Xcode 4.3.2 対応

前回の Xcode 4 リリース対応である「Xcode 4 リリース版との違い .pdf」は、こちらに吸収されます。

この PDF は Xcode のバージョンアップ（2012 年 5 月現在のバージョンは 4.3.2）にともない、本書をそのまま実践できなくなった部分を補完します。

左の列にオリジナルのページ情報と変更点の要約、右の列に対応する変更内容を配置しています。

例）

4 ページ

Mac OS X 10.7（lion）での開発環境インストールが App Store 経由になりました。

1.2 開発環境ダウンロード

Mac OS X 10.7（lion）では、App Store から iOS 開発環境をインストールするようになっています。

本書の 4 ページ目が変更対象であり、その理由を説明。

変更内容。

また、Xcode 3.4.2 用のサンプルソースも用意しました。オリジナルのサンプルソースフォルダと同じ構成です。

國居貴浩　2012 年 5 月

4ページ

Mac OS X 10.7（lion）での開発環境インストールがApp Store経由になりました。

注）Mac OS X 10.6(Snow Leopard) での iOS 開発環境は、従来のどおり iOS Dev Center からのダウンロードとなります。

また、最新の開発環境は Mac OS X 10.7(lion) 以降でしか提供されていません。

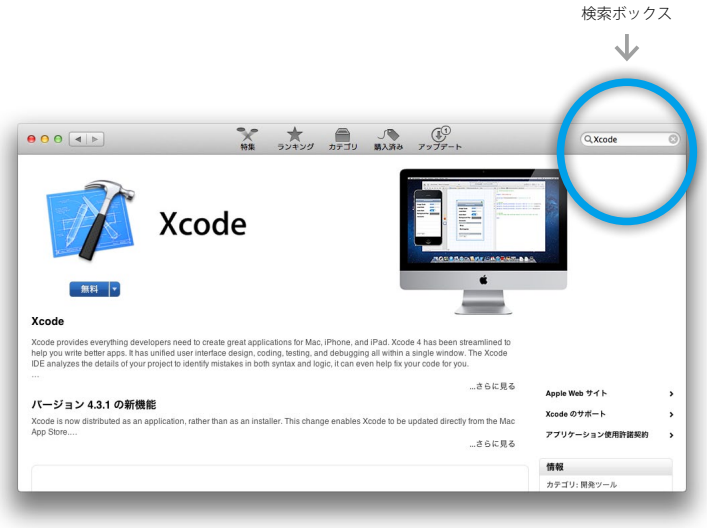
1.2 開発環境ダウンロード

Mac OS X 10.7（lion）では、App Store から iOS 開発環境をインストールするようになっています。

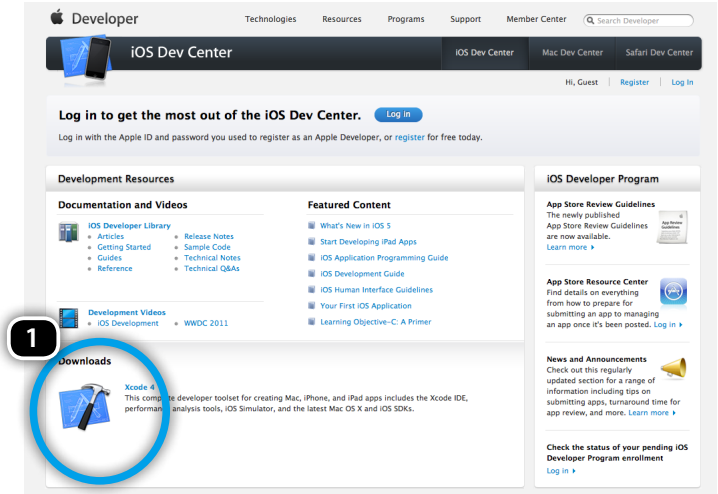
App Store では、検索ボックスに

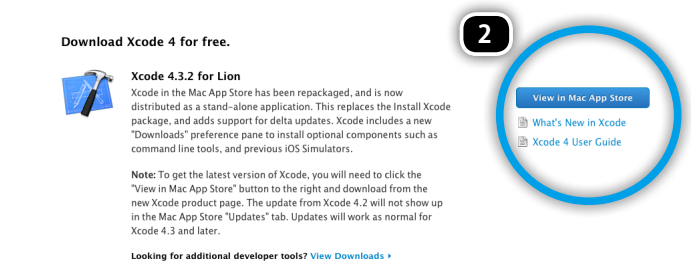
Xcode

と入力して検索すれば見つかります。



開発のメインサイト iOS Dev Center からリンクをたどっていく事でも可能です。

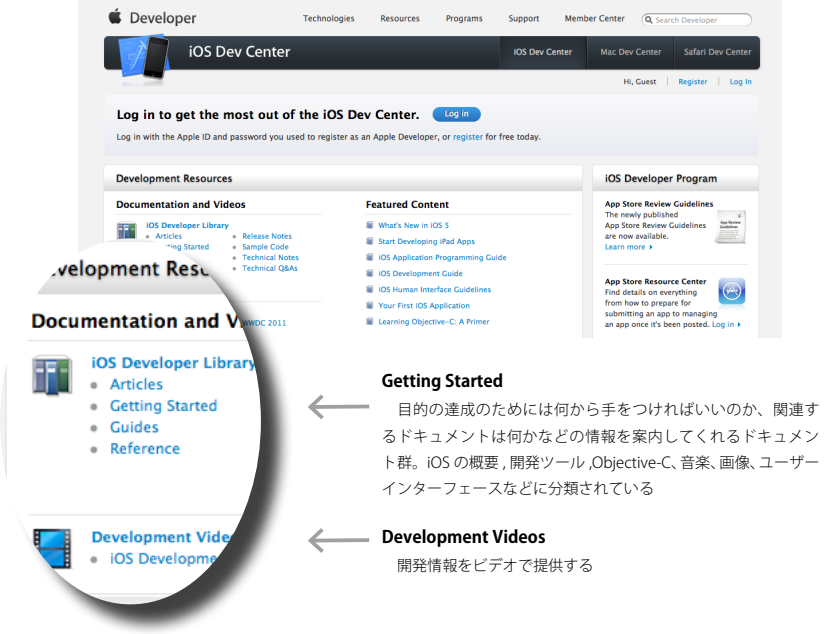




12 ページ

iOS Dev Center サイトの構成が少し変わりました。

Getting Started Documents → Getting Started となりました。
Development Videos というセクションが加わりました。



Getting Started

目的の達成のためには何から手をつければいいのか、関連するドキュメントは何かなどの情報を案内してくれるドキュメント群。iOS の概要 , 開発ツール ,Objective-C、音楽、画像、ユーザーインターフェースなどに分類されている

Development Videos

開発情報をビデオで提供する

13 ページ ・・・ 19 ページ

Xcode バージョン 4.3.2 からはインストール先が異なります。

また、Navigation-Based Application テンプレートが無くなったので、かわりに Empty Application テンプレートを使った手順を説明します。

13 ～ 19 ページ中盤までを、右の内容で差し替えてください。

これから iOS アプリケーションの開発を始めようとする大部分の人達が最初に思うのは

自分の環境で実際に iOS アプリケーションを作ることができるのか？

でしょうから。

さあ、Xcode を立ち上げましょう。

2.2 Xcode を起動

最初に触れたように Xcode は、iOS アプリケーション開発環境を提供してくれるアプリケーションです。

アプリケーションフォルダを開き、Xcode のアイコンをダブルクリックする事で起動できます。

Launchpad から Developer グループを選び、その中の Xcode をクリックでもかまいません。

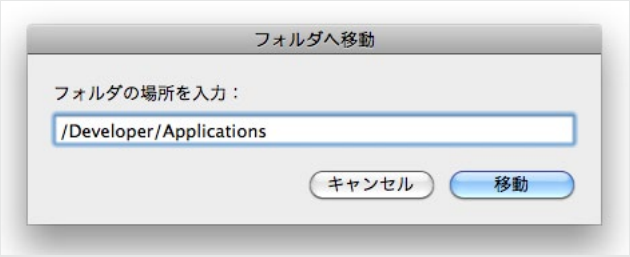
Developer グループには、Xcode の他に iOS/Mac アプリケーション開発で利用するアプリケーション群が集められています。

Mac OS X 10.6(SnowLeopard) の場合、インストール時に別の場所を指定しない限り

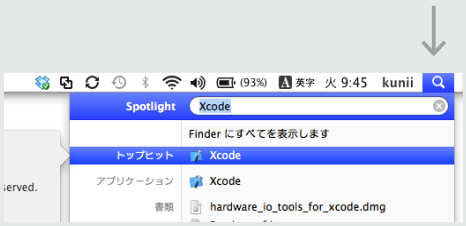
/Developer/Applications

になります(Mac OS X lion でも Xcode のバージョン 4.3.2 より前はこの場所にインストールされていました)。

上記の unix パス表記で示されるこの場所がわからない人は Finder の移動→フォルダへ移動 ... メニューを選んで、表示される以下のダイアログにそのまま上の文字列を入力してください。



Spotlight に Xcode と書き込んで検索、起動することもできます。

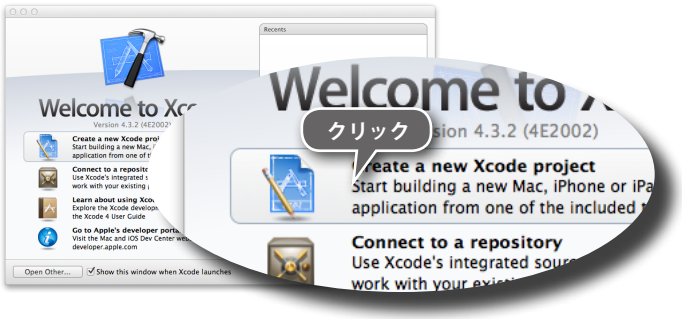


2.3 ようこそ Xcode へ

初めての起動ではウエルカムメッセージがお迎えです。

それでは、始めてみましょう。

Xcode は数ステップで iOS アプリケーションを作成し、iPhone シミュレータを起動することができます。まずは "Create a new Xcode project" をクリックしてください。



プロジェクトテンプレートを選ぶためのダイアログが現れます。

2.4 iOS アプリケーションプロジェクトの新規作成

Xcode は iOS アプリケーションだけではなく、Mac アプリケーションも作成できるようになっています。そのため、ダイアログでは最初に左側の分類一覧から iOS の Application を選んでおく必要があります。

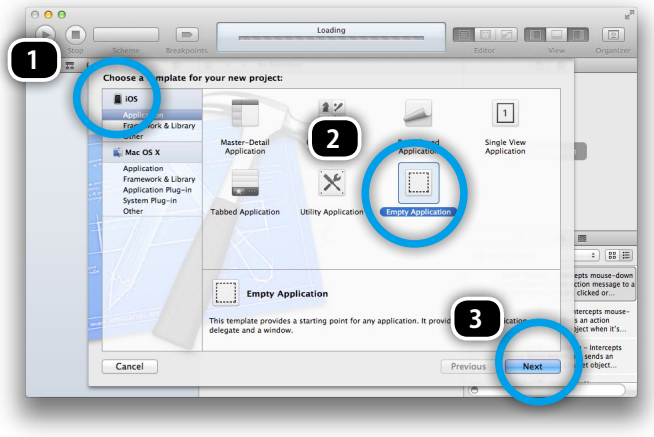
選ぶと、右側のビューにいくつかの iOS アプリケーション用テンプレートアイコンがリスト

アップされます。3D グラフィック用、タブコントローラを使うもの、…

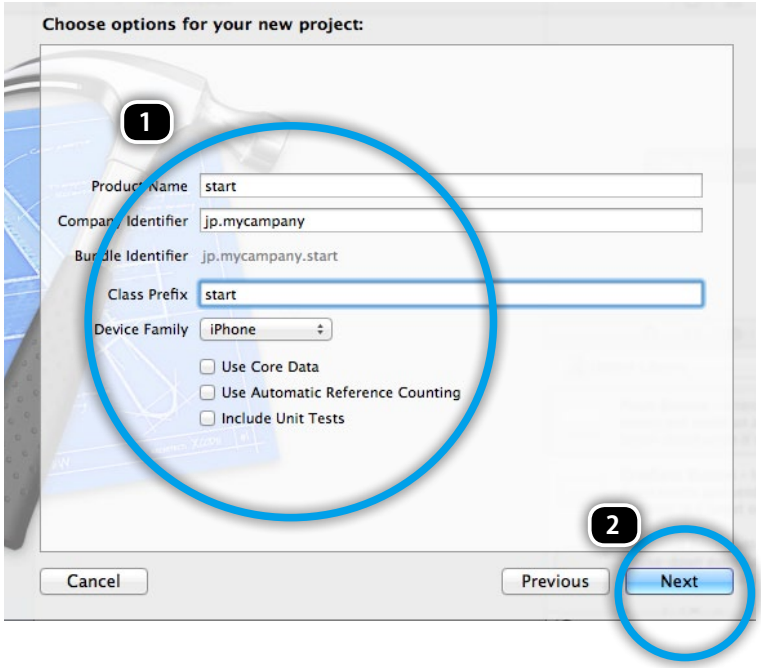
今回は

Empty Application

を選び Next ボタンをクリックしてください。



次に出てくるのはプロジェクトの名前（Product Name）や会社識別子（Company Identifier）を指定するダイアログです。



今回はプロジェクトの名前を

start

とします。

その下の会社識別子は iPhone 実機にアプリケーションを実機にインストールする際に重要な意味を持ちますが、現状は仮の設定でもかまいません。

jp.mycompany

とでも仮決めしておきましょう。

もちろん、すでに会社識別子を決めて利用している人はその文字列を打ち込んでください。詳しくは 8 章の「ハートに火をつけて」の章で説明します。

Class Prefix はテンプレートにより自動的に作成されるファイル名等の接頭辞を指定します。何でもかまいませんが、ここではプロジェクトの名前に合わせ

start

にします。

Device family はアプリケーションを iPad 専用にするか iPhone 専用にするか、それとも iPad では iPad 用、iPhone では iPhone 用になるユニバーサルアプリケーションにするかを指定します。

iPhone

を選びます。

Use CoreData は Apple が提供する情報管理用のフレームワーク（ある目的のために用意された、連携して動作する関数やクラス群をフレームワークと呼びます）です。
例えばゲームの記録など、アプリケーションが終了しても残しておきたい情報を管理するために利用します。非常に便利なのですが、この本では扱いません。

チェックを外します

メモリ管理の基礎を学習するために、あえて Automatic Reference Counting を使わずに手動でメモリを管理します。

Automatic Reference Counting の概略は、234 ページの補完として追記しました。

Use Automatic Reference Counting はメモリ管理を自動化させるスイッチです。
この本では、メモリ管理の基礎を学習するために、あえて Automatic Reference Counting を使わずに手動でメモリを管理します。

チェックを外します

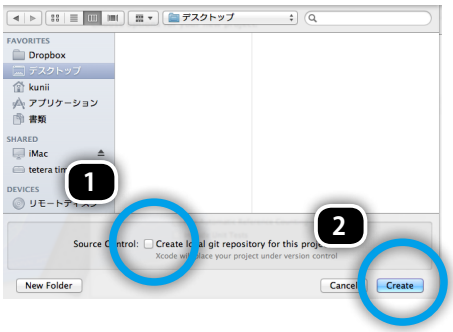
メモリ管理の知識は、アプリケーションのパフォーマンスを調整する上で重要です。
まずは、この本でメモリ管理の基礎を学び、Automatic Reference Counting が代行してくれる処理がどういったものであるかを理解してください。
その後は、積極的に Automatic Reference Counting を利用するとよいでしょう。

Include Unit Test はアプリケーションの部品をテストするために使う機能です。ページの都合上、この本では扱いません。

チェックを外します

用意ができたら Next ボタンを押してください。

次の画面ではプロジェクト一式を置く場所を聞かれるので、好きな場所を指定してください。
ここではデスクトップを選びました。

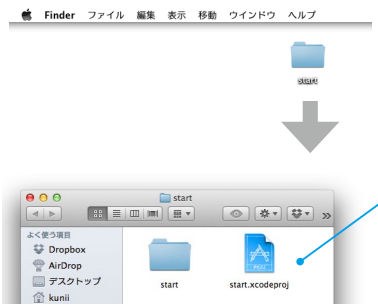


下部にある **Source Control** の指定はプログラムの変更履歴を残していくことができる機能です。そのまま利用してもらっていてもかまいませんが、この本では扱いません。

チェックを外します

保存場所を決めたら Create ボタンを押してください。

デスクトップに "start" というフォルダが作られ、iOS アプリケーション作成用ワークスペースウィンドウが開きます。



start フォルダの中には、関連ファイルをまとめた同名の start フォルダと start.xcodeproj があり、以後はこのアイコンをダブルクリックすると Xcode を起動する事ができる

最初にワークスペースウィンドウのエディタエリアに表示されるのは、今から作成する iOS アプリケーションの情報画面です。アプリケーションが、どのバージョンの OS まで動作を保証させるかなどを指定します。

今は特にさわる必要はありません。

オリジナルのナビゲーションバーは、この PDF ではジャンプバーと表記します。

ナビゲータセレクタバー

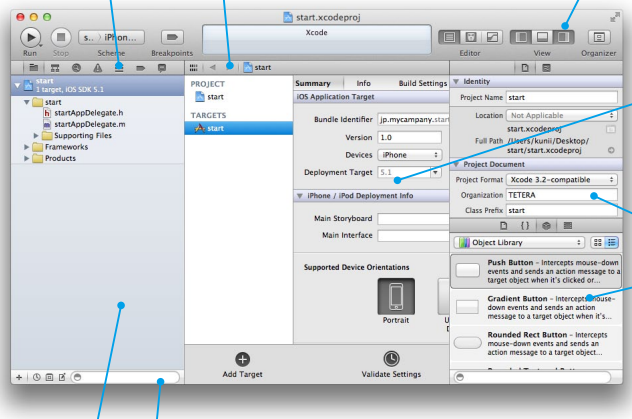
目的に合わせて、下のナビゲータエリアの内容を切り替えるためのバーです。構成ファイル一覧や、注意事項一覧、履歴一覧などがあります

