

# 初めてのiOSアプリ ケーション



Developer

# 目次

## 初めてのiOSアプリケーション開発について 4

### 入門 6

新規プロジェクトを作成、テストする 6

アプリケーションが起動する仕組みを調べる 12

まとめ 17

### View Controllerおよびそのビューの内側 18

インスペクタを使ってView Controllerの動作を調べる 18

ビューの背景色を変更する 22

まとめ 26

### ビューの設定 27

ユーザインターフェイス要素を追加する 27

ボタンのアクションを作成する 34

テキストフィールドおよびラベルのアウトレットを作成する 37

テキストフィールドのデリゲート接続を作成する 41

アプリケーションのテスト 42

まとめ 43

### View Controllerの実装 44

ユーザ名を表すプロパティを追加する 44

changeGreeting:メソッドの実装 47

View Controllerをテキストフィールドのデリゲートとして設定する 48

アプリケーションのテスト 50

まとめ 50

### トラブルシューティングとコードのレビュー 51

コードおよびコンパイラの警告 51

ストーリーボードファイルのチェック 51

デリゲートメソッド名 52

コードリスト 52

    インターフェイスファイル：HelloWorldViewController.h 52

    実装ファイル：HelloWorldViewController.m 52

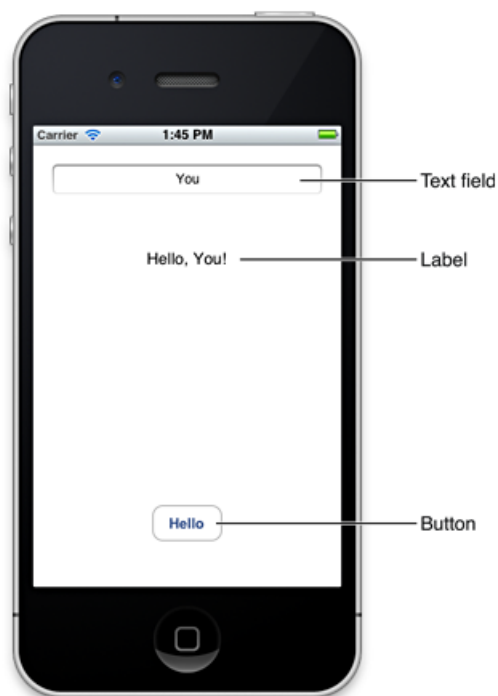
書類の改訂履歴 54

# 初めてのiOSアプリケーション開発について

『Your First iOS App』では、iOSアプリケーション開発に必要な「3つのT」を紹介します。

- **Tools（ツール）**。Xcodeを使用してプロジェクトを作成および管理する方法。
- **Technologies（技術）**。ユーザ入力に応答するアプリケーションを開発する方法。
- **Techniques（技法）**。あらゆるiOSアプリケーション開発の根底にある、基本的なデザインパターンを活用する方法。

このチュートリアルを最後まで進めると、次のような外観のアプリケーションが完成します。



上図のように、主なユーザインターフェイス要素が3つあります。

- テキストフィールド（ユーザが情報を入力）
- ラベル（アプリケーションが情報を表示）
- ボタン（これを押すとラベルに情報が表示される）

完成したアプリケーションを起動し、テキストフィールド内をクリックすると、システム組み込みのキーボードが画面に現れます。このキーボードで名前を入力し、（「Done」キーを押して）キーボードを消した後、「Hello」ボタンを押すと、テキストフィールドとボタンの間にあるラベルに「Hello, *Your Name!*」という表示が現れます。

このチュートリアルを最大限に活かすためには、一般的なコンピュータプログラミングの基礎、特にオブジェクト指向プログラミングやObjective-C言語に多少慣れていなければなりません。

---

**注意** iPad専用の開発を目的としているデベロッパの方も、iOSアプリケーションを開発する、このチュートリアルから始めると良いでしょう。このチュートリアルに掲載しているのはiPhoneのユーザインターフェイスですが、使用するツールとテクニックは、iPadアプリケーションの開発に使用するものとまったく同じです。

---

# 入門

このチュートリアルに従ってiOSアプリケーションを開発するためにはXcode4.3以降が必要です。XcodeはAppleが提供する統合開発環境（IDE、Integrated Development Environment）で、iOS用、Mac OS X用のアプリケーションを開発できます。MacにXcodeをインストールすると、iOSプラットフォームのプログラミングインターフェイスを備えたiOS SDKも入手することになります。

## 新規プロジェクトを作成、テストする

アプリケーション開発を開始するため、新しいXcodeプロジェクトを作成します。

1. Xcode（デフォルトでは/Applications以下）を起動してください。

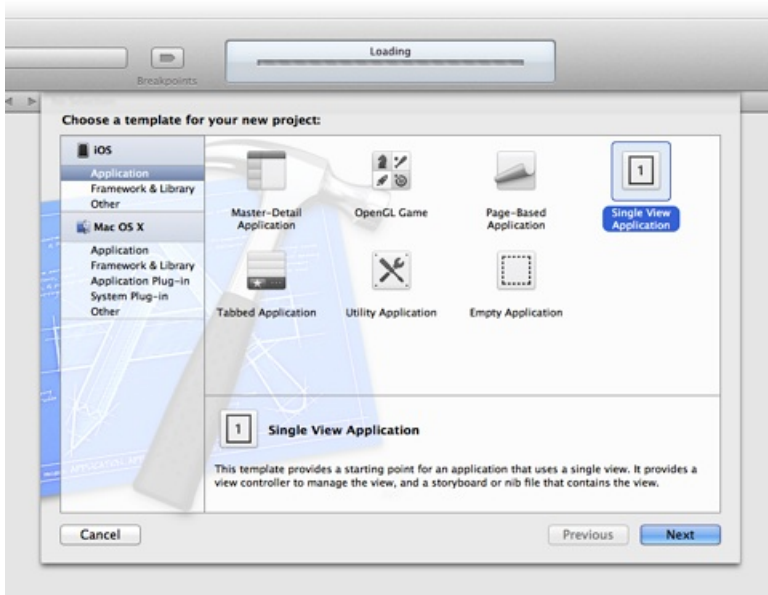
Xcodeでプロジェクトを作成する（または開く）のが初めてであれば、次のような「Welcome to Xcode」ウィンドウが現れます。



一方、すでにプロジェクトを作成した（または開いた）ことがあれば、「Welcome to Xcode」ウィンドウは現れず、最初からプロジェクトウィンドウが開きます。

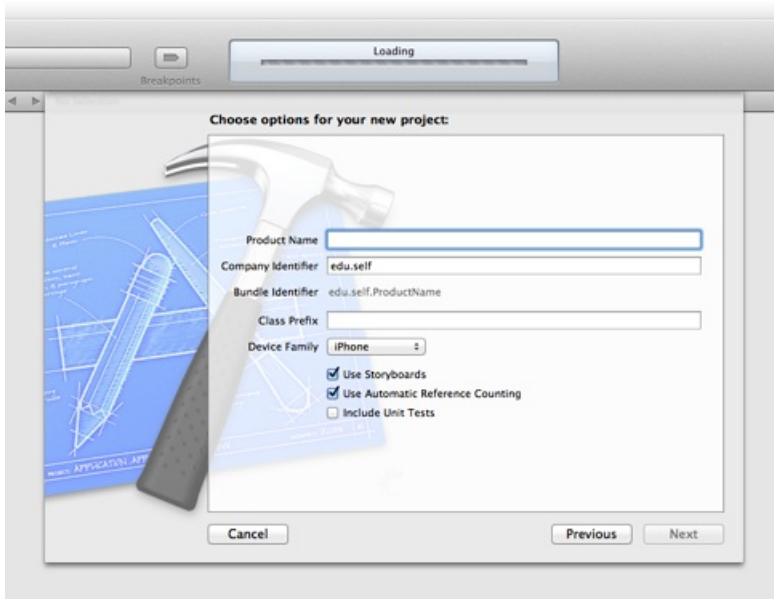
2. 「Welcome to Xcode」 ウィンドウの「Create a new Xcode project」をクリック（または「File」>「New」>「Project」を実行）してください。

新しいウィンドウが開き、テンプレート選択ダイアログが現れます。Xcodeには組み込みのアプリケーションテンプレートがいくつか付属しており、これを使って、一般的なスタイルのiOSアプリケーションを開発できます。たとえば「Tabbed」テンプレートはiTunesのような外観、「Master-Detail」テンプレートはMailのような外観のアプリケーション開発に使えます。



3. ダイアログの左側にあるiOSセクションで、「アプリケーション(Application)」を選択します。
4. ダイアログのメイン領域で「Single View Application」を選択し、「Next」をクリックしてください。

新しいダイアログが表示され、アプリケーションの名前を入力し、プロジェクトの追加オプションを選択するように指示されます。



5. 「製品名(Product Name)」、「会社ID(Company Identifier)」、「クラスプレフィックス(Class Prefix)」の各フィールドに入力します。

下記の値を使用できます。

- **Product Name** : HelloWorld
- **Company Identifier** : 存在する場合は、会社ID会社IDがない場合は、edu.selfを使用できます。
- **Class Prefix** : HelloWorld

**注意** ここで入力した製品名 (Product Name) は、プロジェクト名やアプリケーション名としても使われます。また、クラス接頭辞 (Class Prefix) は、自動生成されるクラス名の一部となります。たとえば、自動生成されるアプリケーションデリゲートクラスは、HelloWorldAppDelegateという名前になります。指定したクラス接頭辞に応じ、「YourClassPrefixNameAppDelegate」という形の名前になるのです (アプリケーションデリゲートについては[「アプリケーションが起動する仕組みを調べる」](#) (12 ページ) を参照)。

以下、話を簡単にするため、製品名として「HelloWorld」、クラス接頭辞として「HelloWorld」と入力したものと説明します。

6. 「デバイスファミリ(Device Family)」ポップアップメニューで、iPhoneが選択状態になっていることを確認します。



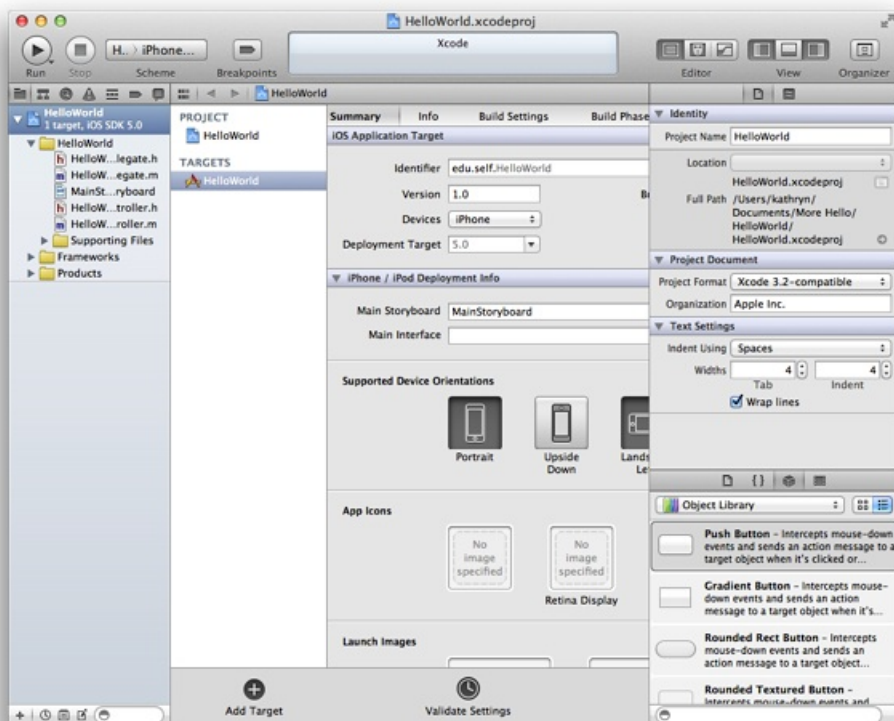
7. 「ストーリーボードを使用(Use Storyboards)」と「自動参照カウントを使用(Use Automatic Reference Counting)」オプションが選択状態になっていること、および「単体テストを含める(Include Unit Tests)」オプションが選択状態になっていないことを確認します。

8. 「次へ(Next)」をクリックします。

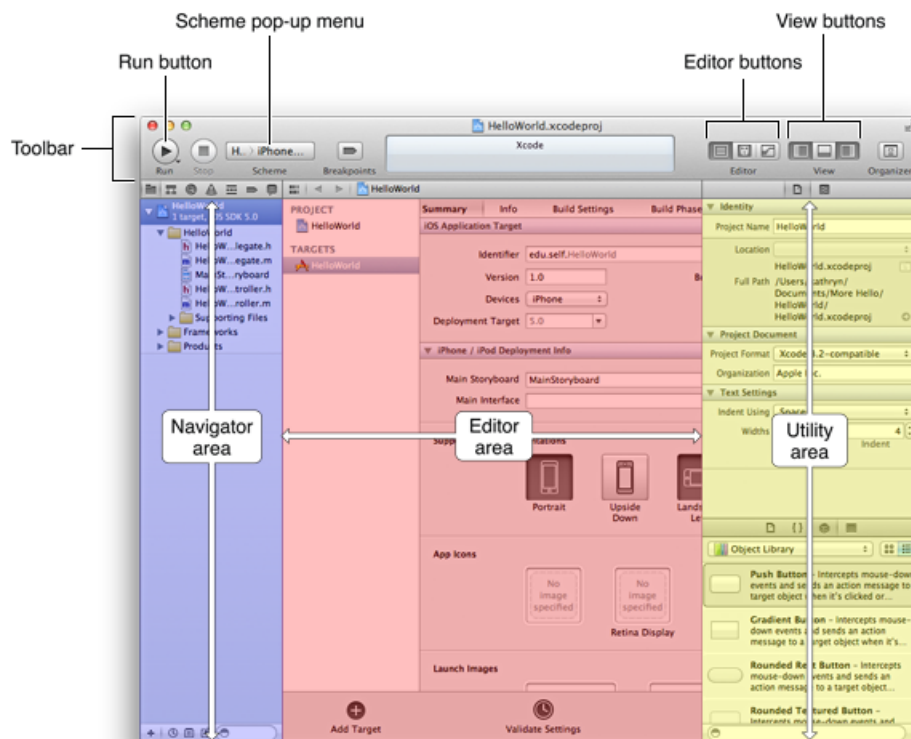
プロジェクトの保存先を指定するダイアログが現れます。

9. プロジェクトの場所を指定して（「ソース管理(source control)」オプションは未選択状態のまま）、「作成(Create)」をクリックします。

次のような新しいプロジェクトが、ウインドウ（ワークスペースウインドウと呼ぶ）内に開きます。



少し時間を割いて、このワークスペースウィンドウを観察してみましょう。このチュートリアルでは全般的に、下のウィンドウ内のボタンや領域を使って作業することになります。



