iPhone開発ガイド

2010-05-28

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

はじめに

iPhoneアプリケーションを開発するには、Appleの優れた統合開発環境(IDE)であるXcodeを使用します。Xcodeは、アプリケーションのユーザインターフェイスの設計と、それを動作させるためのコードの記述に必要なすべてのツールを備えています。アプリケーションを開発しながら、そのアプリケーションをお使いのコンピュータ、iPhone、またはiPod touch上で実行します。

この文書では、iPhoneアプリケーションの開発プロセスについて説明します。また、アプリケーションをテストの目的でデバイス上で実行する際に必要となる、iPhoneデベロッパプログラムのメンバ登録についての情報も提供します。

iPhoneアプリケーションの開発が終了したら、それをApp Storeに投稿します。App Storeは、iPhone OSユーザがアプリケーションを入手するための安全なマーケットプレースサイトです。ただし、アプリケーションを公開する前に、さまざまな使用形態に対応し、自作製品についてのフィードバックを得るために、少人数のユーザ集団でアプリケーションをテストする必要があります。この文書では、アプリケーションのテスターグループを作成する方法と、アプリケーションをテスターグループに配布する方法も説明します。

この文書を利用するには、『*iOS Application Programming Guide*』に解説されている、iPhoneアプリケーションアーキテクチャに精通している必要があります。また、プログラミングの基本概念にも精通している必要があります。

この文書を読むと、iPhoneアプリケーションの開発プロセスの基礎を理解できます。さらに知識を深めるには、この章の最後に列挙されている文書をお読みください。

ソフトウェア要件： この文書は、iPhone SDK 3.2（Xcode 3.2.2を含む）配布版とMac OS X v10.6以降に適用されます。

iPhone Webアプリケーションを開発する場合は、http://developer.apple.com/safari/libraryを参照してください。

この書類の構成

この文書は、次の章で構成されています。

- 「iPhone開発クイックスタート」（13 ページ）では、Xcodeを使用してアプリケーションを設計、ビルド、および実行するために行う主な開発作業の概要を説明します。
- 「アプリケーションの設定」（27 ページ）では、アプリケーションのプロパティとエンタイトルメント(Entitlements)を設定する方法と、iPhoneシミュレータおよびデバイス環境で適切にビルドされるようにアプリケーションを調整する方法について説明します。
- 「アプリケーションの実行」（33 ページ）では、iPhoneアプリケーションの実行やデバッグに必要な手順をそれぞれ説明します。

- 「iPhoneシミュレータの使用」（47ページ）では、コンピュータの入力デバイスを使用して、iPhoneのユーザとデバイス間のやり取りをシミュレートする方法を説明します。
- 「デバイスとデジタルIDの管理」（51ページ）では、開発のためにコンピュータとデバイスを設定する方法を示します。また、Xcodeの「オーガナイザ(Organizer)」を使用して、コンソールログやクラッシュ情報を表示する方法や、デバイス上で実行中のアプリケーションのスクリーンショットを撮る方法、さらに、開発中のアプリケーションをデバイスにインストールするために必要なデジタルIDを保護する方法も示します。
- 「アプリケーションのデバッグ」（59ページ）では、Xcodeのデバッグ機能について説明します。
- 「アプリケーションの単体テスト」（65ページ）では、単体テストとプロジェクトで単体テストを利用する方法について紹介します。
- 「アプリケーションのチューニング」（79ページ）では、アプリケーションのパフォーマンスを測定したり調整したりするために使用するツールである、InstrumentsとSharkについて説明します。
- 「アプリケーションの配布」（83ページ）では、Distributionプロビジョニングプロファイルを使ったアプリケーションのアーカイブの作成方法と、それをアプリケーションのテスターに送信またはiTunes Connectへ投稿する方法について説明します。アプリケーションのテスター向けの説明書についてもここで説明します。
- 「iPhone開発に関するFAQ」（93ページ）では、iPhoneシミュレータについてデベロッパから寄せられるよくある質問を列挙します。
- 「Hello, World!ソースコード」（95ページ）では、「チュートリアル：Hello, World!」（20ページ）で説明したHello, World!アプリケーションのソースコードを示します。
- 「単体テスト結果用マクロのリファレンス」（97ページ）では、単体テストメソッドで使用できるテスト結果マクロについて説明します。

iPhone SDKのインストール

iPhoneアプリケーションの開発に必要なツール（Xcode、iPhoneシミュレータなど）をインストールするには、http://developer.apple.com/iphoneにアクセスしてください。

注： iPhone SDKをインストールするにはIntelベースのMacが必要です。

関連項目

以下の文書は、iPhoneアプリケーションの開発を理解するために不可欠な概念について説明しています。

- 『*iOS Technology Overview*』は、iPhone OSとそのテクノロジーについて説明しています。
- 『*iOS Application Programming Guide*』では、iPhoneアプリケーションのアーキテクチャについて説明し、UIKitなどの重要なシステムフレームワークの主要なカスタマイズポイントを示します。
- 『*Cocoa Fundamentals Guide*』は、Cocoaフレームワークの基本概念、用語、アーキテクチャ、およびデザインパターンと、開発環境について説明しています。

- 『*The Objective-C Programming Language*』は、オブジェクト指向プログラミングについて紹介し、iPhone開発で使われる主要なプログラミング言語について説明しています。
- 『*Dashcode User Guide*』は、iPhone版Safari用に最適化されたWebページの作成方法について説明します。これらのWebアプリケーションはHTML、CSS、JavaScriptなどのWebテクノロジーを使用します。

iPhone開発クイックスタート

iPhoneアプリケーションの開発は、楽しくてやりがいのある仕事です。自分のアイディアを製品にするには、iPhoneアプリケーションの開発に使われる統合開発環境(IDE)、Xcodeを使用します。Xcodeを使用して、ソースファイルの管理および編集、ドキュメントの表示、アプリケーションのビルド、コードのデバッグ、およびアプリケーションパフォーマンスの最適化を行います。

注： iPhoneアプリケーションを開発するには、登録アップルデベロッパになる必要があります。デバイスでアプリケーションを実行するには、iPhoneデベロッパプログラムにメンバ登録する必要があります。詳細については、「デバイスとデジタルIDの管理」（51 ページ）を参照してください。

この章では、Xcodeを使用してアプリケーションを設計、ビルド、および実行するために行う主な開発作業の概要を説明します。また、iPhone OS向け"Hello World!"アプリケーションの開発方法を説明するクイックチュートリアルも含まれています。

必須の開発作業

iPhoneアプリケーションの開発プロセスは、以下の主要なステップに分けられます。

1. プロジェクトを作成する。

   Xcodeには、すぐにプロジェクトに着手できるように、プロジェクトテンプレートがいくつか用意されています。これから開発しようとするアプリケーションタイプを実装したテンプレートを選びます。詳細については、「iPhoneプロジェクトの作成」（14 ページ）を参照してください。

2. ユーザインターフェイスを設計する。

   Interface Builderアプリケーションを利用すると、アプリケーションのユーザインターフェイスをグラフィカルに設計して、リソースファイルとして保存できます。リソースファイルは、実行時にアプリケーションにロードされます。Interface Builderを使用したくない場合は、ユーザインターフェイスをプログラムでレイアウトすることもできます。詳細については、『*iOS Application Programming Guide*』の「User Interface Design Considerations」を参照してください。

3. コードを記述する。

   Xcodeには、クラスおよびデータのモデリング、コード補完、ドキュメントへの直接アクセス、リファクタリングなど、すばやくコードを記述できるように支援する機能があります。詳細については、「コードの編集」（16 ページ）を参照してください。

4. アプリケーションをビルドして実行する。

   コンピュータ上でアプリケーションをビルドして、iPhoneシミュレータアプリケーションまたはデバイス上でそれを実行します。詳細については、「アプリケーションのビルドと実行」（19 ページ）を参照してください。

5. アプリケーションのパフォーマンスを測定して調整する。

   実行するアプリケーションを用意したあと、そのアプリケーションがデバイスのリソースをできる限り効率的に使用すること、およびユーザのジェスチャに対して適切に応答することを保証するために、パフォーマンスを測定する必要があります。詳細については、「アプリケーションパフォーマンスの測定」（20 ページ）を参照してください。

これ以降は、これらのステップについて詳しく説明します。

iPhoneプロジェクトの作成

iPhone SDKには、アプリケーションの開発を始める際に利用できるプロジェクトのテンプレートがいくつか用意されています。次の種類のアプリケーションから選ぶことができます。

- **Navigation-Based Application。**複数の画面を使用して、階層的にデータを表示するアプリケーション。「連絡先(Contacts)」アプリケーションは、ナビゲーションベースアプリケーションの一例です。
- **OpenGL ES Application。**画像やアニメーションを表示するために、OpenGL ESベースのビューを使用するアプリケーション。
- **Tab Bar Application。**ユーザが複数の画面から選択できるラジオインターフェイスを表示するアプリケーション。「時計(Clock)」アプリケーションは、タブバーアプリケーションの一例です。
- **Utility Application。**メインビューを1つ実装しており、ユーザは裏面のビューにアクセスして簡単なカスタマイズができるアプリケーション。「株価(Stocks)」アプリケーションは、ユーティリティアプリケーションの一例です。
- **View-Based Application。**1つのビューを使用してユーザインターフェイスを実装しているアプリケーション。
- **Window-Based Application。**このテンプレートは、すべてのアプリケーションのスタート地点として役割を果たし、1つのアプリケーションデリゲート（delegate、委任）と1つのウインドウを含んでいます。独自の階層ビューを実装する場合に、このテンプレートを使用します。

iPhoneアプリケーションで使用するためにスタティックライブラリを開発する必要がある場合は、スタティックライブラリターゲットをプロジェクトに追加できます。それには、「プロジェクト(Project)」＞「新規ターゲット(New Target)」を選び、「iPhone OS/Cocoa Touch」リストから「Static Library」ターゲットテンプレートを選択します。

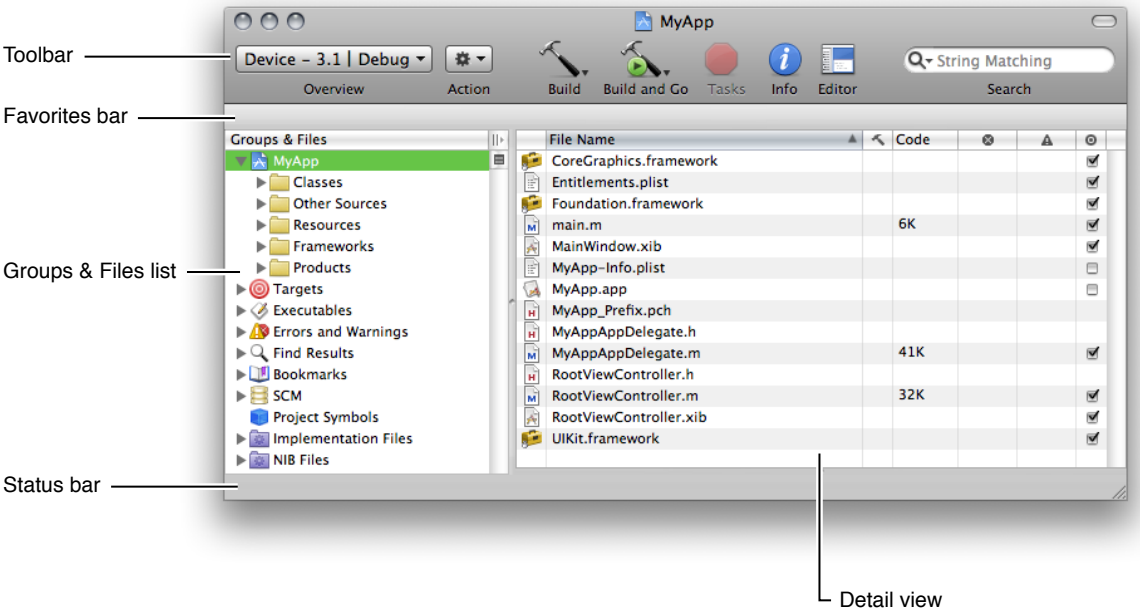
iPhoneアプリケーションで使用するスタティックライブラリは、コード署名する必要がありません。このため、作成したスタティックライブラリターゲットから、「コード署名ID (Code Signing Identity)」ビルド設定を削除する必要があります。それには、次の手順を実行します。

1. スタティックライブラリターゲットの「情報(Info)」ウインドウを開き、「ビルド(Build)」ペインを表示します。
2. 「Code Signing」グループから、「コード署名ID (Code Signing Identity)」ビルド設定用の「Any iPhone OS Device」条件定義を選択します。
3. この条件定義の値を「iPhone Developer」から「Don't Code Sign」に変更します。

iPhoneアプリケーションのアーキテクチャの詳細については、『*iOS Application Programming Guide*』を参照してください。

iPhoneアプリケーションを開発するには、Xcodeプロジェクトを対象に作業を行います。プロジェクトに関する作業のほとんどは**プロジェクトウインドウ**から行います。このウインドウで、アプリケーションのビルドに必要なソースファイルやその他のリソースファイルを表示したり管理したりします。また、プロジェクトのすべての構成要素にアクセスしたり編集したりすることができます。図 1-1は、プロジェクトウインドウを示しています。

図 1-1 プロジェクトウインドウ



プロジェクトウインドウには、プロジェクト内を移動するために以下の主要な領域が含まれています。

- **「グループとファイル(Groups & Files)」リスト。**プロジェクトの内容のアウトラインビューを提供します。このリスト内で、ファイルやフォルダを移動したり、プロジェクト内容を整理したりできます。「グループとファイル(Groups & Files)」リストで現在選択されている要素によって、詳細ビューに表示される内容が決まります。
- **詳細ビュー(Detail view)。**「グループとファイル(Groups & Files)」リストで選択されている項目（複数可）が表示されます。詳細ビューでは、プロジェクトの内容を参照したり、「検索(Search)」フィールドを使用して検索したり、カラム単位で並べ替えたりできます。詳細ビューは、プロジェクトの内容をすばやく検索し、アクセスするのに役立ちます。
- **ツールバー(Toolbar)。**最もよく使うXcodeコマンドにすばやくアクセスできます。
- **よく使う項目バー(Favorites bar)。**プロジェクト内のよくアクセスする場所を保存し、すばやくその場所に戻ることができます。よく使う項目バーは、デフォルトでは表示されません。よく使う項目バーを表示するには、「表示(View)」＞「レイアウト(Layout)」＞「よく使う項目バーを表示(Show Favorites Bar)」メニューを選びます。
- **ステータスバー(Status bar)。**プロジェクトに関するステータスメッセージが表示されます。ビルドやインデックス作成などの処理中に、ステータスバーには、現在のタスクの進捗状況を表す進捗インジケータが表示されます。

プロジェクトの作成の詳細については、『*Xcode Project Management Guide*』の「Creating Projects」を参照してください。

コードの編集

コードを記述するために使用するメインツールは、Xcodeのテキストエディタです。この高度なテキストエディタには、次のような便利な機能があります。

- **ヘッダファイル検索。**Commandキーを押しながらシンボルをダブルクリックすると、そのシンボルが宣言されているヘッダファイルを表示できます。
- **APIリファレンス検索。**Optionキーを押しながらシンボルをダブルクリックすると、そのシンボルの使用法に関する情報を提供するAPIリファレンスに直接アクセスできます。
- **コード補完。**コードの入力に伴って、現在入力しようとしているシンボルの名前をXcodeが判断して、それを完成させるテキストをエディタが自動的に挿入してくれます。Xcodeのこの機能は、ユーザの邪魔にならないように、また、オーバーライド可能な形で動作します。
- **コードの折りたたみ。**コードの折りたたみ機能を利用することで、作業対象になっていないコードを折りたたんで、注目したいコードのみを表示できます。

これらの機能と、テキストエディタのその他の機能の詳細については、『*Xcode Workspace Guide*』の「The Text Editor」を参照してください。

コード補完の使用

このテキストエディタは、コード補完によって、コードのすばやい入力を支援します。コード補完がアクティブになっている場合、Xcodeは、ユーザが入力したテキストと、それが入力された文脈の両方に基づいて、ユーザが入力しようとしているトークンを完成させるための提案を行います。コード補完は、デフォルトでは有効になっていません。

コード補完は、次の手順でアクティブにします。

1. Xcodeの「環境設定(Preferences)」ウインドウを開きます。

   「Xcode」＞「環境設定(Preferences)」を選びます。

2. 「コード入力補助(Code Sense)」ペインの「コードの補完(Code Completion)」セクションの下で、「自動的に候補を表示：すぐに(Immediate from the Automatically Suggest)」ポップアップメニューを選びます。

3. 「OK」をクリックします。

シンボルの名前の入力に応じて、Xcodeがそのシンボルを認識し、図1-2に示すような提案を行います。提案を受理するには、TabまたはReturnを押します。Escapeを押して、補完リストを表示することもできます。

図 1-2 コード補完の使用

コード補完の詳細については、『*Xcode Workspace Guide*』の「Completing Code」を参照してください。

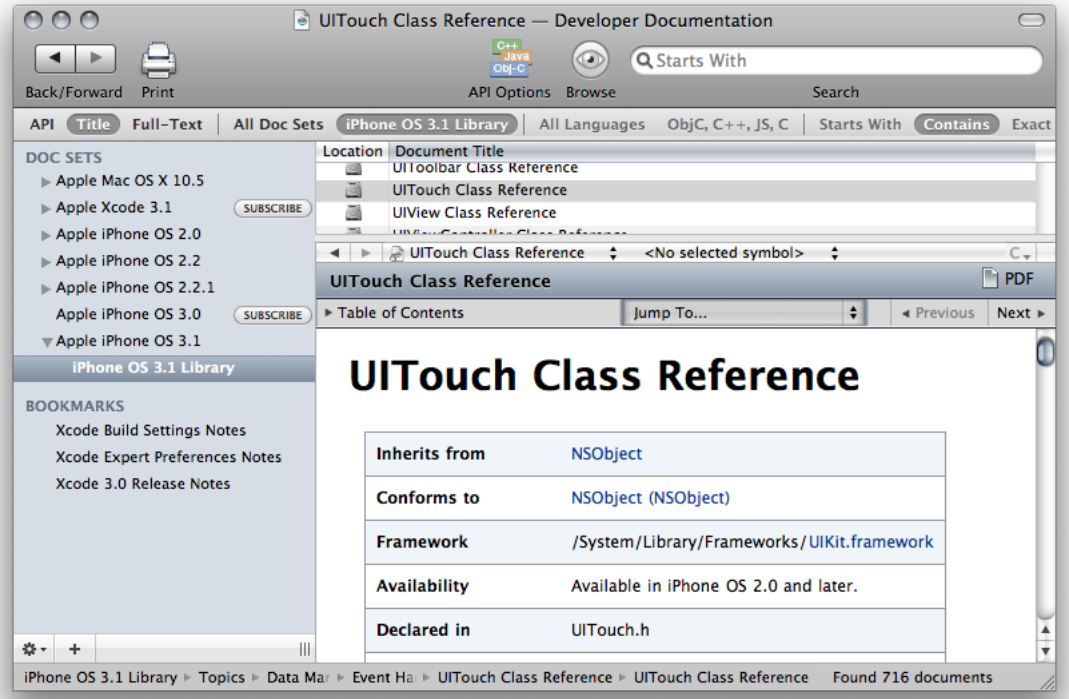
ドキュメントへのアクセス

開発の作業中に、特定のシンボルのリファレンスや、APIの使用法またはiPhone OSテクノロジーについての高度なドキュメントにすばやくアクセスしたい場合があります。Xcodeでは、「リサーチアシスタント(Research Assistant)」ウインドウと「ドキュメント(Documentation)」ウインドウを使用して、そうしたリソースに簡単にアクセスできます。

リサーチアシスタント(Research Assistant)は、図 1-4 （19 ページ）に示すような軽量なウインドウです。ここには、選択されている項目に関するAPIリファレンスが簡潔に表示され、対象項目が表示されているエディタからフォーカスは移りません。このウインドウによって、編集作業の邪魔にならないようにAPIリファレンスを参照できます。ただし、リファレンスを詳しく調べる必要がある場合は、クリック1つで「ドキュメント(Documentation)」ウインドウを開くことができます。

ドキュメント(Documentation)ウインドウ（図 1-3）では、コンピュータにインストールされているデベロッパ向けドキュメント（APIリファレンス、ガイド、特定のツールやテクノロジーに関する一連の記事を含む）を参照したり検索したりできます。詳しいヘルプを参照したい場合には、このウインドウによって、「リサーチアシスタント(Research Assistant)」よりも広範囲で詳しいドキュメント表示が得られます。

図 1-3 「ドキュメント(Documentation)」ウインドウでのAPIリファレンスの参照



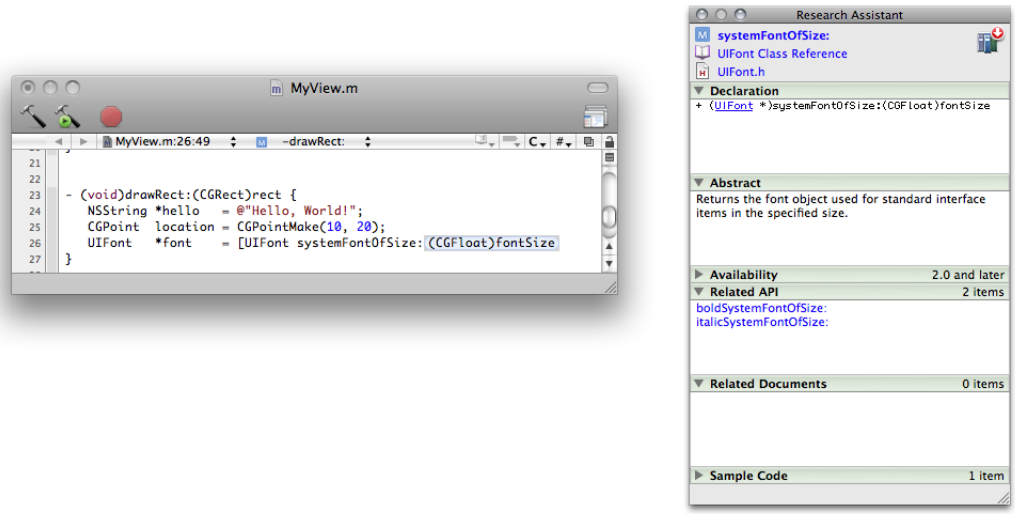
ソースファイル内のシンボルのAPIリファレンスを表示するには、テキストエディタでそのシンボルを選択し、「ヘルプ(Help)」＞「APIリファレンス内で選択されたテキストを検索(Find Selected Text in API Reference)」を選びます（または、Optionを押しながら、シンボル名をダブルクリックします）。このコマンドによって、選択されたシンボルがプロジェクトのSDK用のAPIリファレンスから検索され、「ドキュメント(Documentation)」ウインドウに表示されます。たとえば、ソースファイル内で`UIFont`というクラス名を選択して、「APIリファレンス内で選択されたテキストを検索(Find Selected Text in API Reference)」コマンドを実行すると、Xcodeは「ドキュメント(Documentation)」ウインドウを開き`UIFont`クラスのAPIリファレンスを表示します。

「ドキュメント(Documentation)」ウインドウは、iPhoneドキュメントライブラリを参照するための優れたツールですが、コードを記述している最中は、テキストエディタからフォーカスを移さずにそのシンボルの基本情報を簡潔に知りたい場合もあります。「リサーチアシスタント(Research Assistant)」は、このような情報を小さくて邪魔にならないウインドウに表示します。

「リサーチアシスタント(Research Assistant)」は、ソースコード内のカーソルの移動に応じて動的に動作します。APIリファレンスが見つかったシンボルを認識すると、「リサーチアシスタント(Research Assistant)」はそのリファレンスを表示します。図 1-4は、その様子を示しています。「リサーチアシスタント(Research Assistant)」を一目見れば、そのシンボルの基本情報がわかります。

「リサーチアシスタント(Research Assistant)」を表示するには、「ヘルプ(Help)」＞「リサーチアシスタントを表示(Show Research Assistant)」を選びます。

図 1-4 「リサーチアシスタント(Research Assistant)」でのAPIリファレンスの参照



「リサーチアシスタント(Research Assistant)」から、そのシンボルのより広範囲なリファレンスにすばやく移動できます。また、そのシンボルを宣言しているヘッダを参照することもできます。

Xcodeでのドキュメントアクセスの詳細については、『*Xcode Workspace Guide*』の「Documentation Access」を参照してください。

アプリケーションのビルドと実行

iPhoneシミュレータは、iPhone OS APIを実装しており、デバイス環境とよく似た環境を提供します。これを利用すると、Mac OS Xでアプリケーションを実行できるので、利用可能なデバイスがなくても、アプリケーションの機能をすばやくテストできます。ただし、iPhoneシミュレータでアプリケーションを実行することは、実際のデバイスで実行することと同じではありません。iPhoneシミュレータは、デバイスのパフォーマンスまではエミュレートしません。つまり、iPhoneシミュレータでは、実際のデバイスのメモリ制約やプロセッサのパフォーマンスを組み入れることはできません。第1に、iPhoneシミュレータは、デバイス上で実行されるバージョンではなく、Mac OS Xバージョンの下位レベルシステムフレームワークを使用します。第2に、iPhoneシミュレータ上では利用できないハードウェアベースの機能があります。それでも一般に、iPhoneシミュレータはアプリケーションの初期テストを実行できる優れたツールです。

ユーザのデバイス上でアプリケーションがどのように実行されるかを正確に把握するには、デバイス上でアプリケーションを実行し、Instrumentsやその他のパフォーマンス測定ツールを使用して、パフォーマンスデータを収集する必要があります。

Xcodeでは、コードのコンパイルとデバッグを行うために、GCC、GDBなどのオープンソースのツールを利用しています。また、Subversion、CVS、Perforceなどのソース管理システムを利用したチームベースの開発もサポートしています。

アプリケーションのビルドには、以下の手順が含まれます。

- ソースファイルをコンパイルして、アプリケーションバイナリを生成する。
- そのバイナリをiPhoneシミュレータまたはデバイス上に配置する。

Xcodeでは、「ビルド(Build)」コマンドを実行するとこれらのタスクが実行されます。詳細については、「アプリケーションの実行」（33 ページ）を参照してください。

アプリケーションパフォーマンスの測定

アプリケーションの機能をテストしたら、アプリケーションがデバイス上でうまく動作することを保証する必要があります。これは、アプリケーションがデバイスのリソースをできる限り効率的に使用することを意味します。たとえばメモリは貴重なリソースです。したがって、アプリケーションは、iPhone OSのパフォーマンスを損なわないようにメモリ占有量を少なく維持する必要があります。また、アプリケーションは効率良いアルゴリズムを使用して消費電力をできる限り少なくし、バッテリー持続時間を長くしなければなりません。Xcodeは、アプリケーションパフォーマンスの測定とチューニングを行うために、InstrumentsとSharkという2つの主要なツールを提供しています。

Instrumentsアプリケーションは、動的なパフォーマンス分析ツールです。これを利用すると、コードの実行中にコードを調査して、実行内容についての重要な測定データを収集できます。Instrumentsが収集したデータをリアルタイムに参照または分析したり、そのデータを保存しておいて後で分析することもできます。アプリケーションに関して、CPU、メモリ、ファイルシステム、ネットワーク、その他のリソースの使用状況についてのデータを収集できます。

Sharkアプリケーションは、コード内のパフォーマンスのボトルネックを発見するのに役立つもう1つのツールです。このツールは、ハードウェアおよびソフトウェアのパフォーマンスイベントのプロファイルを作成し、コードの全体的な動作や、iPhone OSとのやり取りの様子を示します。

詳細については、「アプリケーションのチューニング」（79 ページ）を参照してください。

さらに学びたい方へ

Xcodeの開発プロセスの詳細については、『*A Tour of Xcode*』を参照してください。

チュートリアル：Hello, World!