ツールバー

メニューにある主要なコマンドをアイコンボタンで提供します

ジャンプバー

エディタエリアに表示中の内容の特定領域にすばやくアクセスするために使われます



エディタエリア

ナビゲータエリアで選択された項目によってさまざまに変化し、表示内容を確認または編集する機能を提供します

ユーティリティエリア

エディタエリアの内容の詳細などを表示します

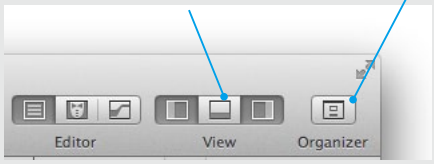
フィルターバー

ナビゲータエリアに表示する一覧を、入力した内容にあわせて制限することができます

ナビゲータエリア

ナビゲータセレクタバーで選択された項目にあわせて表示内容がかわります

オリジナルの図版には付いていないデバッグエリアを表示するための Hide or show the Debug area ボタンが追加されています



オリジナルの図版には付いていない Organizer ウィンドウを表示するための Open the organizer ボタンが追加されています。

Organizer ウィンドウは 129 ページの補完、191 ページから 205 ページの補完で説明します

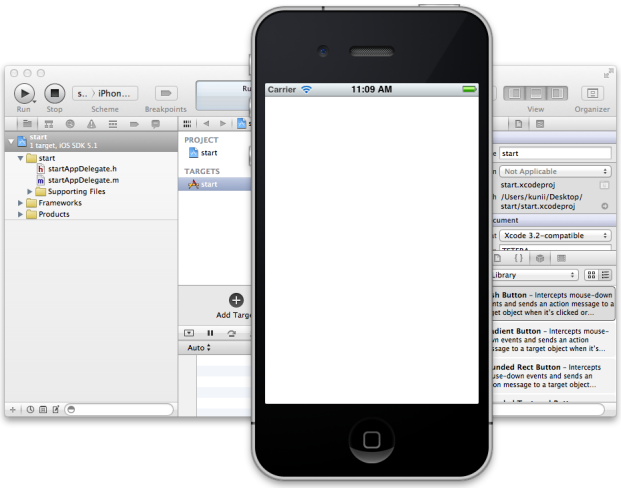
まずは iOS アプリケーションを動かしてみましょう。

実行先が iPhone シミュレータになっているのを確認して、ツールバーの Run ボタンを押してください。

実行先の変更

クリックするとポップアップメニューが表示され、使用するシミュレータが選べます。iOSDevice というのは実機の事ですが、有料開発者登録を済ませていない場合は利用できません。

Mac の画面上に iPhone が出現しましたか？
これが開発用に使う iPhone シミュレータです。

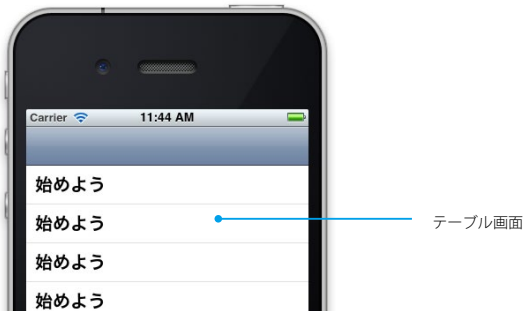


ここまで確認したら、iPhone シミュレータ→ iPhone シミュレータを終了メニューを選んでシミュレータを終了し Xcode に戻ってください。

今回選んだ Empty Application テンプレートプロジェクトは、真っ白の画面を一つ用意した iOS アプリケーションを作成するプロジェクトのテンプレートです。

2.5 自分の iPhone アプリケーションの作成

では、ここに iOS アプリケーションでよく見かけるテーブル画面を表示してみましょう。

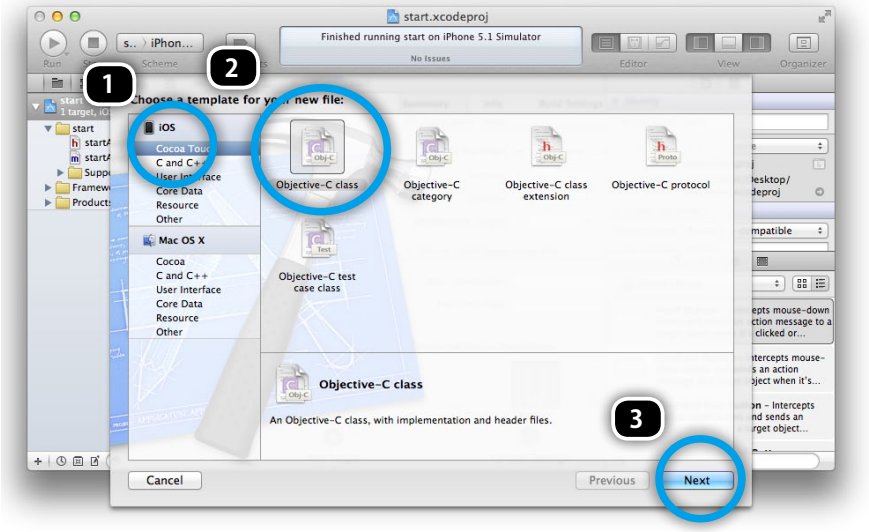


Xcode で File → New → File…メニューを選んでください。

今回の Empty Application テンプレートにはオリジナルで選んだ Navigation-Based Application テンプレートで用意されるテーブル画面管理部品（RootViewController.m/h/xib というファイルで用意される）がありません。

そのため RootViewController を自前で用意する事になるので、その手順を説明します。

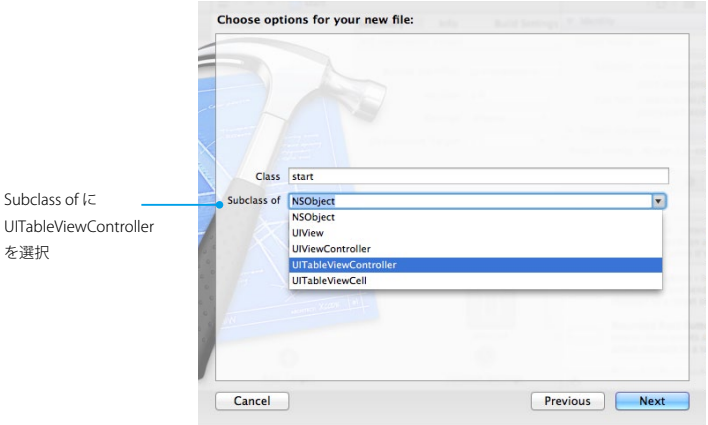
ファイルテンプレート選択画面が現れるので、左側の分類から iOS の Cocoa Touch を選び、右に表示されるファイルテンプレートアイコンの中から Objective-C class を選んで Next ボタンを押してください。



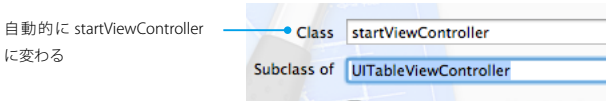
次に現れる画面では Subclass of のドロップダウンされるメニューから

UITableViewController

を選択します。



Subclass of に UITableViewController を選ぶと Class の項目が、自動的に startViewController に変わります。



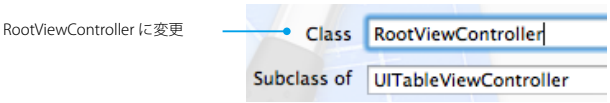
名前はなんでもいいのですが、オリジナルの説明で RootViewController という名称（Navigation-Based Application テンプレートは、この名前で用意する）を使っているので、そちらに合わせています。

プロジェクト作成時に指定した Class Prefix が適用され、選んだ subclass にみあった名前を Xcode が自動的に設定してくれています。

このままでもかまいませんが、ここでは

RootViewController

という名前に変更します。



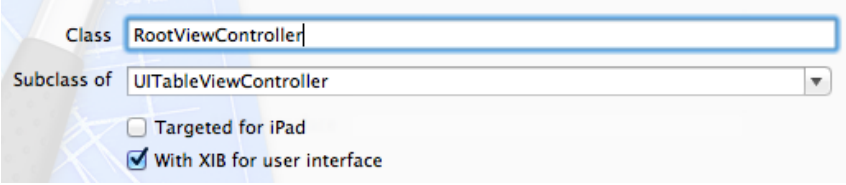
sub class の下にある **Targeted for iPad** は、テーブル画面コントローラを iPad 用に作成するかどうかのスイッチです。iPhone 用を作っているので使いません。

チェックを外します

With XIB User Interface は作成するテーブル画面のレイアウトを XIB ファイルとして提供するかどうかのスイッチです。

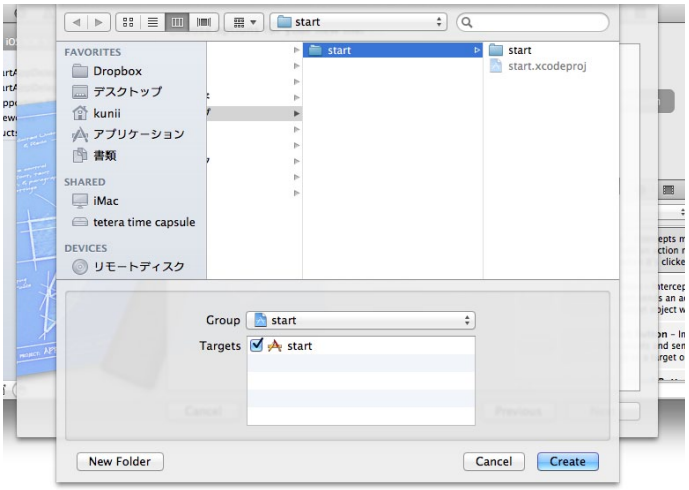
XIB ファイルは後の章で取り扱うので、提供してもらう事にします。

チェックを付けます



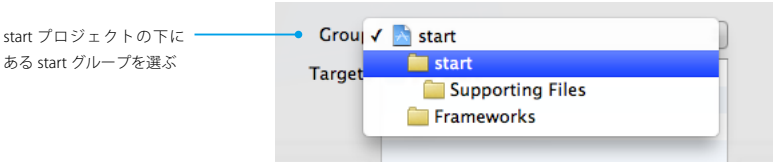
用意ができたら Next ボタンを押してください。

次の画面ではファイルの作成先を問い合わせてきます。

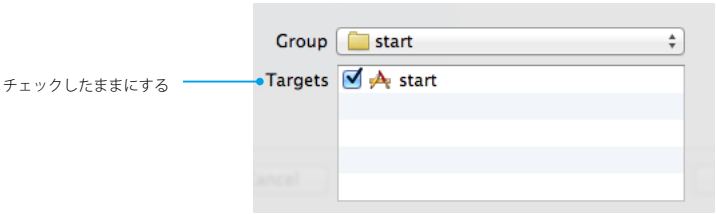


作成先はどこでもいいのですが、関連するプロジェクトのフォルダに収めるのが適切でしょう。

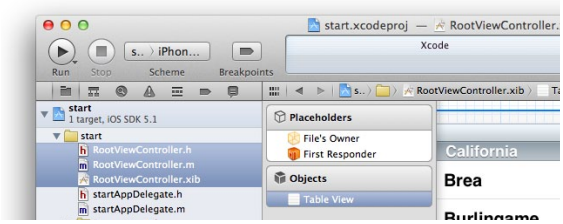
下側にある **Group** のポップアップメニューから start プロジェクトの下にある start グループを選んでください。



その下の **Targets** は、このファイルを登録するアプリケーションの指定です。start にチェックが付いているはずなので、そのまま Create ボタンを押してください。



ナビゲーションエリアには start グループの中に RootViewController.h と RootViewController.m、RootViewController.xib という項目が追加されます。



start グループアイコンが表示されていないときは、start プロジェクトアイコンの 横にある三角形 (デスクロージャ) をクリックする。
同じように start グループ横 のデスクロージャをクリックする事で RootViewController.m 等が表示できる。

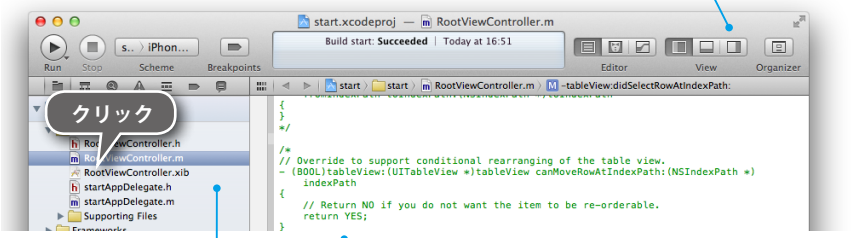
start グループアイコン

これでテーブル画面を表示する準備ができました。
今度は、この RootViewController をアプリケーションから使ってみます。同じ start グループ内に

startAppDelegate.m

という項目があると思うので、クリックしてください。
startAppDelegate.m を選ぶとエディットエリアに内容が表示されます。

Hide or show the Utilities ボタンをクリックして押し下げ状態を解除すれば、ユーティリティエリアが隠されて、エディットエリアを広くもできます



ナビゲーションエリアの startAppDelegate.m をクリックして選択すると、エディットエリアに startAppDelegate.m の内容が表示され編集可能となる

エディットエリアに表示されるテキストは iOS アプリケーションの制御コードを記述したものでプログラムソースといいます。
そして、このプログラムソースを書く行為をコード記述、すなわちコーディングと呼ぶわけです。ただし iPhone やその他コンピュータはこのプログラムソースを理解できません。
コンピュータが理解できるのは 0 と 1 だけなので、このプログラムソースを理解できる形に翻訳する必要があります。
これを翻訳作業 (コンパイル) と呼び、コンパイルした 0、1 記述の制御コード、画像やサウンドファイル、その他アプリケーションに必要な資源をひとまとめにして決められた形式に基づいたひとつのアプリケーションに仕上げる作業をビルドと呼んでいます。
最初にクリックした Run ボタンの「Run」とは、この一連の作業を指していたわけです。

この時点のサンプルソースを

start24-pre

として追加提供しています。

次にエディットエリアの 2 カ所に、以下の文を追加してください。

これは、先ほど用意した RootViewController を使って iPhone 画面上にテーブル画面を表示させるための処理です。

startAppDelegate.m

```
#import "startAppDelegate.h"
#import "RootViewController.h"

@implementation startAppDelegate

@synthesize window = _window;

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

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)
launchOptions
{
    self.window = [[[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]] autorelease];
    // Override point for customization after application launch.
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];

    RootViewController* viewcontroller = [[[RootViewController alloc]init] autorelease];
    self.window.rootViewController = [[[UINavigationController alloc]
        initWithRootViewController:viewcontroller]autorelease];
    return YES;
}
```

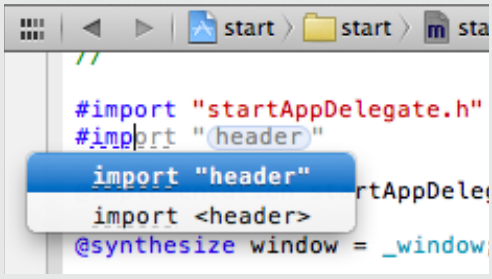
追加

追加

コード補完機能

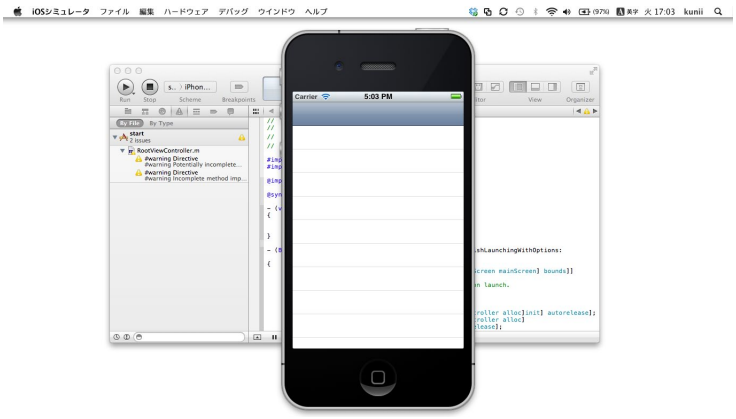
エディットエリアで #import と書こうとして #、i、m、と打ち込んでいくたびに、携帯電話の文字補完のように単語候補が表示されましたか？

このように Xcode にもプログラマ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き、入力しようとしている単語を完成させるための提案を行うコード補完機能があります。



書き込み終わったら、先ほどと同じように Run ボタンを押してみてください。

iOS アプリケーションのビルド作業が実行され、完了すると iPhone シミュレータが起動されテーブル画面が表示されましたか？



せっかくなので、少しばかり、iPhone シミュレータを楽しんでみましょう。

19 ページのシミュレータの説明に続きます

21 ページ ・・・ 28 ページ

自前で用意した RootViewController はオリジナルの Navigation-Based Application テンプレートで用意されたものと少し異なります。