ユーティリティ領域が（上図のように）ワークスペースウィンドウ内に開いているかも知れませんが、以降の作業では使わないので、閉じてしまっても構いません。一番右側の「View」ボタンでこの領域を制御できます。ユーティリティ領域が開いていれば、ボタンは次のような外観になっています。



必要ならばこの「View」ボタンを押して、ユーティリティ領域を閉じてください。

まだコードは1行も書いていませんが、アプリケーションを構築し、Xcodeに付属するSimulatorアプリケーション上で実行できます。名前からも分かるように、Simulatorを使うと、iOSベースのデバイス上でアプリケーションを実行したとき、どのような外観や動作になるかを確認できます。

1. Xcodeツールバーの「Scheme」ポップアップメニューで、「HelloWorld」>「iPhone 5.0 Simulator」をオンにしておきます。

この選択肢が表示されない場合は、同じメニューの「iPhone 5.0 Simulator」をオンにしてください。

2. Xcodeツールバーの「実行(Run)」ボタンをクリックします（または、「プロジェクト(Product)」>「実行(Run)」を選択します）。

ビルド処理の進捗状況に応じて、ツールバーの中央、アクティビティビューアにメッセージが表示されます。


ビルド処理が終わると、自動的にSimulatorが起動します（ワークスペースウインドウの上部にSimulatorが現れるまで、若干待たなければならないかも知れません）。最初に（iPad用ではなく）iPhone用にビルドする旨を指定したので、iPhone風の外観のウインドウになります。この擬似iPhone画面上に、アプリケーションが次のように表示されます。



この時点では、アプリケーションは特に面白いことをするわけではありません。真っ白な画面が現れるだけです。この画面が何に由来するものかを理解するためには、コード中に記述されている各オブジェクトと、それが連携してアプリケーションを起動する仕組みを知る必要があります。今回はそのまま、「iOS Simulator」>「Quit iOS Simulator」コマンドで、Simulatorを終了してください（Xcodeは実行したまま）。

アプリケーションを起動したとき、ワークスペースウインドウの下部に「Debug」領域が開くことがあります。チュートリアルでは、このペインは使いません。画面を広く使えるよう、このペインは閉じておくといよいでしょう。

- ツールバーの「Debug View」ボタンをクリックしてください。

「Debug View」ボタンは「View」ボタン群の中央にあって、次のような外観をしています： .

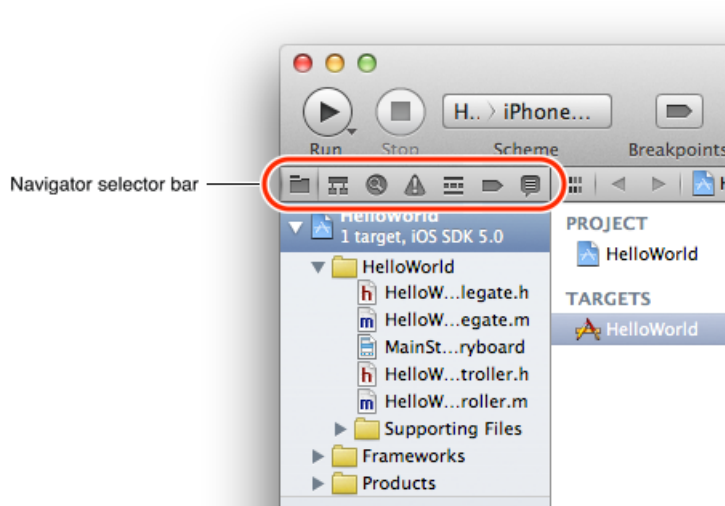
## アプリケーションが起動する仕組みを調べる

Xcodeに付属のテンプレートに基づきアプリケーションを構築したので、基本的なアプリケーション環境は、実行時に自動的に設定されます。具体的には、アプリケーションオブジェクトを作成し、実行ループを開始する、などといった処理です（実行ループは入力源を登録し、入力イベントをアプリケーションに配送できるようにします）。その大部分を実行するのはUIApplicationMain関数で、UIKitフレームワークに組み込まれており、これを呼び出す旨の記述がソースファイルmain.mにあります。

**注意** UIKitフレームワークには、ユーザインターフェイスを構築、管理するために必要なクラスがすべて揃っています。とはいえUIKitフレームワークも、Cocoa Touch（あらゆるiOSアプリケーションの動作環境）が提供する、数多くのオブジェクト指向フレームワークのひとつに過ぎません。

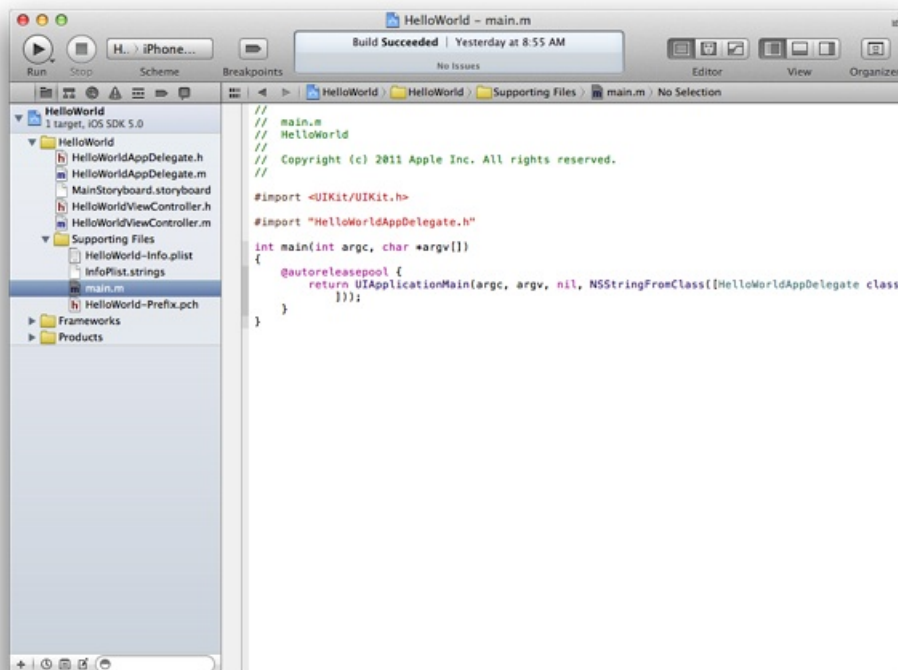
1. プロジェクトナビゲータが、ナビゲーション領域に開いているはずです。

ここにはプロジェクトを構成するファイルがすべて表示されます。開いていない場合は、ナビゲータセクタバーの一番左側にあるボタンを押してください。



2. プロジェクトナビゲータ上で、「Supporting Files」フォルダを展開表示します。フォルダ名の隣にある三角形をクリックしてください。
3. main.mを選択してください。

ソースファイルの内容が、ウインドウの主エディタ領域に、次のように表示されます。



main.mのmain関数は、自動解放プールのブロック内で、次のようにしてUIApplicationMain関数を呼び出します。

```
@autoreleasepool {  
    return UIApplicationMain(argc, argv, nil, NSStringFromClass([HelloWorldAppDelegate  
class]));  
}
```

@autoreleasepool文の記述があるので、自動参照カウント（ARC、Automatic Reference Counting）システムを使うことになります。ARCのオブジェクトライフサイクル自動管理機能により、オブジェクトは、必要である間は確実に存続し、不要になれば廃棄されます。

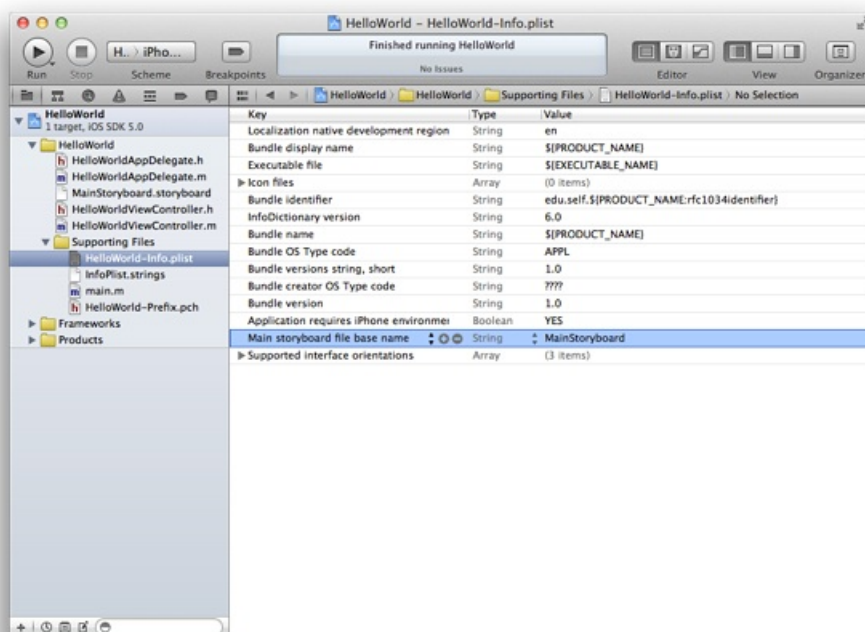
UIApplicationMainを呼び出すと、UIApplicationのインスタンス、アプリケーションデリゲート（このチュートリアルでは、Single Viewテンプレートに組み込まれているHelloWorldAppDelegate）のインスタンスが生成されます。**アプリケーションデリゲート**の第一の役割は、アプリケーションが情報を描画できるよう、ウインドウを用意する、というものです。また、画面表示に先立ち、若干のアプリケーション設定処理も行います（**デリゲーション**はデザインパターンのひとつで、あるオブジェクトが別のオブジェクトに代わって、あるいはそれと連携して動作する、というものです）。

iOSアプリケーションの場合、**ウィンドウオブジェクト**がアプリケーションの可視部分を収容するコンテナとして動作し、イベントをアプリケーションオブジェクトに配送し、デバイスの向き変化に回答できるよう、必要な支援を行います。ウィンドウそれ自体は不可視です。

UIApplicationMainではさらに、アプリケーションのInfo.plistファイルを読み込んで内容を走査します。Info.plistファイルの中身はプロパティリスト（キーと値の組を構造化したリスト）で、アプリケーションに関する情報（名前、アイコンなど）を記述するようになっています。

- プロジェクトナビゲータで、「Supporting Files」フォルダ以下のHelloWorld-Info.plistを選択してください。

ウィンドウのエディタ領域に、次のようにInfo.plistファイルが表示されます。



このチュートリアルでは、「Supporting Files」フォルダ以下にあるほかのファイルを参照する必要はないので、画面が錯綜しないよう、プロジェクトナビゲータでこのフォルダを折り畳んでおくといでしょう。フォルダアイコンの隣にある三角形をもう一度クリックしてください。

このプロジェクトではストーリーボードを使うことにしているので、Info.plistファイルには、アプリケーションオブジェクトが読み込むべきストーリーボードファイル名も記述されています。**ストーリーボード**には、オブジェクト、遷移、接続などの情報がアーカイブの形で記述されており、これがアプリケーションのユーザインターフェイスを定義しています。



HelloWorldアプリケーションでは、ストーリーボードファイル名はMainStoryboard.storyboardとなっています（Info.plistファイルにはこの名前の先頭部分しか記述されていないことに注意）。アプリケーションを起動すると、MainStoryboard.storyboardが読み込まれ、これをもとに、初期View Controllerのインスタンスが生成されます。**View Controller**はコンテンツ領域を管理するオブジェクトです。**初期View Controller**は単に、アプリケーションの起動後、最初にロードされるView Controllerのことです。

HelloWorldアプリケーションにはView Controllerが1つしかありません（具体的にはHelloWorldViewController）。これ以降HelloWorldViewControllerが、唯一のビューとして現れるコンテンツ領域を管理します。**ビュー**は、画面上の指定された矩形領域にコンテンツを描画するオブジェクトであり、ユーザがタッチすることにより発生したイベントを処理します。ビューの中に別のビューが含まれることもあり、これをサブビューと呼びます。サブビューをビューに追加したとき、包含する側のビューを親ビュー、される側を子ビューといいます。親ビューと子ビュー（および、さらにその子のビュー）が**ビュー階層**を形成します。View Controllerは、単一のビュー階層を管理します。