このチュートリアルでは、iPhoneの画面上にテキストを出力する簡単なプロジェクトの作成手順を説明します。

プロジェクトの作成

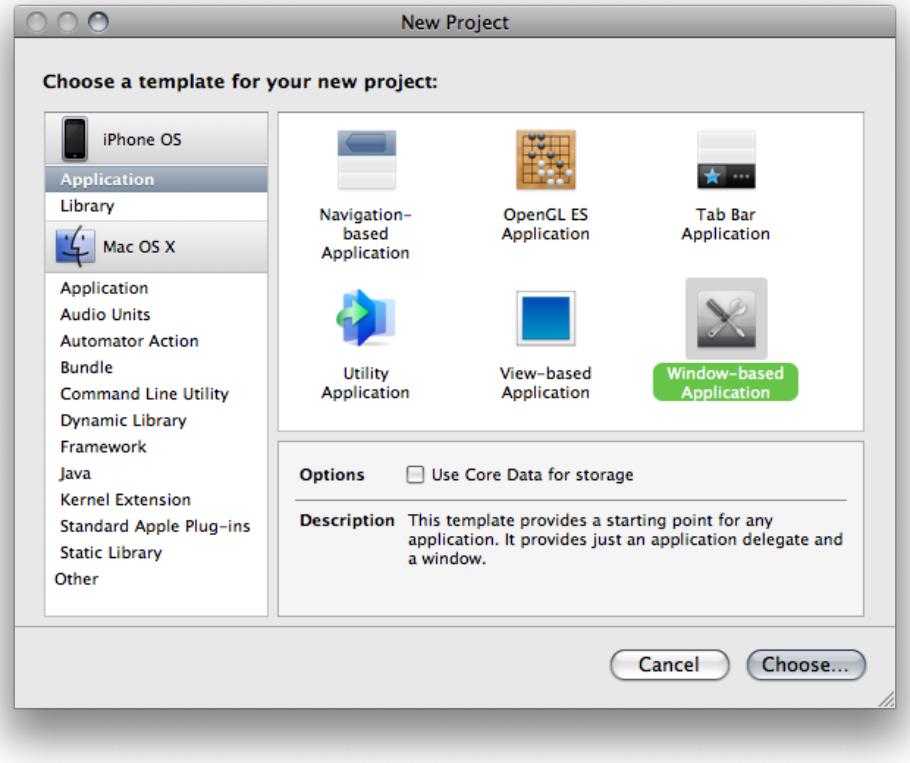
Hello Worldプロジェクトを作成するには、次の手順を実行します。

1. `<Xcode>/Applications`にあるXcodeを起動します。

   `<Xcode>`は、Xcodeツールセットをインストールしたディレクトリを表します。詳細については、『Xcode Installation Guide』の「Xcode Installation Details」を参照してください。

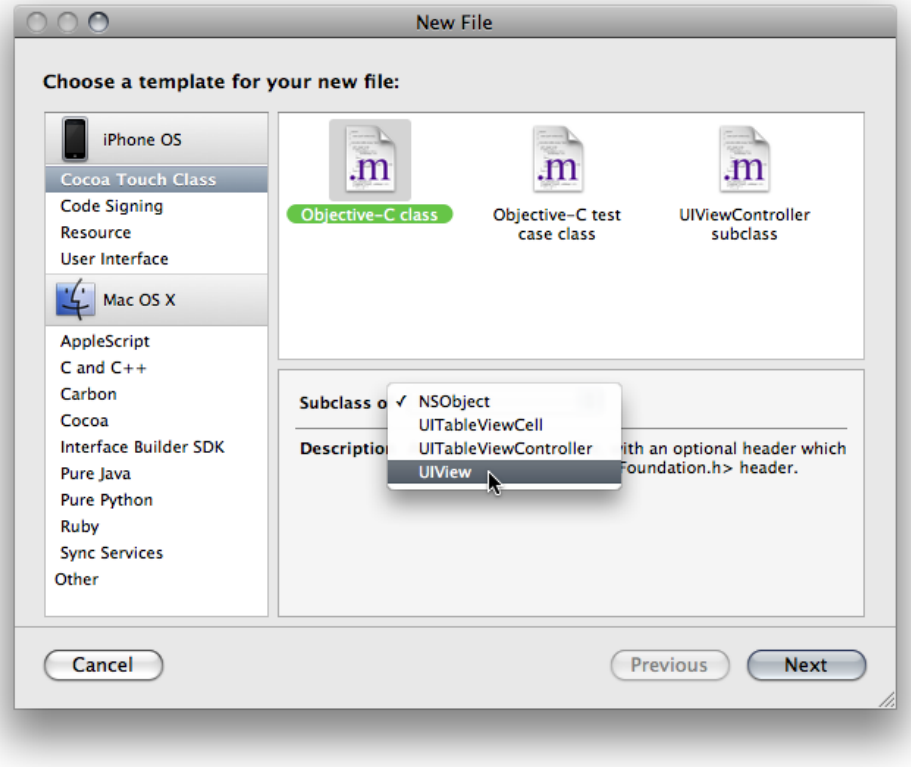
2. 「ファイル(File)」＞「新規プロジェクト(New Project)」を選びます。

3. Window-Based Applicationのテンプレートを選択し、「選択(Choose)」をクリックします。



4. プロジェクトを`HelloWorld`と名付け、ファイルシステム上での保存場所を選びます。
5. プロジェクトに`MyView`クラスを追加します。
    a. 「ファイル(File)」＞「新規ファイル(New File)」を選びます。

b. Cocoa TouchのUIView subclassテンプレートを選択し、「次へ(Next)」をクリックします。



c. 「ファイル名(File Name)」テキストフィールドに、`MyView.m`と入力します。

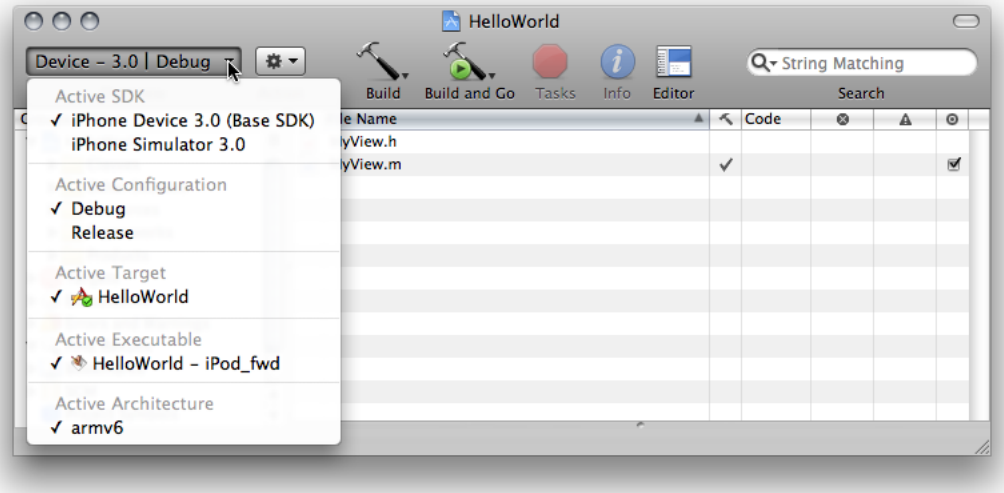
d. 「同時にMyView.hも作成(Also create "MyView.h")」オプションを選択し、「完了(Finish)」をクリックします。

6. プロジェクトにアクティブSDKを選びます。

プロジェクトの作成時に開発デバイスを接続している場合、Xcodeは、アクティブSDK (Active SDK)をデバイス用にビルドするように設定します。それ以外の場合は、iPhoneシミュレータ用にビルドするように設定します。

注： iPhoneデバイスSDKを使用するにはiPhoneデベロッパプログラムのメンバになる必要があります。詳細については、「iPhoneデベロッパプログラムのメンバになる」（51 ページ）を参照してください。

アクティブSDKを設定するには、「プロジェクト(Project)」＞「アクティブSDKを設定(Set Active SDK)」サブメニューまたはプロジェクトウインドウの「概要(Overview)」ツールバーから項目を選びます。



コードの作成

Xcodeでは、テキストエディタでの作業に最も多くの時間を費やします。コードの記述、アプリケーションのビルド、コードのデバッグを行うことができます。コードを記述しながら、Xcodeの支援機能を見ていきましょう。

Xcodeのソースコード編集機能を体験するには、次の手順に従ってアプリケーションのソースコードを入力する必要があります。参考のために、「Hello, World!ソースコード」（95 ページ）に最終的なソースコードがあります。

まず、`MyView`クラスを使用するように`HelloWorldAppDelegate`クラスを修正します。

1. 「グループとファイル(Groups & Files)」リストで、HelloWorldグループを選択します。
2. 詳細ビューで`HelloWorldAppDelegate.m`をダブルクリックします。
3. `HelloWorldAppDelegate`エディタウインドウで、
   a. 既存の`#import`行の下に、次のコード行を追加します。

      ```
      #import "MyView.h"
      ```

   b. `applicationDidFinishLaunching:`メソッドのオーバーライドポイントのコメントの下に、次のコード行を追加します。

```
        MyView *view = [[MyView alloc] initWithFrame:[window frame]];
        [window addSubview:view];
        [view release];
```

これらの変更を加えると、HelloWorldAppDelegate.mファイルのコードは以下のようになっているはずです。

```
#import "HelloWorldAppDelegate.h"
#import "MyView.h"

@implementation HelloWorldAppDelegate

@synthesize window;

- (void)applicationDidFinishLaunching:(UIApplication *)application {

    // アプリケーション起動後のカスタマイズ用、オーバーライドポイント
    MyView *view = [[MyView alloc] initWithFrame:[window frame]];
    [window addSubview:view];
    [view release];

    [window makeKeyAndVisible];
}

- (void)dealloc {
    [window release];
    [super dealloc];
}

@end
```

リスト 1-1に、ウインドウに“Hello World!”と描画するコードを示します。MyView.mのdrawRect:メソッドに、ハイライトされているコード行を追加します。

リスト 1-1　ビューに“Hello, World!”と描画するメソッド

```
- (void) drawRect:(CGRect) rect {
    NSString *hello   = @"Hello, World!";
    CGPoint  location = CGPointMake(10, 20);
    UIFont   *font    = [UIFont systemFontOfSize:24.0];
    [[UIColor whiteColor] set];
    [hello drawAtPoint:location withFont:font];
}
```

コード補完を有効にしている場合（「コード補完の使用」（16 ページ）で説明しています）、テキストエディタはシンボル名が入力されるたびに認識したシンボル名に一致するよう候補を提示します。たとえば、CGPointMと入力すると、テキストエディタは、図 1-2（17 ページ）に示すような候補を提案します。この補完機能は、ここで候補の提案を受け入れ、パラメータのプレースホルダにジャンプして利用できます。

1. 「編集(Edit)」＞「次のプレースホルダを選択(Select Next Placeholder)」を選んで最初のパラメータにジャンプし、次に10と入力します。

「次のプレースホルダを選択(Select Next Placeholder)」コマンドを使用して、入力中のテキストを補完するためにテキストエディタが提案している関数呼び出しやメソッド呼び出し内の引数間を移動できます。

2. 2番目のパラメータにジャンプし、20と入力します。

3. 行の末尾にセミコロン（;）を入力して、Returnを押します。

アプリケーションの実行

Hello Worldアプリケーションをビルドして実行するには、「ビルド(Build)」＞「ビルドと実行(Build and Run)」を選びます（または、プロジェクトウインドウで「ビルドと実行(Build and Run)」ツールバー項目をクリックします）。ビルドエラーがなければ、XcodeはiPhoneシミュレータまたはデバイス（アクティブSDKの設定によります）にアプリケーションをインストールします。

トラブルシューティング：Hello, World!ビルドエラー

このセクションでは、Hello, World!プロジェクトの起こりうるビルドエラーとその原因について説明します。

```
Building ... — 2 errors
   Compiling <project_directory>/Classes/HelloWorldAppDelegate.m (2 errors)
      error: 'MyView' undeclared (first use in this function)
      error: 'view' undeclared (first use in this function)
```

このビルドエラーを修正するには、

```
#import "MyView.h"
```

を`HelloWorldAppDelegate.m`ファイルに追加します。

その他の起こりうるビルドエラーについては、「ビルドエラーの解決」（41 ページ）を参照してください。

さらに学びたい方へ

これでiPhone OS向けの標準の“Hello World!”アプリケーションの作成方法がわかったので、よく目にするこのアプリケーションのCocoa Touchバージョン、*HelloWorld*を試すことができます。

より複雑なアプリケーション開発の手順ごとのチュートリアルについては、『*Your First iOS Application*』を参照してください。

Objective-Cの詳細については、『*Learning Objective-C: A Primer*』を参照してください。

iPhoneアプリケーションの開発の詳細については、『*iOS Application Programming Guide*』を参照してください。

アプリケーションの設定

この章では、アプリケーションのプロパティを設定してアプリケーションの実行環境をカスタマイズする方法、エンタイトルメントを設定してiPhone OSのセキュリティ機能を利用する方法、および複数の異なるSDKやアーキテクチャ用にビルドプロセスを調整する方法について説明します。

この章ではまた、iPhoneアプリケーションをビルドするターゲットをアップデートして、iPhoneとiPad用に最適化されたアプリケーションをビルドする1つのターゲットにする方法と、iPhoneアプリケーションをビルドするターゲットとiPadアプリケーションをビルドするターゲットの2つにする方法も説明します。

開発用にデバイスを設定する方法については、「デバイスとデジタルIDの管理」（51 ページ）を参照してください。

プロパティリストファイルの編集

プロパティリストファイルは、単純なデータ型を使って名前付きの複数の値と値のリストに整理したXMLファイルです。これらのデータ型により、分かりやすく効率良い方法で構造化されたデータを作成、送信、格納できます。Xcodeでは主に2つのタイプのプロパティリストファイルを使い、アプリケーション用にランタイム設定情報を格納します。

- **情報プロパティリスト(Information Property List)。**これらのファイルは、一般的にinfo-plistファイルとして参照され、アプリケーションやiPhone OSで使用する重要な情報を含んでいます。info-plistファイルで定義されるアプリケーションのプロパティの詳細については、『*iOS Application Programming Guide*』の「The Information Property List」を参照してください。
- **エンタイトルメント(Entitlements)。**これらのファイルでは、アプリケーションに、iPhone OSの機能（Push Notificationなど）やセキュアなデータ（ユーザのキーチェーンなど）へのアクセスを提供するプロパティが定義されています。

プロパティリストファイルの詳細については、『*Property List Programming Guide*』を参照してください）。

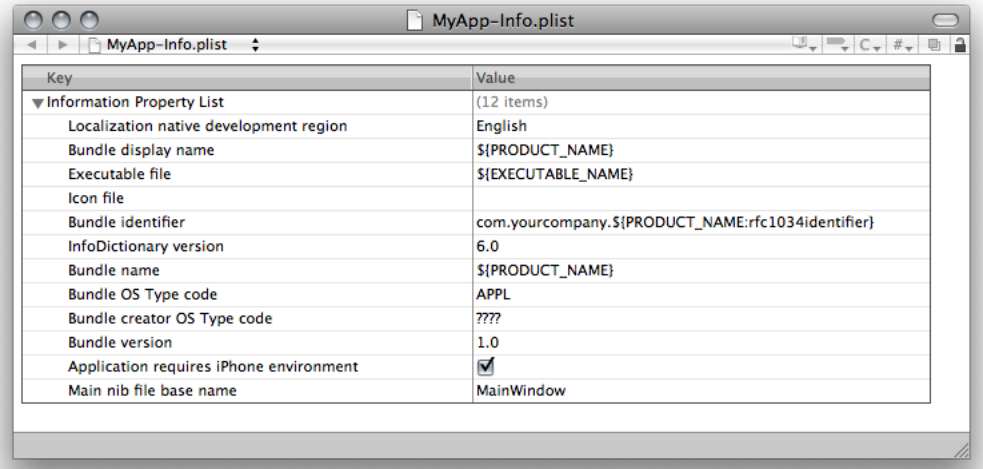
プロパティリストファイルを編集するには、次のいずれかのアクションを実行します。

- プロジェクトウインドウのエディタペインで編集するには、「グループとファイル(Groups & Files)」リストか詳細ビューでファイルを選択する。
- 専用のエディタウインドウで編集するには、「グループとファイル(Groups & Files)」リストか詳細ビューでファイルをダブルクリックする。

「グループとファイル(Groups & Files)」リストの詳細については、『*Xcode Workspace Guide*』の「Project Window Components」を参照してください。

図 2-1にinfo-plistファイルを表示したプロパティリストエディタウインドウを示します。

図 2-1 info-plistファイルを表示したプロパティリストエディタウインドウ

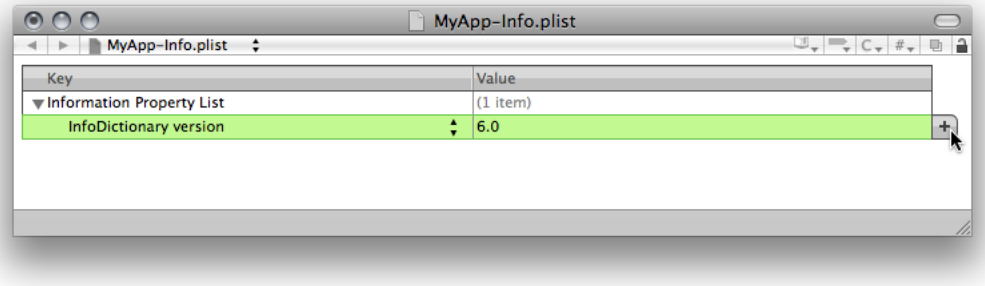


ファイル内の各行はプロパティ定義（またはキーと値のペア）を指定します。「Key」のセルはプロパティ名（およびそのデータ型）を指定します。「Value」のセルはプロパティの値を指定します。

プロパティリストエディタでは、デベロッパはプロパティリストに関して以下のような編集作業を行うことができます。

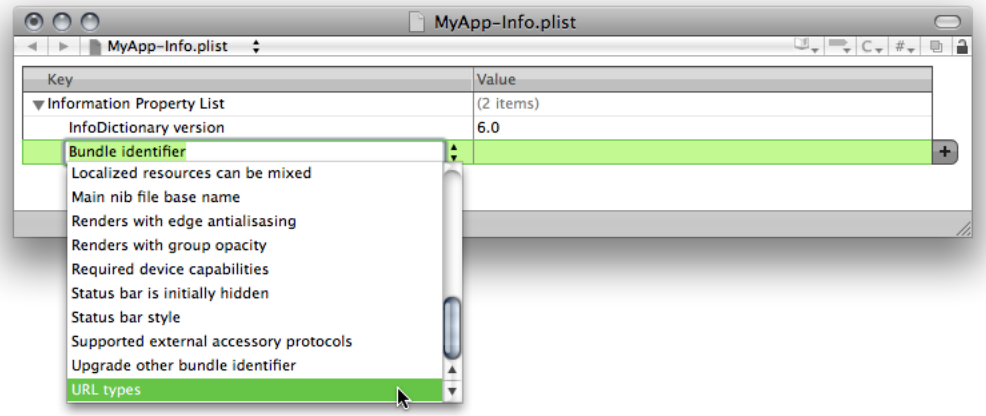
- **兄弟プロパティの追加。**行を選択し三角形（ディスクロージャトライアングル）が閉じた状態で、「Add Sibling」ボタン（図 2-2に示します）をクリックするかReturnキーを押して、プロパティに兄弟を追加できます。

  図 2-2 プロパティリストエディタでの兄弟プロパティの追加



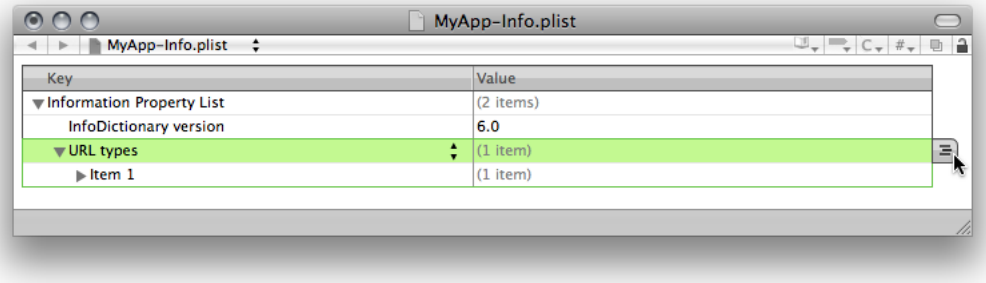
  兄弟行を追加したら、図 2-3に示すように、「Key」セルのプロパティタイプメニューからプロパティタイプを選びます

図 2-3 プロパティリストエディタでのプロパティタイプの指定



- **子プロパティの追加。**複数の値を持つプロパティの行（値が値リストであるプロパティ）を選択しディスクロージャトライアングルが開いた状態で、「Add Child」ボタン（図 2-4に示します）をクリックするかReturnキーを押して、プロパティに子プロパティを追加できます。

図 2-4 プロパティリストエディタでの子プロパティの追加



- **プロパティの削除。**ある行を選択した状態でDeleteキーを押して行を削除します。

プロパティリストファイルの編集の利便性を高めるとともにファイルの構造が必ず正しくなるように、プロパティリストエディタでは、プロパティリストファイルスキーマを使って「Key」カラムにプロパティ名を、書式化された値（ブール型の値のチェックボックスのように）を「Value」カラムに表示します。しかし、プロパティのキーの名前と値を、XML（または“未加工の”）形式で確認したい場合があるかもしれません。書式化されたキー／値の表示と未加工形式での表示を切り替えるには、プロパティリストエディタのショートカットメニューで「Show Raw Keys/Values」オプションを切り替えます。

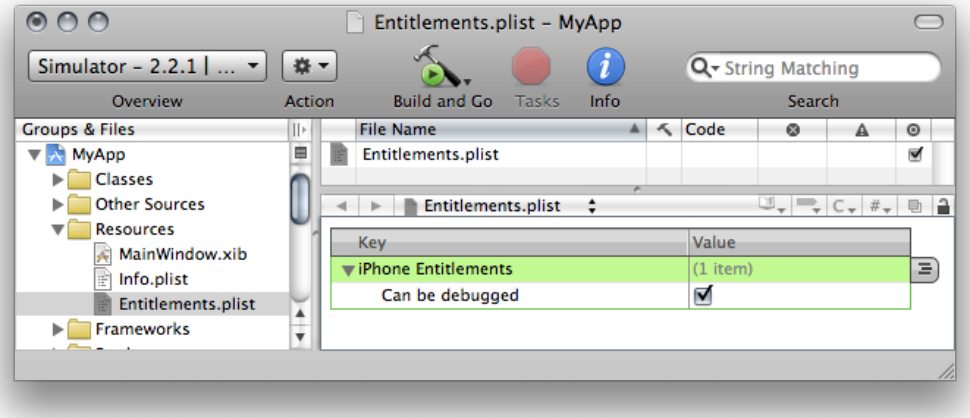
アプリケーションのエンタイトルメントの管理

iPhone OSでは、**エンタイトルメント(Entitlements)**というプロパティを通して、特殊なリソースや機能へのアクセス（アプリケーションのデバッグが可能かなど）を提供します。アプリケーションのエンタイトルメント情報を指定するには、エンタイトルメントの定義（キー／値ペア）を含むEntitlementプロパティリストファイルをプロジェクトに追加します。アプリケーションをビルドすると、Xcodeによって、生成されたアプリケーションバンドルにこのファイルがコピーされます。

Entitlementプロパティリストをプロジェクトに追加するには、次の手順を実行します。

1. 「グループとファイル(Groups & Files)」リストで、「Resources」グループを選択します。
2. 「ファイル(File)」＞「新規ファイル(New File)」を選びます。
3. 「iPhone OS」＞「Code Signing」＞「Entitlements」テンプレートを選びます。
4. このファイルに`Entitlements.plist`という名前を付けます（「Code Signing Entitlements」ビルド設定の値と一致していれば、任意の名前を使用できます。これについてはこの章の後半で説明します）。
5. このプロパティリストファイルのタイプを「iPhone Entitlements」に設定します。

   テキストエディタでこのファイルを選択し、「表示(View)」＞「プロパティリストタイプ(Property List Type)」＞「iPhone Entitlements plist」を選びます。