その点をふまえ Xcode バージョン 4.3.2 に対応した形で説明します。

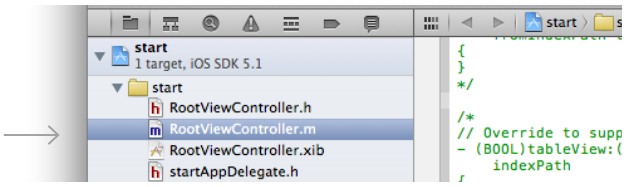
21 ～ 28 ページを、右の内容で差し替えてください。

しかし、できあがった アプリケーションは、見た目はよく見かける iOS アプリケーションのテーブル画面のようですが、触っても何も反応してくれません。

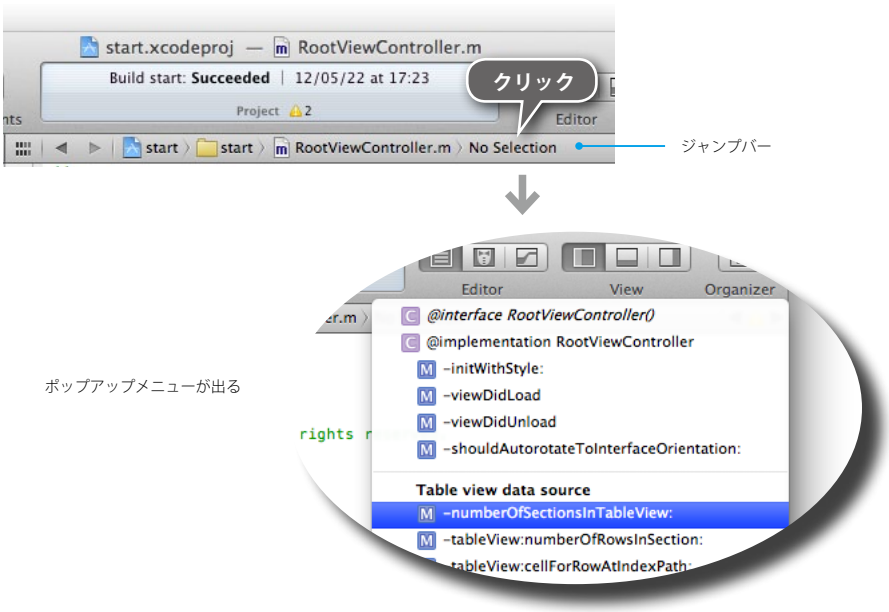
まずはテーブルに 20 個の項目を自力で表示させてみましょう。

シミュレータを終了し Xcode に戻ってください。

Xcode に戻ったら、今度は RootViewController.m を選んでエディットエリアに内容を表示させてください。



次にエディットエリアの上にあるジャンプバーをクリックしてください。メニューがポップアップします。



ポップアップしたメニューから

-numberOfSectionsInTableView:

を選びます。

エディットエリアの内容がスクロールして以下の部分が表示されましたか？

```
#pragma mark - Table view data source

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
#warning Potentially incomplete method implementation.
    // Return the number of sections.
    return 0;
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSectic
{
#warning Incomplete method implementation.
    // Return the number of rows in the section.
    return 0;
}
```

ポップアップしたメニューから選んだものはメソッドというものです。

2.6 メソッド

表示された部分では、(NSInteger) や (UITableView*) など追加情報や実際の動作処理が記述されています。今後、この本でメソッドを説明するときはメニューにならって

-numberOfSectionsInTableView: メソッド

といったように表現するので覚えておいてください。

表示されたメソッドはテーブルのセクション数を返すメソッドです。
そしてその下にある

-tableView:numberOfRowsInSection: メソッド

は、各セクションが持つ項目数を返すメソッドとなります。
このように、テーブル画面はセクションと、そのセクションに含まれる項目という形で構成されています。

注）ここでは、わかりやすいようにセクションごとに分割エリアを表示させている

セクションの数と、各セクションに含まれる項目数からテーブル画面は決定されるわけです。
処理を見ると、-numberOfSectionsInTableView: メソッド、-tableView:numberOfRowsInSection: メソッドどちらも 0 を返しています。
セクション数も項目数も 0 なので、何も表示されないわけです。
セクションを 1 つ用意し、そのセクションに含まれる項目数を 20 にしてみましょう。

また、セクションや項目数が 0 で無い場合、その下の

-tableView:cellForRowAtIndexPath: メソッド

で項目用の画面部品を作って返す必要があります。
その作業も追加します。

各項目用の画面部品を提供しなければならない

RootViewController.m

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
#warning Potentially incomplete method implementation.
    // Return the number of sections.
    return 1;
}


- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
#warning Incomplete method implementation.
    // Return the number of rows in the section.
    return 20;
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)
indexPath
{
    static NSString *CellIdentifier = @"Cell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc]
            initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier] autorelease];
    }
    // Configure the cell...
    return cell;
}
```

セクション数を 1 とする

項目数を 20 とする

項目用画面部品の作成

```
UITableViewCell alloc…
```

が項目部分の画面部品の作成です。

さっきと同じように Run ボタンを押してください。今度は項目をタップすれば青く反転します。

タップすると青く反転 するようになった

確認できたらシミュレータを終了し、Xcode に戻ってください。 今度はテーブルの各項目に文字を表示させてみましょう。

さきほど変更した -tableView:cellForRowAtIndexPath: メソッドの処理に以下の一行を追加します。

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)
indexPath
{
  static NSString *CellIdentifier = @"Cell";
  UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
  if (cell == nil) {
    cell = [[[UITableViewCell alloc]
      initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier] autorelease];
  }
  // Configure the cell...
  cell.textLabel.text = @" 始めよう ";
  return cell;
}
```

表示文字列の指定

= で結んでいますが、数学と違いイコールという意味ではありません。C 言語、C++ 言語、Objective-C 言語では、代入を意味します。この場合 cell.textLabel.text への代入です。 何に何を代入しているかは、また後で。

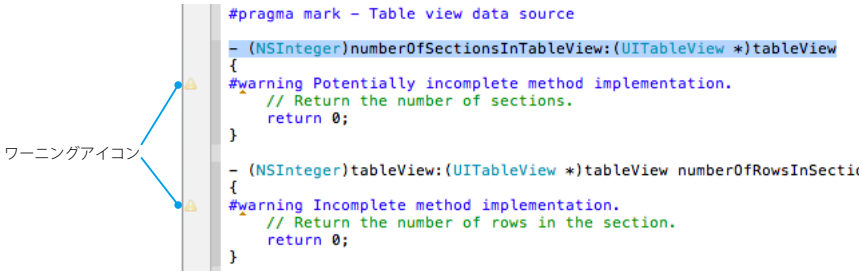
本書はデバッグ作業の扱い方に重点を置いているので、テーブル画面の取り扱い方法などの説明はひかえています。

興味のある人は Apple が提供する日本語ドキュメント「iOS Table View プログラミングガイド」をご覧ください。

https://developer.apple.com/jp/devcenter/ios/library/japanese.html

Run すると、「始めよう」という文字を表示する項目が 20 個現れるはずです。確認できたらシミュレータを終了し Xcode に戻ってください。

ところで、現在エディットエリアは黄色く反転されたり、左側にアイコンが付いていると思います。これはワーニング（注意）というものです。



ワーニングは記述された内容に誤りがあるのではないかという事を、Xcode がユーザーに知らせるためのものです。

いつまでもワーニングアイコンが出ているのも気持ちが悪いので、原因部分を排除しましょう。

実は、ここで表示されているワーニングは強制的に表示するよう指示されたものです。それがワーニングアイコンの横に記述されている

```
#warning Potentially incomplete method implementation.
```

この部分がメッセージとなる

と

```
#warning Incomplete method implementation.
```

この部分がメッセージとなる

という記述です。

セクションや項目をいくつ作るかは制作者しか決められません。そのため Xcode は、ここはそちらにお任せしますというワーニングを、わざと出すようにしているのです。

セクション数は 1、項目数は 20 と決めたので、もはやこれらのワーニングは不要です。削除してください。

```
- (NSInteger)numberOfSections InTableView:(UITableView *)tableView
{
#warning Potentially incomplete method implementation.
    // Return the number of sections.
    return 1;
}


- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
#warning Incomplete method implementation.
    // Return the number of rows in the section.
    return 20;
}
```

削除する

削除する

タップしたら反転するだけというのもつまらないので、タップすると別の画面が出る ようにしましょう。さきほどのエディットエリアのナビゲーションバーから

```
-tableView:didSelectRowAtIndexPath:
```

を選びます。

メソッドの内容がすべてコメントになってしまっています。これではタップしても項目が青くなるだけで何も起こらないはずです。

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)
    indexPath
{
    // Navigation logic may go here. Create and push another view controller.
    /*
     DetailViewController *detailViewController = [[DetailViewController alloc]
         initWithNibName:@"Nib name" bundle:nil];
     // ...
     // Pass the selected object to the new view controller.
     [self.navigationController pushViewController:detailViewController animated:YES];
     [detailViewController release];
     */
}
```

// から改行されるまでの部分や、/* から */ までの範囲がコメント アウトされた状態

このように // から改行されるまでの部分や、/* */ で囲まれた部分をコメントといい、この部分はプログラムには何の影 響も及ぼしません。

コメントは注意書きをプログラムソースの中に残したい場合に利用されます。

また、一時的に一部のプログラムを無効にしたい場合にも用いられ、その場合コメン トアウトなどと表現されます。

2.7 コメントアウト

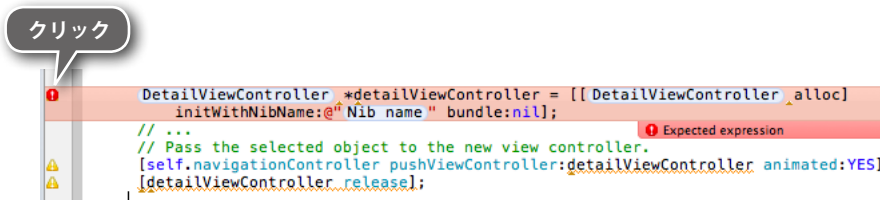
まさに、先のプログラムソースの状態がコメントアウトであり、ここでは項目がタップされた時に実行されるプログラムが無効にされているわけです。

このように、Xcode が作るテンプレートでは、ところどころでコメントアウトされた制御コード記述が存在し、コメントを外すと少しの変更ですぐに使えるように準備されているものがあります。

まずは試しにコメントを外して Run してみましょう。 両端の /* と */ を削除します。
ただし、外したとたんにエラーアイコンが現れます。

// もコメントです。この場合 // のある位置から行の終わりまでがコメント扱いとなります。ここでは、この // は外す必要はありません。

エラーアイコンをクリックすると詳しい情報も表示されます。



どうやら記述ミスを怒られたようです。DetailViewController など存在しないので、この部分を修正しなければいけません。

存在しない DetailViewController の代わりに、必ず存在するものを使いましょう。私たちが現在変更中の

RootViewController

です。以下のように書き直します。

RootViewController.m

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
  // Navigation logic may go here. Create and push another view controller.

  RootViewController *detailViewController = [[RootViewController alloc]
      initWithNibName:@"RootViewController" bundle:nil];
  // ...
  // Pass the selected object to the new view controller.
  [self.navigationController pushViewController:detailViewController animated:YES];
  [detailViewController release];

}
```

RootViewController を使ったものに変更

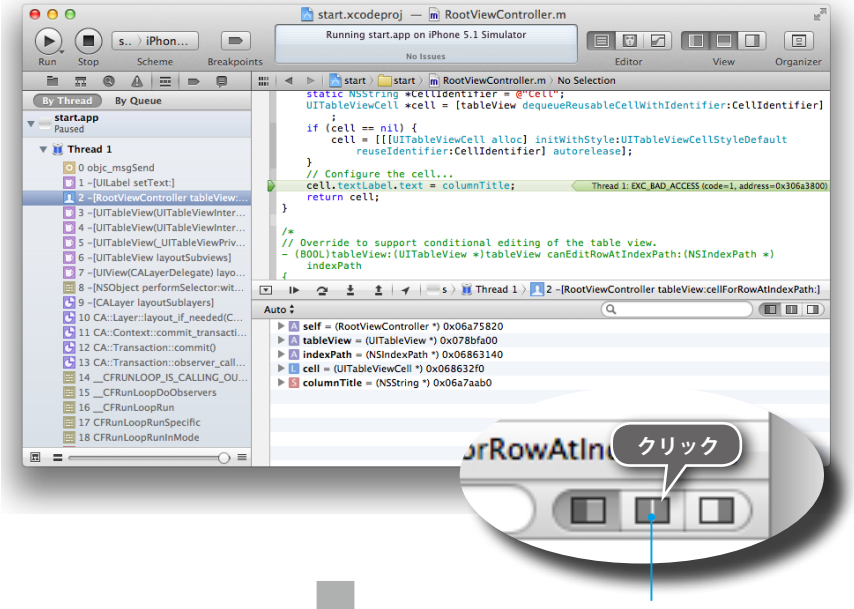
今度は Run できました。項目をタップすると、次々と画面が切り替わります。

しかしこれでは「始めよう」ばかりで、まるで金太郎飴です。
せめて、新しいテーブルの項目は別の文字列にしたいところです。

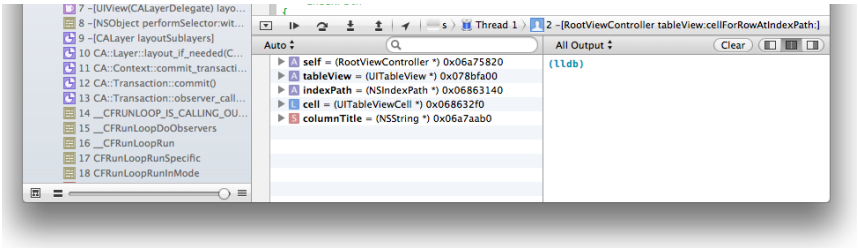
33 ページ

コンソールエリアが表示されていない場合は右のようにしてください。

コンソールエリアが表示されていない人は、Show the Variables View and the Console ボタンをクリックして選択する事で表示されます。

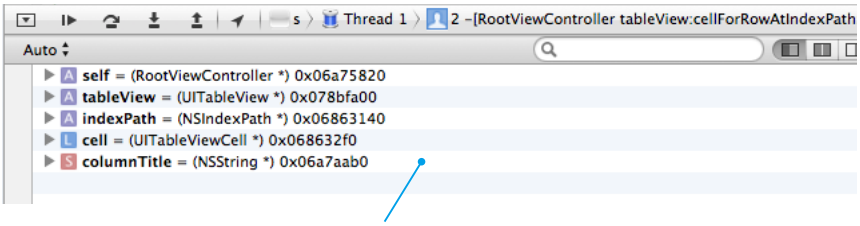


Show the Variables View and the Console ボタン



37 ページ

Variable is not a CFString というメッセージは出なくなりました。

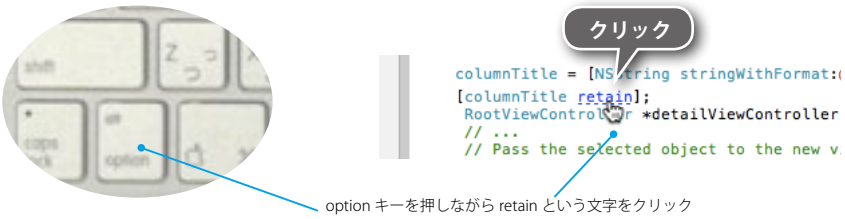


Variable is not a CFString というメッセージは出なくなった

40 ページ

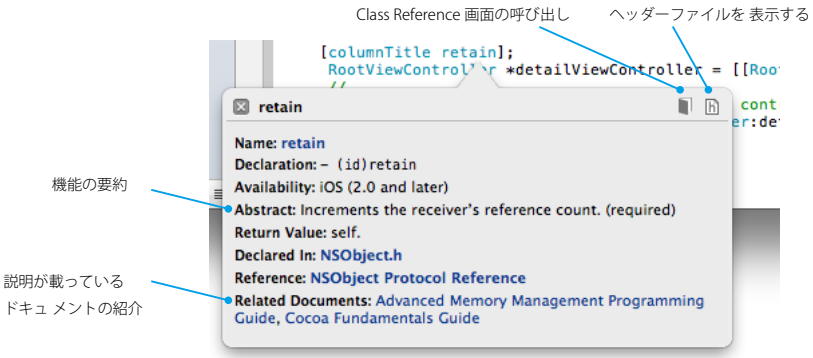
クイックヘルプの出し方が変更されました。

option キーを押しながら追加した retain という文字をクリックしてみてください。



41 ページ

クイックヘルプの内容が少し変わりました。

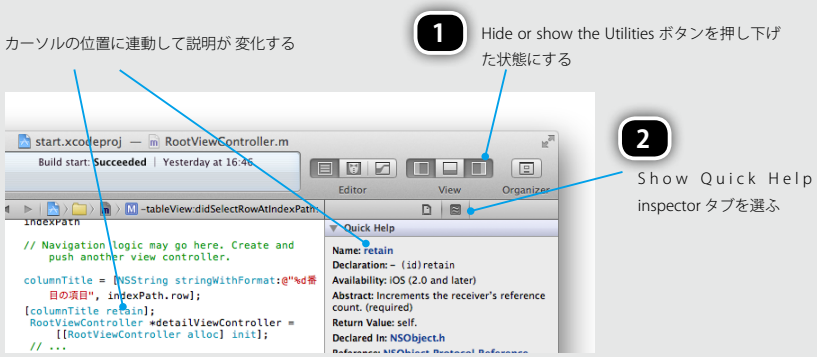


42 ページ

ワークスペースウィンドウのツールバーのボタン構成が少し変わりました。

もうひとつのクイックヘルプ

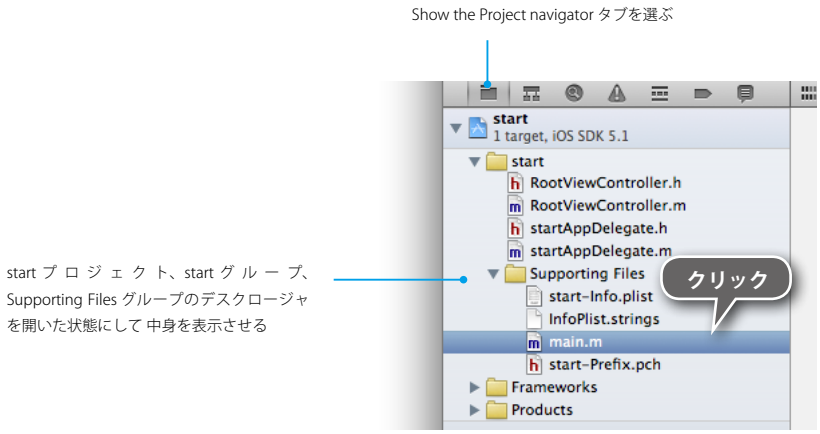
ツールバーにある Hide or show the Utilities ボタンを押し下げ状態にしユーティリティエリアを表示させ、その下の Show Quick Help inspector タブを選ぶ事で、ユーティリティエリアにエディタエリアのカーソル位置に合わせてクイックヘルプを表示してくれる画面（Quick Help inspector 画面）を出しておくこともできます。