---

**注意** HelloWorldアプリケーションでは、ビューとView Controllerが、Model-View-Controller（MVC）と呼ばれるデザインパターンで定義された、アプリケーションオブジェクトに求められる3つの役割のうち2つを担っています。残ったもうひとつの役割はモデルオブジェクトが担います。MVCでは、モデルオブジェクトがデータ（カレンダーアプリケーションの予定事項、描画アプリケーションの図形など）、ビューオブジェクトがそのデータの表示方法を管理し、コントローラオブジェクトが両者を仲介します。HelloWorldアプリケーションでは、モデルオブジェクトはユーザが入力した名前を保持する文字列に過ぎません。当面、MVCについてこれ以上の知識は必要ありませんが、アプリケーションを構成するオブジェクトが、どのようにしてそれぞれの役割を果たすのか、自分なりに考えてみるのもよいでしょう。

---

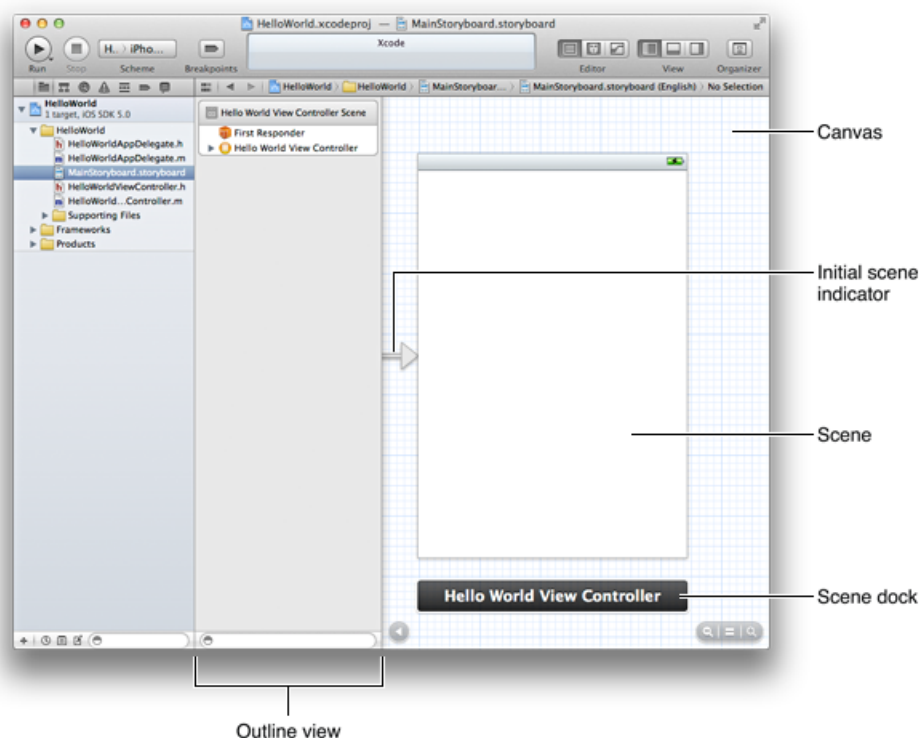
この後のステップで、サブビューを3つ、HelloWorldViewControllerで管理されるビューに追加して、ビュー階層を組み立てます。サブビューとは具体的に言うと、テキストフィールド、ラベル、ボタンのことです。

View Controllerおよびビューは、ストーリーボード上で目で見て確認できます。

- プロジェクトナビゲータでMainStoryboard.storyboardを選択します。

エディタ領域にストーリーボードが開きます（ストーリーボードオブジェクトの奥側に見えるグラフ用紙のような領域を**キャンバス**といいます）。

デフォルトのストーリーボードを開くと、ワークスペースウィンドウは次のようになります。



ストーリーボードにはシーンとセグエが記述されています。**シーン**はView Controller、**セグエ**はシーン間の遷移を表します。

「SingleView」テンプレートの場合、View Controllerは1つだけなので、アプリケーションのストーリーボードにはシーンが1つしかなく、セグエはありません。キャンバス上にある、シーンの左辺を指す矢印は**先頭シーンのインジケータ**で、アプリケーションの起動後、最初にロードされるシーンであることを表します（通常、先頭シーンは初期View Controllerと同じものです）。

キャンバス上のシーンは「Hello World View Controller」という名前になっています。

HelloWorldViewControllerオブジェクトが管理するシーンだからです。「Hello World View Controller」シーンはいくつかの部品から成り、それぞれ、Xcodeの**アウトラインビュー**（キャンバスとプロジェクトナビゲータの間にあるペイン）に表示されます。この時点では、View Controllerには次の部品があります。

- ファーストレスポндаというプレースホルダオブジェクト（橙色の箱のアイコン）。

**ファーストレスポンド**は動的なプレースホルダで、アプリケーションの動作中に発生する、各種のイベントを最初に受け取るオブジェクトを表します。ここで言うイベントには、編集操作に関するイベント（テキストフィールドをタップしてキーボードを表示するなど）、モーションイ



ベント（デバイスを振るなど）、アクションメッセージ（タップされたボタンが送信するメッセージなど）、その他があります。このチュートリアル範囲では、ファーストレスポンドは何もしません。

- HelloWorldViewControllerオブジェクト（黄色の円内に薄い灰色の矩形があるアイコン）。ストーリーボードは、シーンを読み込むと、それを管理するView Controllerのインスタンスを生成します。
- View Controllerの下に列挙されているビュー（アウトラインビューにこれを表示するためには、「Hello World」View Controllerの隣にある三角形をクリックして展開表示にする必要があるかも知れません）。

Simulator上でアプリケーションを起動したときに見えていたのは、このビューの真っ白な背景です。

---

**注意** アプリケーションのウィンドウオブジェクトは、ストーリーボードには表示されません。

---

キャンバス上、シーンの下側にある領域を、**シーンドック**といいます。現時点では、View Controllerの名前（すなわち「Hello World View Controller」）がシーンドックに表示されています。状況に応じて、ファーストレスポンドやView Controllerオブジェクトを表すアイコンが表示されることもあります。

## まとめ

この章では、Xcodeを使って、「Single View」テンプレートをもとに新規プロジェクトを作成し、テンプレートに定義されているデフォルトアプリケーションをビルド、実行しました。次に、ソースファイルmain.m、Info.plistファイル、ストーリーボードファイルなど、プロジェクトの基本的な要素や、アプリケーションの起動処理について説明しました。また、Model-View-Controller（MVC）デザインパターンで、アプリケーション内のオブジェクトの役割がどのように定義されているか、を調べました。

次の章では、View Controllerおよびそのビューについて解説します。

# View Controllerおよびそのビューの内側

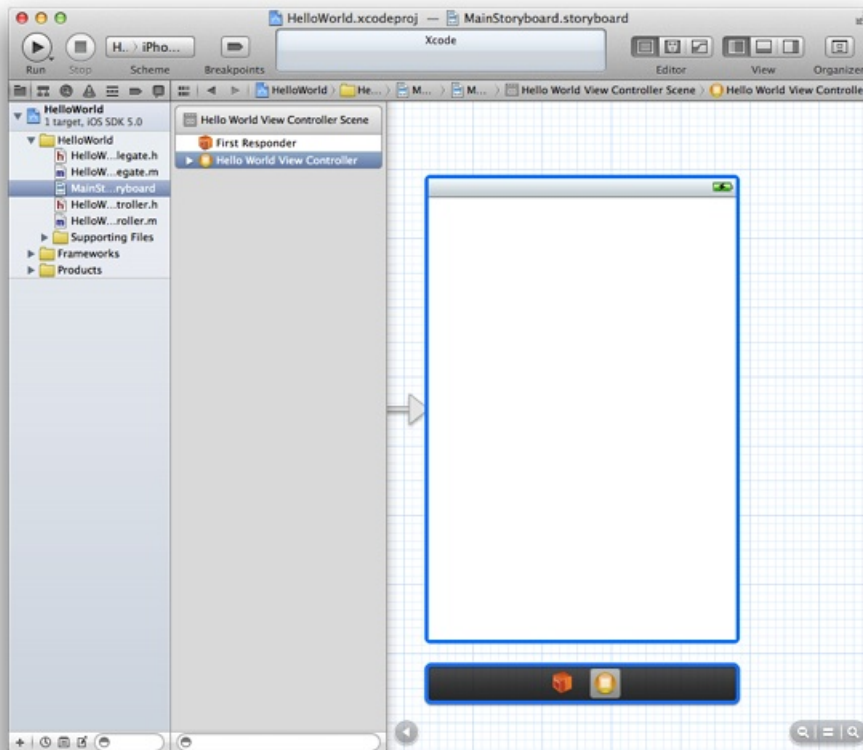
すでに述べたように、各View Controllerはある1つのシーン（情報内容の表示領域）を管理します。ここに表示される内容は、View Controllerのビューで定義するようになっています。この章では、HelloWorldViewControllerが管理するシーンについて詳しく説明し、ビューの背景色を調整する方法を示します。

## インスペクタを使ってView Controllerの動作を調べる

アプリケーションを起動すると、メインストーリーボードファイルが読み込まれ、先頭View Controllerのインスタンスが生成されます。先頭View Controllerは、アプリケーションを起動したとき、最初に表示されるシーンを管理します。「Single View」テンプレートではView Controllerが1つしかないので、当然、これが先頭View Controllerになります。Xcodeインスペクタを使うと、View Controllerの状態その他、さまざまな事項を調べることができます。

1. 必要ならばプロジェクトナビゲータ上でMainStoryboard.storyboardをクリックし、キャンバス上にシーンを表示しておきます。
2. アウトラインビューで、「Hello World View Controller」を選択してください（「First Responder」のすぐ下）。

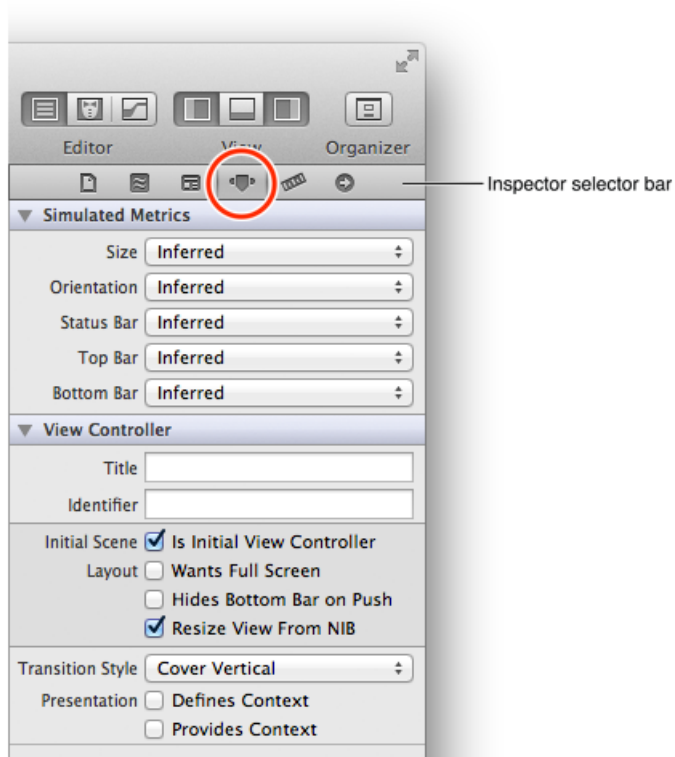
ワークスペースウィンドウは次のような表示になります。



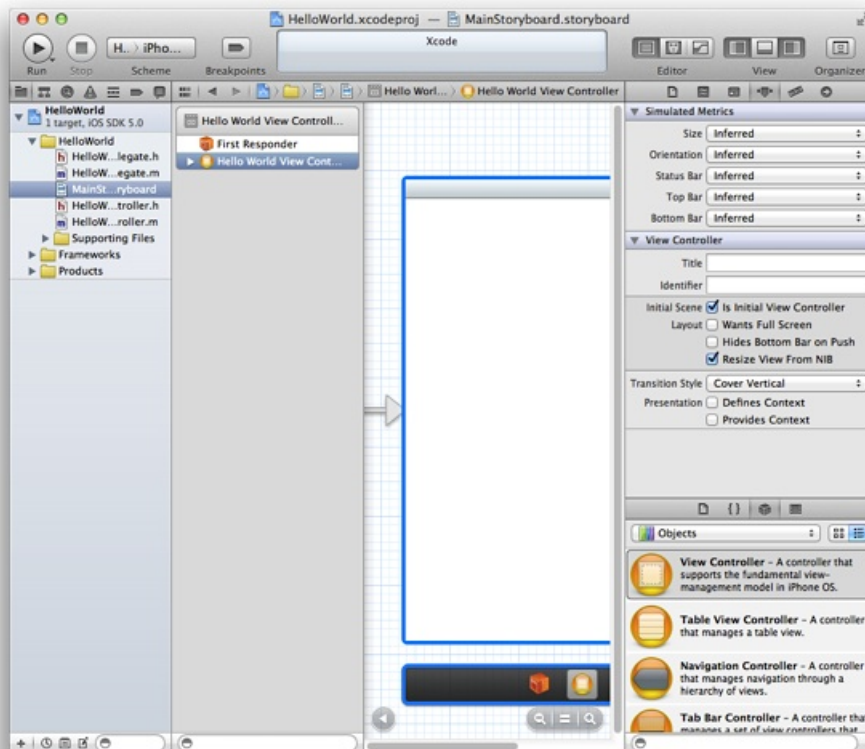
シーンとシーンドックの両方に青枠がついていること、シーンドック上のView Controllerが選択状態になっていることに着目してください。

3. ツールバーの一番右にある「View」ボタンをクリック（または「View」>「Utilities」>「Show Utilities」を実行）すると、ウィンドウの右側にユーティリティ領域が表示されます。
4. ユーティリティ領域の「Attributes」インスペクタボタンをクリックして、Attributesインスペクタを開きます。