6. エンタイトルメントのエントリをこのファイルに追加します。

   エンタイトルメントプロパティごとに、次の定義を行う必要があります。

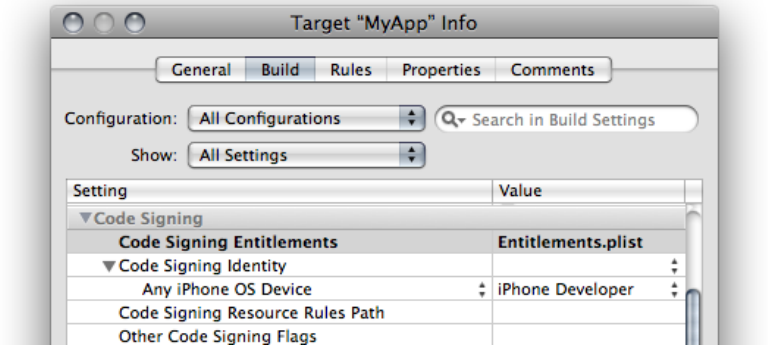
   a. 選択した行の右側の「Add Child」ボタンまたは「Add Sibling」ボタンをクリックします。「Add Child」ボタンには階層を表す3つの線があります。「Add Sibling」ボタンにはプラス記号(+)があります。

   b. 表示されたポップアップメニューからエンタイトルメントプロパティを選びます。

追加する必要があるエンタイトルメントがメニューに表示されない場合は、「表示(View)」＞「プロパティリストタイプ(Property List Type)」＞「ファイルタイプのデフォルト(Default for File Type)」を選びます。次に、エンタイトルメントキーの名前とタイプを入力します。

c. そのプロパティの値を入力します。

7. ターゲットの「Code Signing Entitlements」ビルド設定を、新規に追加したEntitlementsプロパティリストファイルの名前に設定します。



条件付きコンパイルとリンク

iPhone OSランタイム環境には、iPhoneシミュレーション環境とiPhoneデバイス環境の2つがあります。アプリケーションをMac上でテストするには前者を使い、デバイス上でテストするには後者を使います。これらの環境は基本的に異なります。そのため、2つの環境に別々に実装されたテクノロジーを使用する場合、一部は、iPhoneシミュレータアプリケーションで実行しデバイス上で実行しないように、コードを調整する必要があります。CFNetworkフレームワークを使用する場合と同様に、シミュレータとデバイス上で同じ機能を取得するように異なるフレームワークにリンクする必要もあるかもしれません。

このセクションでは、アクティブSDKがiPhoneシミュレータSDKファミリとiPhoneデバイスSDKファミリのどちらに属しているかに応じて、iPhoneシミュレータまたはデバイスをターゲットにする方法と、どのフレームワーク（またはライブラリ）をリンクさせるかを示します。

iPhoneアプリケーションのソースコードの条件付きコンパイル

デバイス上ではなくiPhoneシミュレータ上でコードを実行する必要がある場合と、逆の場合があります。このような場合には、プリプロセッサマクロの`TARGET_OS_IPHONE`および`TARGET_IPHONE_SIMULATOR`を使用して、コードを条件付きでコンパイルします。

リスト 2-1に、`TARGET_IPHONE_SIMULATOR`マクロを使用して、iPhone OS用に作成したコードをシミュレータ向けにコンパイルするか、またはデバイス向けにコンパイルするかを指定する方法を示します。

リスト 2-1　シミュレータ向けコンパイルの指定

```
// helloに"Hello, <deviceまたはsimulator>"!と設定する
#if TARGET_IPHONE_SIMULATOR
   NSString *hello = @"Hello, iPhone Simulator!";
#else
   NSString *hello = @"Hello, iPhone device!";
#endif
```

リスト 2-2は、Mac OS XとiPhone OSで共用するソースにおいてTARGET_OS_IPHONEマクロを使用する方法を示します。

リスト 2-2　iPhone OS向けコンパイルの指定

```
#if TARGET_OS_IPHONE
   #import <UIKit/UIKit.h>
#else
   #import <Cocoa/Cocoa.h>
#endif
```

TARGET_OS_IPHONEマクロおよびTARGET_IPHONE_SIMULATORマクロは、TargetConditionals.hヘッダファイルに定義されています。

iPhoneアプリケーションのフレームワークの条件付きリンク

iPhoneシミュレータ上で実行する場合と、デバイス上で実行する場合とで、異なるフレームワークにリンクするようにアプリケーションターゲットを設定する必要が生じることがあります。

特定のSDKを使用しているときにだけフレームワークをリンクするには、`-framework <framework_name>`に定義を適用させるSDKのすべての構成で「他のリンカフラグ(Other Linker Flags)」ビルド設定を設定します。

必要であれば、「他のリンカフラグ(Other Linker Flags)」のビルド設定に別の条件を追加し、異なるSDKおよびフレームワークを指定できます。

特定のSDKに対するビルド設定の定義の詳細については、『*Xcode Project Management Guide*』の「Editing Conditional Build Settings」を参照してください。

iPhoneからiPadへのターゲットのアップグレード

既存のiPhoneアプリケーションをiPadデバイスで実行するようにアップグレードしたい場合は、iPhoneアプリケーションをビルドするターゲットをiPhoneアプリケーションおよびiPadアプリケーションの両方をビルドできるターゲットにアップグレードするか、iPadアプリケーションをビルドするターゲットをプロジェクトに追加する必要があります。

iPhoneターゲットをiPad開発にアップグレードするには、「グループとリスト(Groups & Files)」リストのターゲットを選択し、「プロジェクト(Project)」＞「現在のターゲットをiPad用にアップグレード(Upgrade Current Target for iPad)」を選びます。

iPadアプリケーションの開発の詳細については、『*iPad Programming Guide*』を参照してください。

アプリケーションの実行

アプリケーションの実行またはデバッグの準備ができたら、Xcodeのビルドシステムを使用してアプリケーションをビルドします。ビルドエラーがなければ、iPhoneシミュレータまたはデバイス上でアプリケーションを実行できます。

注： iPhoneシミュレータでアプリケーションをテストしたら、iPhone OSベースのデバイス上でアプリケーションをテストして、パフォーマンスの測定と調整を行う必要があります。デバイス上でアプリケーションを実行できるようにするには、iPhoneデベロッパプログラムのメンバになる必要があります。詳細については、「iPhoneデベロッパプログラムのメンバになる」（51 ページ）を参照してください。

iPhone SDKは、iPhoneシミュレータSDKとiPhoneデバイスSDKの2つのSDKファミリで構成されています。

- **iPhoneシミュレータSDK：**このSDKは、iPhoneシミュレータで実行するアプリケーションをビルドします。
- **iPhoneデバイスSDK：**このSDKは、デバイスで実行するアプリケーションをビルドします。

アプリケーションをビルドして実行するには、次の手順に従います。

1. ビルド環境を指定する。
2. 実行環境を指定する。
3. ベースSDKを設定する。
4. アプリケーションを実行する場所を指定する。
5. アプリケーションをビルドする。
6. アプリケーションを実行する。

この章では、アプリケーションを実行またはデバッグするために必要な手順をそれぞれ説明します。iPhone OSの機能を紹介したアプリケーションを参照する場合は、「サンプルアプリケーションの実行」（33 ページ）から始めてください。

サンプルアプリケーションの実行

iPhone Dev Centerでは、iPhoneのアプリケーションの開発プロセスについての学習に役立つさまざまなリソースを提供しています。これらのリソースタイプの1つにサンプルコードがあります。サンプルコードはお使いのコンピュータにダウンロードしてiPhoneシミュレータで実行できます。

iPhoneデベロッパプログラムのメンバであれば、お使いのデバイス上でサンプルコードを実行することもできます。詳細については「iPhoneデベロッパプログラムのメンバになる」 （51 ページ）を参照してください。

サンプルコードを実行するには、次の手順を実行します。

1. 使用するサンプルコードを含むZIPアーカイブ（.zipという拡張子のファイル）をダウンロードします。このアーカイブはアプリケーション名にちなんで命名されています（たとえば、HelloWorld.zip）。
2. Xcodeの「ドキュメント(Documentation)」ウインドウからサンプルコードをダウンロードした場合、Xcodeがアーカイブを展開し、プロジェクトを自動的に開きます。自動的に開かない場合は、次の手順を続けます。
3. アーカイブをダブルクリックして展開します。
4. サンプルコードプロジェクトのディレクトリに移動します。このディレクトリもサンプルアプリケーションにちなんで命名されています（たとえば、HelloWorld）。
5. プロジェクトパッケージ（.xcodeprojという拡張子のファイル）をダブルクリックします。たとえば、HelloWorld.xcodeprojです。Xcodeでプロジェクトが開きます。

> **トラブルシューティング：Xcodeが起動しない。：** Xcodeが起動しない場合は、Xcodeをダウンロードしてお使いのコンピュータにインストールする必要があります。Xcodeをダウンロードするには、iPhone Dev Centerにアクセスしてください。

Xcodeでサンプルコードプロジェクトを開いたら、次のセクションの手順に従ってアプリケーションをビルドして実行します。

ビルド環境の設定

アプリケーションをビルドする際、Xcodeは、フレームワーク、ライブラリ、アプリケーション、コマンドラインツールなどのリソースで構成されるビルド環境を使用します。iPhone SDKのリリースのたびにこの環境の改良が行われ、ユーザインターフェイス機能の追加や、コンパイラやリソース処理ツールの強化がなされています。これらのリソースのほかに、アプリケーションのビルドをデバッグ用にするか配布用にするかを指定できます。このセクションでは、ビルド環境を設定する方法について説明します。

ベースSDKを設定する

Xcodeでのアプリケーションのビルド方法を決定する主な要素の1つは、ビルドに使用するSDKです。前述のとおり、iPhone SDKには2つのファミリがあります。1つはデバイスをターゲットとし、1つはiPhoneシミュレータをターゲットとするものです。iPhoneデバイスSDKファミリに属するSDKを使用してビルドしたアプリケーションは、iPhone OSベースのデバイス上でのみ実行できます。逆に、iPhoneシミュレータSDKファミリに属するSDKを使用してビルドしたアプリケーションは、iPhoneシミュレータでのみ実行できます。

プロジェクトの「ベースSDK (Base SDK)」ビルド設定（プロジェクトの「情報(Info)」ウインドウの「一般(General)」ペイン、またはプロジェクトまたはターゲットの「情報(Info)」ウインドウの「ビルド(Build)」ペインから設定できます）で、Xcodeがアプリケーションのビルドに使用するSDKを指定します。図3-1に、プロジェクトの「情報(Info)」ウインドウの「一般(General)」ペインの「すべての構成のベースSDK (Base SDK for All Configurations)」ポップアップメニューを示します。

図 3-1 プロジェクトの「情報(Info)」ウインドウの「すべての構成のベースSDK (Base SDK for All Configurations)」ポップアップメニュー

重要： 補足ですが、特定バージョンのSDK（たとえば、iPhone OS 3.2をターゲットとするiPhone SDK 3.2など）を使用して作成したアプリケーションは、対応するSDKより前の（iPhoneシミュレータまたはデバイスの）OSのバージョンによっては実行できない場合があります。つまり、たとえばiPhone OS 3.1.3でアプリケーションを実行する場合にはiPhone SDK 3.1.3を使ってアプリケーションをビルドする必要があります。

コード署名IDの設定

デバイス上で実行するようにアプリケーションをビルドする場合は、キーチェーンに格納されている開発用証明書（コード署名IDとも呼ばれます）を利用してアプリケーションに署名します。開発用証明書を取得してインストールする方法については、「iPhone開発用のMacの準備」（51 ページ）を参照してください。

「コード署名ID (A Code Signing Identity)」ビルド設定は、Xcodeがバイナリへの署名に使用するコード署名IDを指定します。Xcodeは、デフォルトのキーチェーンからコード署名IDを検索します。図3-2は、図3-3（デフォルトのキーチェーンの名前はキーチェーンリストに太字で表示されています）に示したログイン（デフォルト）キーチェーンから取得した「コード署名ID (Code Signing Identity)」ビルド設定のオプションセットの例を示しています。

図 3-2 「コード署名ID (Code Signing Identity)」ビルド設定オプション

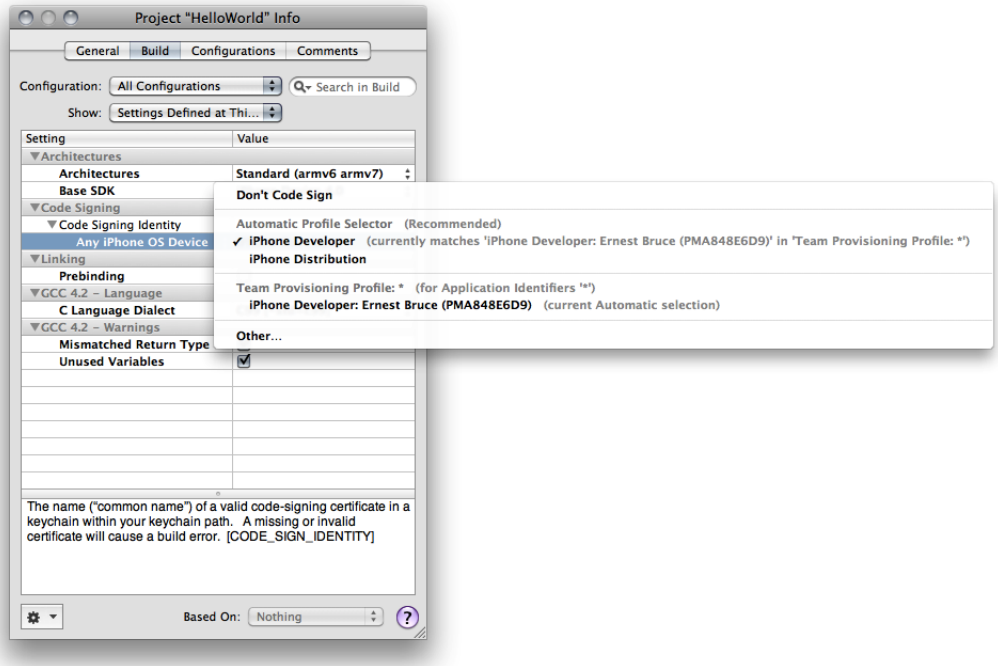
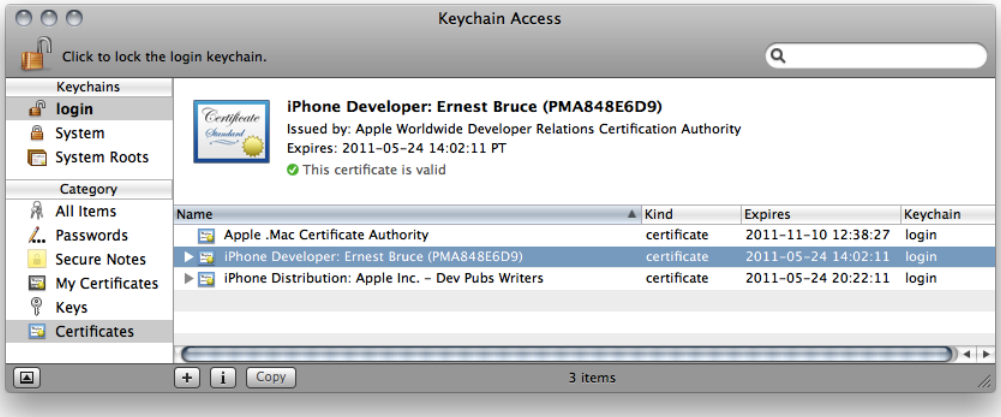


図 3-3 開発用証明書を含むキーチェーン



「コード署名ID (Code Signing Identity)」ビルド設定に設定可能な値は、次のとおりです。

- **Don't Code Sign**。バイナリに署名しない場合はこのオプションを選びます。この値を設定すると、iPhoneデバイスSDKを使用してアプリケーションをビルドした場合にそのビルドは失敗します。
- **Automatic Profile Selectors**。これを選択すると、「iPhone Developer」または「iPhone Distribution」で始まる名前を持つIDが検索されます。

- **コード署名ID (Code Signing Identity)。**特定のコード署名ID。デフォルトのキーチェーン内のコード署名IDは、それらをiPhoneプロビジョニングポータル(iPhone Provisioning Portal)からダウンロードしたときのプロビジョニングプロファイルの名前の下に一覧表示されます。期限切れまたはその他の理由で無効になったIDは、グレイ表示されており選択できません。

Xcodeのプロジェクトテンプレートは、iPhone Developer自動セレクタを使用するように設定されています。

重要： 同じ名前の別のコード署名IDを使用する必要がある場合は、IDごとに別のMac OS Xユーザアカウントを使用しなければなりません。

アーキテクチャの設定

iPhone OSのデバイスは、`armv6`および`armv7`を含むアーキテクチャセットのうちの1つを使用します。「アーキテクチャ(Architectures)」ビルド設定は、アプリケーションをビルドするためのアーキテクチャを指定します。この設定の値を指定する際には、次の2つの選択肢があります。

- **Standard。**サポートされているすべてのiPhone OSデバイスと互換性のある、共通のアーキテクチャによってアプリケーションバイナリを生成します。このオプションは最小のアプリケーションを生成します。ただし、すべてのデバイスで、できる限り高速に実行するための最適化は行われません。
- **Optimized。**サポートされているiPhoneOSデバイスごとに最適化されたアプリケーションバイナリを生成します。ただし、ビルド時間は「Standard」オプションを使用した場合よりも長くなります。また、複数の命令セットがバンドルされるためアプリケーションも大きくなります。

ターゲットまたはプロジェクトの「情報(Info)」ウインドウの「ビルド(Build)」ペインで、「アーキテクチャ(Architectures)」ビルド設定の値を選びます。

これらの事前定義の値によって提供されるものとは異なるアーキテクチャセットの実行可能コードを含むようにアプリケーションをビルドする場合は、「アーキテクチャ(Architecture)」ビルド設定の値リストから「その他(Other)」を選び、希望するiPhone OSデバイスアーキテクチャの名前を入力します。

アクティブなビルド構成を設定する

ビルド構成は、ビルドされたプロダクトの目的をXcodeに知らせます。Xcodeのプロジェクトテンプレートは、Debug構成とRelease構成で構成されており、アプリケーションをデバッグ用またはユーザへのリリース用にそれぞれビルドできるようになっています。Debugビルドには、アプリケーションをデバッグするのに役立つ情報や機能が含まれています。Releaseビルドは、サイズが小さく、高速なバイナリを作成します。開発の初期から中期の間は、Debug構成を使用します。この構成は、最適なデバッグ環境を提供するからです。Release構成は、開発の最終段階でアプリケーションのパフォーマンスを測定し分析するために使用します。

ビルドプロセスを起動すると、Xcodeは、**アクティブなビルド構成**を使用して、アプリケーションをビルドします。アクティブなビルド構成は、次の2つの場所で設定できます。

- 「プロジェクト(Project)」メニューの「アクティブなビルド構成を設定(Set Active Build Configuration)」サブメニュー

■ ツールバーの「概要(Overview)」ポップアップメニュー

Release構成とDebug構成の詳細については、『*Xcode Project Management Guide*』の「Building Products」を参照してください。

デバイスファミリの設定

デバイスファミリはアプリケーションを実行するデバイスの種類を識別します。デバイスファミリは、iPhoneとiPadの2種類あります。iPhoneデバイスファミリには、iPhoneとiPod touchの両方のデバイスが含まれます。iPadデバイスファミリに含まれるのはiPadデバイスです。

アプリケーションを実行できるデバイスファミリを指定するには以下の手順を実行します。

1. プロジェクトの「情報(Info)」ウインドウまたはターゲットの「情報(Info)」ウインドウを開く。
2. 「ビルド(Build)」ペインで「Targeted Device Family」ビルド設定に移動し、デバイスに該当する値を選ぶ。

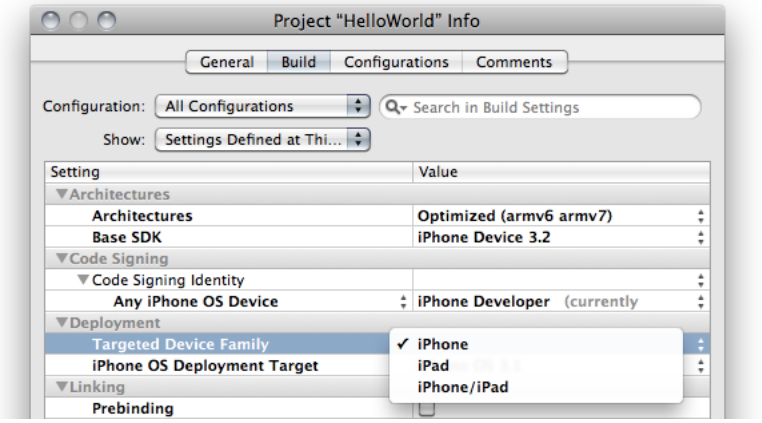


表 3-1に「Targeted Device Family」ビルド設定で指定できる値を一覧に示します。これは、ビルド後のアプリケーションが最適化されるデバイスファミリについても示しています。

表 3-1 「Targeted Device Family」ビルド設定の値

値	アプリケーション最適化の対象デバイス
iPhone	iPhoneとiPod touch
iPad	iPad
iPhone/iPad	iPhone、iPod touch、およびiPad（Universalアプリケーション）

実行環境の指定

iPhone OS（および、それに対応するSDK）の各リリースには、以前のリリースにはなかった機能が含まれています。OSの新しいリリースが公開されると、すぐにアップグレードするユーザもいれば、最新リリースへの移行を延期するユーザもいます。アプリケーションやユーザのニーズに応じて、次の2つの方法のいずれかを選択できます。

- **最新のiPhone OSリリースをターゲットにする。**最新リリースをターゲットにすると、最新バージョンのiPhone OSで使用可能なすべての機能を利用できます。ただし、このアプローチでは、アプリケーションをデバイスにインストールできるユーザが少なくなる可能性があります。なぜなら、ターゲットリリースより前のiPhone OSリリースでは、このアプリケーションを実行できないからです。
- **以前のiPhone OSリリースをターゲットにする。**以前のリリースをターゲットにすると、幅広いユーザ層にアプリケーションを提供できます（アプリケーションをターゲットのOSとそれ以降のリリースで実行できるため）。ただし、アプリケーションが使用できるiPhone OS機能が制限される可能性があります。

「iPhone OS Deployment Target」ビルド設定で、アプリケーションを実行する最も古いiPhone OSリリースを指定します。デフォルトでは、このビルド設定は「ベースSDK (Base SDK)」ビルド設定の値に対応するiPhone OSリリースに設定されています。たとえば、「ベースSDK (Base SDK)」がiPhoneデバイス2.2.1またはiPhoneシミュレータ2.2.1に設定されている場合、「iPhone Deployment Target」ビルド設定は iPhone OS 2.2.1になります。その様子を図 3-4に示します。

図 3-4　「ベースSDK (Base SDK)」ビルド設定と「iPhone Deployment Target」ビルド設定

iPhoneデバイス3.0 SDKを使用してアプリケーションをビルドしてiPhone OS 2.2.1で実行できるようにするには、「ベースSDK (Base SDK)」をiPhoneデバイス3.0に設定し、「iPhone Deployment Target」を「iPhone OS 2.2.1」に設定します。

アプリケーションをビルドすると、配置ターゲットの選択がアプリケーションの`Info.plist`ファイル内の`MinimumOSVersion`エントリに反映されます。アプリケーションをApp Storeに投稿すると、このプロパティの値に基づいて、アプリケーションを実行できるiPhone OSリリースが表示されます。

iPhoneデバイスSDK： iPhoneデバイスSDKのリリースが、アプリケーションのターゲットiPhone OSリリースよりも古い場合（たとえば、アクティブSDKはiPhoneデバイス2.2.1で、iPhone Deployment Targetはiphone OS 3.0の場合）、アプリケーションがターゲットOSリリースで利用できない機能を使用していることを検出すると、Xcodeはビルド警告を表示します。詳細については、「実行環境の指定」（39 ページ）を参照してください。

また、SDK互換開発手法を使用して、使用しているシンボルがアプリケーションの実行環境で利用できることを保証しなければなりません。これらの手法については、『*SDK Compatibility Guide*』で説明しています。

iPhoneシミュレータSDK： iPhoneシミュレータSDKを使用してアプリケーションをテストする場合、iPhoneシミュレータソフトウェアのバージョンは、アプリケーションをビルドしたiPhoneシミュレータSDKに対応するiPhone OSリリースに設定する必要があります。たとえば、iPhoneシミュレータSDK 2.2.1を使用してアプリケーションをビルドした場合は、iPhoneシミュレータのソフトウェアのバージョンをiPhone OS 2.2.1に設定します。詳細については、「アプリケーションのターゲットの指定」（40 ページ）を参照してください。

アプリケーションのターゲットの指定

開発期間中、たとえば、デバイスによるアプリケーションのパフォーマンスをテストするために、実行環境をiPhoneシミュレータからデバイスに切り替えてアプリケーションを実行したい場合があります。アプリケーションの実行をデバイス上からiPhoneシミュレータ上に切り替えるには、プロジェクトウインドウのツールバーにある「概要(Overview)」ツールバーメニューを使います。図 3-5（41 ページ）にそのメニューを示し、これらの手順を説明します。

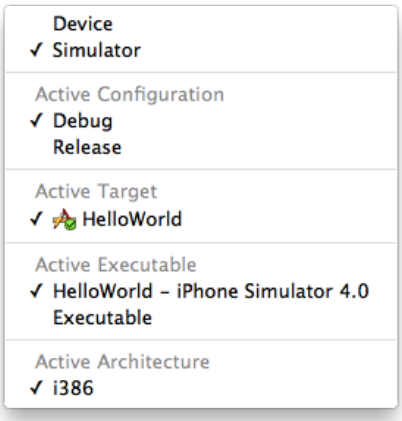
1. アプリケーションを実行する環境を、デバイス上かシミュレーション上か選ぶ。

   「概要(Overview)」ツールバーメニューの最初の2つのオプションで、これらの選択肢から選べます。

2. アプリケーションを実行するデバイスまたはiPhoneシミュレータのバージョンを選ぶ。

   「概要(Overview)」ツールバーメニューの「アクティブな実行可能ファイル(Active Executable)」セクションで、デバイスまたはiPhoneシミュレータのバージョンを指定できます。