70 ページ

main.m の位置が異なります。

ワークスペースウィンドウのナビゲータセレクタバーで Show the Project navigator タブを選び、start プロジェクト /start グループ /Supporting Files グループ内の main.m をクリックしてください。



71 ページ　72 ページ

main 関数の内容、NSAutoreleasePool の扱いが変わりました。

71、72 ページを、右の内容で差し替えてください。

エディタエリアには以下の内容が表示されています。

デバッガでみた内容そのままですね。忘れないでください。

iOS アプリケーションは main 関数から始まる。

ということを。

しかし、なんともシンプルな処理です。これだけで、なぜあれだけ多種多様なアプリケーションに変化するのでしょう？

それは別の章にゆずるとして、ここでは

```
@autoreleasepool {
        ・・・
}
```

に注目してください。

これは @autoreleasepool の {}（波括弧）で囲まれている範囲で、暗黙裏に NSAutoreleasePool オブジェクトが作られ、解放されることを意味します。

```
@autoreleasepool {
        NSAutoreleasePool* pool = [[NSAutoreleasePool alloc]init];
        ・・・
        [pool release];
}
```

注意）これはイメージです。

以前は実際に

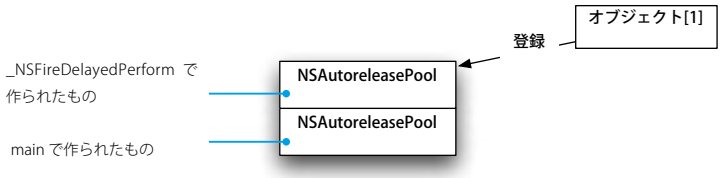
```
int main(int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```

という直接的な記述でしたが、現在は @autoreleasepool {…} という記述形式に変更されました。

4.9 積み重なる自動解放プール

なんと、ここにも NSAutoreleasePool が作られています。

そう、NSAutoreleasePool オブジェクトは積み重ねられるようになっているのです。 そして、いつでも一番上の NSAutoreleasePool が登録対象になります。



少し実験してみましょう。

以下の内容を RootViewController.m の先頭に書き加えます。

RootViewController.m

```
    ・・・
#import "RootViewController.h"

@interface AutoreleasePoolTester : NSObject {
}
@end

@implementation AutoreleasePoolTester
-(void)dealloc {
    [super dealloc];
}
@end


@interface RootViewController ()

@end

@implementation RootViewController
    ・・・
```

新しいオブジェクトのクラス宣言と実装

これは新しいオブジェクトのクラス宣言と実装です。

```
@interface AutoreleasePoolTester : NSObject {
}
@end
```

がクラス宣言部で

```
@implementation AutoreleasePoolTester
-(void)dealloc {
    [super dealloc];
}
@end
```

がその実装部です。

ここでは AutoreleasePoolTester という名前のオブジェクトは NSObject を継承したクラスだと宣言しています。

77 ページ

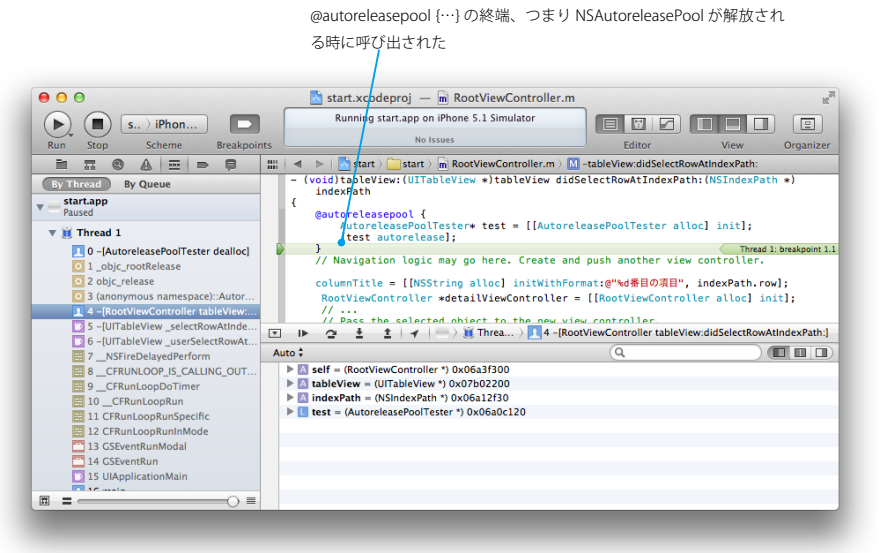
NSAutoreleasePool は @autoreleasepool {…} という形で利用するようになりました。

さて、シミュレータを終了して Xcode に戻りましょう。今度は @autoreleasepool {…} を使い NSAutoreleasePool を作って囲んでみます。

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
  @autoreleasepool {
      AutoreleasePoolTester* test = [[AutoreleasePoolTester alloc] init];
      [test autorelease];
  }
  // Navigation logic may go here. Create and push another view controller.
    columnTitle = [[NSString alloc] initWithFormat:@"%d 番目の項目 ", indexPath.row];
```

79 ページ

NSAutoreleasePool は @autoreleasepool {…} という形で利用するようになりました。



私たちが記述した @autoreleasepool {…} の終端、つまり NSAutoreleasePool が解放される時に、test オブ ジェクトも解放されたのがわかります。しかし、このような事をする必要があるのでしょ うか？

81 ページ

NSAutoreleasePool は @autoreleasepool {…} という形で利用するようになりました。

では、その後にもう一度 autorelease メッセージを投げられた場合どうなるのでしょう？

```
@autoreleasepool {
    AutoreleasePoolTester* test = [[AutoreleasePoolTester alloc] init];
    [test autorelease];
    [test retain];
    [test autorelease];
}
```

82 ページ

NSAutoreleasePool は @autoreleasepool {…} という形で利用するようになりました。

では、これはどうでしょう？

```
@autoreleasepool {
    AutoreleasePoolTester* test = [[AutoreleasePoolTester alloc] init];
    [test autorelease];
    [test release];
    [test retain];
}
```

答えは「わからない」です。

86 ページ ・・・ 87 ページ

Instruments の配置が変わりました。

87 ページで説明されているスキームの変更による起動も可能ですが、ここでは Instruments の診断計測器選択画面を使う方法を紹介します。

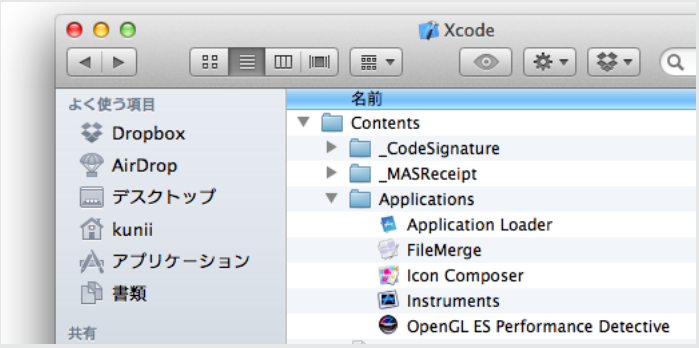
5.2 Instruments

Instruments は Xcode の内部に格納されたアプリケーションで、メモリの使用状況を診断したり、イベントを監視したりする、いろいろな計測器 (Instrument) の集まりです。

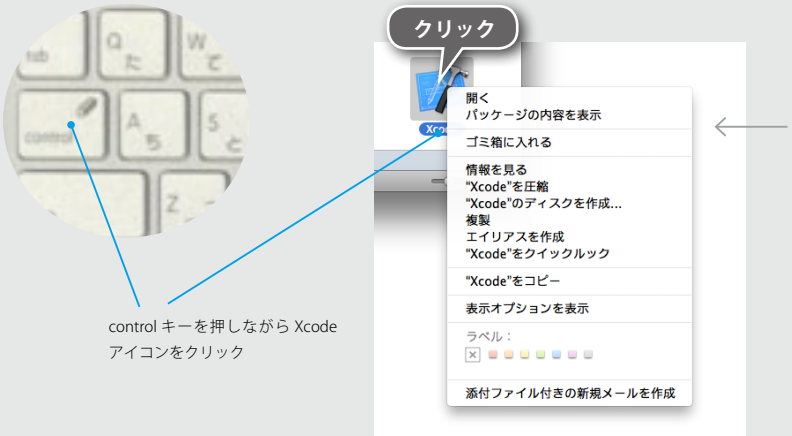
以前、Instruments は Xcode と同じ階層に配置されていたのですが、Xcode バージョン 4.3.2 から Xcode の内部に配置されるようになりました。

Xcode もそうですが、アプリケーションは 1 つのファイルのように振る舞いますが、実際はフォルダです。

この Xcode の内部にある Contents フォルダの中の Applications フォルダに Instruments は格納されています。



control キーを押しながら Xcode をクリックするとコンテキストメニューが現れるので、その中から " パッケージの内容を表示 " を選ぶ事で、中身を見ることができます。

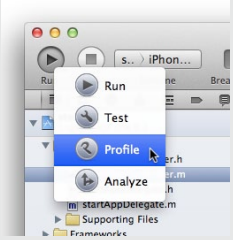


ここでは、その中でよく使われるメモリーリーク計測器の使い方を紹介しましょう。

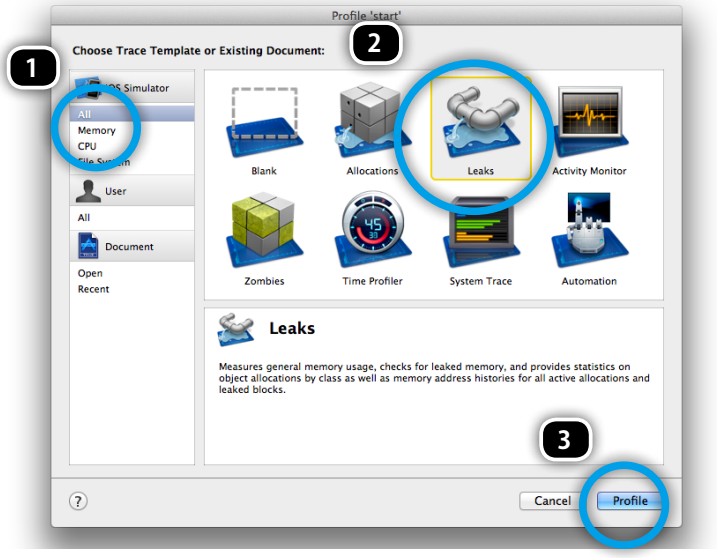
Instruments は単独でも実行できるようになっていますが、基本的に Xcode から連携して使います。もし Xcode が起動していなければ、start.xcodeproj をダブルクリック して Xcode を起動してください。

そして Product → Profile メニューを選びます。

Run ボタンを以下のメニューがポップアップするまで押し続け、Profile メニューを選択する方法もあります。



計測器選択画面が現れるので、左側の分類から iOS Simulator の All を選び、右画面から Leaks 計測器アイコンを選択して Profile ボタンを押してください。



さて、Instruments は立ち上がりましたか？ これがアプリケーションを診断してくれる計測器群です。

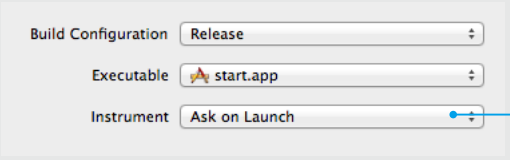
計測器選択画面を出さずに、毎回特定の計測器で Instruments を起動するには

計測器選択画面を出さずに、すぐに特定の計測器で計測を始めるようにもできます。その場合、スキーム編集メニューをポップアップさせて Edit Scheme…を選んでください。

スキーム編集ウィンドウが表示されます。

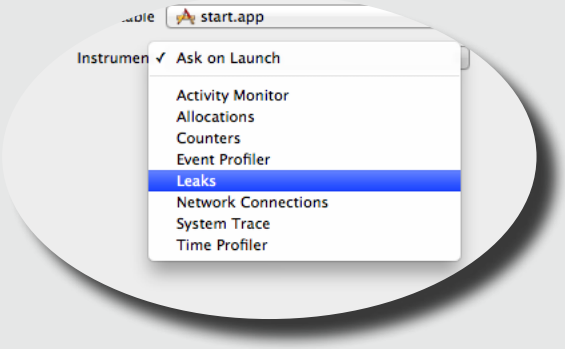
スキーム編集ウィンドウの左のリストから Profile start.app を選ぶと、Product → Profile メニュー実行時の Instruments の振る舞いを設定できる画面が現れます。

ここで、Instruments のポップアップメニューを起動時に尋ねる「Ask on Lanunch」から、特定の計測器に変更する事で、Product → Profile メニューによる実行時は選択された計測器で Instruments が実行される事になります。



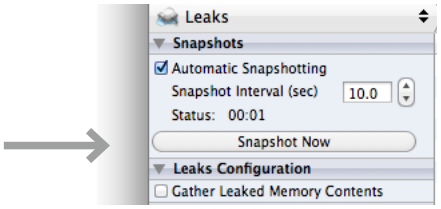
Instruments 起動時の振る舞いを指定している

ポップアップさせ、例えば Learks を選ぶと、計測器選択画面は出さずに Learks 計測器での計測が開始される



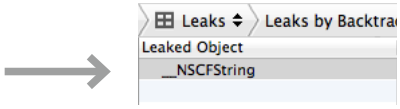
90 ページ

Check for Leaks Now ボタンは Snapshort Now に変更されました。



91 ページ

NSCFString ではなく __NSCFString です。



リーク発生法について

92 ページで補足説明している「一度ナビゲーションバーの戻るボタンを押して、もう一度いずれ かの項目をタップしてください。そして Check for Leaks Now ボタンを押します。」というやり方の方が発生させやすくなっています。

95 ページ

stringWithFormat: と retain の組み合わせでも同じなのですが、4 章からの流れとしては alloc、initWithFormat: で説明した方が適切なので、そちらに変更します。

```
columnTitle = [[NSString alloc] initWithFormat:@"%d 番目の項目 ", indexPath.row]];
```

96 ページ

こちらも alloc、initWithFormat: で説明します。

```
columnTitle = [[NSString alloc] initWithFormat:@"%d 番目の項目 ", indexPath.row]];
```

といっても処理自体は変わりません。しかし今回の columnTitle には先ほど私たちが NSString の alloc、initWithFormat: で所有している NSString オブジェクトが入っています。

102 ページ

履歴リストの項目の並びが違う場合があります。
項目の順序はドラッグして並び替える事が可能です。

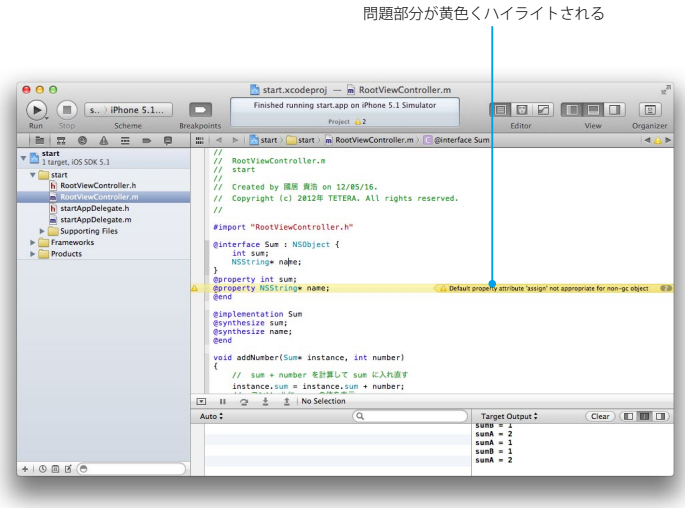
項目の順序はドラッグして並び替える事が可能です

#	Address	Category	Event Type	RefCt	Timestamp	Size	Responsible Lib...
0	0x6b1acf0	CFString (immut...	Malloc	1	00:32.771.304	32	Foundation
1	0x6b1acf0	CFString (immut...	Autorelease		00:32.771.315	0	Foundation
2	0x6b1acf0	CFString (immut...	CFRetain	2	00:32.771.318	0	start