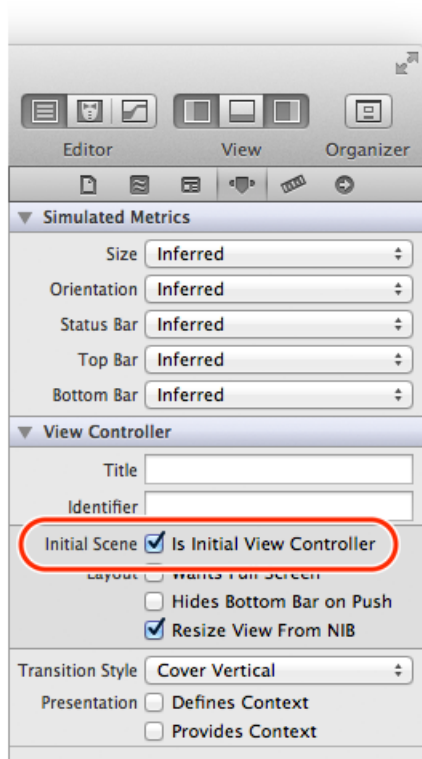
「Attributes」インスペクタボタンは、ユーティリティ領域の上部、インスペクタセクタバーの左から4つめにある、次のようなボタンです：



「Attributes」インスペクタが開くと、ワークスペースウィンドウは次のようになるはずです（全体を表示するにはウィンドウを広げる必要があるかも知れません）。



「Attributes」インスペクタの「View Controller」セクションを見ると、「Initial Scene」がオンになっています。



これをオフにすると、先頭シーンである旨を表すインジケータが、キャンバスに表示されなくなります。このチュートリアルではオンのままにしておいてください。

## ビューの背景色を変更する

先にシミュレータ上でアプリケーションを起動したときには、ビューには真っ白の背景が表示されていました。アプリケーションが正しく動作していることを確認するため、ビューの背景色を白以外の色に設定して、シミュレータ上でアプリケーションを再起動したとき、その色になることを確認してみましょう。

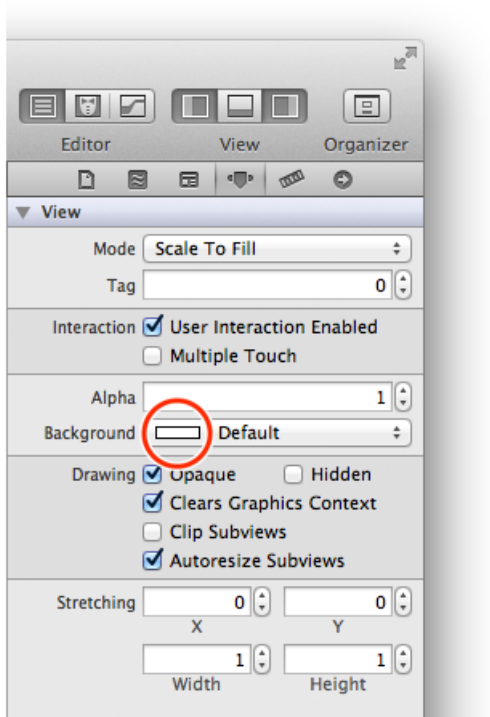
ビューの背景色を変更する前に、ストーリーボードがキャンバス上に開いたままであることを確認してください（必要ならばプロジェクトナビゲータ上でMainStoryboard.storyboardをクリックし、キャンバス上にストーリーボードを開いておきます）。

1. アウトラインビューで、（まだであれば）「Hello World View Controller」の隣の三角形をクリックして展開表示にし、「View」を選択してください。

キャンバス上では、ビュー領域が強調表示になります。

- ユーティリティ領域の上部、インスペクタセクタバーの「Attributes」ボタンをクリックして、「Attributes」インスペクタを開いてください（まだ開いていない場合）。
- 「Attributes」インスペクタで、「Background」ポップアップメニューの白い矩形をクリックすると、「Colors」ウインドウが開きます。

矩形は背景に設定された現在の色を表します。「Background」ポップアップメニューは次のような外観をしています。



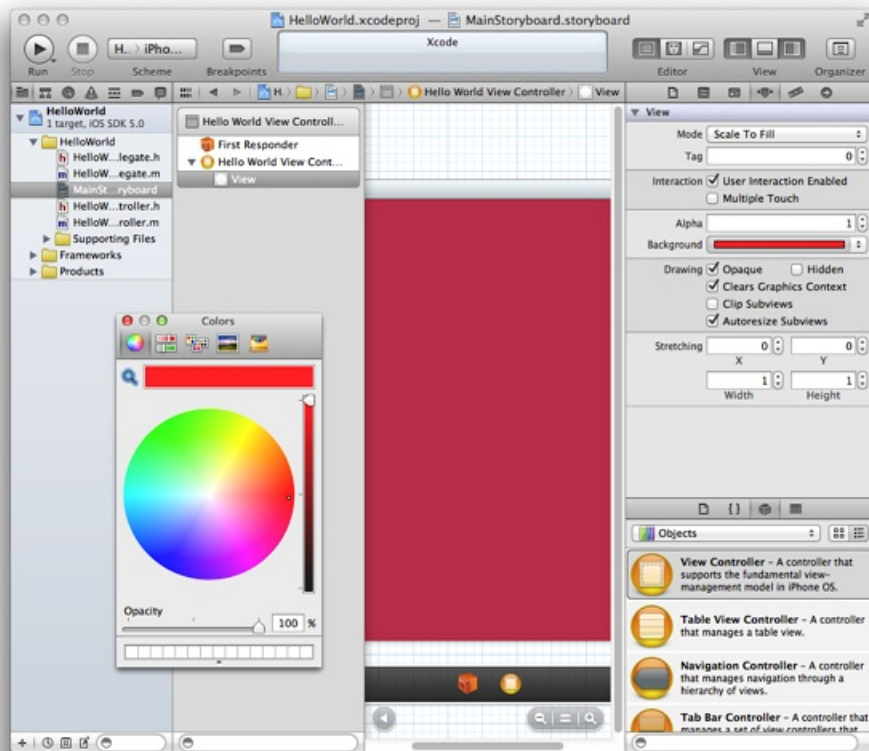
---

**注意** 白い矩形ではなく「Default」をクリックしてポップアップメニューを開いた場合は、このメニューから「Other」を選択します。

---

- 「Colors」ウインドウで、白以外の色を選んでください。

ワークスペースウィンドウ（および「Colors」ウィンドウ）は次のような表示になります。

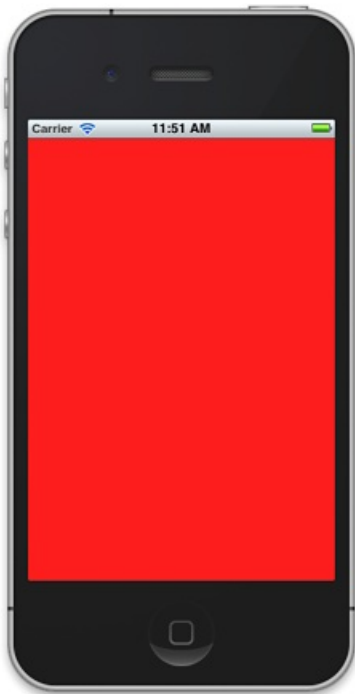



ビューを選択すると強調表示になるので、キャンバス上の色は、「Colors」ウィンドウで指定したものと違って見えるかも知れません。

5. 「Colors」ウィンドウを閉じてください。



「Run」ボタンをクリック（または「Product」>「Run」を実行）すると、シミュレータ上でアプリケーションの動作をテストできます。Xcodeツールバーの「スキーマ(Scheme)」ポップアップメニューに、「HelloWorld」>「iPhone 5.0 Simulator」が表示されていることを確認します。次のようになります。



 **Tip** アプリケーションを起動する前に、修正した内容を保存する必要はありません。「Run」をクリック（または「Product」>「Run」を実行）すれば、自動的にファイルに保存されるからです。

次に進む前に、ビューの背景色を白に戻しておきましょう。

1. 「Attributes」インスペクタで、矢印をクリックして「Background」ポップアップメニューを開きます。  
「Background」ポップアップメニューの矩形は、「Colors」ウインドウで選択した色に変わっているはずです。（矢印ではなく）色のついた矩形をクリックすると、また同じ「Colors」ウインドウが開いてしまいます。ビューの元の背景色をそのまま使いたいのので、「Colors」ウインドウで同じ色を探すより、「Background」メニューから選択する方が容易でしょう。
2. 「Background」ポップアップメニューの「Recently Used Colors」セクションには白の矩形があるはずなので、これを選択してください。
3. 「Run」ボタンをクリックして、アプリケーションをコンパイル、実行（および変更内容を保存）します。

背景が元通り白になったことを確認した後、シミュレータを終了してください。

## まとめ

この章では、シーンの中身を調べた後、ビューの背景色を変え（た後もとに戻し）ました。

次の章では、ビューにテキストフィールド、ラベル、ボタンを追加し、ユーザが操作できるようにします。

# ビューの設定

Xcodeには、ストーリーボードファイルに追加できるオブジェクトのライブラリが付属しています。これらの中には、ボタン、テキストフィールドなどビューに属するユーザインターフェイス要素もあれば、View Controller、ジェスチャリコグナイザなど高度なオブジェクトもあります。

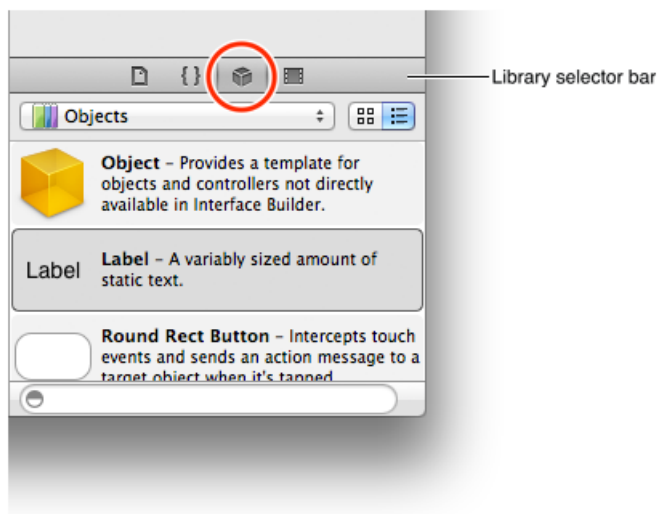
すでに「Hello World View Controller」シーンにはビューが含まれているので、今度は、ボタン、ラベル、テキストフィールドを追加する必要があります。その後、各要素とView Controllerクラスを接続して、要素が意図通りに振る舞うようにします。

## ユーザインターフェイス要素を追加する

ユーザインターフェイス (UI) 要素を、オブジェクトライブラリからキャンバス上のビューにドラッグして追加します。UI要素は、ビューに置いた後も、必要に応じて位置を動かしたり大きさを変えたりできます。

1. 必要であれば、プロジェクトナビゲータ上でMainStoryboard.storyboardを選択し、キャンバス上に「Hello World」View Controllerのシーンを表示してください。
2. 必要ならば、オブジェクトライブラリを開いてください。

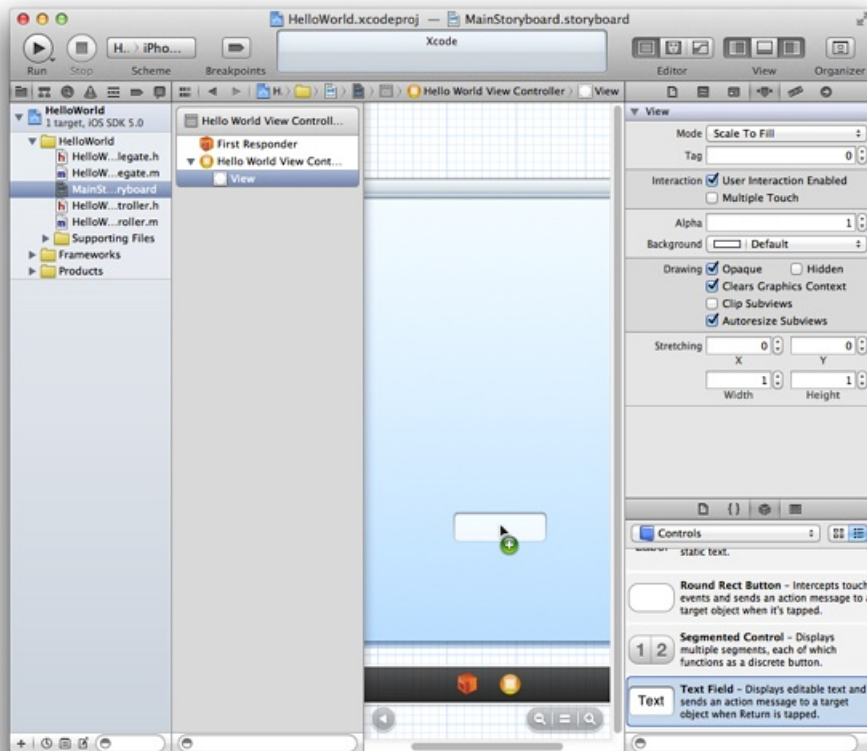
オブジェクトライブラリはユーティリティ領域の下部に現れます。表示されない場合は、ライブラリセレクトタブの左から3番目、次のような外観のボタンをクリックしてください：