図 3-5 プロジェクトウインドウのツールバーにある「概要(Overview)」ポップアップメニュー



アプリケーションのビルド

ビルドプロセスを起動するには、「ビルド(Build)」＞「ビルド(Build)」を選びます。

プロジェクトウインドウのステータスバーに、ビルドが成功したか、または、ビルドエラーや警告が出力されたかが表示されます。ビルドエラーと警告は、テキストエディタまたはプロジェクトウインドウで参照できます。

重要： iPhoneシミュレータ用にビルドすると、生成されたバイナリは、iPhoneシミュレータのターゲットリリースでのみ動作します。それより前、または後のリリースのiPhoneシミュレータでは動作しません。

iPhone OSベースのデバイスは、ARMとThumbの2つの命令セットをサポートしています。デフォルトでは、XcodeはThumb命令セットを使用します。Thumbを使用すると通常、ARMよりもコードサイズを約35%削減できるからです。浮動小数点を扱うコードを多用するアプリケーションの場合は、ThumbよりもARM命令を使用した方が、パフォーマンスが向上する可能性があります。「Compile for Thumb」ビルド設定をオフにすると、Thumbをオフにできます。

ビルドが正常に完了したら、「アプリケーションの実行」（43 ページ）で説明されている、アプリケーションの実行に進むことができます。

ビルドプロセスの詳細については、『*Xcode Project Management Guide*』の「Building Products」を参照してください。

ビルドエラーの解決

このセクションでは、アプリケーションのビルド中に発生する可能性のあるビルドエラーを示し、それらの解決方法を提案します。

プロビジョニングプロファイルエラー

デバイス用にビルドした場合、アプリケーションをデバイスにインストールする際にプロビジョニングプロファイルの問題が原因でXcodeでトラブルが発生したときは、iPhoneデベロッパプログラムのポータル（「iPhoneデベロッパプログラムのメンバになる」（51 ページ）を参照）で、プロビジョニングプロファイルが正しく構成されているかを確認してください。必要であれば、「iPhone開発用のMacの準備」（51 ページ）の説明に従って、アプリケーションをコンピュータやデバイスにインストールし直します。

コード署名エラー

バイナリに署名するために使用しているコード署名IDが期限切れになっているか、その他の理由で無効になっている場合は、次のビルドエラーが発生することがあります。

```
Code Signing Identity 'iPhone Developer' does not match any valid, non-expired,
 code-signing certificate in your keychain.
```

このエラーを解決するには、「コード署名IDの設定」（35 ページ）の説明に従って、有効なコード署名IDを選びます。

アプリケーションIDエラー

アプリケーションIDのビルドエラーは、プロビジョニングプロファイルに設定されているアプリケーションID（プログラムのポータルから取得）と、アプリケーションの`CFBundleIdentifier`プロパティで指定されているアプリケーションIDの不一致が原因で起こることがあります。このようなエラーを避けるには、プロファイルのアプリケーションIDを`com.<organization_name>.*`に設定し、アプリケーションの`CFBundleIdentifier`プロパティを`com.<organization_name>.<application_name>`に設定します。つまり、プロビジョニングプロファイルのアプリケーションIDにドメインセットを指定した場合は、アプリケーションの`CFBundleIdentifier`プロパティで指定するアプリケーションIDでそのドメインセットを再定義することはできません。そのドメインセットを縮小することのみが可能です。表 3-2および表 3-3にこれらの項目の有効なペアと無効なペアを示します。

表 3-2　有効なアプリケーションID/`CFBundleIdentifier`プロパティのペア

プロビジョニングプロファイル	アプリケーションID	`com.mycompany.*`
アプリケーションバンドル	`CFBundleIdentifier`	`com.mycompany.MyApp`

表 3-3　無効なアプリケーションID/`CFBundleIdentifier`プロパティのペア

プロビジョニングプロファイル	アプリケーションID	`com.mycompany.MyApp.*`
アプリケーションバンドル	`CFBundleIdentifier`	`com.mycompany.MyApp`

iPhoneアプリケーションのバイナリの構成（`CFBundleIdentifier`プロパティの詳細を含みます）については、『*iOS Application Programming Guide*』の「The Application Bundle」を参照してください。

その他のビルドエラーの解決については、『*Xcode Project Management Guide*』の「Viewing Errors and Warnings」で説明されている手法を使用します。

アプリーケーションの実行

アプリケーションを実行すると、XcodeはiPhoneシミュレータまたはデバイスにアプリケーションをインストールし、それを起動します。

アプリケーションの実行が開始されたら、デバイスのすべての機能を使用してアプリケーションが想定通りに動作するかどうかをテストできます。特に、アプリケーションがデバイスのリソース（CPU、メモリ、バッテリーなど）を可能な限り効率的に使用することを確認すべきです。詳細については、「アプリケーションのチューニング」（79 ページ）を参照してください。

アプリケーションを実行するには、「実行(Run)」＞「実行(Run)」または「実行(Run)」＞「デバッグ(Debug)」を選びます。

トラブルシューティング： デバイス上でアプリケーションをデバッグしようとしたときに、「Failed to start remote debug server」というメッセージが表示された場合、このデバイスは、iPhone SDKに対応するiPhone OSリリースで稼働していない可能性があります。詳細については「iPhone OSのインストール」（56 ページ）を参照してください。

「ビルドと実行」ワークフローの効率化

「ビルド(Build)」コマンド、「実行(Run)」コマンド、および「デバッグ(Debug)」コマンドのほかに、Xcodeにはこれらの操作を1つのタスクとして実行する便利なコマンドがあります。そのコマンドは、「ビルドと実行(Build and Run)」と「ビルドとデバック(Build and Debug)」です。

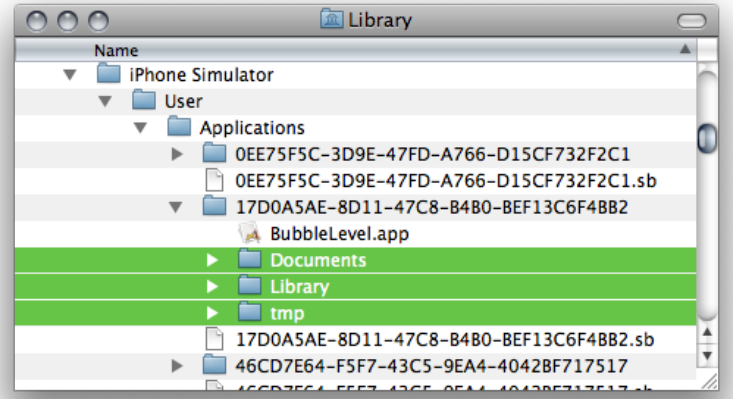
アプリケーションデータの管理

アプリケーションの開発の際には、ビルドをまたいで、iPhoneシミュレーション上あるいは開発デバイス上にユーザ設定やアプリケーションデータを残しておきたい場合があります。Xcodeは、アプリケーションをビルドしたりホストにインストールしたりする際に、ユーザ設定やアプリケーションデータを削除しません。ただし、ユーザが使用するときと同じ状態でアプリケーションをテストする場合は、そのような情報を削除する必要があるかもしれません。それには、ホーム(Home)画面からアプリケーションを削除します。詳細については、「アプリケーションのアンインストール」（49 ページ）を参照してください。

『*iOS Application Programming Guide*』の「File and Data Management」で説明されているとおり、iPhoneアプリケーションはアプリケーションのローカルファイルシステム内に存在するファイルにのみアクセスできます。アプリケーションのローカルファイルシステムは、デバイスおよびiPhoneシミュレータで確認できます。

アプリケーションの、iPhoneシミュレータベースのファイルシステムを表示するには、Finderで`~/ライブラリ/Application Support/iPhone Simulator/User/Applications`ディレクトリに移動します。次に、アプリケーションのバイナリファイルを見つけるには、`Applications`ディレクトリ内の各ディレクトリを開きます。アプリケーションのローカルファイルシステムを構成するディレクトリは、バイナリファイルと一緒に存在します。図 3-6は、iPhone シミュレータのBubbleLevelアプリケーションのローカルファイルシステムを示しています。

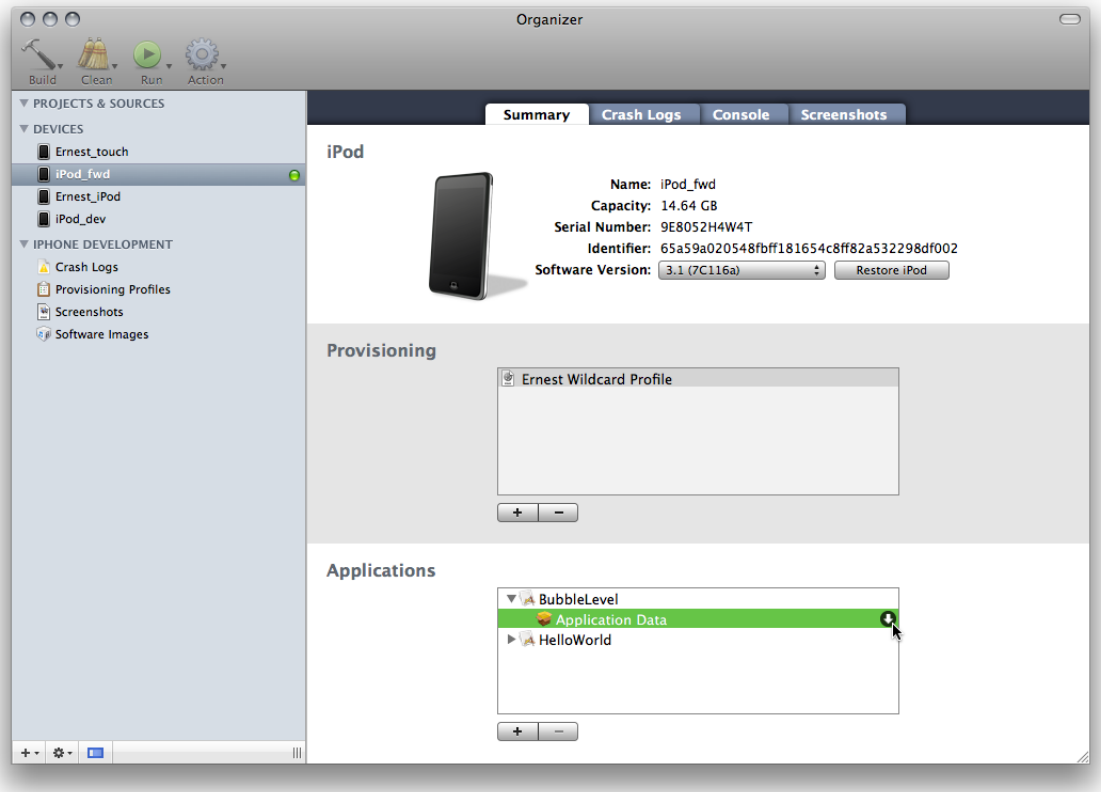
図 3-6 アプリケーションのiPhoneシミュレータベースのローカルファイルシステムの表示



アプリケーションのデバイスベースのファイルシステムをMacにコピーするには、次の手順を実行します。

1. Xcodeで「ウインドウ(Window)」＞「オーガナイザ(Organizer)」を選びます。
2. 「オーガナイザ(Organizer)」で、「DEVICES」リストからデバイスを選択します。
3. 「Summary」ペインで、アプリケーションの横の三角形（ディスクロージャトライアングル）をクリックします。
4. 図3-7に示すように、ダウンロードボタン（「Application Data」パッケージの右側の下向きの矢印）をクリックします。
5. 表示されたダイアログで、ファイルシステムをコピーする場所を選びます。

図 3-7 アプリケーションのデバイスベースのローカルファイルシステムのダウンロード



Mac上にバックアップしたアプリケーションのファイルシステムを復元するには、次の手順を実行します。

1. Xcodeで、「オーガナイザ(Organizer)」を開きます。
2. Finderで、このバックアップを含むディレクトリに移動します。
3. 「オーガナイザ(Organizer)」の「Summary」ペインのアプリケーションにバックアップをドラッグします。

さらに学びたい方へ

Xcodeを使用してアプリケーションをビルドおよび実行する方法の詳細については、『*Xcode Project Management Guide*』を参照してください。

iPhoneシミュレータの使用

iPhoneのシミュレーション環境を利用すると、iPhoneまたはiPadのアプリケーションをコンピュータ上でビルドおよび実行できます。シミュレーション環境を使用して次のことができます。

- 設計や初期のテストの間に、アプリケーションの主な問題点を発見して修正できる。
- iPhoneデベロッパプログラムのメンバになる前に、Xcodeでの開発方法やiPhone OSの開発環境について学ぶことができる。
- アプリケーションのユーザインターフェイスを設計およびテストできる。
- iPhone OSベースのデバイス上で詳細なパフォーマンス分析を実行する前に、アプリケーションのメモリ使用量を測定できる。

iPhoneのシミュレーション環境の主要部分が、**iPhoneシミュレータアプリケーション**です。このアプリケーションは、iPhoneまたはiPadのユーザインターフェイスをコンピュータ上のウインドウに表示します。このアプリケーションは、キーボードとマウスを使用してタップやデバイスの回転、その他のジェスチャをシミュレートすることにより、さまざまな方法でコンピュータとやり取りできます。

重要： iPhone OS 4.0は、Mac OS X v10.6と同じObjective-Cランタイムを使用します。iPhone OS 3.2およびそれ以前のバージョンは、Mac OS X v10.5 Objective-Cランタイムを使用します。この変更により、4.0よりも前のiPhone SDK配布版で生成されたバイナリは、iPhone SDK 4.0と以降の配布版のシミュレータでは実行しません。iPhone SDK 3.2およびそれ以前の配布版を4.0以降の配布版に移行したら、シミュレータで実行するためにiPhoneシミュレータのバイナリをビルドし直さなければなりません。アプリケーションでライセンスされたスタティックライブラリを使用する場合は、iPhone SDK 4.0以降の配布版で生成されたバージョンのライブラリを入手する必要があります。Objective-Cランタイムの詳細については、『*Objective-C Runtime Reference*』を参照してください。

以降の各セクションでは、コンピュータの入力デバイスを使用して、ユーザとデバイス間のやり取りをシミュレートする方法を説明します。また、シミュレータからアプリケーションをアンインストールする方法およびシミュレータのコンテンツをリセットする方法も示します。

シミュレーション環境でのデバイスファミリとiPhone OSバージョンの設定

iPhoneシミュレータは2つのデバイスファミリ（iPhoneとiPad）と、複数のiPhone OSリリースをシミュレートできます。

シミュレートするデバイスファミリを指定するには、「ハードウェア(Hardware)」＞「デバイス(Device)」を選んでデバイスファミリを選びます。

シミュレーション環境で使用するiPhone OSリリースを設定するには、「ハードウェア(Hardware)」＞「バージョン(Version)」を選んでテストするバージョンを選びます。

ハードウェアの操作

iPhoneシミュレータを利用すると、ユーザがデバイス上で実行するほとんどの動作をシミュレートできます。iPhoneシミュレータでアプリケーションを実行した場合、iPhoneシミュレータの「ハードウェア(Hardware)」メニューで以下のハードウェア操作を実行できます。

- **反時計回りに回転(Rotate Left)。**シミュレータを左に回転します。
- **時計回りに回転(Rotate Right)。**シミュレータを右に回転します。
- **シェイクジェスチャー(Shake Gesture)。**シミュレータをシェイクします。
- **ホーム(Home)。**シミュレータをホーム(Home)画面に戻します。
- **ロック(Lock)。**シミュレータをロックします。
- **メモリ警告をシミュレート(Simulate Memory Warning)。**最上位のアプリケーションにメモリ不足の警告を送信します。メモリ不足状態の対処方法については、『*iOS Application Programming Guide*』の「Observing Low-Memory Warnings」を参照してください。
- **着信ステータスバーを切り替える(Toggle In-Call Status Bar)。**ステータスバーを通常状態と着信状態との間で切り替えます。ステータスバーは、通用状態よりも着信状態の方が背が高くなります。このコマンドは、電話中にユーザがアプリケーションを起動すると、アプリケーションのユーザインターフェイスがどうなるかを示します。
- **ハードウェアキーボードをシミュレート(Simulate Hardware Keyboard)。**iPadのシミュレーションでソフトウェアキーボードを切り替えます。iPadデバイスでKeyboard DockまたはWireless Keyboardを使用してシミュレートするにはソフトウェアキーボードをオフにします。

ジェスチャの実行

表 4-1に、シミュレータ上で実行できるジェスチャを示します（ジェスチャについての情報は、『*iPhone Human Interface Guidelines*』を参照してください）。

表 4-1　iPhoneシミュレータでのジェスチャの実行

ジェスチャ	デスクトップアクション
タップ	クリック。
タッチアンドホールド	マウスボタンを押したままにします。
ダブルタップ	ダブルクリック。
スワイプ	1. ポインタを開始位置に置きます。 2. マウスボタンを押したままにします。 3. ポインタをスワイプする方向に移動し、マウスボタンを放します。

ジェスチャ	デスクトップアクション
フリック（はじく）	1. ポインタを開始位置に置きます。 2. マウスボタンを押したままにします。 3. ポインタをすばやくフリックする方向に移動し、マウスボタンを放します。
ドラッグ	1. ポインタを開始位置に置きます。 2. マウスボタンを押したままにします。 3. ポインタをドラッグする方向に移動します。
ピンチ	1. Optionキーを押したままにします。 2. 指でのタッチを表す円を開始位置に移動します。 3. マウスボタンを押したまま、その円を終了位置まで移動します。

アプリケーションのインストール

iPhoneシミュレータSDKを使用してアプリケーションをビルドすると、Xcodeは自動的にiPhoneシミュレータにアプリケーションをインストールします。詳細については、「アプリケーションの実行」（33 ページ）を参照してください。

iPhoneシミュレータには、iPhoneシミュレータSDKを使用してビルドしたアプリケーションしかインストールできません。App Storeから入手したアプリケーションをiPhoneシミュレータにインストールすることはできません。

アプリケーションのアンインストール

iPhoneシミュレータにインストールしたアプリケーションをアンインストールするには、デバイスからアプリケーションをアンインストールするのと同じ方法を使用します。

1. アンインストールしたいアプリケーションのアイコンの上にポインタを置き、アイコンが波打ち始めるまでマウスボタンを押し続けます。
2. アイコンの「閉じる(Close)」ボタンをクリックします。
3. 「ホーム(Home)」ボタンをクリックして、アイコンが波打つのを停止します。

コンテンツと設定のリセット

iPhoneシミュレータのユーザコンテンツと設定を工場出荷時の状態に戻し、インストールしたアプリケーションを削除するには、「iPhoneシミュレータ(iPhone Simulator)」＞「コンテンツと設定をリセット(Reset Content and Settings)」を選びます。

Core Location機能

CoreLocationフレームワークによって報告される位置移動は、iPhoneシミュレータでは、次の位置(Cupertino, CA 95014)に固定されます（精度100メートル）。これは、1つの無限ループに対応します。

- 緯度：北緯37.3317度
- 経度：西経122.0307度

iPhoneシミュレータコンソールログの表示

アプリケーションをiPhoneシミュレータで実行した際にコンソールログを表示する方法については、「コンソールログとクラッシュログの表示」（61 ページ）を参照してください。

ホスト上でのiPhoneシミュレータのファイルシステム

iPhoneシミュレータは、iPhone OSファイルシステムのUserドメインを次のホームディレクトリに格納します。

```
~/ライブラリ/Application Support/iPhone Simulator/User
```

Userドメインは、iPhone OSが保存しているすべてのユーザデータから構成されます。これには、サードパーティのアプリケーションバイナリ、システムデータベース（アドレスブック(Address Book)やカレンダー(Calendar)のデータベースなど）、アプリケーションの環境設定などが含まれます。この場所は、`<iPhoneUserDomain>`と呼ばれます。

iPhoneシミュレータは、システムアプリケーションの環境設定ファイルを`<iPhoneUserDomain>/Library/Preferences`に格納します。

サードパーティアプリケーションの環境設定ファイルは`<iPhoneUserDomain>/Applications/<app_UUID>Library/Preferences`に保存されます。

ハードウェアシミュレーションサポート

iPhoneシミュレータは加速度計やカメラハードウェアをシミュレートしません。

デバイスとデジタルIDの管理

iPhoneシミュレータを利用すると、iPhone OSベースのデバイスを使用せずに、iPhoneアプリケーションの開発を始めることができます。この方法で、アプリケーション開発に使用されるAPIと開発ワークフローに慣れることができます。ただし、アプリケーションを公開する前に実際のデバイス上でアプリケーションをテストして、アプリケーションが確実に意図した通りに動作するようにし、実際のハードウェア上でパフォーマンスを調整する必要があります。

登録アップルデベロッパであれば、iPhone Dev Centerにログインできます。ここでは、iPhoneデベロッパ向けのドキュメントにアクセスしたり、iPhoneシミュレータで動作するiPhoneアプリケーションをビルドできます（登録アップルデベロッパになるには、http://developer.apple.com/jp/programs/register/にアクセスしてください）。ただし、登録アップルデベロッパになってもiPhone OSベースのデバイスでアプリケーションを実行することはできません。それには、iPhoneデベロッパプログラムのメンバになる必要があります。詳細については「iPhoneデベロッパプログラムのメンバになる」（51 ページ）を参照してください。

この章では、iPhone OS開発のためにコンピュータやデバイスを設定する方法を示します。アプリケーションのコンソールログやクラッシュ情報を表示する方法や、アプリケーションの実行中にスクリーンショットをキャプチャする方法も示します。また、開発中のアプリケーションをデバイスにインストールするために必要なデジタルIDを保護する方法についても説明します。

iPhoneデベロッパプログラムのメンバになる

iPhoneデベロッパプログラムは、開発デバイス上でアプリケーションを実行したり、iPhone OSユーザにアプリケーションを配布したりするために必要なツールとリソースを提供しています。iPhoneデベロッパプログラムのメンバになるには、http://developer.apple.com/jp/programs/iphone/にアクセスしてください。

iPhoneデベロッパプログラムのメンバになると、iPhone Dev CenterのiPhoneプロビジョニングポータルにアクセスできます。**iPhoneプロビジョニングポータル**（またはポータル）は、デベロッパの開発デバイスについての情報を格納する、iPhone Dev Centerのアクセス制限領域です。ここでは、iPhoneアプリケーションの開発や配布に使用する、開発用証明書やプロビジョニングプロファイルなどのその他のリソースも管理します。

重要： iPhoneデベロッパプログラムのメンバでなければ、iPhoneプロビジョニングポータルにアクセスできません。

iPhone開発用のMacの準備

デバイスでアプリケーションを実行するには、コンピュータとデバイスをiPhone OS開発用に設定する必要があります。このセクションでは、Xcodeの「オーガナイザ(Organizer)」で管理できるプロセスの概要を説明します。

注： iPhoneを開発用に構成しても、通常の操作の妨げにはなりません。

開発用にデバイスを準備する際には、次のデジタル資産を作成または取得します。

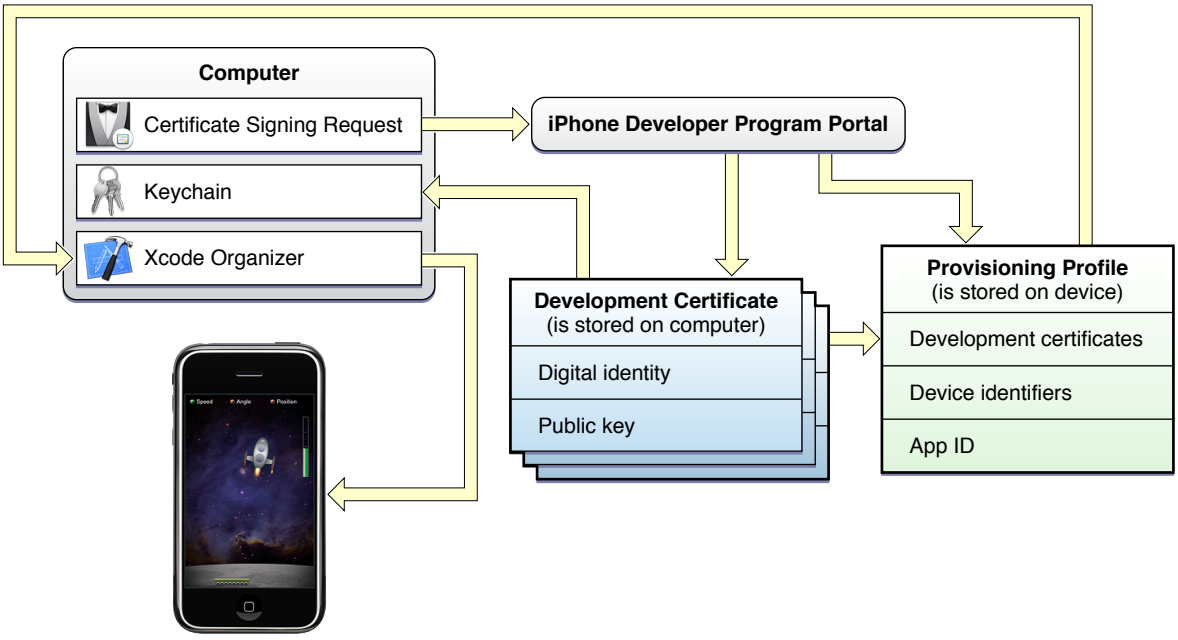
- **証明書署名要求(Certificate Signing Request)。**CSR（証明書署名要求：Certificate Signing Request）には、開発用証明書(Development/Developer Certificate)を生成するために使用される個人情報が含まれています。iPhoneプロビジョニングポータルにこの要求を提出します。
- **開発用証明書(Development Certificate)。**開発用証明書は、iPhoneアプリケーションのデベロッパを識別します。CSRが承認されたら、開発用証明書をポータルからダウンロードし、キーチェーンに追加します。

  XcodeでiPhoneアプリケーションをビルドすると、Xcodeはキーチェーンの中から開発用証明書を探します。証明書が見つかった場合は、Xcodeはアプリケーションに署名します。見つからなかった場合は、ビルドエラーを報告します。
- **プロビジョニングプロファイル(Provisioning Profile)。**プロビジョニングプロファイルは、1つ以上の開発用証明書、デバイス、およびApp ID（iPhoneアプリケーションID）を関連付けます。

