iOS Address Bookプログラミングガイド

Developer

目次

図、リスト

はじめに

iOSのAddress Bookテクノロジーは、連絡先情報やその他の個人情報を1つの集中データベースに格納し、アプリケーション間でこの情報を共有する手段を提供します。このテクノロジーは、いくつかの要素で構成されています。

- Address Bookフレームワークは、連絡先情報へのアクセスを提供します。
- Address Book UIフレームワークは、情報を表示するためのユーザインターフェイスを提供します。
- Address Bookデータベースは、情報を格納します。
- 「連絡先(Contacts)」アプリケーションは、ユーザが連絡先情報にアクセスする手段を提供します。

このドキュメントでは、Address Bookテクノロジーの重要な概念について解説し、実行可能な基本的な操作について説明します。このテクノロジーをアプリケーションに追加すると、ユーザは、「メール(Mail)」や「SMS(Text)」などのほかのアプリケーションで使用している連絡先情報を、そのアプリケーションで使用できます。このドキュメントでは、以下の方法について説明します。

- ユーザのAddress Bookデータベースにアクセスする方法
- 連絡先情報をユーザに入力させる方法
- 連絡先情報をユーザに表示する方法
- ユーザのAddress Bookデータベースに変更を加える方法

このドキュメントを最大限に活用するために、Navigation Controller、View Controller、デリゲートおよびプロトコルについて理解していることが必要です。

> **注意** Mac OS XでAddress Bookテクノロジーを使用するデベロッパは、このテクノロジーのプログラミングインターフェイスがiOSでは異なることに注意してください。

この書類の構成

このドキュメントは、次の章で構成されています。

- "クイックスタートチュートリアル" （6 ページ）では、入門として、Address Bookテクノロジーを使用した簡単なアプリケーションの作成方法を示します。

- "基本要素：レコードとプロパティの操作"（11 ページ）では、Address Bookオブジェクトの作成方法、PersonレコードとGroupレコードの作成方法、プロパティの取得および設定方法について説明します。
- "直接のやり取り：データの選択と表示"（19 ページ）では、Address Book UIフレームワークのビューを使用して、連絡先を表示したり、ユーザに連絡先の選択、新規の連絡先の作成、連絡先の編集をさせる方法について説明します。
- "直接のやり取り：プログラムによるデータベースへのアクセス"（25 ページ）では、アプリケーションによって連絡先情報を直接読み取りおよび書き込みする方法について説明します。

関連項目

以下のドキュメントは、このドキュメントを最大限に活用するために理解しておくべきいくつかの基本概念について解説しています。

- 『*iOS App Programming Guide*』は、iOSプラットフォームは初めてというデベロッパを対象に、利用可能なテクノロジーを紹介し、それらを使用してアプリケーションを作成する方法を説明しています。このドキュメントには、ウインドウ、ビュー、およびView Controllerに関連する解説が含まれています。
- *Interface Builder User Guide*』は、Interface Builderを使用してアプリケーションを作成する方法を説明しています。このドキュメントには、アプリケーションのユーザインターフェイスについてと、ユーザインターフェイスとコードを結び付ける方法に関する解説が含まれています。
- 『*Cocoa Fundamentals Guide*』と『*The Objective-C Programming Language*』は、アプリケーションを記述するために必要なさまざまな基本概念について解説しています。このドキュメントには、デリゲートとプロトコルに関する解説が含まれています。

以下のドキュメントには、Address Bookフレームワークに関する補足情報が含まれています。

- 『*Address Book Framework Reference for iOS*』は、Address Bookデータベースのレコードと直接やり取りをするためのAPIについて説明しています。
- 『*Address Book UI Framework Reference for iOS*』は、Address Bookデータベースのレコードの表示、編集、選択、および作成を支援するコントローラと、それらのデリゲートプロトコルについて説明しています。