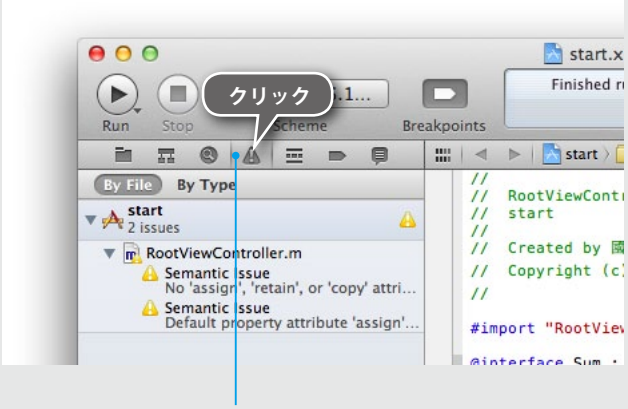
125 ページ

ナビゲーションエリアは自動で Issue Navigator 画面に変わらなくなりました。

エディタエリアには黄色くハイライトされている行があります。



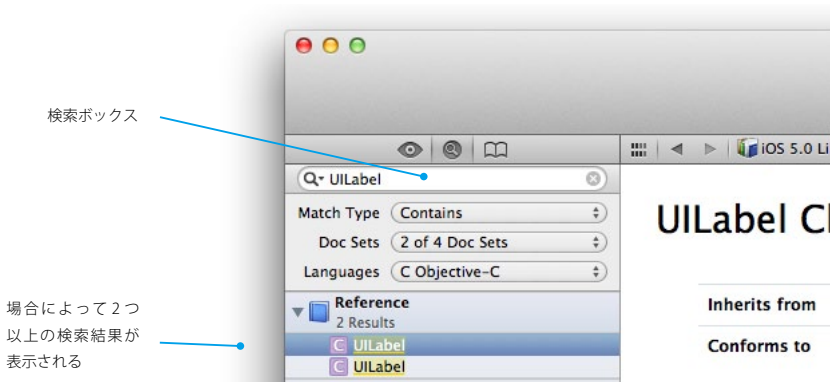
Issue Navigator 画面は、手動で Show the Issue navigator タブをクリックする事で表示されます。



129 ページ

UILabel のクイックヘルプ検索方法を補足しておきます。

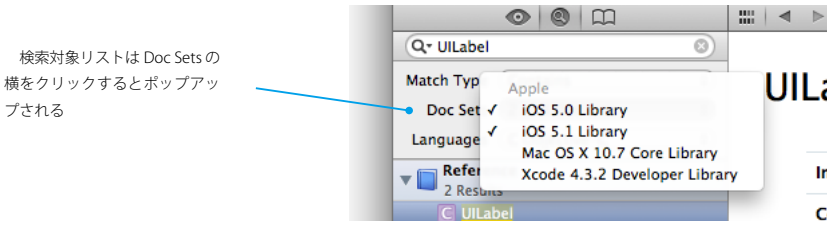
エディタエリアで一時的に UILabel と書いて、Quick Help を呼び出すというのでもいいのですが、UITableView Cell Class Reference が表示されているウィンドウ（Organizer ウィンドウと呼ばれます）の左にある検索ボックスに "UILabel" と書き込み検索する方法が簡単です。



場合によって 2 つ以上の同名の検索結果が表示されます。

これは、検索対象を複数設定している場合に発生します。

例えば下記の場合なら、iOS 5.0 Library、iOS 5.1 Library の 2 つのリファレンスで UILabel の説明が見つかったので 2 つ表示されるわけです。



Doc Sets のポップアップメニューのチェックを外したり付けたりする事で、検索対象を指定する事もできます。

どちらのリファレンスかはジャンプバーで確認できます。

136 ページ ・・・ 137 ページ

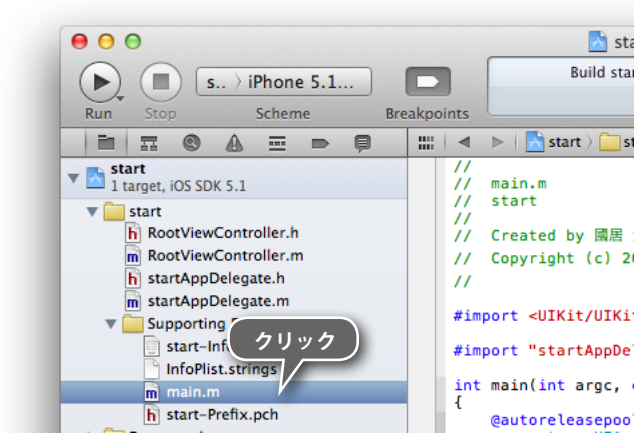
テンプレートで用意される main 関数の内容が変わりました。

136、137 ページを右の内容で置き換えてください。

以前、main 関数をみた時「これだけで、なぜあれだけ多種多様なアプリケーションに変化するのでしょう ?」と言いました。

まず、その疑問から始めましょう。

Project Navigator の start グループ /Supporting Files グループに入っている main.m を選んでください。エディットエリアに内容が表示されます。



main 関数は Empty Application テンプレートに限らず、Apple が用意したすべてのテンプレートでこのようになっています（"startAppDelegate" の部分は変わります）。

main.m

```
#import "startAppDelegate.h"

int main(int argc, char *argv[])
{
  @autoreleasepool {
    return UIApplicationMain(argc, argv, nil, NSStringFromClass([startAppDelegate class]));
  }
}
```

startAppDelegate の部分はプロジェクト作成時に設定する Prefix によって変化する

では、どうしてここからさまざまなアプリケーションに変化するのでしょうか ?

7.1 UIApplicationMain

既に私たちは @autoreleasepool {…} が何を意味するかを知っています。ということは上記疑問を解決するために調べる目標は UIApplicationMain 関数だけということに なります。

クィックヘルプでみた UIApplicationMain 関数の宣言はこうなっています。

```
int UIApplicationMain (
  int argc,
  char *argv[],
  NSString *principalClassName,
  NSString *delegateClassName
);
```

その後、各パラメータの説明が続きます。

各パラメータの詳しい内容は、日本語ドキュメント「iOS アプリケーションプログラミングガイド」の「アプリケーションの状態とマルチタスキンク」の章の「main 関数について」にも記述されています。

各パラメータの役割りをまとめると、次のようになります。

argc
argv 配列の総数。iOS アプリケーションでは 1。

argv
C 文字列の配列。iOS アプリケーションでは argv[0] にアプリケーションファイルの位置が unix パスが入るだけ。

principalClassName
アプリケーションの運用に使用したいオブジェクトのクラス名。UIApplication を継承したクラスでなければならない。nil の場合 UIApplication が使用される。

delegateClassName
アプリケーションの補助運用に使用したいオブジェクトのクラス名。UIApplicationDelegate を継承したクラスでなければならない。nil の場合アプリケーションのメイン nib ファイルからロードされるオブジェクトの 1 つであることを前提にします (Xcode のテンプレートを使用して作成されたアプリケーションの場合)。

まず、main.m では principalClassName が nil ということから、アプリケーションの運用 に UIApplication オブジェクトが使用される事がわかるでしょうか。

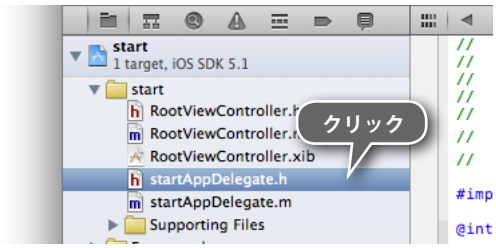
7.2 UIApplication

そうです。このオブジェクトこそが「4 章 泡のように消えるオブジェクト」で説明した、主なイベントにそって処理をおこなうオブジェクトです。

140 ページ ・・・ 141 ページ

現在の Xcode のテンプレートプロジェクトは、第 4 引数の delegateClassName を指定するように変更されています。

140 ページと 141 ページを右の内容で置き換えてください。



その中に UIApplicationDelegate の単語が見つかります。

```
@interface startAppDelegate : UIResponder <UIApplicationDelegate>
```

注意）UIResponder は NSObject を継承したクラスで、ユーザーの入力に対応する機能を持つクラスです。

そうです、この startAppDelegate オブジェクトこそが、私たちのアプリケーションを独自なものにするオブジェクトです。

UIApplicationMain 関数に、startAppDelegate オブジェクトを利用させるには第 4 引数の principalClassName として startAppDelegate クラスの文字列を渡す必要があります。

これが

```
int retVal = UIApplicationMain(argc, argv, nil, NSStringFromClass([startAppDelegate class]));
```

という記述の意味です。

NSStringFromClass 関数はクラスオブジェクト（4.10 クラス宣言で説明しました）を受け取って、対応する文字列を NSString オブジェクトで返します。

クラスオブジェクトは

```
[ クラス名 class]
```

と記述する事で取り出せます。

想像した人もいると思いますが、このような書き方をせずに

```
int retVal = UIApplicationMain(argc, argv, nil, @"startAppDelegate");
```

と記述しても問題なく動きます。ただし、こちらの記述法だとクラス名をスペルミスしていても実行できてしまいます。その場合アプリケーションは異常終了します。

ところで startAppDelegate クラスの継承の仕方がいままでのものと違うようです。

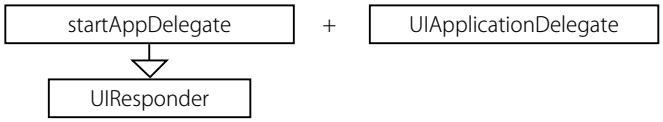
startAppDelegate クラスは UIApplicationDelegate を継承しているように見える のですが、<> は一体何を意味するのでしょうか？

```
@interface startAppDelegate : UIResponder <UIApplicationDelegate>
```

7.4 プロトコル宣言

<> は startAppDelegate クラス宣言が UIResponder クラス宣言を継承し、UIApplication Delegate プロトコル宣言も継承するということを意味します。

注意）Apple のドキュメントではプロトコルの方は継承するとは呼ばず、組み込むと呼んでいます。



UIApplicationDelegate はクラス宣言ではなくプロトコル宣言されたものなのです。ではプロトコル宣言とは何でしょう？

それを知るためには「クラス継承の決まり」を理解する必要があります。

まず、クラス継承では継承元のクラスに定義されたインスタンス変数と同名のインス タンス変数を持てません。

例えば以下のようにstartAppDelegateを継承したexStartAppDelegateクラス宣言をstartAppDelegate.hに書き込んでみてください。

startAppDelegate.h

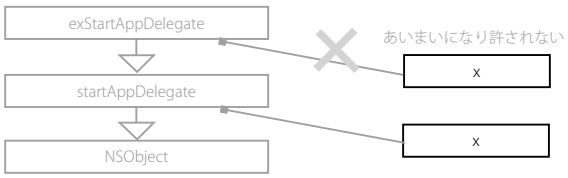
```
#import <UIKit/UIKit.h>

@interface startAppDelegate : UIResponder <UIApplicationDelegate>{
  id x;
}
@property (strong, nonatomic) UIWindow *window;

@end

@interface exStartAppDelegate : startAppDelegate {
  id x;
}
@end
```

書き込んだとたんに左のガターにエラーアイコンが表示されたと思います。

エラーアイコンをクリックすると「xというメンバが複製されている」というメッセージ(Duplicate member "x") が表示されています。

142 ページ

140 ～ 141 ページの変更に合わせ、インスタンス変数windowはxと読み替えてください。



これを確認する方法は、以下のようにexStartAppDelegateクラス宣言からxインスタンス変数を削除して、少々強引なことになりますが「startAppDelegate,NSObject」と書いてビルドしてみれば確認することができます。

144 ページ

コンパイルせずとも、即座にワーニングアイコンが出るようになりました。

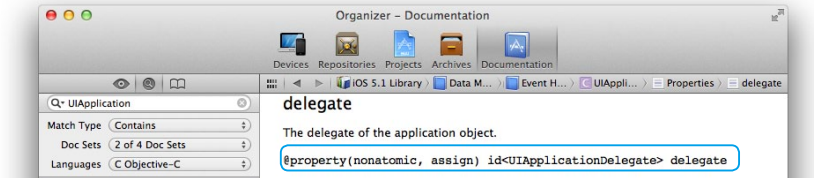
これはプロトコル宣言に対して @implementation は使えないということです。例えば上の @protocol X に対して、以下のような @implementation を使ったメソッドの実装はできません。

書き込むと即座にワーニングアイコンが表示され、クリックすると「X というクラス宣言は見つからない」という ワーニングが表示されます。

145 ページ

説明に UIApplication のヘッダーではなく、クラスリファレンスの delegate プロパティを使います。

解決法のヒントは UIApplication のクラスリファレンスにあります。UILabel と同じように検索してみてください。




```
@property(nonatomic, assign) id <UIApplicationDelegate> delegate
```

148 ページ・・・168 ページ

現在の Xcode では第 4 引数（delegateClassName）にクラスの名を指定するようになっていて、オリジナルのような MainWindow.xib を設定するようにはなっていません。

そのため 148 ページから 168 ページまでの話は不要となります。

ただし、この XIB ファイルからオブジェクトを作る知識は、いろいろなところで利用されるので、学習しておいた方がいいと思います。

start-149

という MainWindow.xib を設定したサンプルプロジェクトを提供しておきますので、これを元に、148 ページから 168 ページを読み進めてください。

また、start-148 から MainWindow.xib を追加して start-149 を作成する方法を右に示します。

start-149 の作成方法

こちらは 148 ページから 168 ページまでの学習を終えた後の復習用として利用してください。

まず、start-148 をフォルダごと複製し start-149 とします。
そして

main.m
startAppDelegate.h
startAppDelegate.m

上記 3 つのソースファイルを以下のように変更します。

main.m

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

~~#import "startAppDelegate.h"~~        ← 不要なので削除する

int main(int argc, char *argv[])
{
  @autoreleasepool {
    return UIApplicationMain(argc, argv, nil, nil);   ← nil にする
  }
}
```

startAppDelegate.h

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

@interface startAppDelegate : UIResponder <UIApplicationDelegate>
@property (strong, nonatomic) IBOutlet UIWindow *window;   ← 追加

@property (strong, nonatomic) IBOutlet UINavigationController *navigationController;   ← 追加

@end
```

注）　Automatic Reference Counting を使わない場合、@property の属性 strong は retain を意味します。

startAppDelegate.m

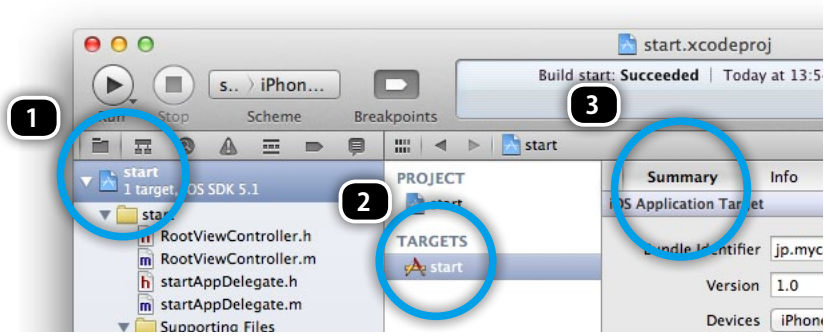
```
    ・・・
#import "startAppDelegate.h"
~~#import "RootViewController.h"~~        不要なので削除する

    ・・・

@implementation startAppDelegate

@synthesize window = _window;
@synthesize navigationController;         追加

    ・・・
- (void)dealloc
{
  [_window release];
  [navigationController release];         追加
  [super dealloc];
}

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)
launchOptions
{
    Tester* tester = [[Tester alloc] init];
    tester.target = [[anotherDelegate alloc] init];
    [tester test];                                    不要なので削除する

~~self.window = [[[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]] autorelease];~~
  // Override point for customization after application launch.
~~self.window.backgroundColor = [UIColor whiteColor];~~   不要なので削除する
  [self.window makeKeyAndVisible];

~~RootViewController* viewcontroller = [[[RootViewController alloc]init] autorelease];~~
~~self.window.rootViewController = [[[UINavigationController alloc]~~
~~initWithRootViewController:viewcontroller]autorelease];~~
                                                      不要なので削除する
  return YES;
}
    ・・・
```

次に、UIApplicationMain 関数の第 4 引数を nil にしたので、利用する xib ファイルを用意、指定します。

Main Interface の設定

利用する xib ファイルの指定には Main Interface を設定します。

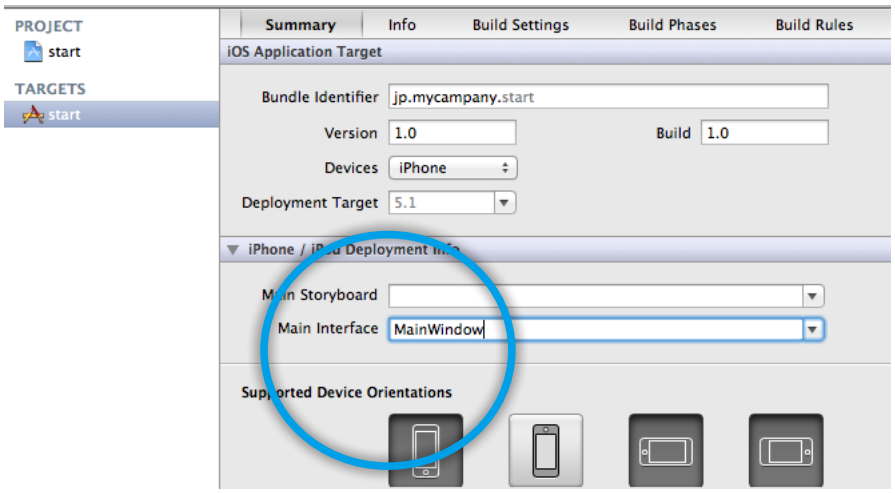
start プロジェクトアイコンを選び、エディタエリアの TARGETS の start タブを選び、右に表示される画面で、上部の Summary タブを選びます。



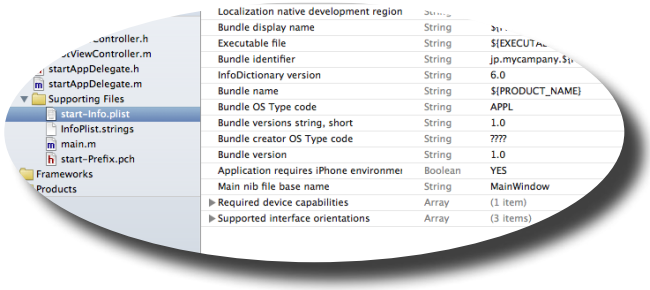
Summary タブ画面を下にスクロールすると Main Interface という項目が現れるので