  開発用証明書を使って署名したiPhoneアプリケーションをデバイスにインストールできるようにするには、デバイスに少なくとも1つ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。このプロビジョニングプロファイルは、デベロッパ（開発用証明書によって識別する）とデバイス（一意のデバイス識別子によって識別する）を識別できなければなりません。iPhoneデベロッパチームに属している場合、チームのメンバは、適切に定義されたプロビジョニングプロファイルがあれば、ほかのメンバがビルドしたアプリケーションを自分のデバイス上で実行できます。

図 5-1に、これらのデジタル資産の関係を示します。

図 5-1　iPhone開発用のコンピュータおよびデバイスの準備



開発用デバイスの設定

デバイス上でアプリケーションを実行するには、デバイスを開発用に設定する必要があります。この処理には、主に次の2つの作業が必要です。

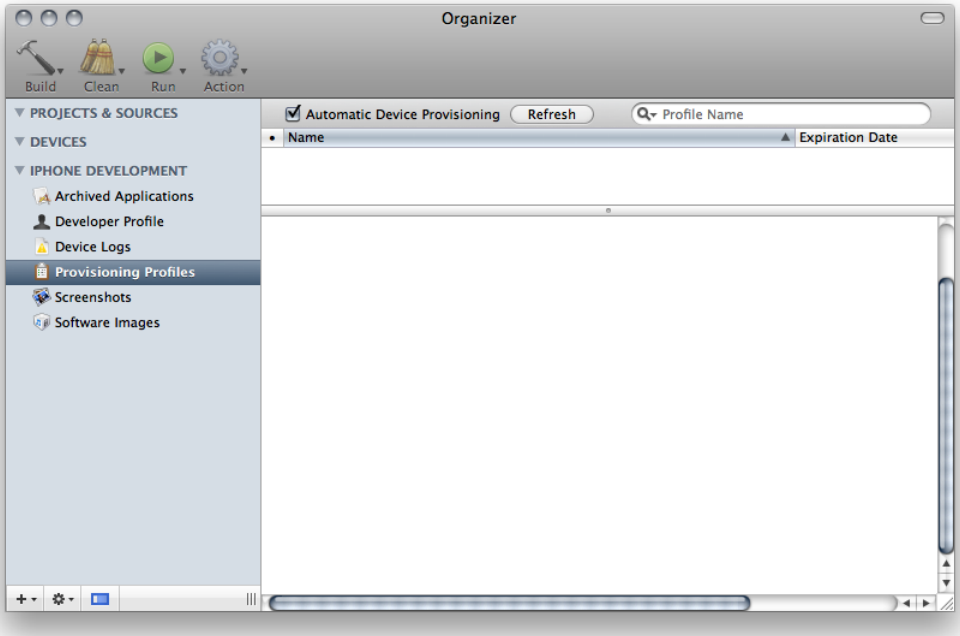
1. iPhoneプロビジョニングポータル上でデベロッパのデバイスを識別するプロビジョニングプロファイルを作成する。
2. そのプロビジョニングプロファイルをデバイスにインストールする。

「汎用的な開発用にデバイスを設定する」（53 ページ）に説明するとおり、新しいデバイスを開発用に指定した場合、Xcodeに自動的にこれらの処理を実行させることができます。ただし、アプリケーションで特殊なプロビジョニングプロファイルを必要とする場合（アプリケーションでPush NotificationやIn App Purchaseを使用する場合）は、「特殊な開発用にデバイスを設定する」（55 ページ）の手順に従う必要があります。

汎用的な開発用にデバイスを設定する

汎用的な開発用に新しいデバイスを設定するには、次の手順に従います。

1. Xcodeの「オーガナイザ(Organizer)」を開きます。
2. 「IPHONE DEVELOPMENT」グループで、「Provisioning Profiles」を選択します。
3. 「Automatic Device Provisioning」オプションを選択します。



4. デバイスを接続します。

5. 「DEVICES」グループで自分のデバイスを選択し、「Use for Development」をクリックします。

6. 表示されたダイアログでiPhoneデベロッパプログラムの資格情報を入力します。

Xcodeがポータルにログインし、新しいデバイスをポータルに追加します。「Team Provisioning Profile:*」という名のプロビジョニングプロファイルもデバイスに追加します。このプロビジョニングプロファイルは、個々にデバイスを設定したチームメンバ全員で自動的に共有されます。

7. Xcodeで、デベロッパに代わって開発用証明書を求めるようにする場合、「Submit Request」をクリックします。

デバイスが正しく設定されたことを確認するために、サンプルプロジェクトをダウンロードし、デバイスSDKを使ってビルドして、生成されたアプリケーションをデバイス上で実行することができます。詳細については、「サンプルアプリケーションの実行」（33 ページ）を参照してください。

特殊な開発用にデバイスを設定する

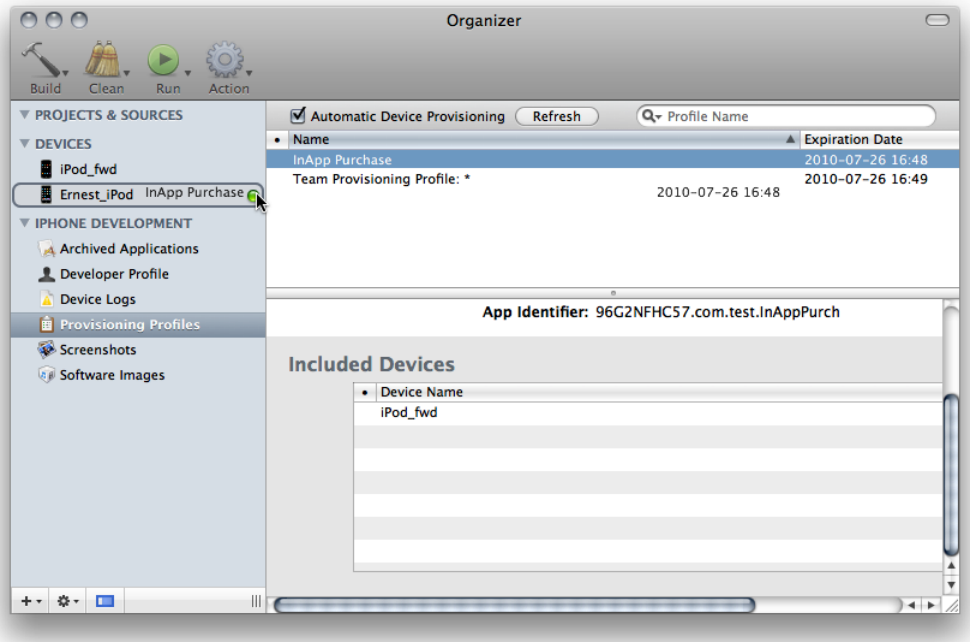
特殊なプロビジョニングプロファイル（たとえば、Push NotificationやIn App Purchaseを使用するアプリケーションの実行に必要なファイルなど）を使用する場合、ポータルで対象のプロビジョニングプロファイルを作成し、開発用デバイスにインストールする必要があります。

まず、「汎用的な開発用にデバイスを設定する」（53 ページ）の手順に従ってポータルにデバイスを追加します。

特殊なプロビジョニングプロファイルをデバイスに追加するには、次の手順に従います。

1. ポータルで、特殊なプロビジョニングプロファイルにデバイスを追加します。
2. Xcodeの「オーガナイザ(Organizer)」で、「IPHONE DEVELOPMENT」グループの「Provisioning Profiles」を選択します。

   プロビジョニングプロファイルのリストにデバイスに追加したいプロビジョニングプロファイルが表示されていない場合は「Refresh」をクリックします。
3. 特殊なプロビジョニングプロファイルを「DEVICES」グループの目的のデバイスにドラッグします。



iPhone OSのインストール

iPhone SDKを使用してアプリケーションを開発する場合は、そのSDKが対象とするiPhone OSバージョンで稼動しているデバイス上で、これらのアプリケーションをテストする必要があります。iPhone SDKの最新リリースは、iPhone Dev Centerからダウンロードできます。最新のiPhone OSをデバイスにインストールするには、iTunesを使用します。

注： 先行期間中は、iPhone SDKおよびiPhone OSの先行リリースをiPhone Dev Centerからダウンロードできます。

デバイスを復元するには、次の手順を実行します。

1. Xcodeを起動し、「オーガナイザ(Organizer)」ウインドウを開きます。
2. デバイスをコンピュータに接続します。
3. 「DEVICES」リストからそのデバイスを選択します。
4. 「Software Version」ポップアップメニューから、デバイスに配置したいiPhone OSのバージョンを選びます。

   iPhone SDKの先行リリースを使用している場合に、インストールするiPhone OSのバージョンが「Software Version」ポップアップメニューに表示されないときは、次の手順を実行します。

   a. iPhone SDKの先行リリースに対応するiPhone OSの先行リリースをhttp://developer.apple.com/jp/からダウンロードします。

      重要： iPhone OSをダウンロードするには、iPhoneデベロッパプログラムのメンバでなければなりません。

   b. 「Software Version」ポップアップメニューから「Other Version」を選びます。

   c. iPhone OSデベロッパソフトウェアを含むディスクイメージに移動し、「開く(Open)」をクリックします。

      Xcodeが、そのディスクイメージからiPhone OSソフトウェアを抽出します。ダウンロードしたディスクイメージは破棄できます。

   d. 「Software Version」ポップアップメニューから、新しくダウンロードしたiPhone OSのバージョンを選びます。

   先行期間中以外は、iTunesを使用してiPhone OSをインストールする必要があります。
5. デバイスの種類に応じて、「Restore iPhone」または「Restore iPod」をクリックします。
6. iTunesを使用して、デバイスに名前を付けます。

デバイス上でのアプリケーションの実行

「iPhone開発用のMacの準備」（51 ページ）と「iPhone OSのインストール」（56 ページ）（必要な場合）の手順を実行したら、開発デバイス上でアプリケーションを実行できます。

XcodeでアクティブSDKをiPhone Device SDKに設定することによって、デバイス用にアプリケーションをビルドして、接続されているデバイス上で実行できます。詳細については、「アプリケーションのビルド」（41 ページ）を参照してください。

スクリーンショットのキャプチャ

スクリーンショットは、アプリケーションの記録に役立ちます。また、この手順は、ユーザがアプリケーションのアイコンをタップしたときに、iPhone OSによって表示されるアプリケーションのデフォルトイメージを作成する手順でもあります。デバイスの画面のスクリーンショットは、「オーガナイザ(Organizer)」から、またはデバイス上で直接キャプチャできます。

「オーガナイザ(Organizer)」からスクリーンショットをキャプチャするには、次の手順を実行します。

1. アプリケーションの画面をスクリーンショット用に設定します。

   アプリケーションのワークフローによっては、コード内にブレークポイントを設定して、そのブレークポイントまでアプリケーションを実行する必要がある場合もあります。

2. 「オーガナイザ(Organizer)」ウインドウを開いてデバイスを選択し、「Screenshot」をクリックします。

3. 「Capture」をクリックします。

   そのスクリーンショットをアプリケーションのデフォルトイメージにするには、「Save As Default Image」をクリックします。

   スクリーンショットのPNGファイルを取得するには、それをデスクトップにドラッグします。

コンピュータにiPhotoがインストールされている場合は、デバイス上で直接スクリーンショットをキャプチャして、iPhotoライブラリに読み込むことができます。

デバイス上でスクリーンショットをキャプチャするには、「ロック(Lock)」ボタンと「ホーム(Home)」ボタンを同時に押します。スクリーンショットが「写真(Photos)」アプリケーションの「カメラロール(Saved Photos)」アルバムに保存されます。

注： スクリーンショットをキャプチャすると、デフォルトイメージには、表示されているとおりのステータスバーが含まれますが、iPhone OSは、アプリケーションの起動時にそれを現在のステータスバーに置き換えます。

デジタルIDの管理

iPhoneプロビジョニングポータルからの証明書を要求すると、公開鍵と非公開鍵のペアが作成されます。公開鍵は証明書に含まれます。非公開鍵はキーチェーンに保存されます。これらのアイテムを使って、Xcodeは作成したアプリケーションにコード署名します。iPhoneアプリケーションの開発に別のコンピュータを使用する必要がある場合、これらのデジタルIDのアイテムをそのコンピュータに転送する必要があります。これは、Xcodeの「オーガナイザ(Organizer)」を使って実行できます。

デジタルIDのアイテムをセキュアなファイルにエクスポートするには、次の手順に従います。

1. Xcodeの「オーガナイザ(Organizer)」を開きます。
2. 「IPHONE DEVELOPMENT」グループで、「Developer Profile」を選択します。
3. 「Export Developer Profile」をクリックします。
4. ファイルに名前を付けて保存場所を選択し、ファイル保護のためにパスワードを入力してから「保存」をクリックします。

別のコンピュータでiPhoneアプリケーションを開発する必要がある場合、次の手順を実行してデジタルIDアイテムをインポートします。

1. デベロッパプロファイル(Developer Profile)のアーカイブを2台目のコンピュータにコピーします。
2. 2台目のコンピュータでXcodeを起動します。
3. 「オーガナイザ(Organizer)」を開きます。
4. 「IPHONE DEVELOPMENT」グループで、「Developer Profile」を選択します。
5. 「Import Developer Profile」をクリックします。
6. アーカイブの場所に移り、アーカイブの保護に使ったパスワードを入力して「開く」をクリックします。

アプリケーションのデバッグ

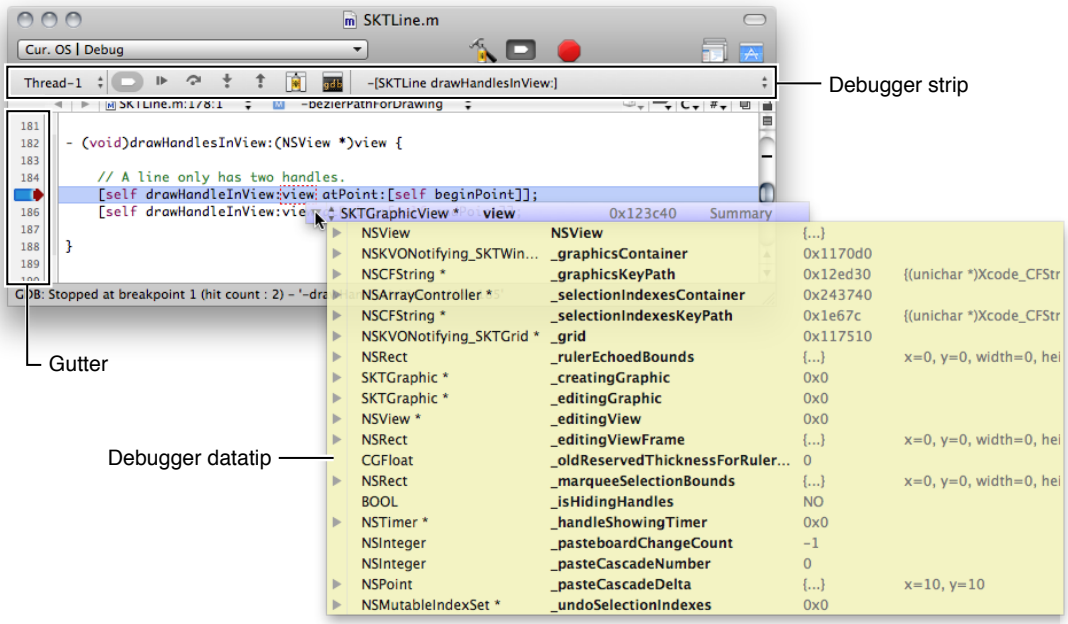
この章では、Xcodeのデバッグ機能について説明します。

デバッグのワークフローにおいて、アプリケーションがファイルシステムに書き込んだデータを表示または操作する必要が生じることがあります。たとえば、テストのための特定の条件を再現するためにアプリケーションが保存したデータを編集する場合があります。アプリケーションのデータの操作方法についての詳細は、「アプリケーションデータの管理」（43 ページ）を参照してください。

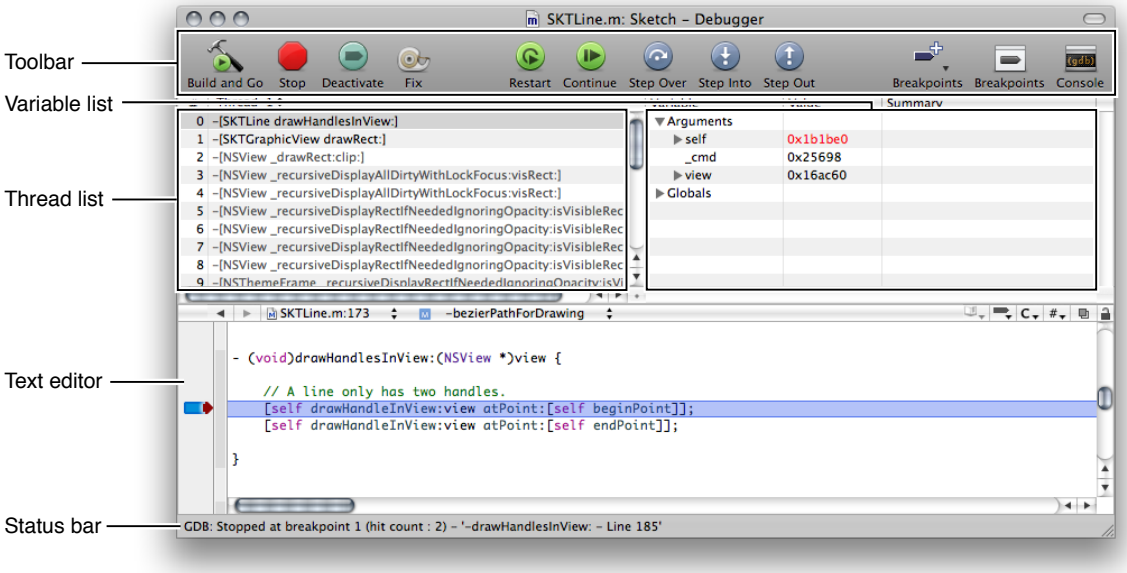
デバッグ機能の概要

Xcodeには、コード内のバグを発見して解消するために、以下のような複数のデバッグ環境があります。

- **テキストエディタ。**テキストエディタを利用すると、まさにコード*内*でコードをデバッグできます。ここでは、ほとんどのデバッグ機能が利用できます。
  - ブレークポイントの追加と設定
  - スレッドごとのコールスタックの表示
  - 変数の上にマウスを置くことによる、変数値の表示
  - 1行のコードの実行
  - 関数やメソッドの呼び出しへのステップイン、ステップアウト、ステップオーバ



■ **「デバッガ(Debugger)」ウインドウ。**より詳細なデバッグを実行する必要がある場合は、「デバッガ(Debugger)」ウインドウを利用すると、従来型のインターフェイスを使用して、テキストエディタが提供しているすべてのデバッグ機能を利用できます。このウインドウには、コールスタックとスコープ内の変数が一目でわかるリストが表示されます。



■ **GDBコンソール。**GDBコンソールウインドウは、テキストベースのデバッグに利用できます。

重要： デバイス上でうまくデバッグするには、デバイスにインストールしたiPhone OSに対応するiPhone SDKをコンピュータにインストールする必要があります。つまり、iPhone 2.2.1 SDKをコンピュータにインストールしていなければ、iPhone OS 2.2.1を実行するデバイス上でアプリケーションをデバッグできないということです。

Xcodeのデバッグ機能の詳細については、『*Xcode Debugging Guide*』を参照してください。

コンソールログとクラッシュログの表示

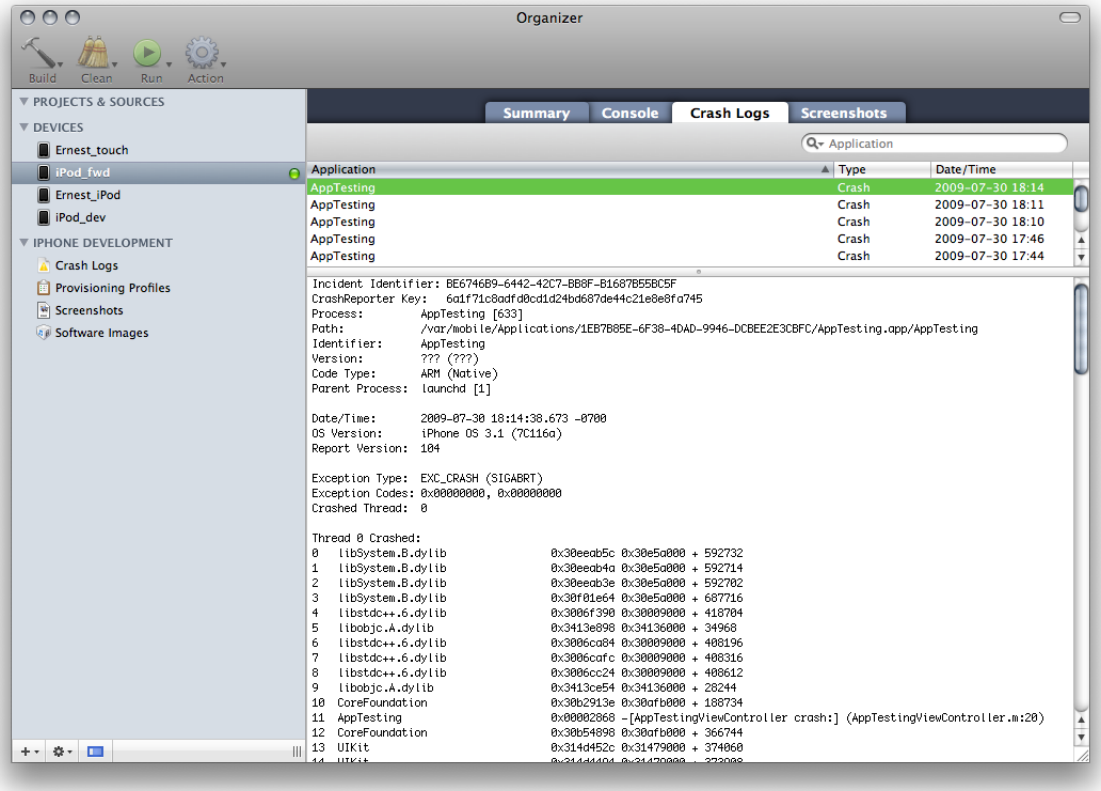
UIKitなどのiPhone OSフレームワークは、予期せぬイベントが発生すると、それを示すためにコンソールにログエントリを出力します。また、iPhoneアプリケーション内でコンソールメッセージを作成することもできます。コンソールログを作成する方法として、`NSLog`関数を使用する方法があります。Xcodeデバッガだけでなく、コンソールログも、アプリケーションのロジックを解析したり、バグを追跡するのに役立ちます。

iPhoneシミュレータでアプリケーションを実行している場合は、コンソールアプリケーション（/アプリケーション/ユーティリティにあります）でコンソールログにアクセスできます。開発用デバイスでアプリケーションを実行する場合、アプリケーションからのログエントリはXcodeの「オーガナイザ(Organizer)」に表示されます。

デバイスのコンソール出力を表示するには、次の手順を実行します。

1. 「オーガナイザ(Organizer)」ウインドウを開きます。
2. 表示したいコンソールログを持つデバイスを選択します。

3. 「Console」をクリックします。



「検索(Search)」フィールドを使用して、ログエントリをフィルタリングすることもできます。また、ログをファイルに保存することもできます。

「オーガナイザ(Organizer)」の「Crash Logs」ペインには、アプリケーションのクラッシュについての情報が含まれています。クラッシュリストをリフレッシュするには、デバイスを一度はずして再接続しなければならない場合があります。

メモリリークの検出

リークを修正した後にプログラムがクラッシュし始めた場合、コードで解放済みのオブジェクトやメモリバッファを使おうとしていることが考えられます。メモリリークの詳細については、『Memory Usage Performance Guidelines』の「Finding Memory Leaks」を参照してください。

NSZombieEnabled機能を使って、解放されたオブジェクトにアクセスするコードを検出できます。NSZombieEnabledが有効な場合、アプリケーションは割り当て解除されたメモリへのアクセスのログを記録します。以下に示します。

```
2008-10-03 18:10:39.933 HelloWorld[1026:20b] *** -[GSFont ascender]:message sent to
deallocated instance 0x126550
```

アプリケーションでNSZombieEnabled機能を有効にするには、次の手順を実行します。

1. 「プロジェクト(Project)」＞「アクティブな実行可能ファイルを編集(Edit Active Executable)」を選び、「実行可能ファイルの情報(Executable Info)」ウインドウを開きます。
2. 「引数(Arguments)」をクリックします。
3. 「環境に設定される変数(Variables to be set in the environment)」セクションの追加（+）ボタンをクリックします。
4. 「名前(Name)」カラムに`NSZombieEnabled`、「値(Value)」カラムに`YES`と入力します。
5. `NSZombieEnabled`エントリのチェックマークが選択されていることを確認します。

実行可能ファイルの環境設定の詳細については、『*Xcode Project Management Guide*』の「Configuring Executable Environments」を参照してください。

iPhone 3.1 SDKではLeaks Instrumentが導入されおり、Instrumentsアプリケーション内のこの機能を使って簡単にメモリリークを検出できます。詳細については、『*Instruments User Guide*』を参照してください。

アプリケーションの単体テスト

単体テストは、堅牢でセキュアなコードを書く際に役立ちます。Xcodeは使いやすく柔軟な単体テスト環境を提供しており、そのテスト環境を使うことによりコードに変更が重ねられてもそのコードを確実に設計どおりに動作するようにできます。

ソフトウェア要件： この機能はiPhone OS 3.0以降で利用できます。

この章では、単体テストの概要と、プロジェクト内での利用方法を解説します。

単体テストの採用および使用の事例については、次のサイトを参照してください。

http://www.macdevcenter.com/pub/a/mac/2004/04/23/ocunit.html

Xcodeは単体テスト用にあらかじめ設定されています。追加のソフトウェアをインストールする必要はありません。

単体テストの概要

単体テストにより、コードが示すべき動作を指定し、たとえばパフォーマンス向上やバグ修正のためにコードを変更しても、その機能性が変わらないようにすることができます。**テストケース**は特定の方法でコードを動作させます。その結果が期待される結果と異なる場合、テストケースは失敗します。**テストスイート**は、一連のテストケースから構成されます。1つまたは複数のテストスイートを開発して、コードのさまざまな側面をテストできます。