クイックスタートチュートリアル

このチュートリアルでは、連絡先リストから一人の人物を選ぶようユーザに求め、選択された人物の名前と電話番号を表示する簡単なアプリケーションを作成します。

プロジェクトの作成と設定

1. Xcodeで、Single View Applicationテンプレートから新規プロジェクトを作成します。
2. Address Book UIおよびAddress Bookの2つのフレームワークをプロジェクトにリンクしてください。

> **Important** これを怠るとプロジェクトはビルドできません（リンカのエラー）。

UIの構築とヘッダファイルの生成

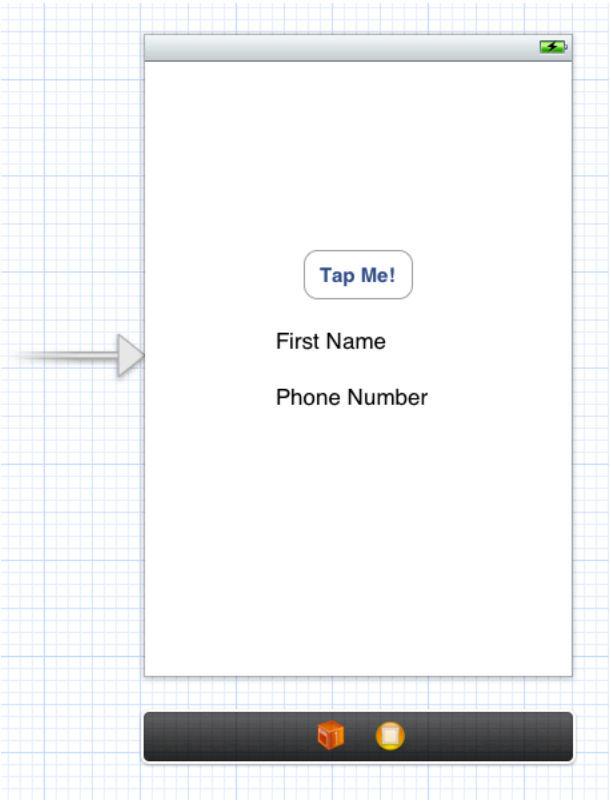
ユーザインターフェイスを構築する際には、必要なアクションやプロパティを宣言してヘッダファイルの大部分を生成する、というXcodeの機能を活用できます。

1. メインストーリーボードファイル（`MainStoryboard.storyboard`）を開きます。
2. 「ライブラリ(library)」パネルから1つのボタンと2つのラベルをビューにドラッグして追加します。図 1-1に示すようにこれらを配置します。
3. Assistant editorを開きます。
4. View Controller上で、`showPicker:`という新しいアクションメソッドに、ボタンを接続します。

   この結果、ストーリーボード上でボタンのターゲットとアクションを設定し、メソッド宣言をヘッダファイルに追加し、そのスタブ実装を実装ファイルに追加したことになります。スタブ実装の実際の中身は後で記述します。
5. `firstName`および`phoneNumber`という、View Controllerの新しいプロパティを、2つのラベルにそれぞれ接続します。

この結果、ストーリーボード上で接続を作成し、プロパティ宣言をヘッダファイルに追加し、プロパティの`@synthesize`行を実装ファイルに追加したことになります。

図 1-1 インターフェイスのレイアウト



この時点で、View Controllerのヘッダファイルである`ViewController.h`はほとんど完成しています。`@interface`行の末尾に次の記述を追加して、View Controllerクラスが`ABPeoplePickerNavigationControllerDelegate`プロトコルに準拠している旨を宣言します。

```
<ABPeoplePickerNavigationControllerDelegate>
```

リスト 1-1に完成したヘッダファイルを示します。

リスト 1-1 完成したヘッダファイル

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


@interface ViewController : UIViewController
<ABPeoplePickerNavigationControllerDelegate>
```

```
@property (weak, nonatomic) IBOutlet UILabel *firstName;
@property (weak, nonatomic) IBOutlet UILabel *phoneNumber;

- (IBAction)showPicker:(id)sender;

@end
```

実装ファイルの記述

View Controllerの実装ファイルViewController.mには、showPicker:メソッドのスタブ実装ができています。リスト 1-2に完全な実装例を示します。この実装では、新しいPeopleピッカーを作成し、このView Controllerをそのピッカーのデリゲート（delegate、委任）に設定します。そして、ピッカーをモーダルなView Controllerとして表示します。

リスト 1-2　Peopleピッカーの表示

```
- (IBAction)showPicker:(id)sender
{
    ABPeoplePickerNavigationController *picker =
            [[ABPeoplePickerNavigationController alloc] init];
    picker.peoplePickerDelegate = self;

    [self presentModalViewController:picker animated:YES];
}
```

Peopleピッカーはユーザのアクションに応答して、デリゲートのメソッドを呼び出します。リスト 1-3にこのメソッドの実装を示します。ユーザが操作をキャンセルした場合は、最初のメソッドが呼び出されてPeopleピッカーが消去されます。ユーザが人物を選択した場合は、2番目のメソッドが呼び出されて、その人物の名前と電話番号がそれぞれのラベルにコピーされ、Peopleピッカーが消去されます。

ユーザがピッカー内で選択されている人物のプロパティをタップしたときに、Peopleピッカーは3つめのメソッドを呼び出します。このアプリケーションでは、ユーザが人物を選択すると、Peopleピッカーは常に消去されます。したがって、ユーザがその人物のプロパティを選択する手段はありません。つまり、このメソッドが呼び出されることはありません。しかし、この関数を入れなければ、プロトコルの実装が不完全になります。

リスト 1-3　Peopleピッカーのユーザアクションに対する応答