MainWindow

と記入します。名称は任意ですが慣例で MainWindow としました。



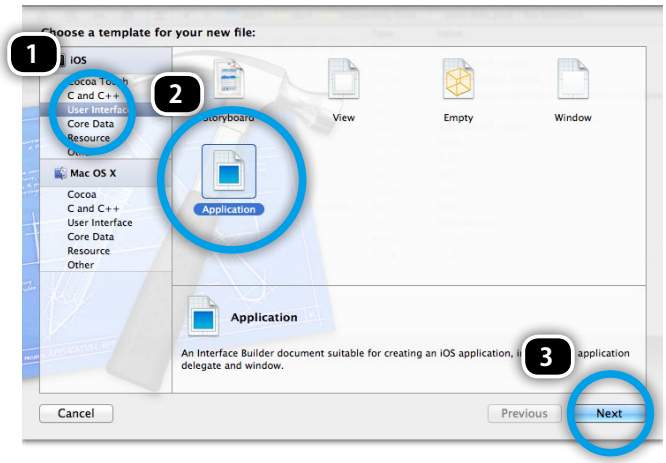
連動し start-Info.plist に Main nib file base name の項目が自動的に追加される



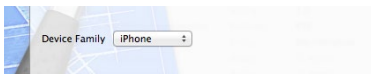
今度は Main Interface で指定した MainWindow.xib ファイルを作成します。

MainWindow.xib ファイルの作成

File → New → File メニューを選び、現れたファイルテンプレート選択画面で、iOS の User Interface タブを選び、表示されたテンプレートから Application を選び Next ボタンを押します。



次の画面では iPhone 用を選択し、Next ボタンを押します。



ファイル出力先指定画面ではファイル名を

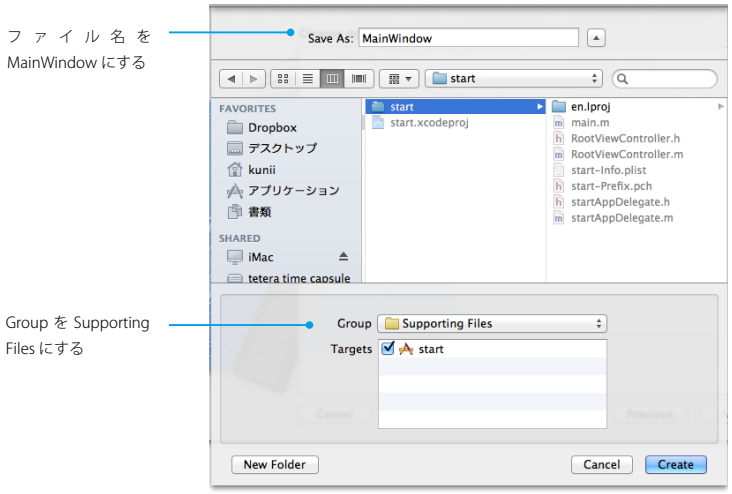
MainWindow

にし、Group に

Supporting Files

を選んで Create ボタンを押します。

> Group を Supporting Files にするのは慣例に従っているだけです。必ず Supporting Files でなければいけないわけではありません。



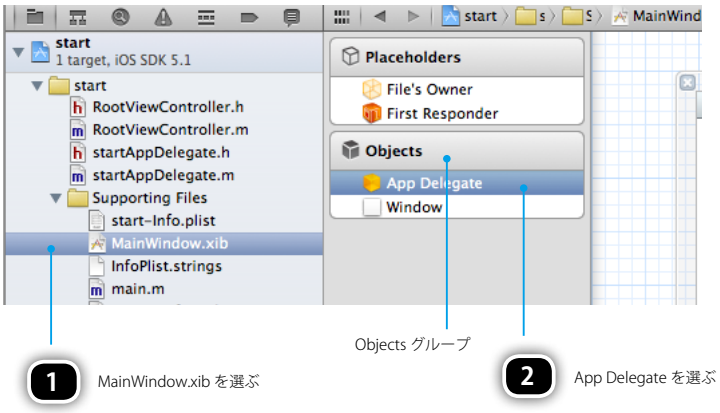
MainWindow.xib ファイルがプロジェクトに登録されるので、最後に MainWindow.xib 側の設定をおこないます。

MainWindow.xib の設定

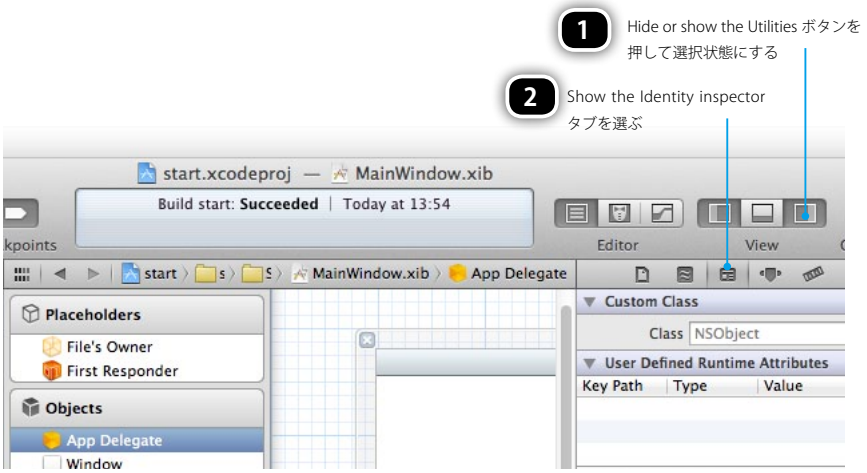
ナビゲーションエリアのリストから、追加された MainWindow.xib を選びます。

1、App Delegate オブジェクトのクラスを startAppDelegate に変更する

エディタエリアの Objects グループに MainWindow.xib が含んでいるオブジェクトのリストが現れるので、その中の App Delegate を選択します。



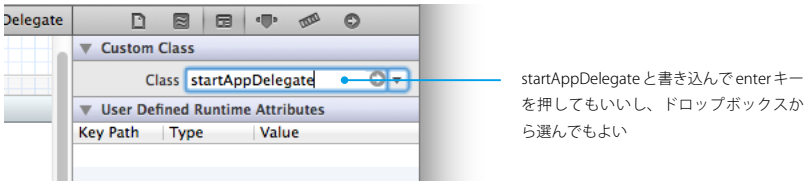
そして Hide or show the Utilities ボタンを押し選択状態にし、ユーティリティエリアを表示させ、Show the Identity inspector タブを選びます。



表示された Identity inspector の Class 項目を

startAppDelegate

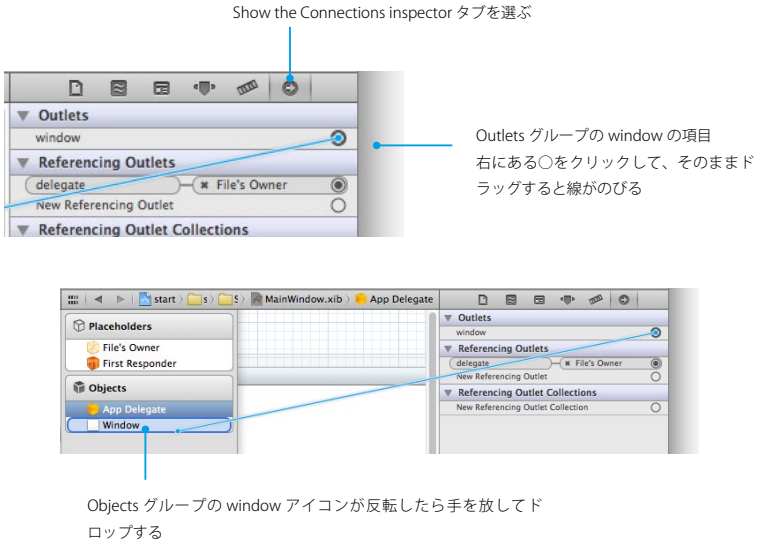
に変更します。



2、App Delegate オブジェクトの window outlet を Window オブジェクトと結びつける

Objects グループの App Delegate が選ばれた状態で、Show the Connections inspector タブを選びます。そして表示された Connections 画面の Outlets グループにある window の項目と Objects グループの window アイコンを結んでください。

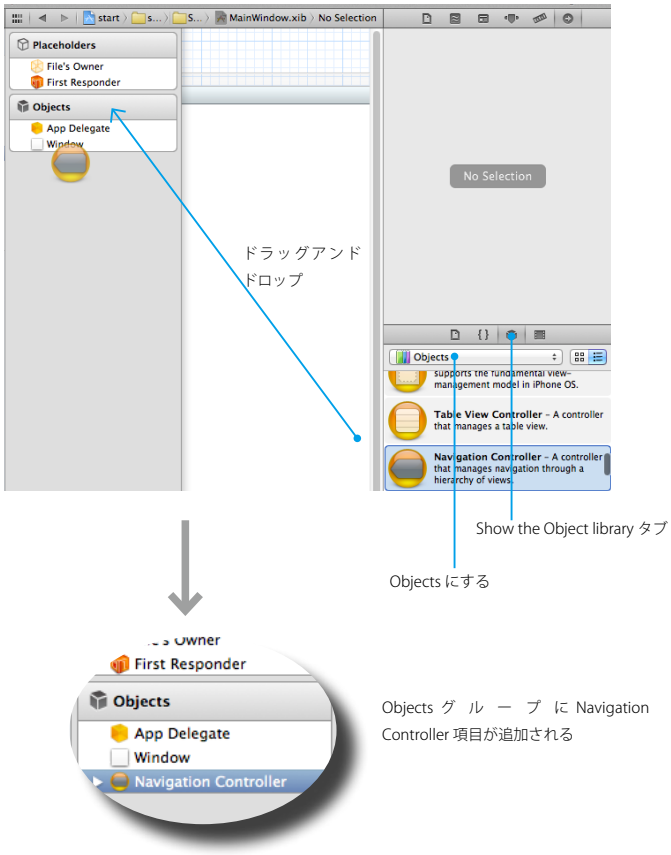
Outlets グループの window の項目右にある○をクリックして、そのままドラッグすると線がのびます。



3、Navigation Controller オブジェクトの挿入

Show the Object library タブを選んでください。

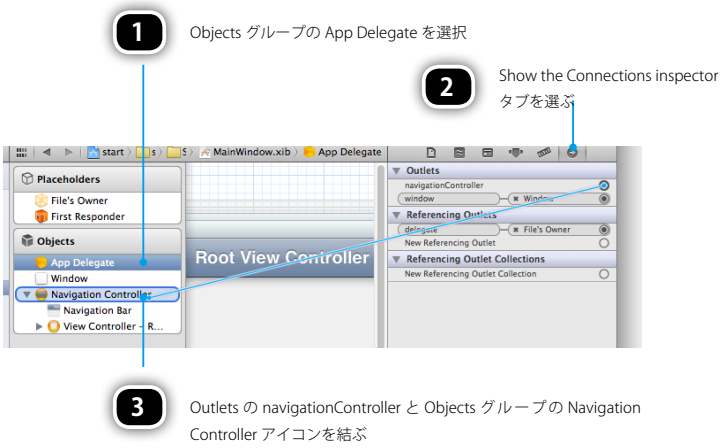
表示された Object library 画面の Objects リストの中から、Navigation Controller アイコンを Objects グループにドラッグアンドドロップします。



Navigation Controller が Objects グループに追加されます。

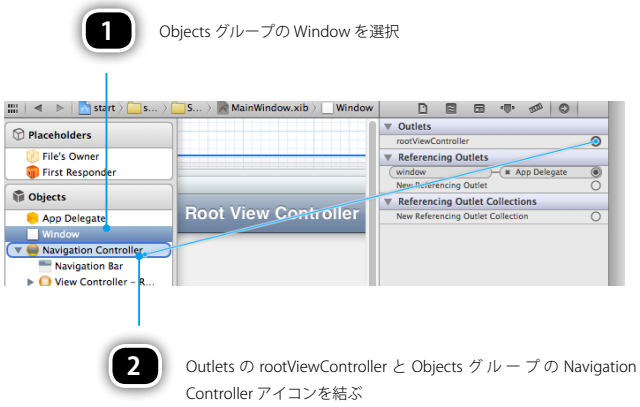
4、App Delegate の navigationController outlet を Navigation Controller オブジェクトと結びつける

Objects グループの App Delegate を選択し、Show the Connections inspector タブを選びます。そして表示された Outlets グループの navigationController と先ほどドロップした Navigation Controller アイコンを window を結んだ時の要領で結びつけてください。



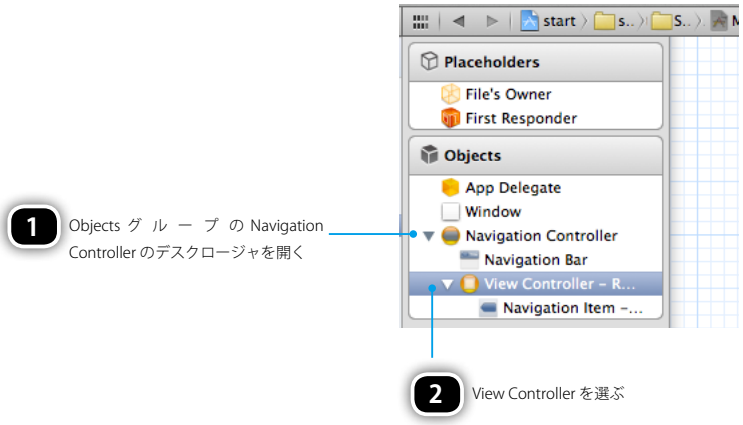
5、Window オブジェクトの rootViewController outlet を Navigation Controller オブジェクトと結びつける

Objects グループの Window を選択し、Outlets の rootViewController と Navigation Controller アイコンを結びます。



6、View Controller オブジェクトのクラスを RootViewController に変更する

Objects グループの Navigation Controller のデスクロージャを開き、中に収まっている View Controller を選びます。

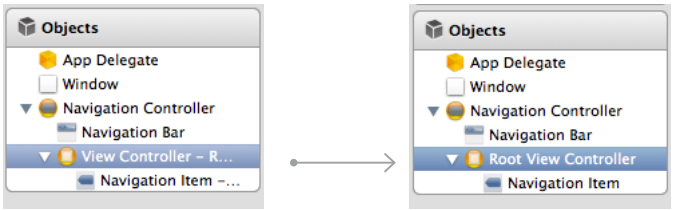


Show the Identity inspector タブを選び Class に

RootViewController

を設定します。

これで Objects グループの View Controller は Root View Controller と表示されるようになります。

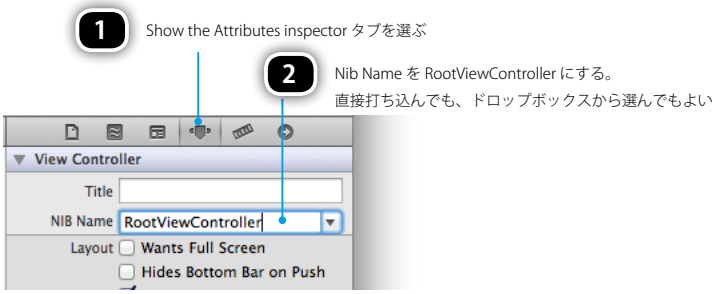


7、Root View Controller オブジェクトの NIB ファイルを設定する

Show the Attributes inspector タブを選び NIB Name に

RootViewController

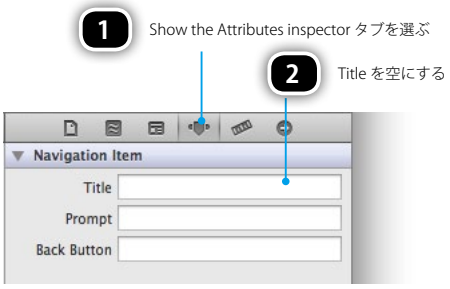
を設定します。



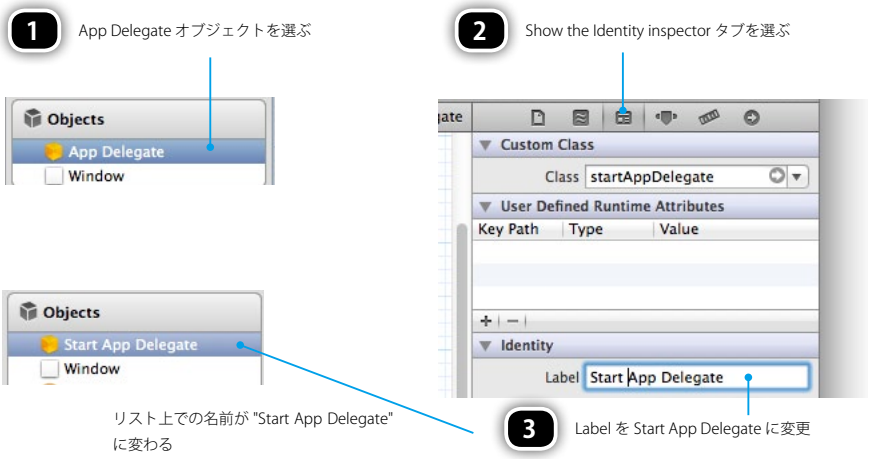
8、Navigation Item オブジェクトの Title を空にする

Objects グループの Root View Controller のデスクロージャを開き、中に収まっている Navigation Item を選びます。

Show the Attributes inspector タブを選び、Title を空にします。



最後に、特に必要ではありませんが、本文と合わせるために startAppDelegate の Objects グループリスト上での名前を "Start App Delegate" にするために、startAppDelegate の Label を変更しておきます。