単体テストは、テスト駆動開発、すなわちコードよりも先にテストケースを書くスタイルをとる際の基本です。この開発アプローチでは、コードを書き始める前にコードの要件とエッジケースを体系化できます。テストケースを記述したら、テストケースに合格するためのアルゴリズムを開発します。コードがテストケースに合格したら、コードを改良するための土台ができ、期待される動作への（結果としてプロダクトにバグが生じる可能性のある）変更はすべて、次回のテスト実行時に特定できるようになります。

テスト駆動開発を使用していない場合でも、単体テストによってコードへのバグの混入を減らすことができます。現在作業中のアプリケーションに単体テストを組み込むことで、今後ソースコードを変更してもアプリケーションの動作は変わらないことを保証できます。バグを修正したときには、そのバグが修正されていることを確認するテストケースを追加できます。ただし、単体テストを考慮して設計されたものではないプロジェクトに単体テストを追加すると、テストのためにコードの一部を再設計したり、リファクタリングしたりする必要が生じる場合があります。

Xcodeの単体テスト環境は、オープンソースのSenTestingKitフレームワークに基づいています。このフレームワークは、テストスイートを設計し、それをコードに対して実行できるようにする一連のクラスとコマンドラインツールを提供しています。

Xcodeは、*ロジックテスト*と*アプリケーションテスト*の2種類の単体テストを提供します。

- **ロジックテスト。**このテストは、クリーンルーム環境（つまり、コードがアプリケーション内で実行されない状態）で、そのコードの正しい機能をチェックします。ロジックテストによって、非常に細かいレベル（クラスの1つのメソッド）で、またはワークフローの一部（複数のクラスのいくつかのメソッド）として、コードを動作させる特定のテストケースをまとめることができます。ロジックテストを使用してコードのストレステストを実行し、アプリケーションを実行する際にはありえないような極端な条件でコードが正しく動作することを保証することができます。ロジックテストは、予期しない使いかたをされたときでも正しく動作する堅牢なコードを作成するのに役立ちます。ロジックテストはiPhoneシミュレータSDKベースですが、アプリケーションはiPhoneシミュレータでは実行しません。テスト対象のコードは対応するターゲットのビルドフェーズの間に実行されます。
- **アプリケーションテスト。**このテストは、実行中のアプリケーション内のコードの機能をチェックします。アプリケーションテストを使用して、ユーザインターフェイスコントロールの接続（接続口とアクション）が適切であることと、コントロールとコントローラオブジェクトが、アプリケーションに適用したオブジェクトモデルに基づいて正しく動作することを保証できます。アプリケーションテストを実行するのはデバイス上だけであるため、このテストを使用して、デバイスの位置を取得するなどのハードウェアテストも実行できます。

テストのセットアップ

ロジック単体テスト（「単体テストの概要」（65 ページ）で紹介）を利用すると、網羅的で高度にカスタマイズされたテストを実行できます。ロジックテストを実行するには、iPhoneシミュレータSDKを使用して単体テスト(Unit Test)バンドルをビルドします。単体テストバンドルをビルドすると、Xcodeはそのバンドルに含まれるテストスイートを実行します。テストスイートをテストバンドルに組み込むには、そのバンドルに`SenTestCase`サブクラスを追加します。これらのクラスには、APIを呼び出して、期待した結果が得られたかをテストしてレポートするテストケースメソッドが含まれています。Xcodeは、テストが成功したか失敗したかをテキストエディタウインドウと「ビルド結果(Build Results)」ウインドウにレポートします。

アプリケーション単体テストによって、iPhone OSデバイス上で実行中のアプリケーション内のコードを、Cocoa Touchフレームワークで利用可能なリソースにアクセスしながらテストできます。

このセクションでは、それぞれの単体テスト用にプロジェクトをセットアップする方法を説明します。

ロジックテストのセットアップ

ロジック用の単体テストバンドルをプロジェクトにセットアップするには、次の手順を実行します。

1. iPhone OS単体テストバンドルターゲットをプロジェクトに追加します。このターゲットに`LogicTests`という名前を付けます（単体テストバンドルには任意の名前を使用できますが、テストであることが識別できるように*Tests*という接尾辞を付けるとよいでしょう）。

   プロジェクトへのターゲットの追加の詳細については、『*Xcode Build System Guide*』の「Creating Targets」を参照してください。

2. LogicTestsターゲットをアクティブターゲットに設定します。

3. 「Tests」というグループを「グループとファイル(Group & Files)」リストに追加し、リストからそのグループを選択します。

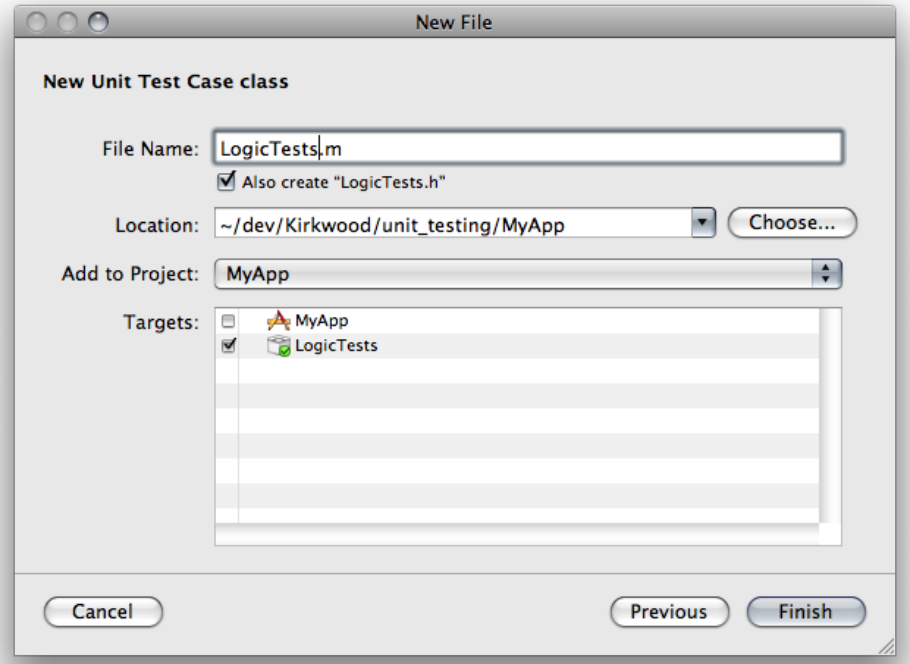
4. 単体テストバンドルターゲットに単体テストクラスを追加します。バンドル内の各単体テストクラスは、1つのテストスイートを構成します。

   a. 「ファイル(File)」＞「新規ファイル(New)」を選び、「iPhone OS Unit Test Case」テンプレートを選択して「次へ(Next)」をクリックします。

   b. このクラスに`LogicTests`という名前を付けます（ここでも任意の名前を使用できます）。

   c. ヘッダファイルを作成するオプションを選択します。

   d. LogicTestsがターゲットリストで選択されている唯一のターゲットであることを確認します。



5. `LogicTests.h`を次のように変更します。

```
#import <SenTestingKit/SenTestingKit.h>
#import <UIKit/UIKit.h>

@interface LogicTests :SenTestCase {
}
@end
```

6. `LogicTests.m`を次のように変更します。

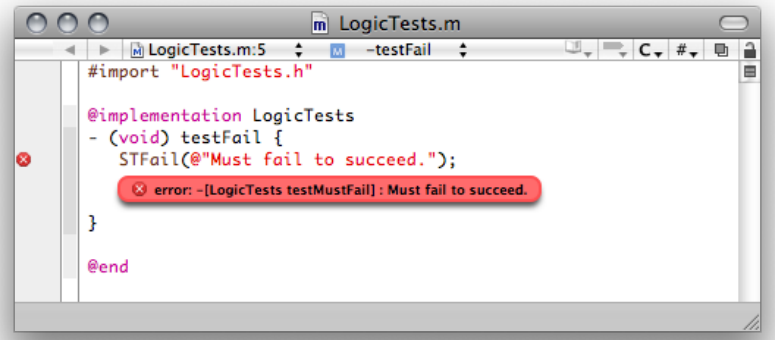
```
#import "LogicTests.h"

@implementation LogicTests
```

```
- (void) testFail {
      STFail(@"Must fail to succeed.");
}
@end
```

7. プロジェクトのベースSDKをiPhoneシミュレータ3.0以降に設定します。

   ベースSDKビルド設定の詳細については、「ベースSDKを設定する」（34 ページ）を参照してください。

8. アクティブターゲットをLogicTestsに設定し、「ビルド(Build)」＞「ビルド(Build)」を選びます。単体テストバンドルが正しく設定されている場合は、ビルドが失敗してテキストエディタにエラーメッセージが表示されます。



9. LogicTest.mのハイライトされている行を次のように変更し、テストケースが成功するようにします。

```
#import "LogicTests.h"

@implementation LogicTests
- (void) testPass {
      STAssertTrue(TRUE, @"");
}
@end
```

これで、ロジック単体テストバンドルは正しく設定されました。テストケースをバンドルに追加する方法については、「テストの記述」（73 ページ）を参照してください。

アプリケーションテストのセットアップ

アプリケーション用の単体テストバンドルをプロジェクトにセットアップするには、次の手順を実行します。

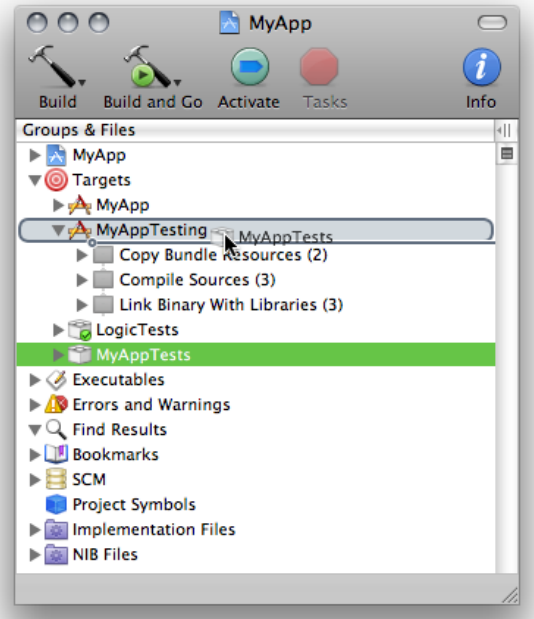
1. ショートカットメニューから「複製(Duplicate)」を選び、テスト対象のアプリケーションをビルドするためのターゲットをコピーします。それに<application_name>Testingという名前（たとえば、MyAppTesting）を付けます（このターゲットには任意の名前を使用できますが、アプ

リケーション単体テストの実行に使われるターゲットであることが識別できるように*Testing*という識別子を付けるとよいでしょう）。このターゲットの目的は、アプリケーション単体テストを実行することだけです。

2. iPhone OS単体テストバンドルターゲットをプロジェクトに追加します。このターゲットに`<application_name>Tests`という名前（たとえば、MyAppTests）を付けます。

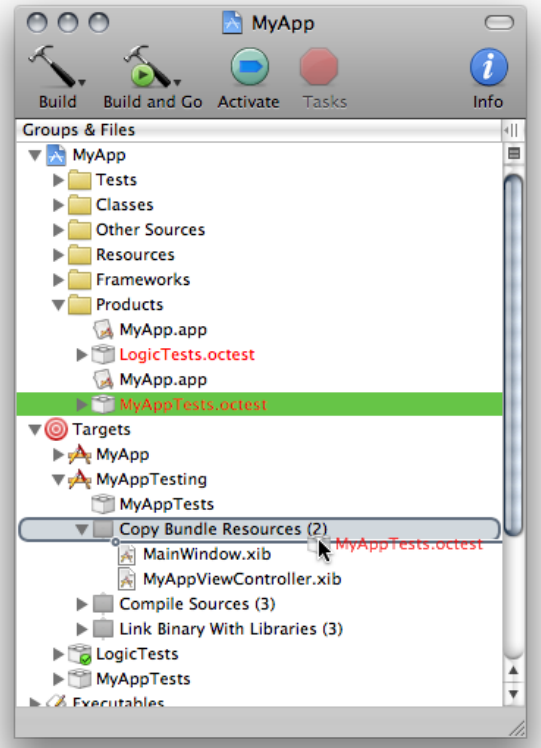
   プロジェクトへのターゲットの追加の詳細については、『*Xcode Build System Guide*』の「Creating Targets」を参照してください。

3. MyAppTestsをMyAppTestingにドラッグし、MyAppTestingターゲットをMyAppTestsターゲットに依存させます。



MyAppTestingターゲットをMyAppTestsターゲットに依存させることで、MyAppTestingをビルドしたときに、（最終ビルド以降にMyAppTestsを変更した場合は）確実に先にMyAppTestsがビルドされます。

4. `MyAppTests.octest`プロダクトをMyAppTestingターゲットのCopy Bundle Resourcesビルドフェーズにドラッグして、MyAppTestsバンドルをMyAppTestingバンドルに組み込みます。



5. 「Tests」というグループを「グループとファイル(Group & Files)」リストに追加し（そのグループがまだリストに存在していない場合）、リストからそのグループを選択します。

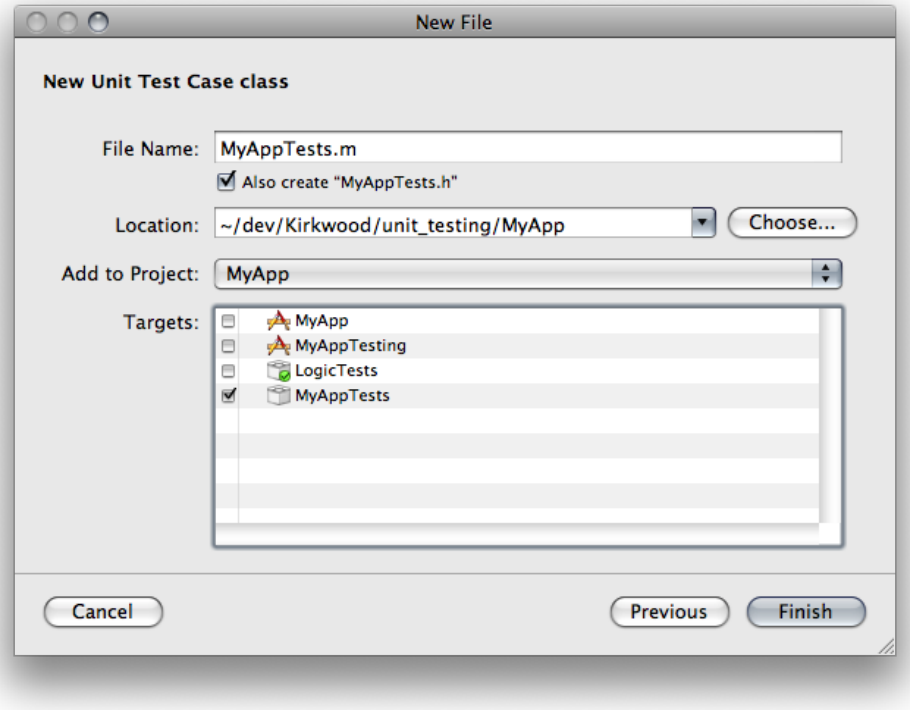
6. MyAppTestsターゲットに単体テストクラスを追加します。バンドル内の各単体テストクラスは、1つのテストスイートを構成します。

   a. 「ファイル(File)」＞「新規ファイル(New)」を選び、「iPhone OS Unit Test Case」テンプレートを選択して「次へ(Next)」をクリックします。

   b. このクラスに`MyAppTests`という名前を付けます（このクラスには任意の名前を使用できますが、識別しやすいように*Tests*という接尾辞を付けるとよいでしょう）。

   c. ヘッダファイルを作成するオプションを選択します。

d. MyAppTestsがターゲットリストで選択されている唯一のターゲットであることを確認します。



7. MyAppTests.hを次のように変更します。

```
#import <SenTestingKit/SenTestingKit.h>
#import <UIKit/UIKit.h>

@interface MyAppTests :SenTestCase {
}
@end
```

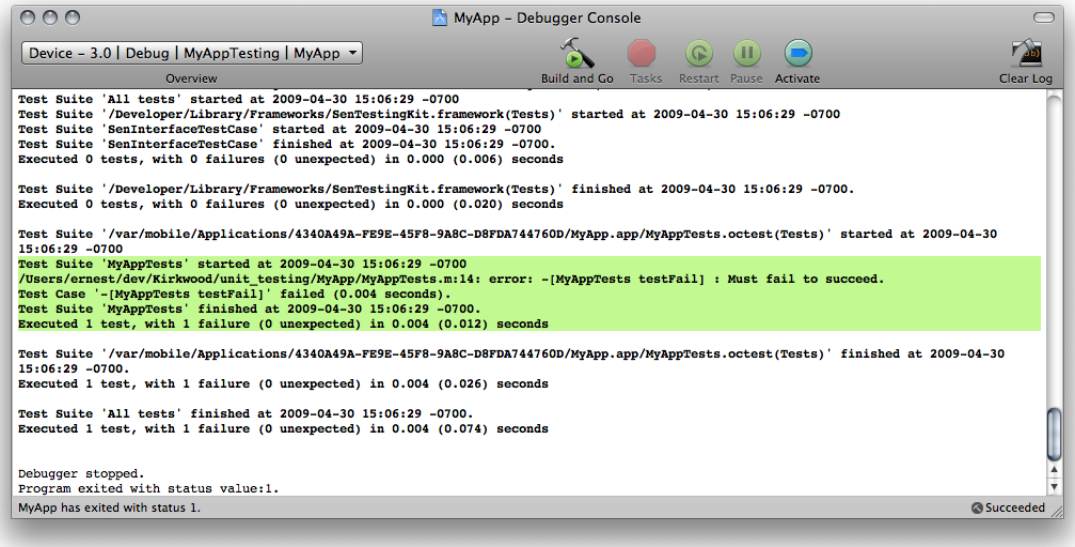
8. MyAppTests.mを次のように変更します。

```
#import "MyAppTests.h"

@implementation MyAppTests
- (void) testFail {
      STFail(@"Must fail to succeed.");
}
@end
```

9. プロジェクトのベースSDKをiPhoneデバイス3.0以降に設定します。

ベースSDKビルド設定の詳細については、「ベースSDKを設定する」（34 ページ）を参照してください。

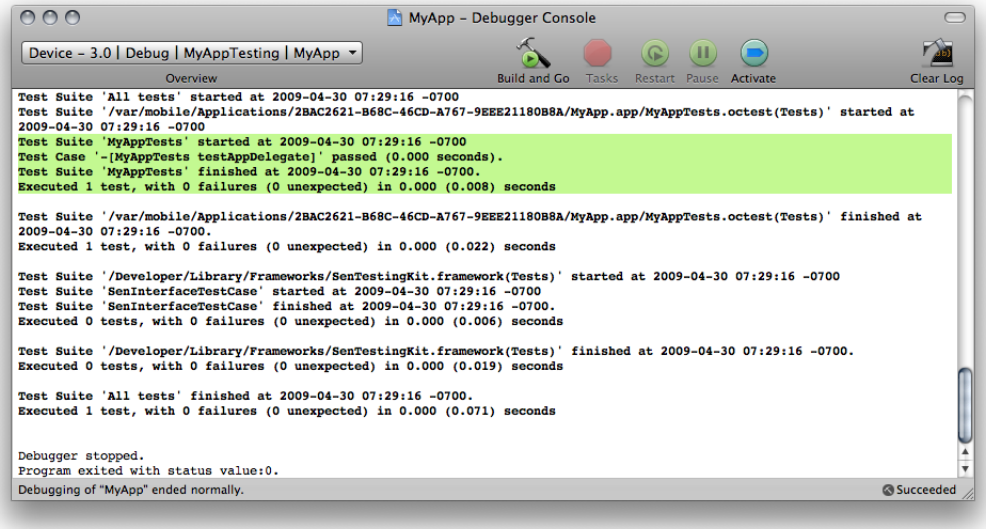
10. アクティブターゲットをMyAppTestingに設定し、「ビルド(Build)」＞「ビルドと実行(Build and Run)」を選びます。Xcodeはアプリケーションをビルドし、デバイスにインストールして起動します。次にテストスイートを実行しますがこれは失敗します。テスト結果はコンソールに表示されます。この出力によって、アプリケーション単体テストが正しくセットアップされていることを確認できます。



11. MyAppTests.mのハイライトされている行を次のように変更し、テストスイートが成功するようにします。

```
#import "MyAppTests.h"

@implementation LogicTests
- (void) testAppDelegate {
    id app_delegate = [[UIApplication sharedApplication] delegate];
    STAssertNotNil(app_delegate, @"Cannot find the application delegate.");
}
```

```
@end
```



これで、アプリケーション単体テストバンドルは正しく設定されました。テストケースをバンドルに追加する方法については「テストの記述」（73 ページ）を参照してください。

テストの記述

（テストスイートを実装する）単体テストクラスによって単体テストバンドルを構成したら、このクラスにテストケースメソッドを追加することによって、テストケースをテストスイートに追加します。**テストケースメソッド**は、`test...`という名前の単体テストクラスのインスタンスメソッドで、パラメータはなく、戻り値の型は`void`です。テストケースメソッドはAPIを呼び出して、そのAPIが期待どおりに動作するか（たとえば、期待した戻り値が戻るかどうか、例外が発生するかなど）をテストしてレポートします。テストケースメソッドは、期待された条件を確認したりその結果をレポートするために、一連のマクロを使用します。これらのマクロについては、「単体テスト結果用マクロのリファレンス」（97 ページ）で説明しています。

APIにアクセスするテストケースの場合は、適切な実装ファイルを単体テストバンドルに追加して、それに対応するヘッダファイルを単体テストクラスにインポートする必要があります。単体テストを使用するプロジェクトの例については、*iPhoneUnitTests*サンプルコードプロジェクトを参照してください。

> **注：** iPhoneの単体テストケースはObjective-Cで記述されます。

次に示すのはテストケースメソッドの構造体です。

```
- (void) test<test_case_name> {
    ...      // テストケースの対象のAPIをセットアップして呼び出す。
    ST...    // テストフレームワークに成功／失敗をレポートする。
    ...      // 破棄する。
```

```
}
```

各テストケースメソッドは個別に呼び出されます。したがって、各メソッドは、対象のAPIとやり取りするために必要な補助変数、構造体、オブジェクトのセットアップと破棄を行う必要があります。便利なことに、各テストケースメソッドの前後に呼び出されるsetUpとtearDownの1組のメソッドを単体テストクラスに追加できます。テストケースメソッドと同様に、この2つのメソッドの型はvoidで引数はありません。

setUp/tearDownメソッドペアの例を次に示します。

```
- (void) setUp {
   test_subject = [[MyClass new] retain];
   STAssertNotNil(test_subject, @"Could not create test subject.");
}

- (void) tearDown {
   [test_subject release];
}
```

注： setUpまたはtearDownの呼び出しの中で失敗がレポートされている場合、その失敗は最終的には呼び出し元のテストケースメソッドの中でレポートされます。

テストの実行

コードに変更を加えてもそのコードの正しい動作を変えないようにするには、定期的に（特に、重要な変更を加えた後は）テストスイートを実行する必要があります。このセクションでは、ロジックテストとアプリケーションテストを実行する方法を示します。また、ビルドするたびにロジックテストが実行されるように、アプリケーションターゲットをセットアップする方法も示します。

ロジックテストの実行

ロジックテストを実行するために必要な作業は、適切なロジックテストターゲットをビルドすることだけです。このようなターゲットをビルドするには、次の2つの方法があります。

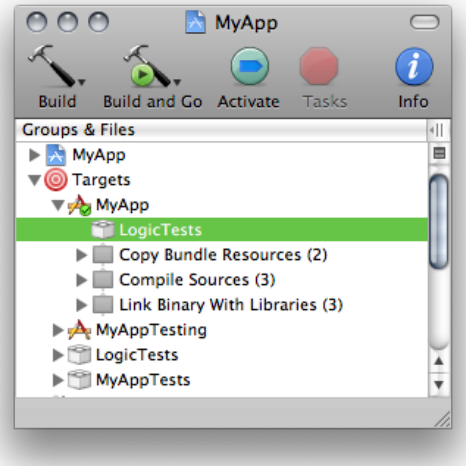
- ターゲットをアクティブにし、「ビルド(Build)」＞「ビルド(Build)」を選ぶ。
- ターゲットのショートカットメニューから「ビルド(Build)」を選ぶ。

「ビルド結果(Build Results)」ウインドウにテスト結果が表示されます。

アプリケーションをビルドするたびにロジックテストが実行されるようにXcodeを設定できます。このようにすると、ロジックテストを実行することを覚えておく必要がありません。

ビルドのたびにロジックテストが実行されるようにアプリケーションターゲットをセットアップするには、次の手順を実行します。

1. ロジックテストターゲットをアプリケーションターゲットにドラッグします。これによって、アプリケーションターゲットはロジックテストターゲットに依存するようになります。



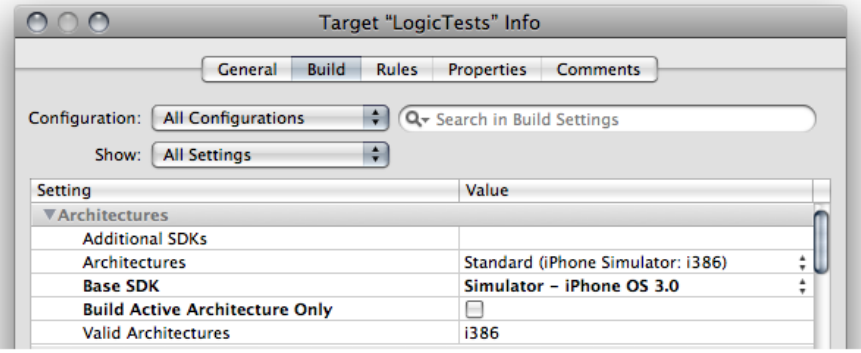
アプリケーションターゲットをビルドすると、Xcodeはまずロジックテストターゲットをビルドし、テストの失敗をレポートします。