```
- (void)peoplePickerNavigationControllerDidCancel:
            (ABPeoplePickerNavigationController *)peoplePicker
{
    [self dismissModalViewControllerAnimated:YES];
}



- (BOOL)peoplePickerNavigationController:
            (ABPeoplePickerNavigationController *)peoplePicker
      shouldContinueAfterSelectingPerson:(ABRecordRef)person {

    [self displayPerson:person];
    [self dismissModalViewControllerAnimated:YES];

    return NO;
}

- (BOOL)peoplePickerNavigationController:
            (ABPeoplePickerNavigationController *)peoplePicker
      shouldContinueAfterSelectingPerson:(ABRecordRef)person
                                property:(ABPropertyID)property
                              identifier:(ABMultiValueIdentifier)identifier
{
    return NO;
}
```

最後にもうひとつ、名前と電話番号を表示するメソッドを実装します（リスト 1-4を参照）。名前と電話番号で実装が異なることに注意してください。名前は文字列プロパティで、Personレコードには1つだけ存在しますが、値をNULLとすることも可能です。一方、電話番号は多値プロパティです。同一Personレコードに、0個以上何個でも存在して構いません。この例では、リストの最初に現れる電話番号を使います。

リスト 1-4　Person情報の表示

```
- (void)displayPerson:(ABRecordRef)person
```

```
{
    NSString* name = (__bridge_transfer NSString*)ABRecordCopyValue(person,
                                                kABPersonFirstNameProperty);
    self.firstName.text = name;

    NSString* phone = nil;
    ABMultiValueRef phoneNumbers = ABRecordCopyValue(person,
                                    kABPersonPhoneProperty);
    if (ABMultiValueGetCount(phoneNumbers) > 0) {
       phone = (__bridge_transfer NSString*)
               ABMultiValueCopyValueAtIndex(phoneNumbers, 0);
    } else {
        phone = @"[None]";
    }
    self.phoneNumber.text = phone;
}
```

アプリケーションのビルドと実行

アプリケーションを実行すると、1つのボタンと2つの空のテキストラベルが最初に表示されます。ボタンをタップするとPeopleピッカーが表示されます。人物を選択すると、Peopleピッカーは消えて選択した人物の姓名が各ラベルに表示されます。

基本要素：レコードとプロパティの操作

Address Bookデータベースと本格的にやり取りするには、4つの基本的なオブジェクト、すなわちAddress Book、レコード、単一値プロパティ、多値プロパティを理解しておく必要があります。この章では、これらのオブジェクトにどのようにデータが格納されているかについて解説し、それらとやり取りをするための関数について説明します。

Address Bookデータベースと直接やり取りをする方法（たとえば、Personレコードの追加や削除の方法）については、“直接のやり取り：プログラムによるデータベースへのアクセス”（25 ページ）を参照してください。

Address Book

Address Bookオブジェクトを使用すると、Address Bookデータベースとやり取りできます。Address Bookを使用するには、`ABAddressBookRef`のインスタンスを宣言し、それを`ABAddressBookCreate`関数から返された値に設定します。複数のAddress Bookオブジェクトを作成できますが、それらはすべて同一の共有データベースに基づきます。

> **Important** `ABAddressBookRef`のインスタンスを複数のスレッドで使用することはできません。各スレッドは、それ自身のインスタンスを作成する必要があります。

Address Bookの参照を作成した後は、そこからデータを読み込んだり、変更を保存できたりします。変更を保存するには、`ABAddressBookSave`関数を使用します。変更を破棄するには、`ABAddressBookRevert`関数を使用します。保存されていない変更があるかを確認するには、`ABAddressBookHasUnsavedChanges`関数を使用します。

以下のコードは、変更を加えてから、変更をAddress Bookデータベースに保存する一般的なコードパターンを示しています。

```
ABAddressBookRef addressBook;
bool wantToSaveChanges = YES;
bool didSave;
CFErrorRef error = NULL;

addressBook = ABAddressBookCreate();
```

```
/* ... Address Bookを対象とした操作 ... */

if (ABAddressBookHasUnsavedChanges(addressBook)) {
    if (wantToSaveChanges) {
        didSave = ABAddressBookSave(addressBook, &error);
        if (!didSave) {/*ここでエラーを処理 */}
    } else {
        ABAddressBookRevert(addressBook);
    }
}

