoreAudio.framework	2.0	Audio	Core Audio全体で使われるオーディオデータタイプを提供します詳細については、『*Core Audio Framework Reference*』を参照してください。
CoreBluetooth.framework	5.0	CB	省電力Bluetoothハードウェアへのアクセスを提供します。
CoreData.framework	3.0	NS	アプリケーションのデータモデルを管理するためのインターフェイスが含まれています詳細については、『*Core Data Framework Reference*』を参照してください。
CoreFoundation.framework	2.0	CF	基本ソフトウェアサービスを提供します（一般的なデータ型の抽象化、文字列ユーティリティ、コレクションユーティリティ、リソース管理、環境設定など）詳細については、『*Core Foundation Framework Reference*』を参照してください。
CoreGraphics.framework	2.0	CG	Quartz 2D用のインターフェイスが含まれています詳細については、『*Core Graphics Framework Reference*』を参照してください。
CoreImage.framework	5.0	CI	ビデオ画像、静止画像を操作するためのインターフェイスが含まれています。詳細については、『*Core Image Reference Collection*』を参照してください。

Name	最初のリリース	プレフィックス	説明
CoreLocation.framework	2.0	CL	ユーザの位置を決定するためのインターフェイスが含まれています。詳細については、『*Core Location Framework Reference*』を参照してください。
CoreMedia.framework	4.0	CM	オーディオおよびビデオを操作する下位レベルのルーチンが含まれています。詳細については、『*Core Media Framework Reference*』を参照してください。
CoreMIDI.framework	4.2	MIDI	MIDIデータを処理する下位レベルのルーチンが含まれています詳細については、『*Core MIDI Framework Reference*』を参照してください。
CoreMotion.framework	4.0	CM	加速度計およびジャイロデータにアクセスするためのインターフェイスを含まれています詳細については、『*Core Motion Framework Reference*』を参照してください。
CoreTelephony.framework	4.0	CT	電話関連の情報にアクセスするためのルーチンが含まれています詳細については、『*Core Telephony Framework Reference*』を参照してください。
CoreText.framework	3.2	CT	テキストのレイアウトとレンダリングのエンジンが含まれています詳細については、『*Core Text Reference Collection*』を参照してください。
CoreVideo.framework	4.0	CV	オーディオおよびビデオを操作する下位レベルのルーチンが含まれています。このフレームワークを直接使用しないでください。
EventKit.framework	4.0	EK	ユーザのカレンダーイベントデータにアクセスするためのインターフェイスが含まれています詳細については、『*Event Kit Framework Reference*』を参照してください。
EventKitUI.framework	4.0	EK	標準のシステムカレンダーインターフェイスを表示するクラスが含まれています詳細については、『*Event Kit UI Framework Reference*』を参照してください。
External-Accessory.framework	3.0	EA	接続されているハードウェアアクセサリと通信するためのインターフェイスが含まれています詳細については、『*External Accessory Framework Reference*』を参照してください。
Foundation.framework	2.0	NS	文字列、コレクション、およびその他の下位レベルデータ型を管理するためのインターフェイスが含まれています詳細については、『*Foundation Framework Reference*』を参照してください。

Name	最初のリリース	プレフィックス	説明
GameKit.framework	3.0	GK	ピアツーピア接続を管理するためのインターフェイスが含まれています詳細については、『*Game Kit Framework Reference*』を参照してください。
GLKit.framework	5.0	GLK	複雑なOpenGL ESアプリケーションの構築に用いる、Objective-Cのユーティリティクラスが含まれています。詳細については、『*GLKit Framework Reference*』を参照してください。
GSS.framework	5.0	gss	セキュリティ関係の標準的なサービス群を提供します。
iAd.framework	4.0	AD	アプリケーション内に広告を表示するためのクラスが含まれています詳細については、『*iAd Framework Reference*』を参照してください。
ImageIO.framework	4.0	CG	画像データの読み取りおよび作成のためのクラスが含まれています詳細については、『*Image I/O Reference Collection*』を参照してください。
IOKit.framework	2.0	特になし	デバイスによって使用されるインターフェイスが含まれています。このフレームワークを直接インクルードしないでください。
MapKit.framework	3.0	MK	アプリケーションへのマップインターフェイスの埋め込みと、逆ジオコーダ座標のためのクラスが含まれています詳細については、『*Map Kit Framework Reference*』を参照してください。
MediaPlayer.framework	2.0	MP	フルスクリーンビデオを再生するためのインターフェイスが含まれています詳細については、『*Media Player Framework Reference*』を参照してください。
MessageUI.framework	3.0	MF	電子メールメッセージの作成とキューイングのためのインターフェイスが含まれています詳細については、『*Message UI Framework Reference*』を参照してください。
MobileCore-Services.framework	3.0	UT	システムによってサポートされるUTI (Uniform Type Identifier)を定義します
NewsstandKit.framework	5.0	NK	雑誌や新聞をバックグラウンドでダウンロードするためのインターフェイスを提供します。詳細については、『*Newsstand Kit Framework Reference*』を参照してください。