2. LogicTestsの「情報(Info)」ウインドウに「ビルド(Build)」ペインが表示されます。

3. 「構成(Configuration)」ポップアップメニューから、「すべての構成(All Configurations)」を選びます。

4. 「表示(Show)」ポップアップメニューから、「すべての設定(All Settings)」を選びます。

5. ビルド設定リストの「アーキテクチャ(Architectures)」グループで、ベースSDKを「iPhone Simulator 3.0」に設定し、「アクティブアーキテクチャのみをビルド(Build Active Architecture Only)」を選択解除します。



アプリケーションテストの実行

アプリケーションテストを実行するには、アプリケーションテストターゲットをビルドして、実行またはデバッグする必要があります。このターゲットは、テストケースの実行またはデバッグのためにだけ使用できることを忘れないでください。デバイス上で対話的にアプリケーションを実行するために、このターゲットを使用することはできません。

コンソールにはアプリケーションテストの結果が表示されます。

テスト可能なコードの記述

Xcodeに組み込まれている単体テストのサポートにより、開発作業を任意の方法でサポートするテストスイートをビルドできます。これを使用して、コード中のリグレッションの可能性を検出したりアプリケーションの動作を検証したりできます。この機能によってプロジェクトに素晴らしい価値を付加できます。特に、個々のAPIを確実に期待どおりに動作させることによって、コードの安定性を大幅に向上させることができます。

もちろん、単体テストによってもたらされる安定性のレベルは、デベロッパが作成するテストケースの品質に大きく依存します。ここでは、テストしやすいコードを記述するために考慮すべきガイドラインを示します。

- **APIの要件を定義する。**プログラムに追加した各メソッドや関数の要件と結果を定義する必要があります。要件には入力と出力の範囲、発生する例外と例外が発生する条件、および戻り値の型（特に、戻り値がオブジェクトの場合）を含めます。要件を指定し、コード内でその要件が満たされていることを確認することは、堅牢でセキュアなコードを書くうえで役立ちます。

  例外を使用してクライアントによるAPIの不適切な使用を識別してレポートする例については、*iPhoneUnitTests*サンプルコードプロジェクトを参照してください。

- **コードを記述するときにテストケースを記述する。**APIを記述するときに、そのAPIの要件を確実に満たすようなテストケースを1つまたは複数記述します。既存のコードの単体テストを記述するのは、まだ書いていないコードや最近書いたコードの単体テストを記述するよりも困難です。

- **境界条件をチェックする。**メソッドのパラメータが特定の範囲の値を期待している場合は、テストではその範囲の最小値と最大値を含む値を渡すべきです。たとえば、整数パラメータが0から100までの値をとる場合は、0、50、100の値をそれぞれ渡す3つのパターンのテストを用意します。

- **ネガティブテストを使用する。**ネガティブテストでは、コードがエラー条件に適切に対応するようにします。無効または予期しない入力値を受け取った場合に、コードが適切に動作することを検証します。また、そのときにエラーコードを返したり、例外を発生させることを検証します。たとえば、整数パラメータが0から100までの値を受け付ける場合は、-1と101の値を渡すテストケースを作成して、APIが例外を発生させたり、エラーコードを返したりすることを確認します。

- **網羅的なテストケースを記述する。**網羅的なテストでは、APIのより複雑な動作を実装するためにさまざまなコードモジュールを結合します。単純で独立したテストは値を返しますが、結合されたテストは複雑な動作を行わせてより多くの問題点を検出しようとします。この種のテストは、より現実的な条件下でのコードの動作を模擬します。たとえば、オブジェクトを配列に追加するだけでなく、配列を作成していくつかのオブジェクトを追加し、別々のメソッドを使用していくつかのオブジェクトを削除します。そして残っているオブジェクトセットとその数が正しいことを確認します。

- **バグ修正をテストケースでカバーする。**バグを修正したら、修正結果を確認するテストケースを1つまたは複数記述します。

アプリケーションのチューニング

アプリケーションのパフォーマンスを最適化することは、開発プロセスにおける重要なフェーズであり、iPhone OSベースのデバイスでは特に重要です。強力なコンピューティングデバイスとはいえ、iPhone OSベースのデバイスは、デスクトップコンピュータやポータブルコンピュータのようなメモリやCPUパワーを備えていないからです。また、ユーザのバッテリー持続時間に直接影響を与えるため、アプリケーションのバッテリー使用にも注意を払う必要があります。

この章では、アプリケーションのパフォーマンスを測定したり調整するために使用するツールである、InstrumentsとSharkについて説明します。

お読みになる前に： この章のアプリケーションのチューニング手法をアプリケーションに試す前に、「アプリケーションの実行」（33 ページ）の手順に従ってください。

一般的なパフォーマンスのガイドラインについては、『*iOS Application Programming Guide*』を参照してください。

Instrumentsアプリケーション

Instrumentsアプリケーションを利用すると、メモリやネットワークの使用状況などのアプリケーションのパフォーマンス測定値を収集できます。iPhoneシミュレータまたは開発デバイス上で実行中のアプリケーションからデータを収集できます。

ユーザが満足できる体験を提供するには、iPhoneアプリケーションが、iPhone OSベースのデバイスのリソースをできる限り効率的に使用することが重要です。たとえば、ユーザに動作が遅いと感じさせたり、バッテリーがすぐに消費されてしまうと感じさせるようなリソースの使いかたをしてはいけません。メモリを使い過ぎるアプリケーションは、動作が遅くなります。動作するためにネットワークに依存するアプリケーションは、できる限りネットワークの使用頻度を下げる必要があります。ネットワーク通信用の無線を作動させると大量のバッテリーを消費するからです。

Instrumentsは高度なデータ収集インターフェイスを備えており、CPU、メモリ、ファイルシステムなど、アプリケーションによるリソースの使用状況を正確に把握できます。

Instrumentsは、ソフトウェアベースのデータ収集ツール（instrumentとも呼ばれる）を使用してパフォーマンスデータを収集します。**instrument**は、ネットワーク活動やメモリ使用状況など、特定の種類のデータを収集します。iPhone OSで利用可能なinstrumentは、Instruments Libraryで見つけることができます。

ほとんどのiPhoneアプリケーションはiPhoneシミュレータで動作し、設計に関する決定のほとんどはそこでテストできますが、シミュレータはデバイスをエミュレートしているわけではありません。特に、CPUの速度やメモリのスループットなど、デバイスのパフォーマンス特性を複製しているわけではありません。ユーザがデバイス上で使用する可能性のあるアプリケーションのパフォーマンスを効果的に測定するには、iPhoneまたはiPod touchを使用する必要があります。そのため、（プロセッサ速度、メモリ制限、特殊なハードウェアなどの点から見て）実際のデバイスの実行環境を正確に表すものを取得できるのは、デバイス上でのみになります。

iPhoneシミュレータにはいくつかの制限があります。

- **使用できる指の本数は最大で2本。**アプリケーションのユーザインターフェイスが3本以上の指によるタッチイベントに対応できる場合、その機能のテストはデバイスでのみ行えます。
- **加速度センサー。**UIKitフレームワークを介して、コンピュータの加速度センサーにアクセスできますが（加速度センサーがある場合）、その測定値と、デバイス上の加速度センサーの測定値は異なります。その差違は、コンピュータとiPhone OSベースのデバイスでは、ハードウェアの残りの部分に対する画面の位置が異なることが大きな原因です。
- **OpenGL ES。**OpenGL ESはデバイス上のレンダラを使用します。これは、iPhoneシミュレータで使用しているものとは若干異なります。このため、シミュレータ上の場面とデバイス上の同じ場面では、ピクセルレベルで同じでない場合があります。詳細については、『*iOS Application Programming Guide*』の「Drawing with OpenGL ES」を参照してください。

デバイス上でアプリケーションのパフォーマンスを測定するには、次の手順を実行します。

1. 「アプリケーションの実行」（33 ページ）の説明に従って、デバイス上でアプリケーションをビルドして実行します。
2. Instrumentsを起動します。

   Instrumentsアプリケーションは`<Xcode>/Applications`にあります（`<Xcode>`は開発ツールのインストール場所を示します）。
3. テンプレート（「Activity Monitor」など）を選んで、トレースドキュメントを作成します。

   トレースドキュメントには、1つのプロセスに関するデータを収集するinstrumentが1つ以上が含まれています。
4. ツールバーの「Default Target」メニューから、パフォーマンスデータの収集対象となるアプリケーションを含むiPhone OSベースのデバイスを選択します。
5. トレースドキュメントにinstrumentを追加または削除して、必要なデータを収集します。
6. 「Default Target」ポップアップメニューを使用して、ターゲットアプリケーションを起動したり、アプリケーションに接続したりします。
7. 「Record」をクリックしてデータ収集を開始し、アプリケーションを使用して調査したい部分を動作させます。

アプリケーションパフォーマンスの測定および分析の詳細については、『*Instruments User Guide*』を参照してください。同文書では、Instrumentsの使用に関する一般情報が記載されています。

Sharkアプリケーション

Instrumentsで収集したパフォーマンスデータを補うために、Sharkアプリケーションを利用してシステムコール、スレッドのスケジューリング、割り込み、仮想メモリフォルトなどのシステムレベルのイベントを表示できます。コードのスレッドが相互にどのようにやり取りしているか、またアプリケーションがiPhone OSとどのようにやり取りしているかを参照できます。

コード内のパフォーマンスの問題が、コード、iPhone OS、およびデバイスのハードウェアアーキテクチャ間のやり取りに関係する場合は、Sharkを使用してこれらのやり取りについての情報を収集し、パフォーマンスのボトルネックを発見できます。

iPhoneアプリケーションでのSharkの使用法については、『*Shark User Guide*』を参照してください。

アプリケーションの配布

App Storeを通して、テスト用または一般配布用にアプリケーションを配布する準備が整ったら、Distributionプロビジョニングプロファイルを使用してアプリケーションのアーカイブを作成し、アプリケーションのテスターに送付するかiTunes Connectに投稿する必要があります。この章ではこれらの作業の進めかたを説明します。

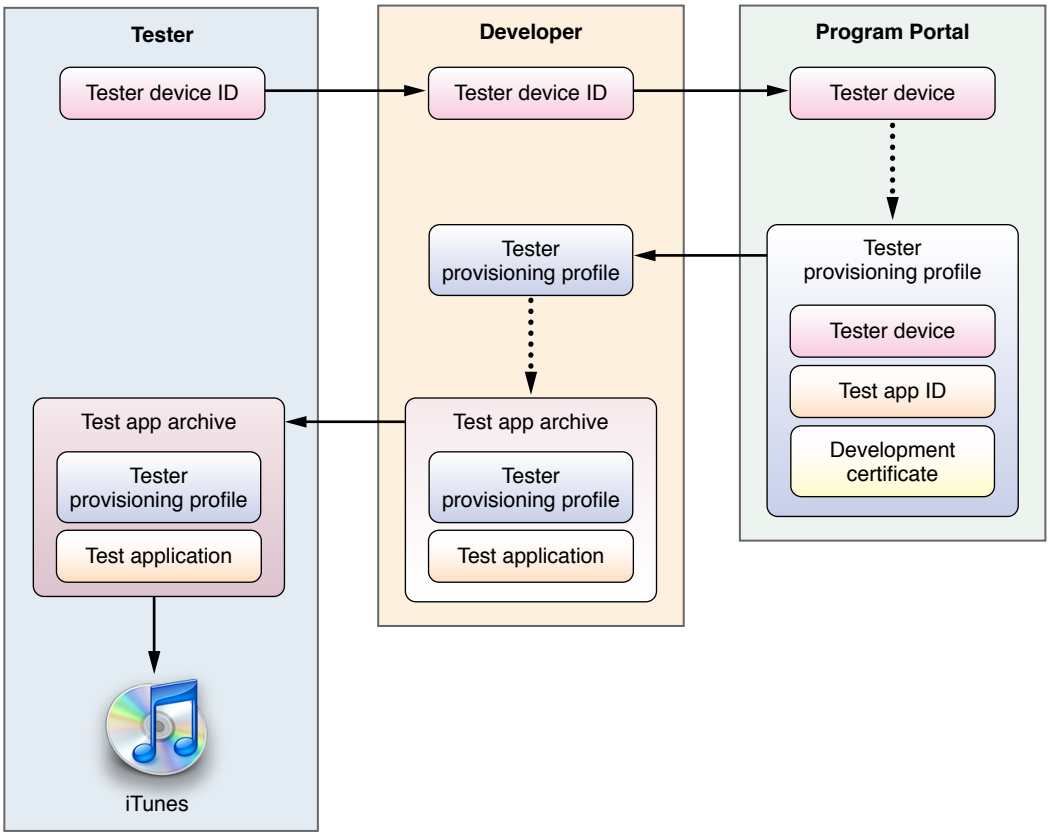
テストのためのアプリケーションの公開

自分自身やチームメイトの支援によって、アプリケーションのテストとチューニングが終了したら、アプリケーションを使用する可能性がある代表的なサンプルユーザに、より広範囲なテストを実施してもらうのが良いでしょう。このようなテストによって、特定の使用パターンでのみ発生する問題が明らかになる場合があります。デベロッパでないユーザがテスト計画に参加することによって、アプリケーションをさまざまな使用形態に触れさせることができ、そうした使用形態によってアプリケーションがクラッシュした場合に、テストに参加したユーザからクラッシュレポート（クラッシュログとも呼ばれる）を収集して、実行時の問題の解決に役立てることができます。

開発中のiPhoneアプリケーションは、アプリケーションデベロッパによって生成されたプロビジョニングプロファイルを持つデバイスでのみ実行できます。iPhoneデベロッパプログラムのメンバであれば、自身とそのチームメンバは、開発プロセスの一部としてプロビジョニングプロファイルをデバイスにインストールします（「デバイスとデジタルIDの管理」で説明しています）。チームメンバ以外のユーザ（テスターと呼ぶ）をテスト計画に追加するには、iPhoneプロビジョニングポータルでユーザをチームの一員として追加して、**テストプロビジョニングプロファイル**（暫定プロビジョニングプロファイルとも呼ばれる）を発行する必要があります。これによって、チームに属していないユーザもApp Storeにまだ公開されていないアプリケーションをデバイスにインストールできます。

図 9-1は、テスターとしてユーザを追加して、テストアプリケーションを配布するプロセスを示しています。

図 9-1 チームへのテスターの追加



お読みになる前に：

- テスターにアプリケーションを送信する前に、「アプリケーションの実行」（33 ページ）の情報を参照してください。
- チームにテスターを追加するには、ポータルで配布の証明書を取得する必要があります。

テスト計画にテスターを追加するために必要な情報をテスターに入手してもらい、テスターにクラッシュログの送信方法を知らせるには、「アプリケーションのテスター向けの説明書」（87 ページ）の情報をテスターに送信します。

重要： 自分のチームにテスターを追加するには、自身がiPhoneデベロッパプログラムのメンバである必要があります。詳細については、「デバイスとデジタルIDの管理」（51 ページ）を参照してください。

この章の残りのセクションでは、自分のチームにテスターを追加する手順を説明し、テスターから送られたクラッシュログを「オーガナイザ(Organizer)」にインポートする方法を示します。

チームへのアプリケーションのテスターの追加

iPhone OSユーザをテスターとしてチームに追加するには、以下の手順を実行します。

1. テスターのデバイスIDを取得します。

   この情報を得るには、電子メールを利用するのが最も簡単な方法です。「デベロッパへのデバイスIDの送信」（88 ページ）に説明されている、デバイスIDを送信する手順をテスターに実行してもらいます。

2. テスターのデバイスIDをポータルに追加します。

   ヒント： テスターの電子メールアドレスをデバイス名として使用します。

3. ポータルで、すでにアプリケーションのテスト用プロビジョニングプロファイルがあれば、テスターのデバイスIDをそこに追加します。そうでなければ、次の特性でプロファイルを作成します。

配布メソッド	暫定
プロファイル名	<アプリケーション名> テスト用プロファイル
アプリケーションID	テスト対象アプリケーションの適切なアプリケーションID
デバイス	テスターのデバイスID

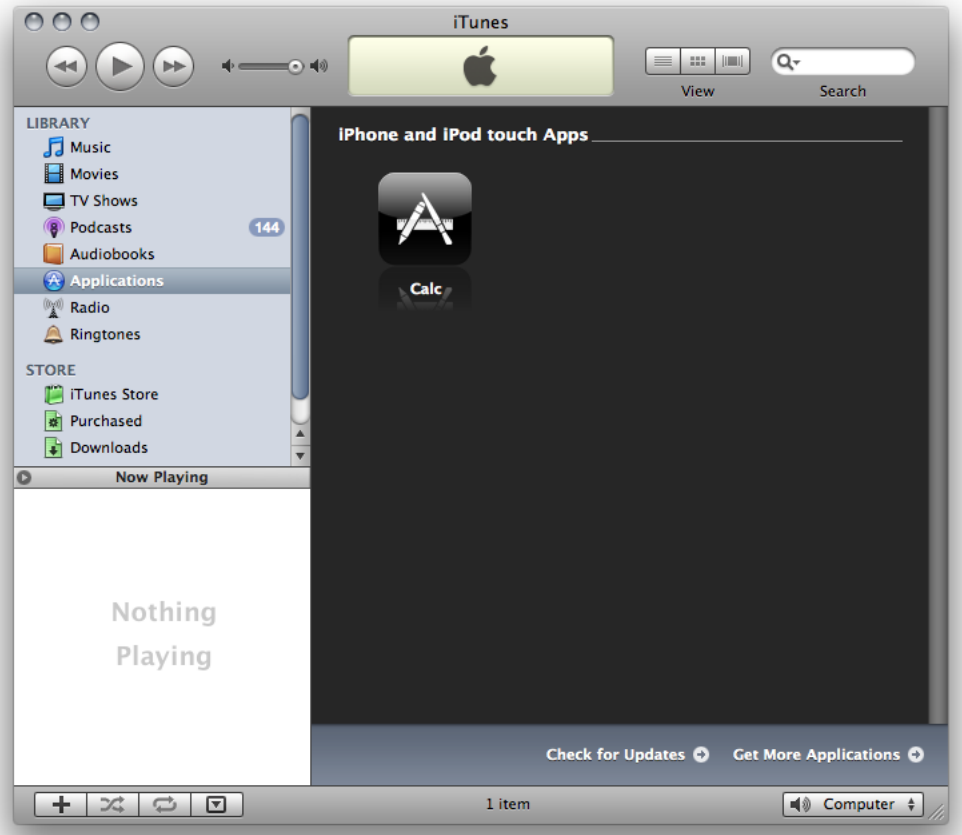
4. テスト用プロファイルをダウンロードし、Xcodeの「オーガナイザ(Organizer)」にインストールします。

   <プロファイル名>.mobileprovisionファイルを、「IPHONE DEVELOPMENT」下の「Provisioning Profiles」のリストにドラッグします。

アプリケーションへのiTunesアートワークの追加

テスト用アプリケーションには、iTunesがアプリケーションの識別に使用するアートワークを含める必要があります。そうしないと、ユーザがアプリケーションをiTunesライブラリに追加したときに、iTunesは、図 9-2に示すような汎用のアートワークを使用します。

図 9-2 テストアプリケーション用の汎用のiTunesアートワーク



テスターに表示されたiTunesアートワークが、アプリケーションのアイコンになります。このアートワークは、`iTunesArtwork`という名前の、512×512のJPEGまたはPNGのファイルでなければなりません。このファイルには拡張子を付けないでください。

アプリケーションのアイコンのファイルを生成したら、次の手順に従ってこれをアプリケーションに追加します。

1. Xcodeでプロジェクトを開きます。
2. 「グループとファイル(Groups & Files)」リストで、「Resources」グループを選択します。
3. 「プロジェクト(Project)」＞「プロジェクトに追加(Add to Project)」を選び、`iTunesArtwork`ファイルに移動して、「追加(Add)」をクリックします。
4. 表示されたダイアログで、「項目をコピーする(Copy items)」オプションを選択して、「追加(Add)」をクリックします。

テストのためのアプリケーションのアーカイブ

テスターへの配布用にアプリケーションをアーカイブするには、次の手順に従います。

1. 「コード署名ID (Code Signing Identity)」ビルド設定をテストプロビジョニングプロファイルに設定します。
2. プロジェクトウインドウの「概要(Overview)」ツールバーメニューで、「アクティブな実行可能ファイル(Active Executable)」をデバイスに設定します。
3. 「ビルド(Build)」＞「ビルドとアーカイブ(Build and Archive)」を選びます。

注： このプロセスを効率化するには、（Release構成を複製して）Distribution構成を作成し、その構成内の「コード署名ID (Code Signing Identity)」ビルド設定を変更します。こうすることで、Development構成とDistribution構成を別々にしておくことができます。ビルド構成の詳細については、「ビルド構成」を参照してください。

テスターへのアプリケーションの送信

テスターにアプリケーションを送信するには、次の手順を実行します。

1. 「オーガナイザ(Organizer)」の「Archived Applications」リストで、テスターに送信するアプリケーションのアーカイブを選択し、「Share Application」をクリックします。
2. 「E-Mail」をクリックします。

   電子メールメッセージで、アプリケーションのテストに必要な情報をテスターに提供します。

テスターからのクラッシュログのインポート

シンボル情報を表示するために「オーガナイザ(Organizer)」にテスターのクラッシュログを追加するには、クラッシュログを「IPHONE DEVELOPMENT」セクションの「Crash Logs」グループへドラッグします。

重要： Xcodeでクラッシュログをシンボル化する（使用されたAPIに関する情報をクラッシュログに追加するため）には、Spotlightはアーカイブしたアプリケーションとそれらに対応するdSMYファイルが保存されているボリュームの索引付けを行う必要があります。

アプリケーションのテスター向けの説明書

このセクションは、テスターが各自のデバイス上でiPhoneアプリケーションをテストする手順についての、テスター向けの説明書です。アプリケーションのテスターは、App Storeでリリースされる前にアプリケーションをテストしたいと考える潜在ユーザです。

自分のアプリケーションのテストに興味を持ってくれたユーザに、アプリケーションのテストに必要な特殊な作業と一緒に、これらの説明書を送信できます。

デベロッパへのデバイスIDの送信

テスト用アプリケーションをテスターに送信する前に、デベロッパはAppleのアプリケーションテストプログラムにテスターのデバイスを登録する必要があります。

テストプログラムの登録のために、デベロッパにデバイスIDを送信するには、次の手順を実行します。

1. iTunesを起動します。
2. デバイスをコンピュータに接続します。
3. 「デバイス(Devices)」リストからそのデバイスを選択します。
4. 「概要(Summary)」ペインで、「シリアル番号(Serial Number)」ラベルをクリックします。ラベルが「識別子(Identifier)」に変わります。
5. 「編集(Edit)」＞「コピー(Copy)」を選びます。
6. デバイス識別子をデベロッパに電子メールで送信します。電子メールには、忘れずに自分の名前とデバイス名を含めます。

テスト用アプリケーションのインストール

デベロッパのテストプログラムへ登録したら、デベロッパはテスターにテストアプリケーションのアーカイブを送信します。デバイス上でアプリケーションを実行するには、アーカイブをiTunesにインストールする必要があります。

テストアプリケーションをデバイスにインストールするには、以下の手順に従います。

1. Finderでアプリケーションのアーカイブ(<アプリケーション名>.ipa)をダブルクリックします。

   iTunesの「アプリケーション(Applications)」リストにアプリケーションが表示されます。
2. デバイスを同期します。

   デバイスのiPhone OSのバージョンが、テストアプリケーションを実行できるバージョンより古い場合は、最新バージョンのiPhone OSでデバイスを更新する必要があります。

デベロッパへのクラッシュレポートの送信

テスト中のアプリケーションがクラッシュした場合、iPhone OSはそのイベントのレコードを作成します。次回デバイスをiTunesに接続したときに、iTunesはそれらのレコード（クラッシュログと呼ばれる）をコンピュータにダウンロードします。問題の解決を支援するために、テスト中のアプリケーションのクラッシュログをデベロッパに送信する必要があります。

Macからのクラッシュレポートの送信

クラッシュレポートをデベロッパに送信するには、次の手順を実行します。

1. Finderで、新しいウインドウを開きます。

2. 「移動」＞「フォルダへ移動」を選びます。
3. ~/ライブラリ/Logs/CrashReporter/MobileDeviceと入力します。
4. デバイス名から始まる名前のフォルダを開きます。
5. テスト中のアプリケーションから始まる名前を持つクラッシュログを選択します。
6. 「Finder」＞「サービス」＞「Mail」＞「ファイルを送信」を選びます。
7. 「新規メッセージ」ウインドウで、「宛先」フィールドにデベロッパの電子メールアドレスを入力し、「件名」フィールドに<アプリケーション名> crash logs from <ユーザ名>（たとえば、「MyTestApp crash logs from Anna Haro」）を入力します。
8. 「メッセージ」＞「送信」を選びます。
9. 同じレポートの重複送信を避けるために、Finderで送信済みのクラッシュログを削除します。

Windowsからのクラッシュレポートの送信

デベロッパにクラッシュレポートを送信するには、Windowsの「検索」フィールドにクラッシュログのディレクトリ（リスト 9-1またはリスト 9-2）を入力します（<ユーザ名>の部分は自分のWindowsユーザ名に置き換えます）。

リスト 9-1 Windows Vistaの場合のクラッシュログの格納場所

```
C:\Users\<ユーザ名>\AppData\Roaming\Apple computer\Logs\CrashReporter/MobileDevice
```

リスト 9-2 Windows XPの場合のクラッシュログの格納場所

```
C:\Documents and Settings\<user_name>\Application Data\Apple
computer\Logs\CrashReporter
```

デバイス名で始まる名前のフォルダを開き、テスト中のアプリケーションのクラッシュログをアプリケーションのデベロッパにメールで送信します。その際、件名の形式を<アプリケーション名> crash logs from <自分の名前>（たとえば、「MyTestApp crash logs from Anna Haro」）にします。