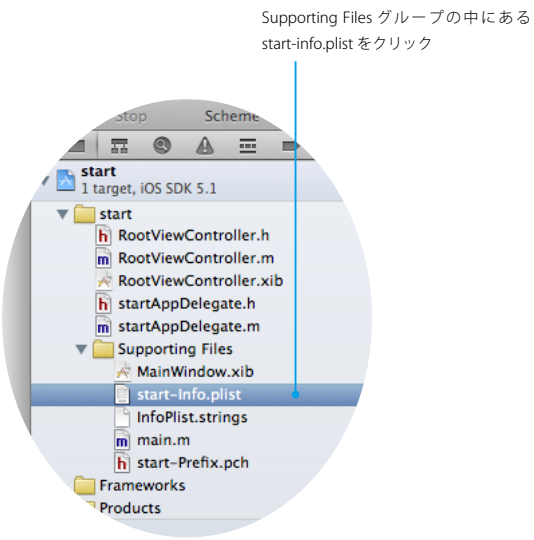


以上です。

149 ページ

start-info.plist の位置が変わりました。

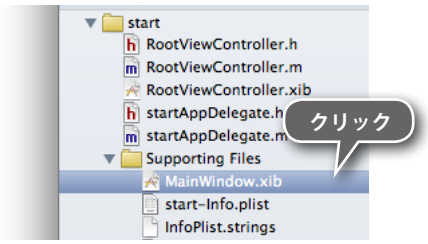
start グループ内の Supporting Files グループの中にある start-info.plist をクリックしてください。



150 ページ

MainWindow.xib の位置が変わりました。

start グループ内の Supporting Files グループの中にある MainWindow.xib をクリックしてください。

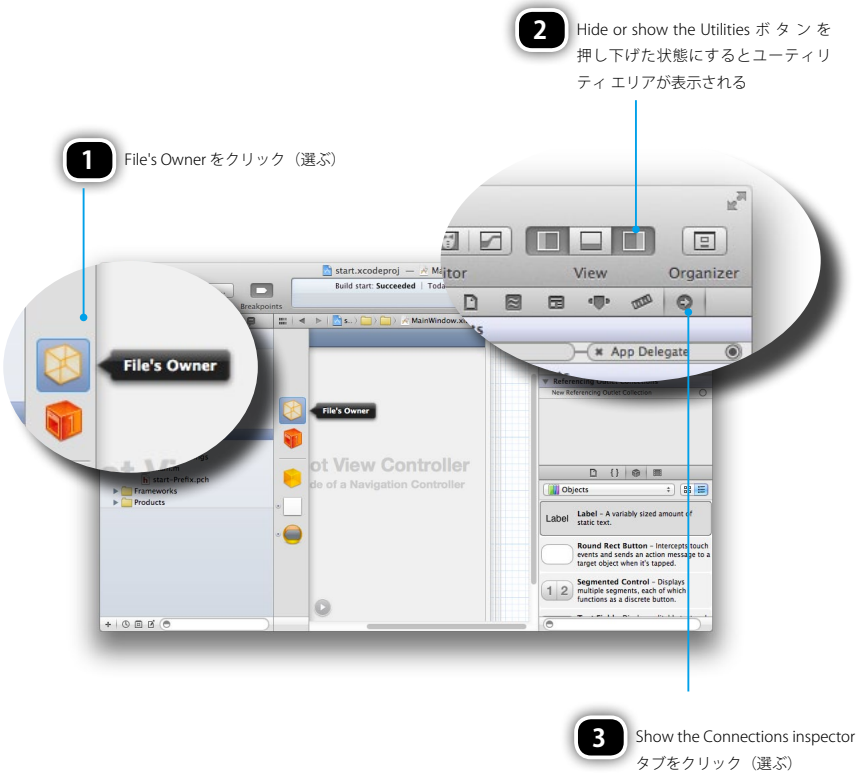


152 ページ

Icon 表示、List 表示切り替えボタンのデザインが変わりました。

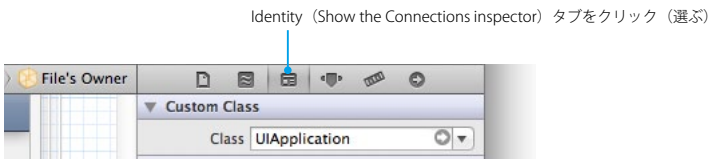
ツールバーのボタンが少し増えました。
タブの位置も変わりました。

しばらく Icon 表示で説明を進めていくことにします。準備ができたらアイコンリスト から File'sOwner を選び、ツールバーにある Hide or show the Utilities ボタンを 押し、表示されたユーティリティ エリアの Show the Connections inspector タブを選びます。



153 ページ

Identity（Show the Identity inspector）タブの位置が変わりました。

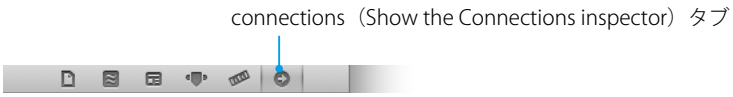


154 ページ

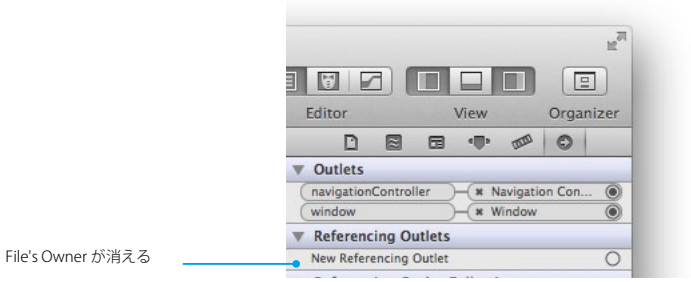
connections（Show the Connections inspector）タブの位置が変わっています。

また、Xcode 4.3.2 では File's Owner との接続を切っても、Window およびテーブル画面が表示されてしまいます。

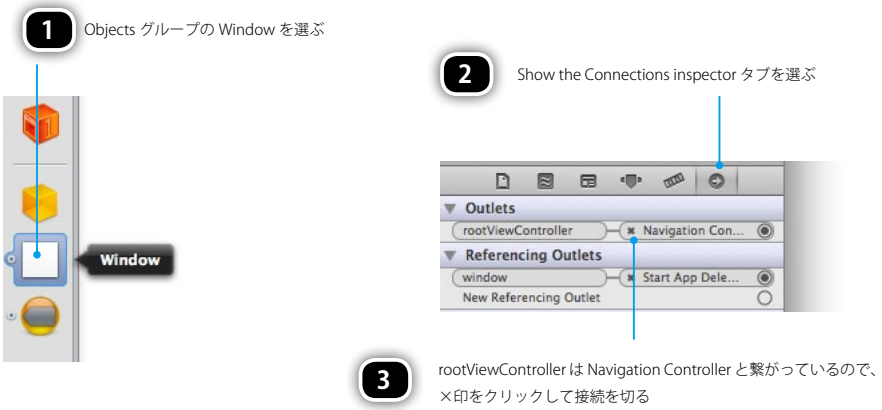
Window オブジェクトの rootViewController の接続も切り離す必要があります。
この場合でも Window は表示されるので、ステータスバーだけが表示されるアプリケーションではなく、ステータスバーと白い画面だけが表示されるアプリケーションとなります。

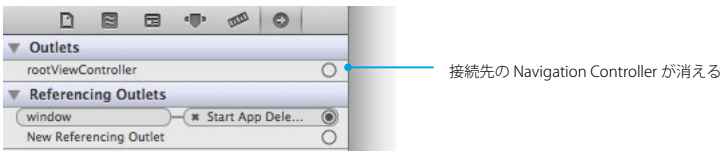


以下のように Referenceing Outlets の項目から File's Owner が消えた事を確認したら、今度は Window オブジェクトの rootViewController の接続を切り離します。



さきほどの要領で、Navigation Controller オブジェクトを選んで、Referenceing Outlets の項目から rootViewController の接続を切ってもいいですし、以下のように Window オブジェクトの Outlets の項目から rootViewController の接続を切る方法でもかまいません。





このやり方は、File's Owner オブジェクトと Start App Delegate オブジェクトの接続を切る場合にも使えます。Outlets、Referenceing Outlets、どちらからでも接続を切る事ができるわけです。

rootViewController と Navigation Controller オブジェクトの接続が切れたのを確認したら Product → Clean メニューを選んだあと、Run ボタンを押して実行させてください。

Xcode 4.3.2 では、接続の変更だけだと、Product → Clean メニューを選んでから Run させないと変更が反映されません。

ステータスバーと白い画面だけが表示されるアプリケーションになってしまいました。
もちろんホームボタンで終了します。確認できたらシミュ レータを終了してください。

166 ページ 167 ページ

iOS バージョン4.0以降では window に navigationController の持つ view を addSubview して、画面と navigationController を関係付けるのではなく、window の rootViewController プロパティに navigationController を設定する事で結びつけます。

そして、この作業は、Interface Builder 側で可能となりました。

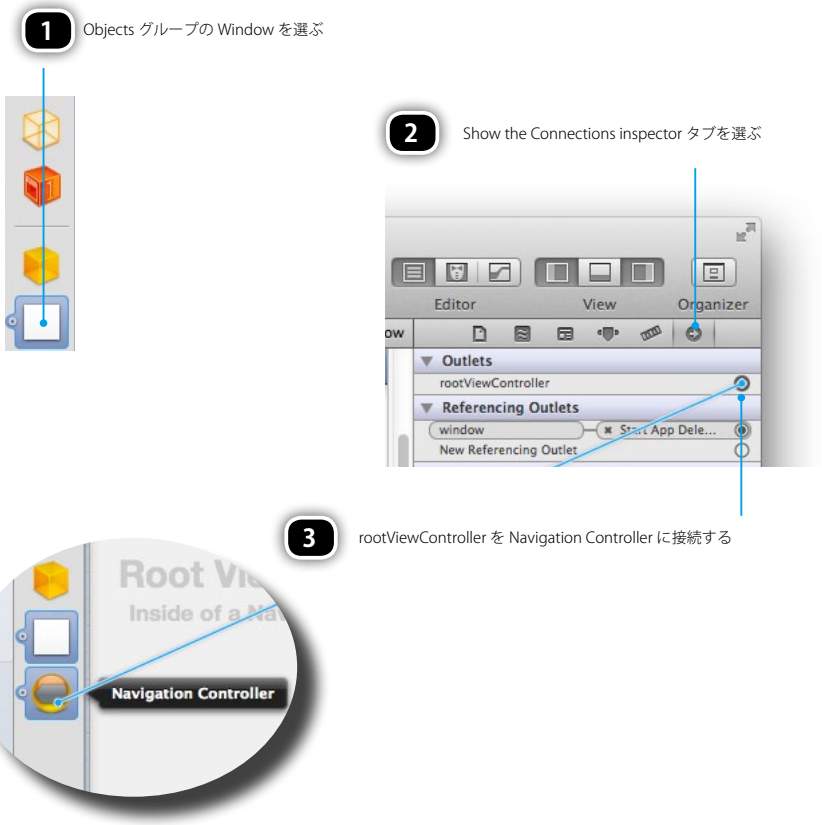
そして私たちの startAppDelegate は、window に navigationController の持つ view を addSubview というメッセージを投げて埋め込む事で、画面と navigationController を関係付けます。