- オブジェクトライブラリで、「Objects」ポップアップメニューから「Controls」を選択してください。

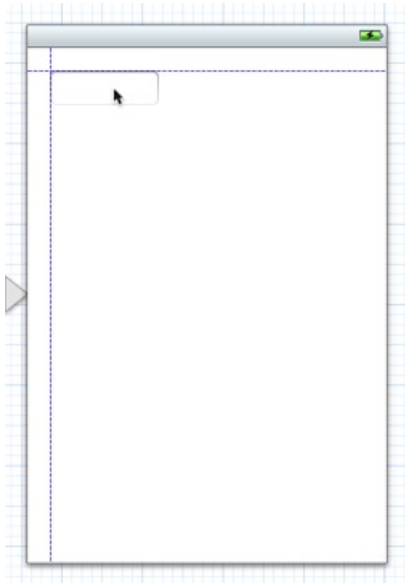
ポップアップメニューの下側に、コントロールが一覧表示されます。各コントローラの名前と外観のほか、機能の簡単な説明も現れます。

- テキストフィールド、角丸の矩形のボタン、ラベルを、リストからひとつずつドラッグし、ビュー上にドロップしてください。



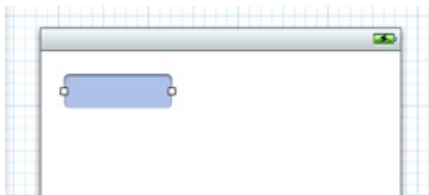
- テキストフィールドをビューの左上隅付近に移動（ドラッグ）してください。

テキストフィールド（やその他のUI要素）を動かすと、ビューの中央や端に位置を合わせやすいよう、青の破線（**位置揃えガイド**）が現れます。次のようにビューの左と上の位置揃えガイドが現れたところで、テキストフィールドのドラッグ操作をやめてください。



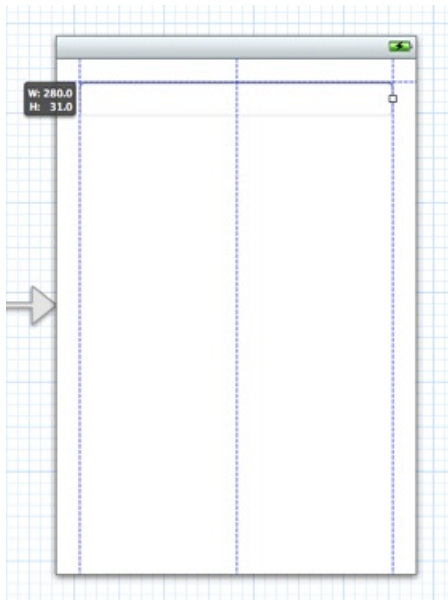
6. ビュー上で、テキストフィールドの大きさを調整します。

UI要素の大きさ調整は、**リサイズハンドル**（要素の境界に現れる白い小矩形）を掴んでドラッグすることにより行います。一般に、要素のリサイズハンドルは、キャンバス上あるいはアウトラインビュー内で当該要素を選択すると現れます。今回はちょうどテキストフィールドをドラッグしてきたところなので、初めから選択状態になっていたはずですが、次のような表示になっていれば、大きさを調整できる状態になっています。そうでなければ、キャンバス上またはアウトラインビュー内で選択してください。



7. テキストフィールドの右側のリサイズハンドルを、ビューの一番右の位置揃えガイドが現れるまでドラッグします。

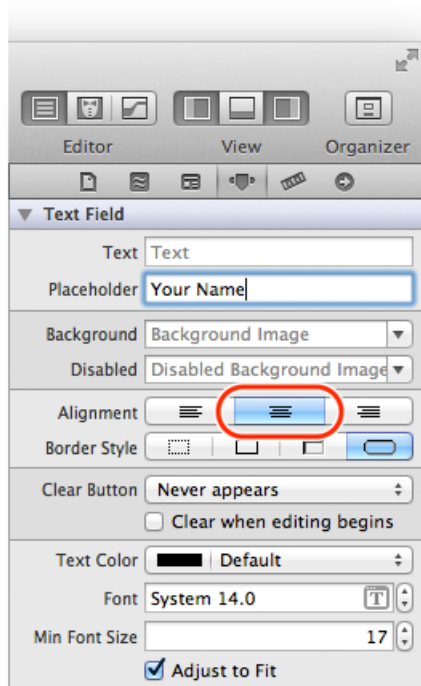
次のような表示になったら、大きさ調整の操作を終えてください。



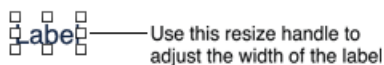
8. テキストフィールドが選択されたままの状態、（必要ならば）**Attributes**インスペクタを開きます。
9. 「Text Field Attributes」インスペクタの上方にある「Placeholder」フィールドに「Your Name」と入力してください。

名前からもわかるように、「Placeholder」フィールドに設定した文字列は、何を入力すればよいのかわかりやすいよう、テキストフィールドに明灰色で表示されます。アプリケーションを実行し、テキストフィールド内でタップすると、このプレースホルダ文字列は消えてしまいます。
10. 「Text Field Attributes」インスペクタの、中央の「Alignment」ボタンをクリックすると、テキストフィールド内のテキストが中央寄せになります。

プレースホルダ文字列を入力し揃えの設定を変更すると、「Text Field Attributes」インスペクタは次のようになるはずです。

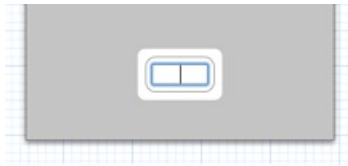


11. ビュー内でラベルをテキストフィールドの下側にドラッグし、左端が揃うように配置してください。
12. 次にラベルの右側のリサイズハンドルをドラッグして、テキストフィールドと同じ幅にします。  
ラベルは幅と高さが調整できるので、（幅しか調整できない）テキストフィールドよりも、リサイズハンドルの数は多くなっています。高さは変えたくないの、隅ではなく右辺中央のリサイズハンドルをドラッグするとよいでしょう。

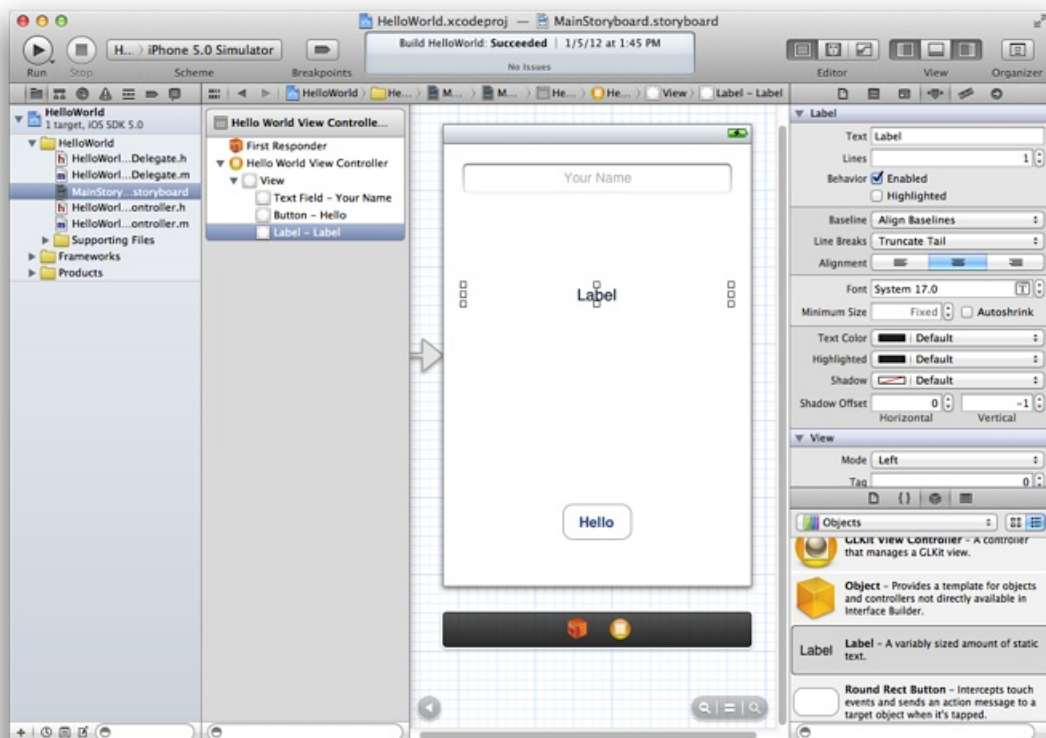


13. 同様に、「Label Attributes」インスペクタの、中央の「Alignment」ボタンをクリックして、ラベル文字列を中央寄せにしてください。
14. ボタンをビューの下方中央付近にドラッグします。
15. キャンバス上のボタンをダブルクリックし、「Hello」という文字列を入力してください。

ビュー内のボタンをダブルクリックした状態（テキスト入力はまだ）では、次のようになっているはずです。



テキストフィールド、ラベル、ボタンを追加し、以上のレイアウト変更を施すと、プロジェクトは次のようになるはずです。



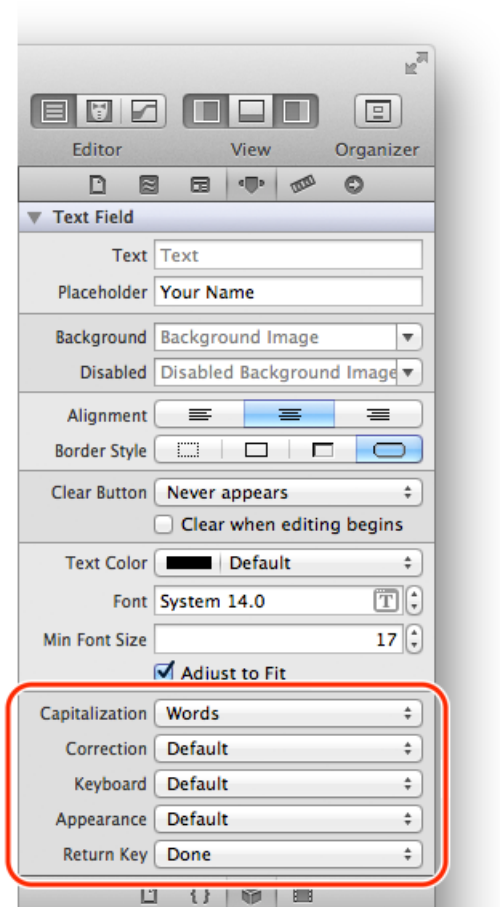
ほかにもテキストフィールドにさまざまな変更を施して、ユーザが想定するような動作にすることができます。まず、ユーザはここに自分の名前を入力することになるので、各語の先頭が自動的に大文字に変わるようにすることが考えられます。次に、テキストフィールドに関連付けられているキーボードを、（たとえば数字ではなく）名前の入力用に設定するとともに、キーボードに「Done」ボタンが表示されるようにします。



このように設定を変更するのは、設計段階で、テキストフィールドに入る情報の種類が分かっている  
ので、実行時の外観や動作をそれに合わせ、ユーザが作業しやすくなるようにしよう、という考え方  
に基づきます。以上の設定変更はすべて、「Attributes」インスペクタ上で行います。

1. ビュー上のテキストフィールドを選択しておきます。
2. 「Text Field Attributes」インスペクタで、次のように設定してください。
  - 「大文字化(Capitalization)」ポップアップメニューで「単語(Words)」を選択します。
  - 「キーボード(Keyboard)」ポップアップメニューが「デフォルト(Default)」に設定されている  
ことを確認します。
  - 「Return Key」ポップアップメニューで「Done」を選択します。

以上の変更を施すと、「Text Field Attributes」インスペクタは次のようになるはずです。



シミュレータ上でアプリケーションを起動し、追加したUI要素が想定通りに現れることを確認してく  
ださい。「Hello」ボタンをクリックすると、それがハイライト状態になり、テキストフィールド内を  
クリックすると、キーボードが表示されます。ただしこの時点では、ボタンを押しても何も起こら

ず、ラベルには「Label」と表示されたままであり、また、表示されたキーボードを閉じる手段也没有ありません。こういった機能を追加するためには、UI要素とView Controllerとの間に、適切な接続を作成する必要があります。この接続について次に説明します。

---

**注意** シミュレータ上でアプリケーションを実行する場合は、実際のデバイスと違い、タップではなくクリックによりボタン類を操作する必要があります。

---


## ボタンのアクションを作成する

ユーザがUI要素を操作する（アクティブ化する）と、その要素はアクションメッセージ（たとえば「この連絡先情報をユーザの連絡先リストに追加せよ」）を、その実行方法を知っているオブジェクトに送信します。このやりとりは、Cocoa Touchの別のデザインパターンである、**ターゲット-アクション**機構の一部を成すものです。

このチュートリアルでは、ユーザが「Hello」ボタンをタップしたとき、「change the greeting」メッセージ（アクション）がView Controller（ターゲット）に送信されるようにします。このメッセージには、View Controllerが管理する文字列（すなわちモデルオブジェクト）を変更する働きがあります。次にView Controllerは、モデルオブジェクトの値変化を反映するよう、ラベルに表示されているテキストを更新します。

Xcode上で、UI要素にアクションを追加し、対応するアクションメソッドをセットアップすることができます。キャンバス上の要素から適切なソースファイル（通常はView Controllerに対応するソースファイル）に、Controlキーを押しながらドラッグする、という方法でセットアップします。ストーリーボードはこのようにして、作成された接続をアーカイブします。すると実行時には、アプリケーションがストーリーボードを読み込んだ時点で、接続が再現されます。