Name	最初のリリース	プレフィックス	説明
OpenAL.framework	2.0	AL	OpenAL（クロスプラットフォームの定位オーディオライブラリ）用のインターフェイスが含まれています。詳細については、http://www.openal.orgを参照してください。
OpenGLES.framework	2.0	EAGL, GL	OpenGL ES（クロスプラットフォームのOpenGL 2Dおよび3Dグラフィックスレンダリングライブラリの組み込み版）用のインターフェイスが含まれています詳細については、『*OpenGL ES Framework Reference*』を参照してください。
QuartzCore.framework	2.0	CA	Core Animationインターフェイスが含まれています詳細については、『*Quartz Core Framework Reference*』を参照してください。
QuickLook.framework	4.0	QL	ファイルをプレビューするためのインターフェイスが含まれています詳細については、『*Quick Look Framework Reference*』を参照してください。
Security.framework	2.0	CSSM, Sec	証明書、公開鍵と非公開鍵、および信用ポリシーを管理するインターフェイスが含まれています詳細については、『*Security Framework Reference*』を参照してください。
StoreKit.framework	3.0	SK	In-App Purchaseに関連付けられた会計処理を扱うためのインターフェイスが含まれています詳細については、『*Store Kit Framework Reference*』を参照してください。
System-Configuration.framework	2.0	SC	デバイスのネットワーク設定を判別するインターフェイスが含まれています詳細については、『*System Configuration Framework Reference*』を参照してください。
Twitter.framework	5.0	TW	Twitterサービスを介してツイートを送るためのインターフェイスが含まれています。詳細については、『*Twitter Framework Reference*』を参照してください。
UIKit.framework	2.0	UI	iOSアプリケーションユーザインターフェイスレイヤ用のクラスとメソッドが含まれています詳細については、『*UIKit Framework Reference*』を参照してください。

シミュレータのフレームワーク

コードを記述しているときは、常にデバイスのフレームワークをターゲットにする必要がありますが、テスト中は特別にシミュレータ用にコードをコンパイルする必要があります。デバイスとシミュレータで利用可能なフレームワークはほとんど同じですが、少しだけ違いがあります。たとえば、シミュレータは、自身の実装の一部にMac OS Xのフレームワークをいくつか使用しています。さらに、デバイスのフレームワークとシミュレータのフレームワークで利用可能な同じインターフェイスでも、システムの制約によって多少異なる場合があります。フレームワークのリストやデバイスのフレームワークとシミュレータのフレームワークの特定の違いの詳細については、『*iOS App Development Workflow Guide*』を参照してください。

システムライブラリ

Core OSおよびCore Servicesレベルの特殊なライブラリの中には、フレームワークとしてパッケージ化されていないものもあります。その代わりに、iOSではシステムの`/usr/lib`ディレクトリにたくさんのダイナミックライブラリが含まれています。共有ダイナミックライブラリは、`.dylib`拡張子で識別されます。これらのライブラリのヘッダファイルは、`/usr/include`ディレクトリにあります。

iOS SDKの各バージョンには、この共有ダイナミックライブラリのローカルコピーが含まれており、これらはシステムと一緒にインストールされます。これらのコピーは、Xcodeプロジェクトからリンクできるように開発用のシステムにインストールされます。特定のバージョンのiOS用のライブラリのリストを調べるには、`<`*Xcode*`>/Platforms/iPhoneOS.platform/Developer/SDKs/<`*iOS_SDK*`>/usr/lib`を参照してください。*<Xcode>*は、Xcodeのインストールディレクトリ、*<iOS_SDK>*は、ターゲットになっている特定のSDKバージョンを表します。たとえば、iOS 4.2 SDK用の共有ライブラリは、`/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS4.2.sdk/usr/lib`ディレクトリにあります。それに対応するヘッダは、`/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS4.2.sdk/usr/include`にあります。

iOSでは、シンボリックリンクを使用してほとんどのライブラリの最新バージョンを参照できます。共有ダイナミックライブラリにリンクする場合は、特定のバージョンのライブラリへのリンクではなくこのシンボリックリンクを使用します。ライブラリのバージョンは、iOSの今後のバージョンで変更される可能性があります。したがって、ソフトウェアを特定のバージョンにリンクした場合、そのバージョンがユーザのシステム上で常に利用できるとは限りません。

書類の改訂履歴

この表は「*iOSテクノロジーの概要*」の改訂履歴です。

日付	メモ
2011-10-12	iOS 5で導入されたテクノロジーを追加しました。
2010-11-15	iOS 4.1およびiOS 4.2の新しい機能を反映するように文書全体を更新しました。
2010-07-08	『iPhone OSテクノロジーの概要』から文書名を変更しました。
2010-06-04	iOS 4.0の機能を反映するように更新しました。
2009-10-19	付録に参考文書へのリンクを追加しました。
2009-05-27	iOS 3.0用に更新しました。
2008-10-15	iOSとそのテクノロジーを紹介する新規文書。

書類の改訂履歴