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *) launchOptions {
    [window addSubview:[navigationController view]];
    [window makeKeyAndVisible];
    return YES;
}
```

注意 今のところ、Interface Builder 側で UINavigationController の view を window に結びつける事はできません。

MainWindow.xib を選び、Objects グループの Window を選んでください。

そして、Show the Connections inspector タブを選び、Outlets グループの rootViewController を Objects グループの Navigation Controller に接続してください。



184 ページ

Firework プロジェクト作成手順が変わりました。

184 ページを右の内容で置き換えてください。

これまでに利用したものでは NSString の initWithFormat: メソッドがこれにあたります。詳しく知りたい人は「Objective-C 2.0 プログラミング言語」を読んでください。

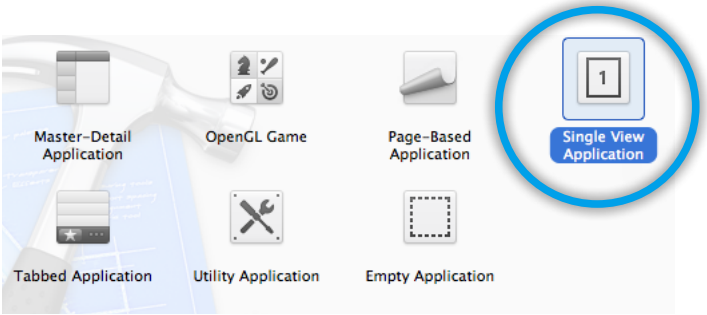
これで FireworkView クラスの設計も終わりです。次から実際の制作にとりかかりましょう。

8.6 Xcode での作業

Xcode で新しいプロジェクトを作りましょう。 今回選択するテンプレートは

Single View Application

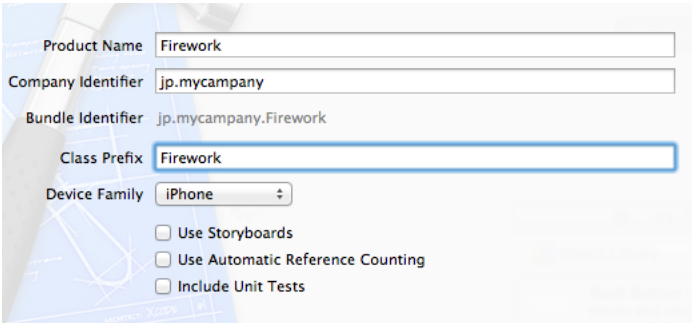
です。



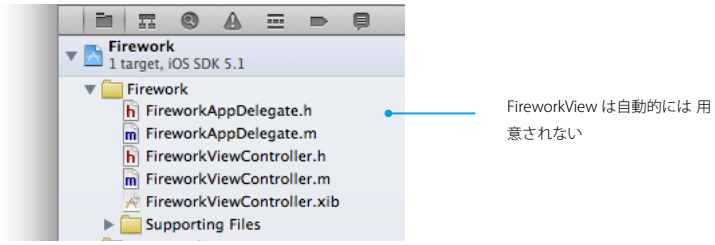
Product Name と Class Prefix はどちらも

Firework

とします。その他の設定は start の時と同じです。



作成されるプロジェクトは、自動的に UIViewController を継承した FireworkView Controller を用意しますが、UIView を継承した FireworkView は用意してくれません。

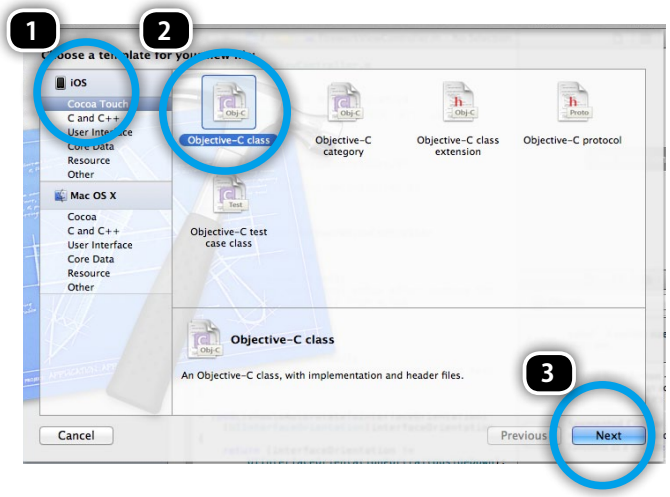


自動作成される UIViewController は「Prefix で指定した名称 ViewController」となります。もし Prefix を Firework 以外として考えた場合、例えば MyTest すると、MyTestViewController となります。

184 ページ　186 ページ

FireworkView.h/m の作成手順が変わりました。

RootViewController.m/h を作った時のように、File → New → File…メニューを選んで、iOS の Cocoa Touch タブから Objective-C class を選んで Next ボタンを押してください。



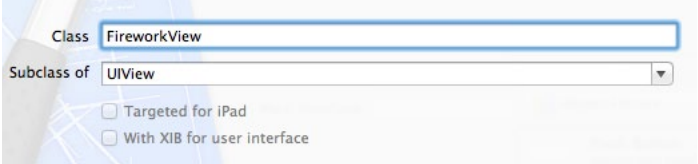
次の画面では Class を

FireworkView

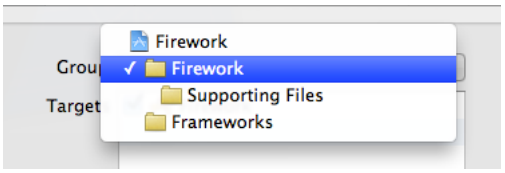
にし、Subclass of は

UIView

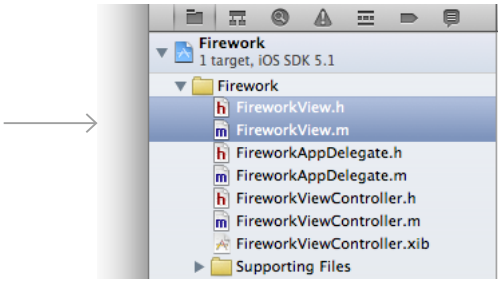
とします。



Group は Firework グループとしました。



Create ボタンを押すと FireworkView.h、FireworkView.m がプロジェクトに追加されます。

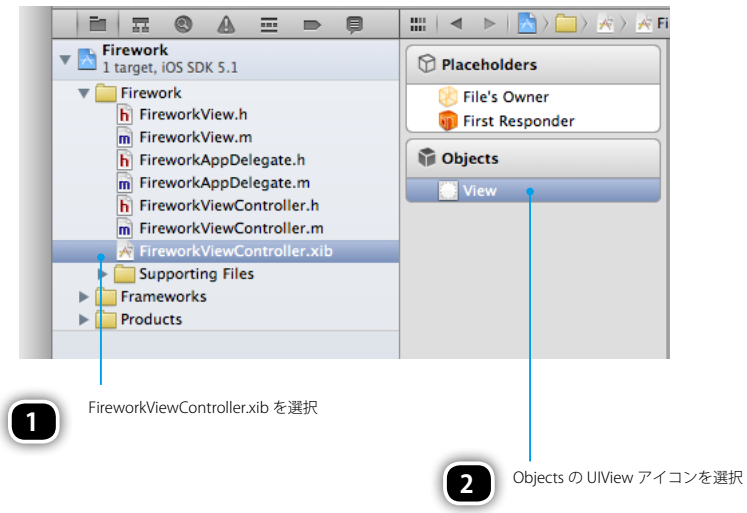


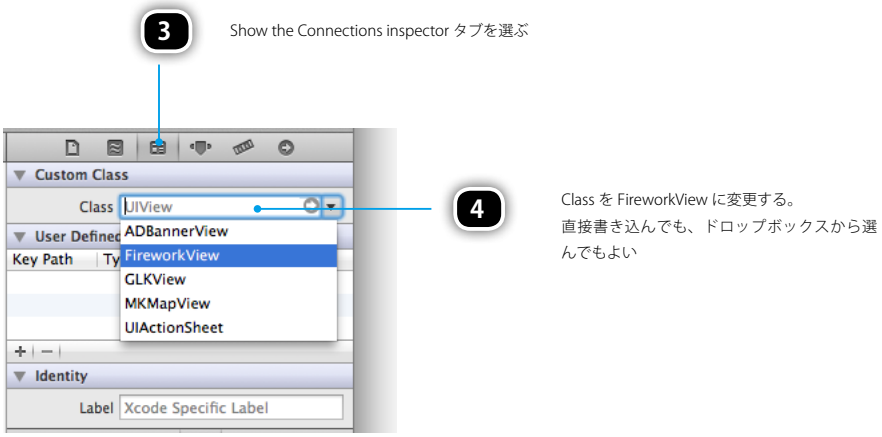
189 ページ

Interface Builder エリアの扱いが変わりました。

ただし、この実装だけでは何も変化しません。なぜなら FirewokViewController の view は UIView のままだからです。

FireworkViewController.xib をクリックして、Interface Builder エリアで Vew オブジェクトのクラスを UIView から FireworkView に変更します。



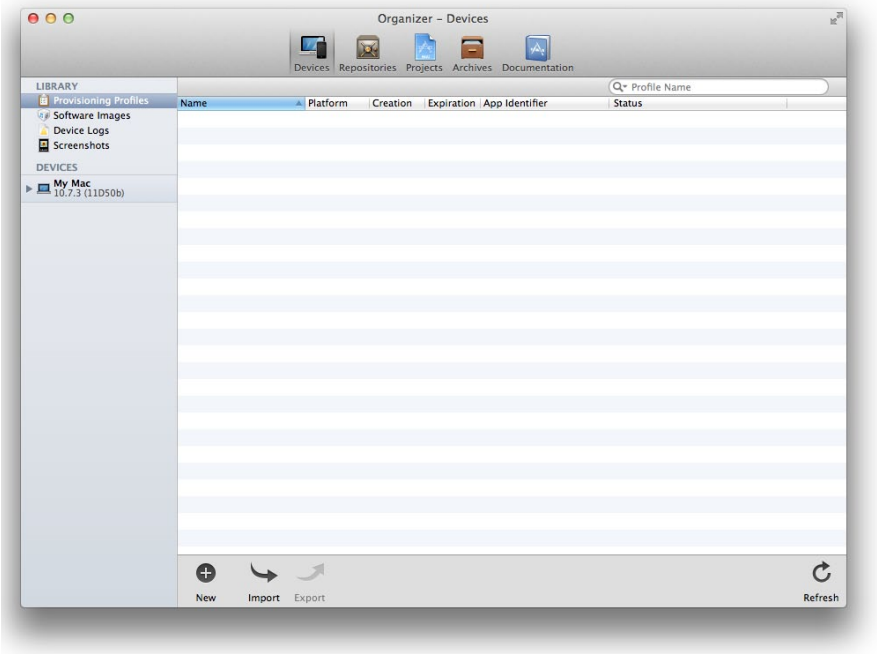


191 ページ ・・・ 205 ページ

191 ページから 205 ページにかけての「8.9 正規開発者証明書請求書類の作成」～「8.10 正規開発者証明書の自分の開発マシン (Mac) への登録」の説明は現在でも有効ですが、Xcode にこの作業を自動する機能が追加されているので右ページで説明します。

iOS Developer Program への入会が完了すると、自分の作ったアプリケーションを実機で動かせる準備が整います。後は正規開発者の証明書を Apple に発行してもらい、Xcode や開発用 Mac、iPhone/iPad/iPod touch 実機に設定します。

まず Xcode を起動して、Window → Organizer メニューを選び Organizer ウィンドウを表示させます。

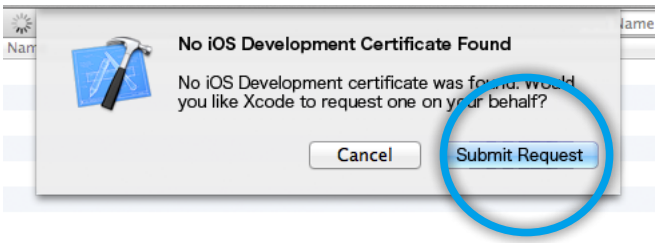


次に Devices タブをクリックし、現れた左のエリアから Provisioning Profiles を選びます。

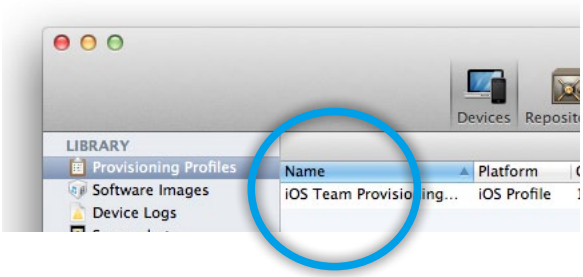
そして右下の Reflesh ボタンをクリックします。

Xcode は iOS Developer Program にサインインすると言い、AppleID とパスワードを聞いてきます。iOS Developer Program に登録した AppleID とパスワードを入力し Log in ボタンを押します。

証明書が見つからないので、Xcode が代行して Apple に要求していいかと尋ねてくるので、Submit Request（要求を送る）ボタンを押します。



Xcode は証明書の準備をおこなってくれ iOS Team Provisioning Profile:* という項目が出現します。



これで Xcode、開発用 Mac への証明書の登録は完了です。
あとは USB ケーブルで iPhone/iPad/iPod touch 実機を Mac とつなぎます。
つないだ実機のアイコンが Organizer ウィンドウに現れるので、アイコンを選び、表示された左の画面で Use for Development ボタンを押します。

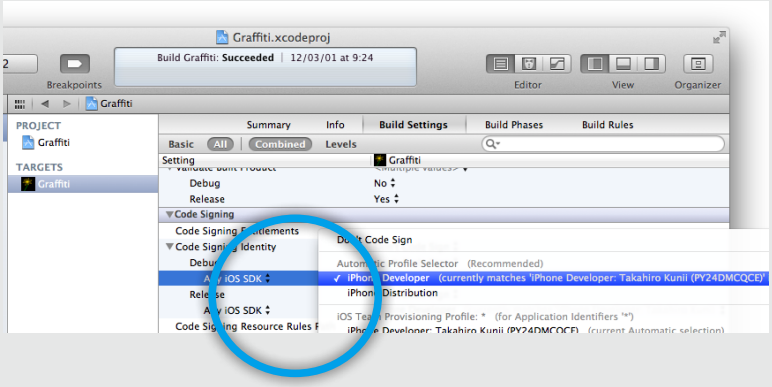
これで実機で自分の作ったアプリケーションを動かす準備ができました。

実機をつないだまま、実行させたいアプリケーションのプロジェクトを開き、実行先を実機に変更します。

iOS Device ではなく、つないだ実機の名前が表示されます。

Run させるとシミュレーションではなくつないだ実機でアプリケーションが実行されます。

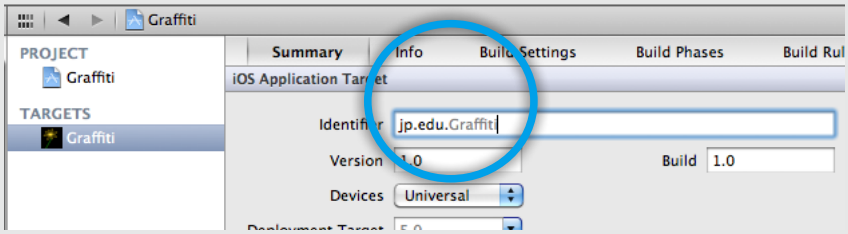
エラーになる場合は、プロジェクトの TARGETS の Build Settings から Code Signing 項目の Code Signing Identity を選びなおす必要があるかもしれません。



通常は、テンプレートが設定している設定のままで大丈夫です。

特別なアプリケーションでは手動での設定が必要です

また、Xcode が用意してくれる iOS Team Provisioning Profile は、アプリケーションの Identifier（識別子）がなんであれ動作させるようになっていますが、プッシュ機能やアプリケーション内課金機能を使うアプリケーションなどは、そのアプリケーション専用の Provisioning Profile を自分で用意する必要があります。



アプリケーションの Identifier（識別子）

アプリケーションごとに違う必要がある。テンプレートを選びプロジェクトを作成した直後は Company Identifier と Product Name が .「ドット」で結ばれたものが設定されている。

詳細は Apple のドキュメント「Your First App Store Submission」の「Provisioning Your Devices for Development」の章で説明されています。また「Technical Note TN2250」も役に立つでしょう。

残念ながら、現在のところ、どちらも英文のドキュメントしかありません。

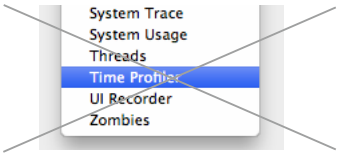
和訳されている Apple のドキュメントで参考になるのは「App Store での公開に向けた開発」です。「iOS ツール ワークフ ローガイト」にも「証明書に関する問題」という形で説明が載っています。App Store への公開手続きに関しては「iTunes Connect デベロッパガイド」が役立ちます。プッシュ機能の詳細は「Local および Push Notification プログラミングガイド」、アプリケーション内課金機能の詳細は「In App Purchase プログラミングガイド」を読むとよいでしょう。

208 ページ

スキーム編集は不要になりました。

スキーム編集は、計測器選択画面を出さずに任意の計測をおこないたい場合に利用してください。

まずは Time Profiler 計測器を指定して Instruments を動かしてみましょう。~~スキーム編集ダイアログを開き、利用する計測器を Time Profiler にします。~~

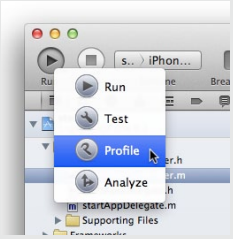


210 ページ

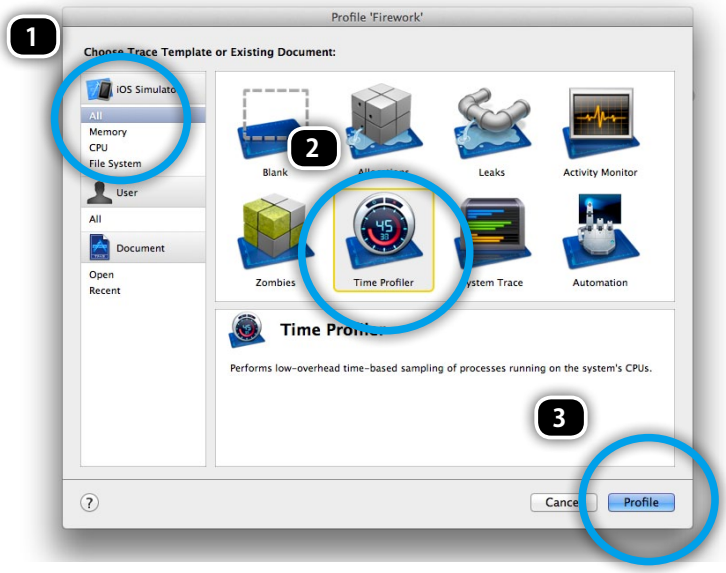
スキーム編集で計測器選択をしていない場合、Instruments を呼び出す作業が右のようになります。

1. ケーブルを繋いだまま Product → Profile メニューを選びます。

Run ボタンを以下のメニューがポップアップするまで押し続け、Profile メニューを選択する方法もあります。



2.Instruments が起動されて、計測器選択画面が出るので、分類を All とし、Time Profiler を選択して Profile ボタンを押します。

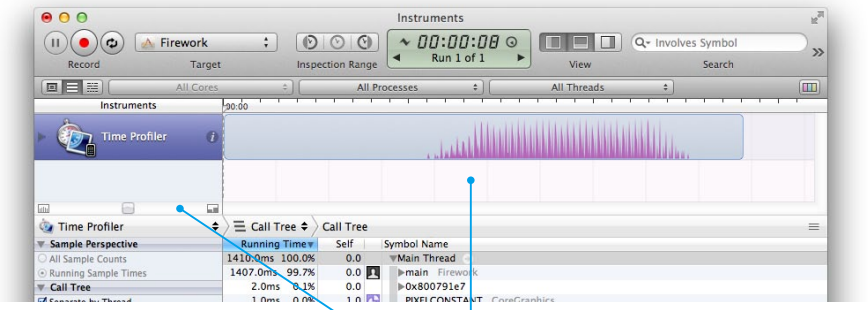


212 ページ

スケールスライダのデザインや Call Tree の設定が少し異なります。

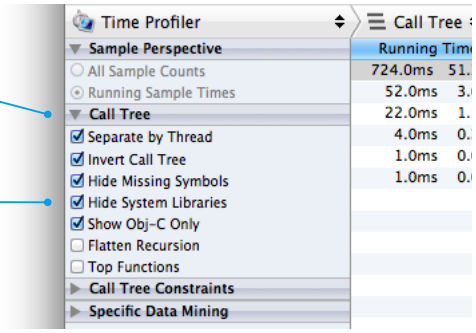
Call Tree の設定では Separate by Thread が最初からチェックされている状態です。

作業には支障がないので、そのまま付けたままでかまいません。



時間スケールスライダを左右にスラ イドさせることで、グラフの時間軸 の間隔を調整できる

Call Tree のデスクロージャを開き、設定項目を表示させる



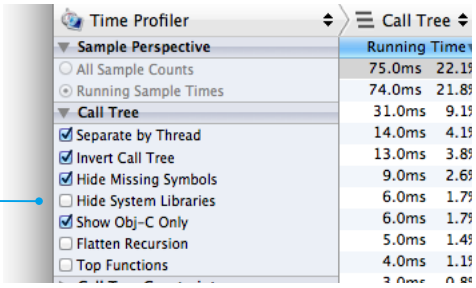
Hide System Libraries

チェックすると UIView など iOS 側のメソッドは表示しなくなる Show Obj-C Only: チェックすると C 言語の関数などは表示されなくなる

221 ページ

Separate by Thread のチェックは作業に支障がないので、そのまま付けたままとします。

Hide System Libraries のチェックを外す



228 ページ

開発環境の Xcode は Mac OS X 10.7（lion）から App Store で提供されるようになりました。

開発環境

・
・
・

Mac OS X 10.7（lion）からは、Xcode は App Store で提供されるようになりました。

229 ページ

開発者証明書要求の機能が Xcode に組み込まれました。

開発者証明書要求

・
・
・

なお、Xcode 自身に一連の作業を自分の開発マシン上で代行する機能が追加されています。

234 ページ

Automatic Reference Counting という、メモリ管理を Xcode 側に代行させる機能がつきました。

メモリ管理

・
・
・

Automatic Reference Counting（以後 ARC と略記）という、メモリ管理を Xcode 側に代行させる機能がつきました。

この機能は、オブジェクトの所有権ポリシー（-retain、-release による所有権の確保、放棄）の煩わしさからプログラマを解放するために、それらの作業を Xcode 側にやらせようというものです。

このため、ARC の利用を指定すると、-retain、-release、-autorelease メッセージは使えなくなり、-dealloc メソッド内では super に -dealloc が送れなく（エラーにされる）なります。

と同時に -dealloc メソッドでおこなっていた所有権放棄の作業も不要になります。

また、プログラマは変数宣言時やプロパティ宣言時に、コンパイラに管理用のヒントを与えれば、所有権ポリシーの作業から解放されます。これがプロパティの

strong

と

weak

という 2 つの属性の役割りです。

strong 属性のプロパティは、retain 属性によく似ていますが、-dealloc メソッドでの -release 送信が必要無くなる点が異なります。

weak 属性のプロパティは assign 属性に似ていますが、変数に代入したオブジェクトがメモリから消える時に変数側に nil が代入される点が異なります。

また、変数宣言でも __strong、__weak を型の前に記述すれば、プロパティと同じように所

有権ポリシーの面倒をみてくれます。省略時は __strong とみなされます。

```
@interface XObject : NSObject
            ・・・
@property (strong) id friend;
@end

@implementation XObject
@synsethise friend;
- (void)dealloc
{
            ・・・・
[friend release];   ← 必要ない
[super dealloc];    ← 記述できない
}
            ・・・・
@end
```

Edit → Refactor → Convert to Objective-C ARC…メニューで既存のプロジェクトの ARC 対応への変換も可能（ただし完全な自動ではありません）です。

今のところ英語のドキュメントのみが提供されています。iOS Dev Center で以下のタイトルで検索してみてください。

Transitioning to ARC Release Notes

240 ページ

NSStringFromClass と対をなす NSClassFromString を紹介します。

principalClassName

渡しているのは NSString 文字列ですが、文字列からクラスを指定できるのでしょうか？

できます。

```
Class NSClassFromString(NSString *aClassName)
```

という Apple の提供する API は、受け取った NSString 文字列で指定されるクラスオブジェクトを返します。

```
id class = NSClassFromString(@"NSString");
NSString* str = [class stringWithFormat:@"%d", 100];
```

は以下のように書く事と同じ動きをします。

```
NSString* str = [NSString stringWithFormat:@"%d", 100];
```