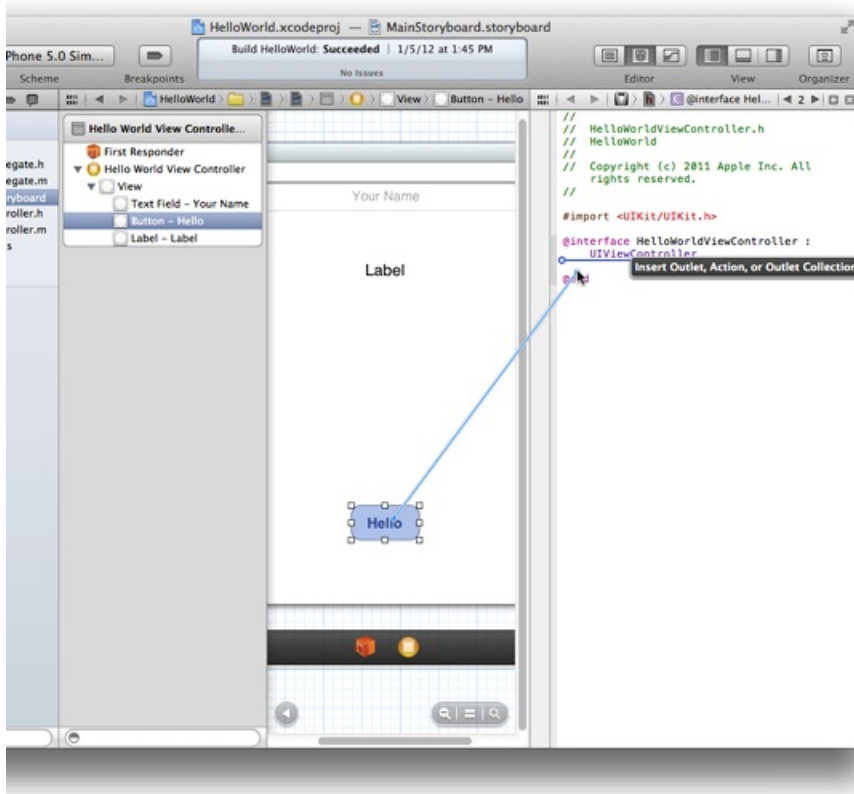
1. 必要であれば、プロジェクトナビゲータ上でMainStoryboard.storyboardを選択し、キャンバス上にシーンを表示してください。
2. Xcodeツールバーの「Utilities」ボタンをクリックしてユーティリティ領域を非表示にし、次に「アシスタントエディタ」ボタンをクリックして、「Assistant」エディタペインを表示します。

これは「Editor」ボタン群のうち中央にあるもので、次のような外観をしています：.

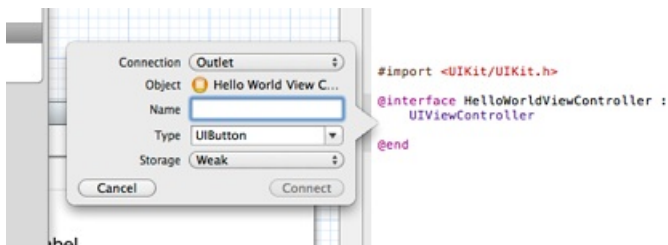
3. AssistantがView Controllerヘッダファイル（つまり、HelloWorldViewController.h）を表示したことを確認します。
4. キャンバスで、「Hello」ボタンからHelloWorldViewController.hのメソッド宣言領域（@interface文と@end文の間）まで、Controlキーを押しながらドラッグしてください。

すなわち、Controlキーを押しながらボタンを掴み、「Assistant」エディタペイン内のヘッダファイルまでドラッグします。

Controlキーを押しながらドラッグすると、次のようになるはずです。



ドラッグ操作を終えるとポップオーバーが現れます。ここで、今作成したアクション接続の設定をしていくことになります。



**注意** HelloWorldViewController.hのメソッド宣言領域以外でドラッグ操作を終えると、別の種類のポップオーバーが現れるか、または何も起こらないことになります。この場合、（必要ならば）キャンバス上のビュー内をクリックしてポップオーバーを閉じ、もう一度最初から実行してください。

5. ポップオーバー上で、次のように、ボタンのアクション接続の設定をしてください。
  - 「接続(Connection)」ポップアップメニューで「アクション(Action)」を選択します。
  - 「Name」フィールドに「changeGreeting:」と入力（コロンも入力することに注意）。

後のステップでchangeGreeting:メソッドを実装して、ユーザがテキストフィールドに入力した文字列が、ラベルに表示されるようにします。

- Typeフィールドにidがあることを確認します。

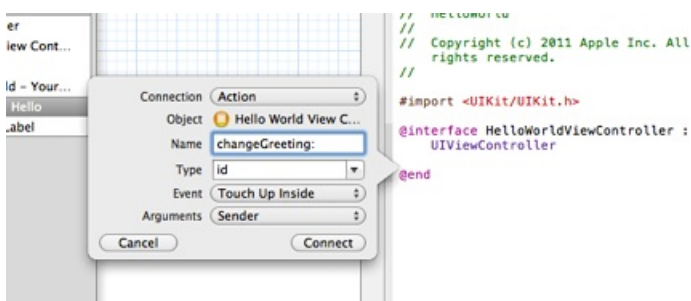
idデータ型は任意のCocoaオブジェクトを表せます。ここで「id」を使うのは、どのような型のオブジェクトがメッセージを送信してもよいようにするためです。

- 「Event」ポップアップメニューから「Touch Up Inside」を選択。

ここで「Touch Up Inside」イベントを選択するのは、ボタンの内側でユーザが指を持ち上げたときに、メッセージが送信されるようにするためです。

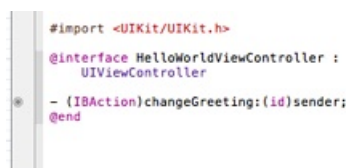
- 「Arguments」ポップアップメニューから「Sender」を選択。

アクション接続の設定が終わると、ポップオーバーは次のようになるはずです。



6. Popoverで「接続(Connect)」をクリックします。

するとchangeGreeting:メソッドのスタブ実装が追加されます。また、メソッドの左側にある中央が塗りつぶされた円は、接続が作成された旨を表します。



Controlキーを押しながら、「Hello」ボタンからHelloWorldViewController.hファイルまでドラッグし、作成されたアクションの設定を施したことにより、次の2つを達成したことになります。

- 適切なコードをView Controllerクラスに追加しました。具体的には、次のようなアクションメソッドの宣言を、HelloWorldViewController.hに追加しました。

```
- (IBAction)changeGreeting:(id)sender;
```

また、次のようなスタブ実装を、HelloWorldViewController.mに追加しました。

```
- (IBAction)changeGreeting:(id)sender {  
}  

```

**注意** IBActionは、このメソッドをターゲット／アクション接続のアクションとして扱うよう、Xcodeに指示するために使用される特殊なキーワードです。IBActionはvoidと定義されています。

アクションメソッドの引数senderは、アクションメッセージを送信したオブジェクトを表します（このチュートリアルではボタン）。

- ボタンとView Controllerの間に接続を作成しました。

次に、View Controllerと、残りの2つのUI要素（ラベルとテキストフィールド）との間に、接続を作成します。

## テキストフィールドおよびラベルのアウトレットを作成する

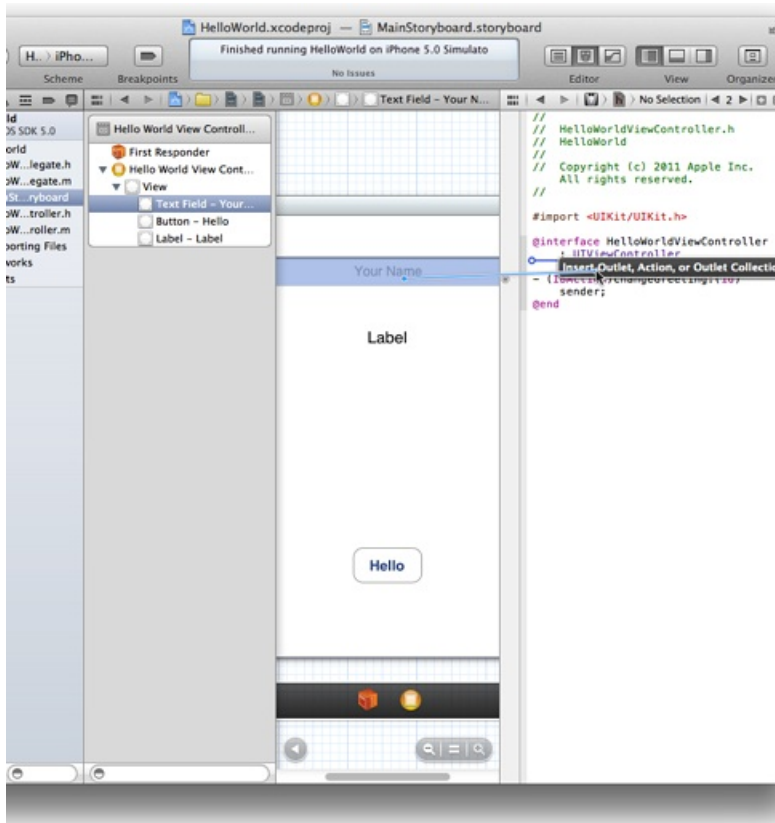
**アウトレット**は2つのオブジェクト間の接続を表します。（View Controllerなどの）オブジェクトが、それに含まれる（テキストフィールドなどの）オブジェクトと通信できるようにしたい場合は、含まれる側のオブジェクトをアウトレットと指定します。アプリケーション実行時には、Xcode上で作成したアウトレットがそのまま再現され、オブジェクト同士が相互に通信できるようになります。

このチュートリアルでは、View Controllerがテキストフィールドから、ユーザが入力した文字列を取得し、ラベルに表示できるようにしたいのです。View Controllerが他のオブジェクトと通信できるよう、相互間にアウトレット接続を作成します。

テキストフィールドやラベルにアウトレットを追加する手順は、ボタンのアクションを追加した際の手順とほとんど同じです。あらかじめ、メインストーリーボードファイルがキャンバス上に表示されたままであること、HelloWorldViewController.hが「Assistant」エディタ上に開いたままであることを確認してください。

1. Controlキーを押しながらビュー上のテキストフィールドを掴み、ヘッダファイルの宣言領域までドラッグしてください。

Controlキーを押しながらドラッグすると、次のようになるはずです。



ドラッグ後は、メソッド宣言領域内であればどこで放しても構いません。このチュートリアルでは、テキストフィールドやラベルのアウトレット宣言が、「Hello」ボタンのメソッド宣言の上に見えています。

2. ドラッグ操作を終えるとポップオーバーが現れます。ここで、テキストフィールドの接続の設定をしていくことになります。

- 「接続(Connection)」ポップアップメニューに「アウトレット(Outlet)」があることを確認します。
- 「名前(Name)」フィールドに、`textField`と入力します。

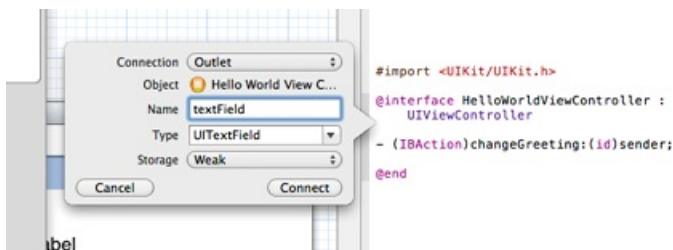
アウトレットの名前はどのように決めても構いませんが、それが表す項目と何らかの関係がある名前にしておけば、コードが分かりやすくなるでしょう。

- TypeフィールドにUITextFieldがあることを確認します。

「Type」フィールドから「UITextField」を選択したことにより、アウトレットがテキストフィールドだけに接続されます。

- 「Storage」ポップアップメニューから「Weak」（デフォルト値）を選択。

以上の設定が終わると、ポップオーバーは次のようになるはずです。



### 3. Popoverで「接続(Connect)」をクリックします。

テキストフィールドにアウトレットを追加したことにより、次の2つを達成したことになります。

- 適切なコードをView Controllerクラスに追加しました。特に、次の宣言をHelloWorldViewController.hに追加しました。

```
@property (weak, nonatomic) IBOutlet UITextField *textField;
```

**注意** IBOutletは、このオブジェクトをアウトレットとして扱うように、Xcodeに指示するためだけに使用される特殊なキーワードです。実際には何も定義されないので、コンパイル時には何の影響もありません。

また、次の文をHelloWorldViewController.mのviewDidLoadメソッドに追加しました。

```
self.setTextField:nil;
```

viewDidLoadメソッドは、先に選択したXcodeテンプレートに組み込まれていたものです（UIKitフレームワークによる実装）。View ControllerはviewDidLoadを、自分自身に含まれるビューを解放する必要があるときに呼び出します。したがって、ビューのアウトレットをnilに設定する場所としてちょうどよいことになります。

- View Controllerからテキストフィールドへの接続を確立しました。

View Controllerとテキストフィールド間の接続を確立すると、ユーザが入力したテキストがView Controllerに渡されるようになります。changeGreeting:メソッドの宣言と同様に、Xcodeは接続が作成された旨を、テキストフィールド宣言の左側に中央を塗りつぶした円を表示することにより示します。



次にラベルにアウトレットを追加し、接続の設定をしましょう。View Controllerとラベル間の接続を確立すると、View Controllerは、ユーザが入力したテキストに合わせてラベル文字列を更新するようになります。手順はテキストフィールドのときとほとんど同じですが、設定内容は若干異なります（HelloWorldViewController.hが「Assistant」エディタに開いたままになっているはずです）。