App Store配布のためのアプリケーションの公開

App Storeを通して一般配布用にアプリケーションを公開する準備が整ったら、iTunes Connectにアプリケーションを投稿します。このセクションでは、投稿用にアプリケーションを準備する方法、およびiTunes Connectにアプリケーションを投稿する方法について説明します。

アプリケーション用のDistributionプロファイルの作成

アプリケーションのDistributionプロファイルを作成するには、次の手順に従います。

1. 次の特性でポータルにDistributionプロビジョニングプロファイルを作成します。

配布メソッド	App Store
プロファイル名	<アプリケーション名> Distributionプロファイル
アプリケーションID	アプリケーションに適したアプリケーションID

2. Distributionプロファイルをダウンロードし、Xcodeの「オーガナイザ(Organizer)」にインストールします。

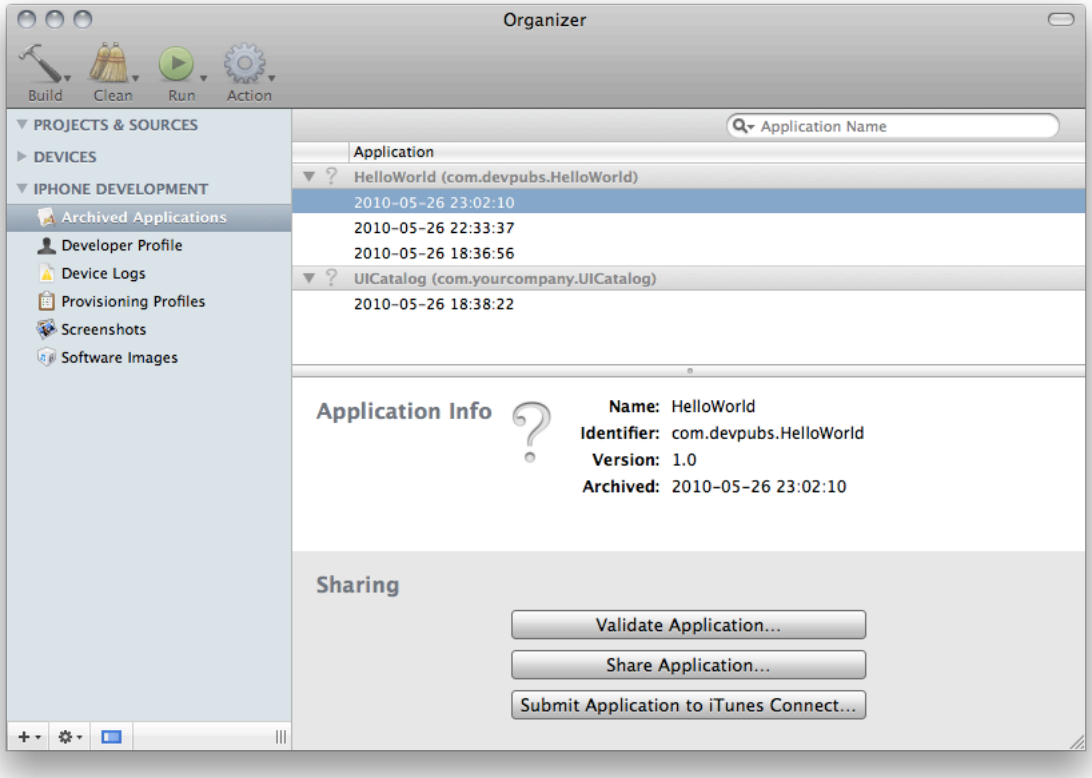
   <プロファイル名>.mobileprovisionファイルを、「オーガナイザ(Organizer)」の「IPHONE DEVELOPMENT」下の「Provisioning Profiles」リストにドラッグします。

iTunes Connectへの投稿のためのアプリケーションのアーカイブ

アプリケーションの配布用アーカイブを作成するには、次の手順に従います。

1. 「コード署名ID (Code Signing Identity)」ビルド設定をDistributionプロファイルに設定します。
2. プロジェクトウインドウの「概要(Overview)」ツールバーメニューで、「アクティブな実行可能ファイル(Active Executable)」をデバイスに設定します。
3. 「ビルド(Build)」＞「ビルドとアーカイブ(Build and Archive)」を選びます。

「オーガナイザ(Organizer)」の「Archived Applications」に新しいアプリケーションのアーカイブが表示されます。アーカイブはそれぞれ、作成された日付と時間で識別されます。



iTunes Connectへのアプリケーションの投稿

iTunes Connectへアプリケーションを投稿するには、次の手順に従います。

1. 「オーガナイザ(Organizer)」の「Archived Applications」リストで、テスターに送信するアプリケーションのアーカイブを選択し、「Submit Application to iTunes Connect」をクリックします。
2. 表示されるダイアログで、iTunes Connectの証明書を入力し、「Submit」をクリックします。

iPhone開発に関するFAQ

以下は、iPhoneの開発についてデベロッパから寄せられるよくある質問です。

- App Storeを通してアプリケーションを配布するために、iTunes Connectにはどのように投稿すれば良いですか。

  「ビルドとアーカイブ(Build and Archive)」コマンドを使用します。詳細については「アプリケーションの配布」（83 ページ）を参照してください。

- iPhoneシミュレータアプリケーションは、ネットワークホームディレクトリで動作しますか。

  いいえ。

- Objective-C言語のプロパティを利用できるようにするには、インスタンス変数またはアクセサメソッドで支える必要がありますか。

  はい。

- スタティックライブラリは、iPhoneアプリケーションで使用する前にコード署名する必要がありますか。

  いいえ。

- なぜアプリケーションでPNGファイルの処理に問題が発生するのですか。

  PNGファイルを使おうとしているコードがPNGの圧縮ファイルを理解できていない可能性があります。

  「Compress PNG Files」ビルド設定をオフにしてください。ビルド設定の詳細については、『*Xcode Project Management Guide*』の「Editing Build Settings」を参照してください。

- WindowsでiPhoneアプリケーションを開発できますか。

  いいえ。iPhoneアプリケーションを開発できるのは、Mac OS Xのみです。

- スタティックライブラリのすべてのObjective-Cクラスをどのようにリンクすれは良いですか。

  「他のリンカフラグ(Other Linker Flags)」ビルド設定を`-ObjC`に設定します。このようにしてもすべてのクラスをリンクしない場合は、`-all_load`に設定します。

- 廃止されたAPIはいつ置き換えるべきですか。

  アプリケーションを実行させるiPhone OSバージョンを検討し、できる限り迅速にアップデートします。詳細については、『*SDK Compatibility Guide*』を参照してください。

- iPhoneシミュレータでコンピュータのカメラを使用できますか。

  いいえ。

- iPhoneの開発に必要な最低限のハードウェア要件は何ですか。

  Intelプロセッサを搭載したMacです。

Hello, World!ソースコード

この付録では「チュートリアル：Hello, World!」（20 ページ）で説明したHello, World!アプリケーションのソースコードを示します。

リスト A-1 main.m

```
// main.m
#import <UIKit/UIKit.h>

int main(int argc, char *argv[]) {

   NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
   int retVal = UIApplicationMain(argc, argv, nil, nil);
   [pool release];
   return retVal;
}
```

リスト A-2 HelloWorldAppDelegate.h

```
// HelloWorldAppDelegate.h
#import <UIKit/UIKit.h>

@interface HelloWorldAppDelegate :NSObject <UIApplicationDelegate> {
    UIWindow *window;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;

@end
```

リスト A-3 HelloWorldAppDelegate.m

```
// HelloWorldAppDelegate.m
#import "HelloWorldAppDelegate.h"
#import "MyView.h"

@implementation HelloWorldAppDelegate

@synthesize window;

- (void)applicationDidFinishLaunching:(UIApplication *)application {

   // アプリケーション起動後のカスタマイズ用、オーバーライドポイント
   MyView *view = [[MyView alloc] initWithFrame:[window frame]];
   [window addSubview:view];
   [view release];

   [window makeKeyAndVisible];
}
```

```
- (void)dealloc {
    [window release];
    [super dealloc];
}

@end
```

リスト A-4　MyView.h

```
// MyView.h
#import <UIKit/UIKit.h>

@interface MyView :UIView {
}

@end
```

リスト A-5　MyView.m

```
// MyView.m
#import "MyView.h"

@implementation MyView

- (id)initWithFrame:(CGRect)frame {
    if (self = [super initWithFrame:frame]) {
        // 初期化コード
    }
    return self;
}

- (void)drawRect:(CGRect)rect {
   NSString *hello   = @"Hello, World!";
   CGPoint  location = CGPointMake(10, 20);
   UIFont   *font    = [UIFont systemFontOfSize:24];
   [[UIColor whiteColor] set];
   [hello drawAtPoint:location withFont:font];
}

- (void)dealloc {
    [super dealloc];
}

@end
```

単体テスト結果用マクロのリファレンス

SenTestingKitフレームワークには一連のテストケース結果用マクロが定義されています。これを利用するとテストケース結果をフレームワークにレポートできます。テストが失敗するとこのフレームワークはテスト失敗のメッセージを発行します。このメッセージは、実行中の単体テストの種類に応じて、Xcodeのテキストエディタ、「ビルド結果(Build Results)」ウインドウ、またはコンソールウインドウに表示されます。

以降のセクションでは、テストケースメソッド内で使用できるテスト結果用マクロについて説明します。これらのマクロは`SenTestCase.h`で宣言されています。

重要： テストケースで評価した式によって例外が発生した場合は、未知のエラーが発生します。コードが例外を発生するかどうかを確認するには、例外の有無や例外を明示的にチェックするテストケースを作成します（「例外テスト」（101 ページ）を参照）。

無条件の失敗

STFail

テストケースを失敗させます。

```
STFail(failure_description, ...)
```

パラメータ

`failure_description`

エラーメッセージを指定する書式文字列。`nil`を渡すこともできます。

`...`

（オプション）`failure_description`の代わりにコンマ区切りの引数リストを渡すこともできます。

等価テスト

STAssertEqualObjects

2つのオブジェクトが異なる場合、このテストケースは失敗します。

```
STAssertEqualObjects(object_1, object_2, failure_description, ...)
```

パラメータ

`object_1`

オブジェクト。

`object_2`

オブジェクト。

`failure_description`

エラーメッセージを指定する書式文字列。`nil`を渡すこともできます。

`...`

（オプション）`failure_description`の代わりにコンマ区切りの引数リストを渡すこともできます。

詳細

`[object_1 isEqualTo:object_2]`がFalseの場合、このテストは失敗します。

STAssertEquals

2つの値が異なる場合、このテストケースは失敗します。

```
STAssertEquals(value_1, value_2, failure_description, ...)
```

パラメータ

`value_1`

スカラ、構造体、共用体。

`value_2`

スカラ、構造体、共用体。

`failure_description`

エラーメッセージを指定する書式文字列。`nil`を渡すこともできます。

`...`

（オプション）`failure_description`の代わりにコンマ区切りの引数リストを渡すこともできます。

詳細

`value_1`と`value_2`が等しくない場合、このテストは失敗します。

STAssertEqualsWithAccuracy

2つの値の差が指定された値より大きい場合、このテストケースは失敗します。

```
STAssertEqualsWithAccuracy(value_1, value_2, accuracy, failure_description, ...)
```

パラメータ

`value_1`

整数値または浮動小数点数値。

`value_2`

整数値または浮動小数点数値。

`accuracy`

整数値または浮動小数点数値。

`failure_description`

エラーメッセージを指定する書式文字列。`nil`を渡すこともできます。

`...`

（オプション）`failure_description`の代わりにコンマ区切りの引数リストを渡すこともできます。

詳細

`value_1`と`value_2`の差が`accuracy`より大きい場合、このテストは失敗します。

Nilテスト

STAssertNil

指定された式がnilでない場合、このテストケースは失敗します。

```
STAssertNil(expression, failure_description, ...)
```

パラメータ

`expression`

テスト対象の式。

`failure_description`

エラーメッセージを指定する書式文字列。`nil`を渡すこともできます。

`...`

（オプション）`failure_description`の代わりにコンマ区切りの引数リストを渡すこともできます。

STAssertNotNil

指定された式がnilの場合、このテストケースは失敗します。

```
STAssertNotNil(expression, failure_description, ...)
```

パラメータ

`expression`

テスト対象の式。

`failure_description`

エラーメッセージを指定する書式文字列。`nil`を渡すこともできます。

`...`

（オプション）`failure_description`の代わりにコンマ区切りの引数リストを渡すこともできます。

ブールテスト

STAssertTrue

指定された式がFalseの場合、このテストケースは失敗します。

```
STAssertTrue(expression, failure_description, ...)
```

パラメータ

`expression`

テスト対象の式。

`failure_description`

エラーメッセージを指定する書式文字列。`nil`を渡すこともできます。

`...`

（オプション）`failure_description`の代わりにコンマ区切りの引数リストを渡すこともできます。

STAssertFalse

指定された式がTrueの場合、このテストケースは失敗します。

```
STAssertFalse(expression, failure_description, ...)
```

パラメータ

`expression`

テスト対象の式。

`failure_description`

エラーメッセージを指定する書式文字列。`nil`を渡すこともできます。

`...`

（オプション）`failure_description`の代わりにコンマ区切りの引数リストを渡すこともできます。

例外テスト

STAssertThrows

式が例外を発生させない場合、このテストケースは失敗します。

```
STAssertThrows(expression, failure_description, ...)
```

パラメータ

`expression`

テスト対象の式。

`failure_description`

エラーメッセージを指定する書式文字列。`nil`を渡すこともできます。

`...`

（オプション）`failure_description`の代わりにコンマ区切りの引数リストを渡すこともできます。

STAssertThrowsSpecific

式が特定のクラスの例外を発生させない場合、このテストケースは失敗します。

```
STAssertThrowsSpecific(expression, exception_class, failure_description, ...)
```

パラメータ

`expression`

テスト対象の式。

`exception_class`

例外クラス

`failure_description`

エラーメッセージを指定する書式文字列。`nil`を渡すこともできます。

`...`

（オプション）`failure_description`の代わりにコンマ区切りの引数リストを渡すこともできます。

詳細

`expression`が`exception_class`クラスの例外を発生させない場合、このテストは失敗します。

STAssertThrowsSpecificNamed

式が、指定された名前の特定のクラスの例外を発生させる場合、このテストケースは失敗します。

```
STAssertThrowsSpecificNamed(expression, exception_class, exception_name,
failure_description, ...)
```

パラメータ

`expression`

テスト対象の式。

`exception_class`

例外クラス

`exception_name`

例外の名前を表す文字列。

`failure_description`

エラーメッセージを指定する書式文字列。`nil`を渡すこともできます。

`...`

（オプション）`failure_description`の代わりにコンマ区切りの引数リストを渡すこともできます。

詳細

expressionがexception_nameという名前のexception_classクラスの例外を発生させない場合、このテストは失敗します。

STAssertNoThrow

式が例外を発生させる場合、このテストケースは失敗します。

```
STAssertNoThrow(expression, failure_description, ...)
```

パラメータ

expression

テスト対象の式。

failure_description

エラーメッセージを指定する書式文字列。nilを渡すこともできます。

...

（オプション）failure_descriptionの代わりにコンマ区切りの引数リストを渡すこともできます。

STAssertNoThrowSpecific

式が特定のクラスの例外を発生させる場合、このテストケースは失敗します。

```
STAssertNoThrowSpecific(expression, exception_class, failure_description, ...)
```

パラメータ

expression

テスト対象の式。

exception_class

例外クラス

failure_description

エラーメッセージを指定する書式文字列。nilを渡すこともできます。

...

（オプション）failure_descriptionの代わりにコンマ区切りの引数リストを渡すこともできます。

詳細

expressionがexception_classクラスの例外を発生させる場合、このテストは失敗します。

STAssertNoThrowSpecificNamed

式が、指定された名前の特定のクラスの例外を発生させる場合、このテストケースは失敗します。

```
STAssertNoThrowSpecificNamed(expression, exception_class, exception_name,
failure_description, ...)
```

パラメータ

`expression`

テスト対象の式。

`exception_class`

例外クラス

`exception_name`

例外の名前を表す文字列。

`failure_description`

エラーメッセージを指定する書式文字列。`nil`を渡すこともできます。

`...`

（オプション）`failure_description`の代わりにコンマ区切りの引数リストを渡すこともできます。

詳細

`expression`が`exception_name`という名前の`exception_class`クラスの例外を発生させる場合、このテストは失敗します。

STAssertTrueNoThrow

式がFalseまたは例外を発生させる場合、このテストケースは失敗します。

```
STAssertTrueNoThrow(expression, failure_description, ...)
```

パラメータ

`expression`

テスト対象の式。

`failure_description`

エラーメッセージを指定する書式文字列。`nil`を渡すこともできます。

`...`

（オプション）failure_descriptionの代わりにコンマ区切りの引数リストを渡すこともできます。

STAssertFalseNoThrow

式がTrueまたは例外を発生させる場合、このテストケースは失敗します。

```
STAssertFalseNoThrow(expression, failure_description, ...)
```

パラメータ

expression

テスト対象の式。

failure_description

エラーメッセージを指定する書式文字列。nilを渡すこともできます。

...

（オプション）failure_descriptionの代わりにコンマ区切りの引数リストを渡すこともできます。

用語解説

アクティブなビルド構成(active build configuration) プロダクトをビルドするためのビルド構成。**ビルド構成(build configuration)**も参照。

アクティブSDK (Active SDK) プロダクトをビルドするために使われるSDK、およびそのプロダクトを実行する実行環境。**SDKファミリ(SDK family)**も参照。

アプリケーションID (Application ID) 1つのベンダのiPhoneアプリケーションまたはiPhoneアプリケーションセットを識別する文字列。これは、バンドル識別子に類似しています。アプリケーションIDの例：`GFWOTNXFIY.com.mycompany.MyApp`。

ベースSDK (Base SDK) プロジェクトのターゲットをビルドするときに使用するデフォルトのSDKを指定するプロジェクト設定。この設定は、「iPhone OS Deployment Target」ビルド設定によってオーバーライドできます。

ビルド構成(build configuration) ビルド設定の名前付きのコレクション。これによって、1つのプロジェクト内で、異なる特定の目的のために（デバッグ用とリリース用など）1つ以上のプロダクトをビルドできるようになります。

CSR（証明書署名要求：Certificate Signing Request） 開発用証明書を生成するために使われる個人情報を含むファイル。CSRは、キーチェーンアクセスアプリケーションによって作成されます。

コード補完(code completion) 識別子やキーワードが入力されるたびに、考えられる完成形を自動的に提案するショートカット機能。提案は、入力されたテキストとその周辺の文脈に基づいて行われます。

開発用証明書(Development Certificate) iPhoneアプリケーションデベロッパを識別するファイル。Xcodeは、開発用証明書を使用して、アプリケーションバイナリに署名します。

デバイスファミリ(Device Family) iPhone OSが稼働できるデバイスのタイプ。iPhoneとiPadの2種類があります。

Entitlement アプリケーションが、保護されたiPhone OS機能などにアクセスできるようにするプロパティ。

instrument Instrumentsアプリケーションを使用して開発されたデータ収集エージェント。Instrumentsは、1つのアプリケーションまたはシステム全体に関するパフォーマンス情報を収集します。

Instrumentsアプリケーション(Instruments application) アプリケーションのパフォーマンスデータを収集したり発掘したりするパフォーマンス分析ツール。

iPhone Dev Center iPhoneアプリケーションの開発に必要なすべてのリソースを提供するAppleのデベロッパセンター。このデベロッパセンターにアクセスするには、ADCのメンバになる必要があります。**Apple Developer Connection**も参照。

iPhoneデベロッパプログラム(iPhone Developer Program) iPhoneアプリケーションの開発、デバイス上でのテスト、App Storeを通してのユーザへの配布を可能にするプログラム。

iPhoneプロビジョニングポータル(iPhone Provisioning Portal) iPhoneアプリケーションをテストするためのデバイスを構成できるようにする、iPhone Dev Centerのアクセス制限領域。

iPhoneシミュレータアプリケーション(iPhone Simulator application) 開発の初期の段階でiPhoneアプリケーションをテストするために、iPhone OSのランタイム環境とユーザ体験をMac OS Xでシミュレートするアプリケーション。

プロジェクトウインドウ(Project window) Xcodeのプロジェクトを構成するファイルを表示したり整理するウインドウ。

プロビジョニングプロファイル(Provisioning Profile) 開発中のアプリケーションをiPhone OSベースのデバイスにインストールできるようにするファイル。このファイルには、1つ以上の開発用証明書、1つのアプリケーションID、および1つ以上のデバイスIDが含まれています。

SDKファミリ(SDK family) 特定のAppleプラットフォーム用にソフトウェア製品をビルドするために使われるSDKリリースのグループ。利用可能なSDKファミリには、Phone Device SDK、iPhone Simulator SDK、およびMac OS X SDKがあります。

テストケース(test case) テスト対象コードを実行してそれが期待通りに動作することを確認するためのコード。

テストケースメソッド(test-case method) `test...`という名前の単体テストクラスのインスタンス。テストにAPIを実行し、期待する結果が得られるかレポートします。

テストプロビジョニングプロファイル(Test Provisioning Profile) iPhoneアプリケーションの開発チームに属していないユーザに発行されるプロビジョニングプロファイル。これを利用すると、App Storeに発行されていないアプリケーションのインストールとテストができます。

テストスイート(test suite) テストケースセット。**テストケース(test case)**も参照。

Xcode iPhoneとMacのアプリケーションの開発に使われる一連のツールとリソース。

Xcodeアプリケーション(Xcode application) Xcode統合開発環境(IDE)のメインアプリケーション。Xcodeに含まれるほかのアプリケーションを管理し、ソフトウェア製品の開発に使われるメインのユーザインターフェイスを提供します。

書類の改訂履歴

この表は「*iPhone開発ガイド*」の改訂履歴です。

日付	メモ
2010-05-28	プロビジョニングプロファイルの自動管理、アプリケーションアーカイブ、およびアプリケーションの配布についての情報を追加しました。
	「アプリケーションの実行」（33 ページ）を更新し、ベースSDK (Base SDK)と、「iPhone OS Deployment Target」ビルド設定、およびプロジェクトウインドウの「概要(Overview)」ツールバーメニューの使いかたの詳細について記述しました。
	「iPhoneシミュレータの使用」（47 ページ）を更新し、iPhone OS 4.0のObjective-Cランタイムの変更が既存のiPhoneシミュレータバイナリに及ぼす影響についての情報を追加しました。
	「デバイスとデジタルIDの管理」（51 ページ）を更新し、プロビジョニングプロファイル自動管理と、Xcodeの「オーガナイザ(Organizer)」での開発者プロファイルの管理方法について記述しました。
	「アプリケーションの配布」（83 ページ）を更新し、「ビルドとアーカイブ(Build and Archive)」コマンドおよびテスト用アプリケーションの配布のための推奨されるワークフローについて詳細を追加しました。
	「iPhoneシミュレータの使用」（47 ページ）にハードウェアシミュレーションのサポート情報を追加しました。
2010-03-19	iPadの情報を追加しました。
	アプリケーションを実行するデバイスファミリを指定する方法を説明する「デバイスファミリの設定」（38 ページ）を追加しました。
	iPhoneターゲットをiPadアプリケーションのビルド用にアップグレードする方法を説明する「iPhoneからiPadへのターゲットのアップグレード」（32 ページ）を追加しました。
	「iPhoneシミュレータの使用」（47 ページ）を更新し、iPadの情報を追加しました。
2010-01-20	誤字の訂正や、読者からのフィードバックの反映を行いました。
2009-08-06	プロパティリストファイルの編集、スタティックライブラリのリンク、およびiPhoneシミュレータのバージョンについての情報を追加しました。若干の変更をしました。
	「プロパティリストファイルの編集」（27 ページ）を追加しました。

日付	メモ
	デバイス上でのアプリケーションのデバッグについての重要な情報を「デバッグ機能の概要」（59 ページ）に追加しました。
	「シミュレーション環境でのデバイスファミリとiPhone OSバージョンの設定」（47 ページ）を追加しました。
2009-05-28	アプリケーションをビルドするアーキテクチャの設定方法について説明しました。
	「アーキテクチャの設定」（37 ページ）を追加し、アプリケーションのビルドが可能なアーキテクチャの選択方法を説明しました。
2009-05-14	単体テストの説明と効率化されたビルドワークフローを追加しました。
	「アプリケーションの単体テスト」（65 ページ）を追加しました。
	「iPhoneシミュレータのフレームワークとライブラリ」を追加しました。
	「アプリケーションの実行」（33 ページ）を効率化されたビルドフローで更新しました。バックアップからのアプリケーションデータの復元についての情報を追加しました。
2009-04-08	Entitlementプロパティリストファイルの作成についての情報を追加しました。内容を若干再編成しました。
	「アプリケーションのエンタイトルメントの管理」（30 ページ）を追加しました。
2009-03-04	内容を若干変更しました。
2009-02-04	内容を若干変更しました。
	「スクリーンショットのキャプチャ」（57 ページ）を更新し、キャプチャしたスクリーンショットのPNGファイルを「オーガナイザ(Organizer)」に取り込む方法を示しました。
	「アプリケーションのビルド」（41 ページ）をアプリケーションIDのビルドエラー情報で更新しました。
2009-01-06	内容を若干追加しました。
	iPhoneシミュレータバイナリは、シミュレータの1つのリリース上でのみ使用できることを説明しました。
2008-11-14	iPhoneシミュレータの新しい機能について情報を追加しました。
	「アプリケーションへのiTunesアートワークの追加」（85 ページ）を追加しました。
	「ハードウェアの操作」（48 ページ）に「メモリ警告をシミュレート(Simulate Memory Warning)」および「着信ステータスバーを切り替える(Toggle In-Call Status Bar)」のコマンドについての情報を追加しました。

日付	メモ
	「Core Location機能」（50 ページ）を追加しました。
	iPhoneアプリケーションでのスタティックライブラリの使用についての情報を、「iPhoneプロジェクトの作成」（14 ページ）に追加しました。
2008-10-15	Xcodeを使ったiPhoneアプリケーションの開発方法について説明する新規文書。
	『*iPhone OS Programming Guide*』および『*iPhone Simulator Programming Guide*』に以前公開された内容を組み込みました。

書類の改訂履歴