1. Controlキーを押しながらビュー上のラベルを掴み、「Assistant」エディタ上に開かれている、HelloWorldViewController.hのメソッド宣言領域までドラッグしてください。
2. ドラッグ操作を終えるとポップオーバーが現れます。ここで、ラベルの接続の設定をしていくことになります。
  - 「接続(Connection)」ポップアップメニューに「アウトレット(Outlet)」があることを確認します。
  - 「名前(Name)」フィールドに、labelと入力します。
  - TypeフィールドにUILabelがあることを確認します。
  - 「Storage」ポップアップメニューから「Weak」を選択。
3. Popoverで「接続(Connect)」をクリックします。

ここまでの作業で、View Controllerへの接続を3つ作成しました。


- ボタンへのアクション接続
- テキストフィールドへのアウトレット接続
- ラベルへのアウトレット接続

以上の接続を、「Connections」インスペクタで確認できます。

1. 「Standard」エディタボタンをクリックして、「Assistant」エディタを閉じ、標準エディタビューに切り替えてください。

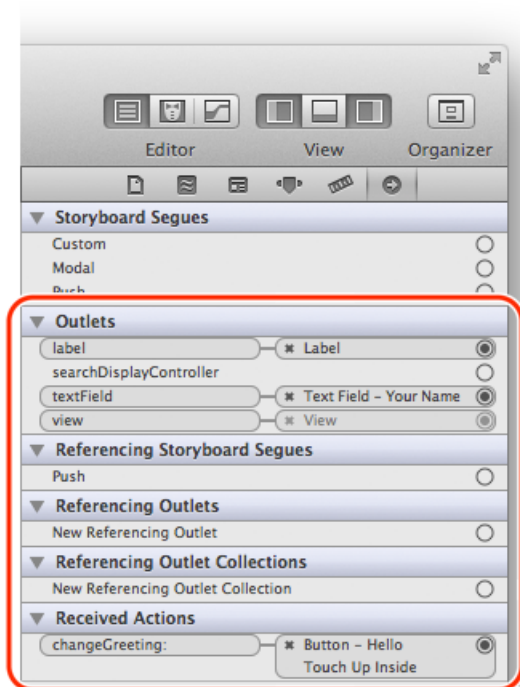
「Standard」エディタボタンは、一連の「Editor」ボタンの一番左側にあり、次のような外観をしています：

2. 「Utilities」ビューボタンをクリックして、ユーティリティ領域を開きます。
3. アウトレットビューで「Hello World View Controller」を選択してください。
4. ユーティリティ領域に「Connections」インスペクタを表示します。

「Connections」インスペクタボタンは、インスペクタセレクトバーの一番右側にあり、次のような外観をしています：



「Connections」インスペクタには、選択状態のオブジェクト（この場合はView Controller）への接続が表示されます。ワークスペースウィンドウの表示は次のようになるはずです。



先に作成した3つの接続に加え、View Controllerとビューを結ぶ接続もあることに注意してください。これはXcodeが自動的に作成するものであり、通常、意識する必要はありません。

## テキストフィールドのデリゲート接続を作成する

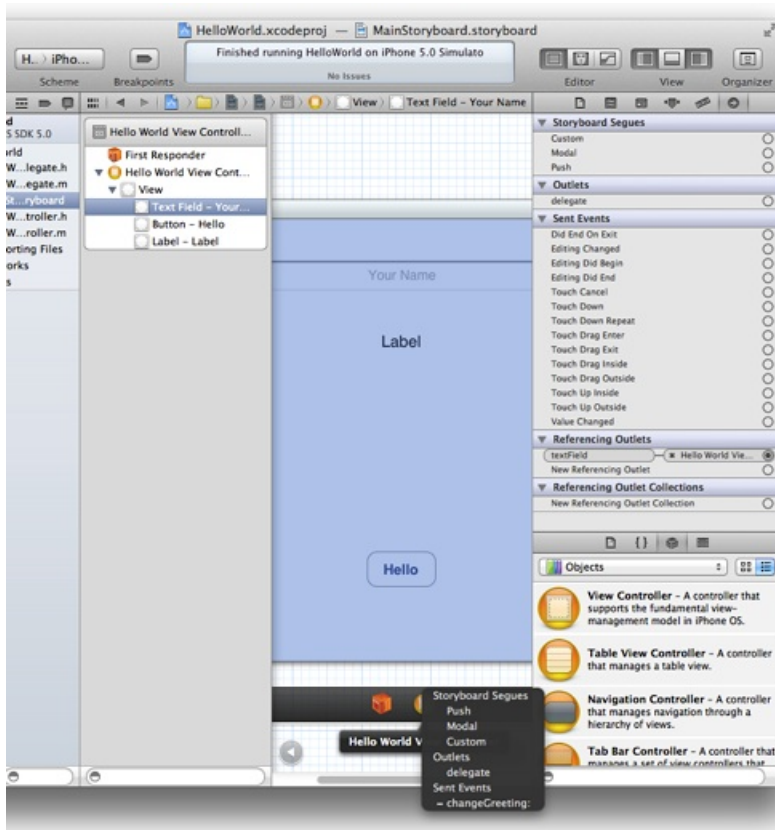
さらにもう1つ、接続を作成します。テキストフィールドを、そのデリゲートとして指定したオブジェクトに接続する必要があります。このチュートリアルでは、View Controllerをテキストフィールドのデリゲートとして使います。

テキストフィールドのデリゲートオブジェクトを指定する必要があるのは、ユーザがキーボードの「Done」ボタンをタップしたとき、テキストフィールドはそのデリゲートにメッセージを送信するからです（デリゲートとは、別のオブジェクトに代わってある処理を行うオブジェクトのことでした）。後のステップで、このメッセージに対応づけられたメソッドを使って、キーボードを画面から消すこととなります。

ストーリーボードファイルがキャンバス上に開いたままになっていることを確認してください。開いていない場合は、プロジェクトナビゲータでMainStoryboard.storyboardを選択します。

1. ビュー上で、Controlキーを押しながらテキストフィールドを掴み、シーンドック内の黄色い円内にドラッグしてください（黄色の円はView Controllerオブジェクトを表します）。

ドラッグ後、マウスボタンを放すと、次のようになるはずですが。



- ここに現れている透明なパネルの「Outlets」セクションで、「delegate」を選択してください。

## アプリケーションのテスト

「Run」をクリックするとアプリケーションの動作をテストできます。

「Hello」ボタンをクリックするとハイライト状態になるはずですが。また、テキストフィールド内をクリックすると、キーボードが表示されてテキストを入力できます。しかしこのキーボードを閉じる手段はまだありません。そのためには、それに関連するデリゲートメソッドを実装する必要があります。これは、次の章で行います。ここで、シミュレータを終了します。

## まとめ

キャンバス上の**View Controller**と、「**Assistant**」エディタで開いた**HelloWorldViewController.h**ファイルとの間に、適切な接続を作成したことにより、実装ファイル（すなわち**HelloWorldViewController.m**）も、アウトレットやアクションに対応できるよう更新されました。実装ファイルにどのような変更が施されたか調べたい場合は、プロジェクトナビゲータで開いてください。

Xcodeには、**Control**キーを押しながらキャンバスからソースファイルにドラッグして接続を確立すれば、自動的にコードが追加される、という機能がありますが、どうしてもこの方法によらなければならないわけではありません。プロパティ宣言やメソッド宣言を、自分でヘッダファイルに追加し、接続を作成してもよいのです（今回はテキストフィールドのデリゲートとの接続）。しかし通常は、こういった処理をXcodeに委ねることにより、つまらない誤り（およびタイピング量）を減らすことができます。

# View Controllerの実装

View Controllerを実装するには、いくつかの手順があります。ユーザ名を表すプロパティを追加し、`changeGreeting:`メソッドを実装し、ユーザが「Done」をタップするとキーボードが消えるようにする必要があります。

## ユーザ名を表すプロパティを追加する

ユーザ名を保持する文字列のプロパティ宣言を追加して、コードからいつでも参照できるようにする必要があります。この宣言は、View Controllerのヘッダファイル（すなわち `HelloWorldViewController.h`）に追加します。

**プロパティ宣言**とは、（たとえばユーザ名を保持する）変数のアクセサメソッドをどのように生成するか、コンパイラに対して指示するための記述子です（アクセサメソッドについてはプロパティ宣言を追加した後で説明します）。

この時点では、ストーリーボードファイルにさらに変更を施す必要はありません。この後の手順でコードを追加する際に画面を広く使えるよう、「Utilities View」ボタンをもう一度クリック（または「View」>「Utilities」>「Hide Utilities」コマンドを実行）して、ユーティリティ領域を非表示にしておいてください。

1. プロジェクトナビゲータで、`HelloWorldViewController.h`を選択します。
2. `@end`文の前に、文字列を宣言する`@property`文を追加します。

プロパティ宣言は次のようになるはずです。

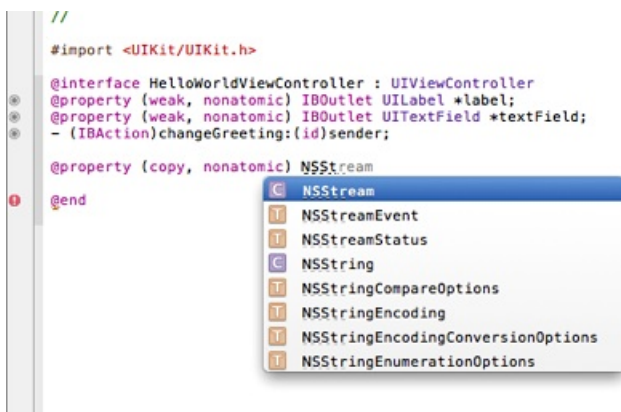
```
@property (copy, nonatomic) NSString *userName;
```

上記のコードをエディタペインにコピーしても、直接入力しても構いません。直接入力の場合、途中まで入力すればXcodeが自動的に補ってくれます。たとえば「@pro...」まで入力すれば、Xcodeは「@property」と入力するものと推測し、次のように、インラインのサジェスションパネルにこのシンボルを表示します。



(上図のように) 推測が正しければ、Returnキーを押して確定します。

入力の途中でいくつか候補が示され、選択できるようになることもあります。たとえば「NSSt...」まで入力すると、次のような候補リストが表示されます。



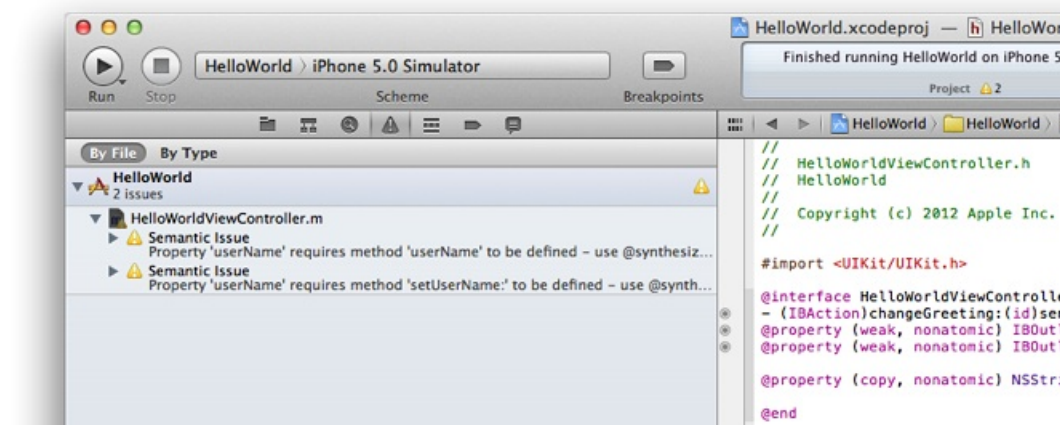
ここでReturnキーを押せば、強調表示されている候補に確定します。しかし（上図のように）この候補が望むものでない場合は、矢印キーで選択してください。

userNameプロパティの実装を完了するためには、アクセサメソッドを合成するようにコンパイラに指示する必要があります。**アクセサメソッド**とは、オブジェクトのプロパティ値を取得/設定するメソッドのことです（「ゲッタ」、「セッタ」と呼ぶこともあります）。

Xcodeはアクティビティビューアに黄色の警告アイコンを表示して注意を喚起します。



この場合、何を警告されているかは明らかなので、詳細を調べる必要はないでしょう。しかし必要な場合は、アクティビティビューアの警告アイコンをクリックすれば、イシューナビゲータに詳細が表示されます。



このチュートリアルでこれ以上イシューナビゲータを使うことはないので、ナビゲータセクタバーの一番左にあるボタンをクリックして、プロジェクトナビゲータに戻ってください。

このチュートリアルでは、**View Controllerの実装ファイル** (HelloWorldViewController.m) にアクセサメソッドを合成するよう、コンパイラに指示するコードを記述します。

1. プロジェクトナビゲータで、HelloWorldViewController.mを選択します。
2. @implementation HelloWorldViewController行の後に、次のコード行を追加します。

```
@synthesize userName = _userName;
```

このコードを追加すれば、アクセサメソッドが必要であるとの警告に対処したことになるので、アクティビティビューアの警告アイコンは消えます。

コンパイラは@synthesize記述子を見つけると、次の2つのアクセサメソッドを自動的に生成します。

```
- (NSString *)userName  
  
- (void)setUserName:(NSString *)newUserName
```

@synthesizeコード行のuserNameにアンダースコアを追加して、\_userNameを、userNameプロパティのインスタンス変数の名前として使用するようコンパイラに指示します。クラスでは\_usernameというインスタンス変数を宣言していないため、このコード行では、コンパイラに合成させるための指示もしています。

**注意** コンパイラはアクセサメソッドを、コンパイル済みコードの方に生成します。ソースコードに追加されるわけではありません。

## changeGreeting:メソッドの実装

“ビューの設定” (27 ページ) では、ユーザが「Hello」 ボタンをタップしたとき、View ControllerにchangeGreeting:メッセージが送信されるよう設定しました。View Controllerはこれに応じて、ユーザがテキストフィールドに入力したテキストを、ラベルに表示するようにしなければなりません。具体的には、changeGreeting:メソッドは次の処理を行う必要があります。

1. テキストフィールドから文字列を取得し、これをView ControllerのuserNameプロパティに設定する。
  2. userNameプロパティに基づき新しい文字列を生成し、ラベルに表示する。
1. 必要ならばプロジェクトナビゲータで、HelloWorldViewController.mを選択しておきます。  
自動的に追加されたchangeGreeting:のスタブ実装を確認するためには、ファイル末尾までスクロールしなければならないかも知れません。
  2. changeGreeting:メソッドのスタブ実装に、次のコードを追加してください。

```
- (IBAction)changeGreeting:(id)sender {  
  
    self.userName = self.textField.text;  
  
    NSString *nameString = self.userName;  
    if ([nameString length] == 0) {  
        nameString = @"World";  
    }  
}
```



```
NSString *greeting = [[NSString alloc] initWithFormat:@"Hello, %@!",
nameString];

self.label.text = greeting;
}
```

changeGreeting:メソッドにはいくつか興味深い点があります。

- 「self.userName = self.textField.text;」というコードは、テキストフィールドからテキストを取得して、その結果をView ControllerのuserNameプロパティに代入します。

このチュートリアルでは、ユーザ名を保持する文字列をほかのどこかで使用するわけではありません。ただし、その役割を覚えておくことは重要です。この文字列インスタンス変数は、View Controllerが管理している非常に単純なModelオブジェクトです。一般に、コントローラは、アプリケーションデータを、固有のModelオブジェクト内に保持しなければなりません。アプリケーションデータをユーザインターフェイス要素（たとえばHelloWorldアプリケーションのテキストフィールド）内に格納してはいけません。

- 「NSString \*nameString = self.userName;」というコードは、（NSString型の）新しい変数を生成し、View ControllerのuserNameプロパティを設定します。
- @“World”は、NSStringのインスタンスで表される文字列定数です。ユーザがアプリケーションを起動したけれどもテキストを何も入力しなかった（すなわち、[nameString length] == 0）場合、nameStringの値は「World」となります。
- initWithFormat:メソッドはFoundationフレームワークに組み込まれています。このメソッドは、（おそらく使い慣れているであろう、Cのprintf関数と同様に、）書式文字列で指定された書式に従って新規文字列を作成します。

書式文字列中の「%@」は、文字列オブジェクトのプレースホルダとして働きます。書式文字列の二重引用符内にあるほかの文字はすべて、そのまま画面に表示されます

## View Controllerをテキストフィールドのデリゲートとして設定する

アプリケーションをビルドして実行すると、ボタンをクリックしたとき、ラベルに“Hello, World!”と表示されます。テキストフィールドを選択して、キーボード入力を開始すると、テキスト入力が完了したことを示す方法とキーボードを閉じる方法がないことに気付くでしょう。

iOSアプリケーションでは、キーボードは、テキスト入力を許可する要素がファーストレスポンドになると自動的に表示され、要素がファーストレスポンドステータスでなくなると自動的に閉じられます（ファーストレスポンドとは、テキストフィールドをタップしてキーボードを表に出すなど、各種



のイベントを最初に受け取るオブジェクトのことでした)。アプリケーションからキーボードに、メッセージを直接送信する方法はありません。しかし、テキストエントリ要素のファーストレスポンドステータスの切り替えによる副次的な効果として、キーボードを表示／非表示にすることができます。

UITextFieldDelegateプロトコルは、UIKitフレームワークで定義されているもので、ユーザが「Return」ボタンをタップしたときにテキストフィールドを呼び出す（ボタンに表示されるテキストに関わらず）、textFieldShouldReturn:メソッドが含まれています。View Controllerはテキストフィールドのデリゲートとして設定したため（[“テキストフィールドのデリゲートを設定するには”](#)（? ページ）を参照）、このメソッドを実装して、resignFirstResponderメッセージを送信する（キーボードを閉じる効果を持つ）ことによって、テキストフィールドからファーストレスポンドステータスを強制的になくすることができます。

---

**注意** プロトコルは、基本的には、単なるメソッドのリストです。クラスがプロトコルに従っていれば（つまり「採用」していれば）そのクラスが、プロトコルの必須メソッドを実装していることが保証されます（プロトコルにはオプションメソッドも含まれている場合があります）。**デリゲートプロトコル**は、1つのオブジェクトがデリゲートに送信できるすべてのメッセージを定義しています。

---

1. 必要ならばプロジェクトナビゲータで、HelloWorldViewController.mを選択しておきます。
2. textFieldShouldReturn:メソッドを実装します。

このメソッドは、テキストフィールドに対し、ファーストレスポンドステータスをなくすよう指示しなければなりません。実際の実装は次のようになるはずです。

```
- (BOOL)textFieldShouldReturn:(UITextField *)theTextField {
    if (theTextField == self.textField) {
        [theTextField resignFirstResponder];
    }
    return YES;
}
```

このアプリケーションでは、テキストフィールドが1つしかないので、実際にはtheTextField == self.textFieldのテストを含める必要はありません。それでも、このテストを入れておくのは、よいパターンであると言えます。このオブジェクトが、同じタイプの複数のオブジェクトのデリゲートになる可能性があり、その場合、どのオブジェクトのデリゲートかを区別する必要が生じるからです。

3. プロジェクトナビゲータでHelloWorldViewController.hを選択します。

4. @interface行の末尾に「<UITextFieldDelegate>」を追加します。  
インターフェイス宣言は次のようになります。

```
@interface HelloWorldViewController : UIViewController <UITextFieldDelegate>
...

```

この宣言は、HelloWorldViewControllerクラスがUITextFieldDelegateプロトコルを採用していることを表します。

## アプリケーションのテスト

アプリケーションをビルドして実行します。今回は想定通りに動作するはずです。シミュレータ上で、名前の入力完了して「Done」をクリックすると、キーボードが閉じます。次に「Hello」ボタンをクリックすると、ラベルに“Hello, 入力した名前!”と表示されます。

アプリケーションが意図した通りに動作しない場合は、問題を修正する必要があります（まず確認すべき事項については[“トラブルシューティングとコードのレビュー”](#)（51 ページ）を参照）。

## まとめ

View Controllerの実装が完了したので、初めてのiOSアプリケーションは完成です。おめでとうございます。

# トラブルシューティングとコードのレビュー

アプリケーションが正しく機能しないトラブルが発生した場合は、この章に記載された問題解決のアプローチを試します。アプリケーションが期待どおりに動作しない場合は、自分のコードと、この章の末尾に示したリストを比較してください。

## コードおよびコンパイラの警告

コードは、どのような警告も表示されることなく、コンパイルされなければなりません。警告に対しては、エラーが表示された場合と同様に対処するようお勧めします。**Objective-C**言語は、非常に柔軟な言語なので、コンパイラから出力されるメッセージがせいぜい警告どまりである場合もあります。

## ストーリーボードファイルのチェック

アプリケーションが正しく動作しない場合、普通はまず、ソースコードにバグがあるのではないかと疑うでしょう。しかし、**Cocoa Touch**を使用しているときは、別の面が加わります。アプリケーションの設定の多くの部分が、ストーリーボードに“エンコード”されている場合があります。たとえば、正しい接続ができていなければ、アプリケーションは期待通りに動作しません。

- ボタンがクリックされたときにテキストが更新されない場合は、ボタンのアクションが**View Controller**に接続されていないか、**View Controller**のアウトレットがテキストフィールドまたはラベルに接続されていない可能性があります。
- 「**Done**」をクリックしたときにキーボードが閉じない場合は、テキストフィールドのデリゲートを接続していないか、あるいは**View Controller**の**textField**アウトレットをテキストフィールドに接続している可能性があります。ストーリーボード上で、テキストフィールドの接続を確認してください。**Control**キーを押しながらテキストフィールドをクリックすると、半透明の接続パネルが現れます。**delegate**アウトレットと、これを参照する**textField**の隣に、黒丸が表示されているはずです。

デリゲートを接続している場合は、もっと些細な問題が原因かも知れません（“デリゲートメソッド名”を参照）。

## デリゲートメソッド名

デリゲートに関してよくある間違いは、デリゲートメソッド名のスペル間違いです。たとえデリゲートオブジェクトを正しく設定しても、そのデリゲートがメソッドの実装で正確な名前を使用していなければ、正しいメソッドを呼び出すことはできません。通常は、`textFieldShouldReturn:`など、デリゲートメソッドの宣言をこのドキュメントからコピー＆ペーストするのが一番良い方法です。

## コードリスト

この節では、`HelloWorldViewController`クラスのインターフェイスファイル、実装ファイルのコードリストを示します。このコードリストには、Xcodeテンプレートのコメントおよびその他のメソッドの実装を示しません。

### インターフェイスファイル：HelloWorldViewController.h

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

@interface HelloWorldViewController : UIViewController <UITextFieldDelegate>

@property (weak, nonatomic) IBOutlet UITextField *textField;
@property (weak, nonatomic) IBOutlet UILabel *label;
@property (nonatomic, copy) NSString *userName;

- (IBAction)changeGreeting:(id)sender;

@end
```

### 実装ファイル：HelloWorldViewController.m

```
#import "HelloWorldViewController.h"

@implementation HelloWorldViewController

@synthesize textField=_textField;
@synthesize label=_label;
```

```
@synthesize userName=_userName;

- (BOOL)textFieldShouldReturn:(UITextField *)textField {
    if (textField == self.textField) {
        [textField resignFirstResponder];
    }
    return YES;
}

- (IBAction)changeGreeting:(id)sender {
    self.userName = self.textField.text;

    NSString *nameString = self.userName;
    if ([nameString length] == 0) {
        nameString = @"World";
    }
    NSString *greeting = [[NSString alloc] initWithFormat:@"Hello, %@!", nameString];
    self.label.text = greeting;
}

@end
```

# 書類の改訂履歴

この表は「初めてのiOSアプリケーション」の改訂履歴です。

日付	メモ
2012-02-16	Updated for Xcode 4.3.
2011-10-12	ストーリーボードおよびARCを使うよう、内容を更新しました。
2011-06-06	誤字を訂正しました。
2011-03-08	Xcode 4用に更新しました。
2010-10-12	リンクの間違いを修正しました。
2010-07-01	iOS 4用に更新しました。
2010-03-15	iOS 3.2用に更新しました。
2009-10-08	Interface Builderでテキストフィールドを接続する必要性を強調しました。
2009-08-10	アクセサメソッドを直接使用した例のドット構文への参照を削除しました。
2009-06-15	iOS 3.0用に更新しました。
2009-01-06	誤字を訂正しました。

日付	メモ
2008-10-15	ターゲットアクションデザインパターンの説明を分かりやすくしました。
2008-09-09	細かい点を訂正し、説明を分かりやすくしました。
2008-06-09	iPhone向けアプリケーションの開発を紹介する新規ドキュメント。



Apple Inc.  
© 2012 Apple Inc.  
All rights reserved.

本書の一部あるいは全部を Apple Inc. から書面による事前の許諾を得ることなく複写複製（コピー）することを禁じます。また、製品に付属のソフトウェアは同梱のソフトウェア使用許諾契約書に記載の条件のもとでお使いください。書類を個人で使用する場合に限り1台のコンピュータに保管すること、またその書類にアップルの著作権表示が含まれる限り、個人的な利用を目的に書類を複製することを認めます。

Apple ロゴは、米国その他の国で登録された Apple Inc. の商標です。

キーボードから入力可能な Apple ロゴについても、これを Apple Inc. からの書面による事前の許諾なしに商業的な目的で使用すると、連邦および州の商標法および不正競争防止法違反となる場合があります。

本書に記載されているテクノロジーに関しては、明示または黙示を問わず、使用を許諾しません。本書に記載されているテクノロジーに関するすべての知的財産権は、Apple Inc. が保有しています。本書は、Apple ブランドのコンピュータ用のアプリケーション開発に使用を限定します。

本書には正確な情報を記載するように努めました。ただし、誤植や制作上の誤記がないことを保証するものではありません。

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/>

Apple, the Apple logo, Cocoa, Cocoa Touch, iPad, iPhone, iTunes, Mac, Mac OS, Objective-C, OS X, and Xcode are trademarks of Apple Inc., registered in the United States and other countries.

iOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used under license.

Apple Inc. は本書の内容を確認しておりますが、本書に関して、明示的であるか黙示的であるかを問わず、その品質、正確さ、市場性、または特定の目的に対する適合性に関して何らかの保証または表明を行うものではありません。その結果、本書は「現状有姿のまま」提供され、本書の品質または正確さに関連して発生するすべての損害は、購入者であるお客様が負うものとなります。

いかなる場合も、Apple Inc. は、本書の内容に含まれる瑕疵または不正確さによって生じる直接的、間接的、特殊的、偶発的、または結果的損害に対する賠償請求には一切応じません。そのような損害の可能性があらかじめ指摘されている場合においても同様です。

上記の損害に対する保証および救済は、口頭や書面によるか、または明示的や黙示的であるかを問わず、唯一のものであり、その他一切の保証にかわるものです。Apple Inc. の販売店、代理店、または従業員には、この保証に関する規定に何らかの変更、拡張、または追加を加える権限は与えられていません。

一部の国や地域では、黙示あるいは偶発的または結果的損害に対する賠償の免責または制限が認められていないため、上記の制限や免責がお客様に適用されない場合があります。この保証はお客様に特定の法的権利を与え、地域によってはその他の権利がお客様に与えられる場合もあります。