CFRelease(addressBook);
```

アプリケーションでは、別のアプリケーション（または同じアプリケーション内の別のスレッド）によってAddress Bookデータベースに変更が加えられたときに通知を受け取れるように要求することができます。一般に、既存の連絡先を表示しているときに、アプリケーションの実行中に発生した連絡先への変更を反映してUIを更新したい場合は、通知を登録する必要があります。

`ABExternalChangeCallback`プロトタイプの関数を登録するには、`ABAddressBookRegisterExternalChangeCallback`関数を使用します。異なるコールバックやコンテキストを渡して`ABAddressBookRegisterExternalChangeCallback`を複数回呼び出すことによって、複数の変更コールバックを登録することもできます。`ABAddressBookUnregisterExternalChangeCallback`を使用して、関数の登録を取り消すこともできます。

変更のコールバックを受け取ったら、2つのことができます。未保存の変更がなければ、コードでは単純にAddress Bookを元の状態に戻して最新のデータを取得するようにします。未保存の変更がある場合には、元の状態に戻してそれらの変更を失うことはしたくないでしょう。その場合には、保存を行います。Address Bookデータベースは、加えた変更と外部の変更を統合するよう最善を尽くします。ただし、変更をマージできずに保存に失敗した場合に、適切なアクションが取れるように準備しておく必要があります。

レコード

Address Bookデータベースでは、情報はレコードに格納され、`ABRecordRef`オブジェクトで表されます。各レコードは、1人の人物または1つのグループを表します。`ABRecordGetRecordType`関数は、レコードが人物の場合には`kABPersonType`を返し、グループの場合には`kABGroupType`を返します。Mac OSのAddress Bookテクノロジーに精通しているデベロッパであれば、これらが異なるタイプのレコードの別々のクラスではないことに気付くはずです。PersonオブジェクトもGroupオブジェクトも同じクラスのインスタンスです。

Important　レコードオブジェクトは、スレッド同士で安全に受け渡しを行うことができません。代わりに、対応するレコードの識別子を渡す必要があります。詳細については、“レコード識別子の使用”（25 ページ）を参照してください。

通常、レコードはAddress Bookデータベースの一部ですが、データベースの外部に存在する場合もあります。そのほうが、アプリケーションで操作中の連絡先情報を格納するのに便利な場合もあります。

レコード内では、データはプロパティのコレクションとして格納されます。Personオブジェクトと Groupオブジェクトで利用可能なプロパティは異なりますが、それらにアクセスするために使用する関数は同じです。`ABRecordCopyValue`関数と`ABRecordSetValue`関数は、それぞれプロパティの取得と設定を行います。`ABRecordRemoveValue`関数を使用して、プロパティを完全に削除することもできます。

Personレコード

Personレコードは、単一値プロパティと多値プロパティの両方で構成されています。姓、名など、1人の人物が1つしか持つことができないプロパティは、単一値プロパティとして格納されます。住所、電話番号など、1人の人物が複数の値を持つ可能性があるプロパティは、多値プロパティです。Personレコードのプロパティは、「Constants」 in *ABPerson Reference* の複数のセクションで示しています。

Personレコードの内容を直接編集する関数の詳細については、“Personレコードを対象とした操作”（26 ページ）を参照してください。

Groupレコード

ユーザは、さまざまな理由で連絡先をグループに整理することがあります。たとえば、プロジェクトに関わる同僚のグループや、一緒にプレーするスポーツチームのメンバのグループを作成します。グループを利用すると、ユーザは、Address Bookの複数の連絡先に同時に特定のアクションを実行できます。

Groupレコードには、kABGroupNamePropertyというプロパティが1つだけあります。これは、グループの名前です。1つのグループ内のすべての人物を取得するには、ABGroupCopyArrayOfAllMembers関数またはABGroupCopyArrayOfAllMembersWithSortOrdering関数を使用します。これらは、ABRecordRefオブジェクトのCFArrayRefを返します。

Groupレコードの内容を直接編集する関数の詳細については、“Groupレコードを対象とした操作”（27 ページ）を参照してください。

プロパティ

プロパティには、単一値と多値の2種類があります。単一値プロパティは、人物の名前など、単一の値だけを持つことができるデータを含みます。多値プロパティは、人物の電話番号など、複数の値を持つことができるデータを含みます。多値プロパティは、可変にも不変にもできます。

Personレコードのプロパティのリストについては「Constants」 in *ABPerson Reference* 内のいくつかのセクションを参照してください。Groupレコードのプロパティについては、「Group Properties」 in *ABGroup Reference* を参照してください。

単一値プロパティ

以下に、単一値プロパティの値を取得したり設定するコードを示します。

```
ABRecordRef aRecord = ABPersonCreate();
CFErrorRef anError = NULL;
bool didSet;


didSet = ABRecordSetValue(aRecord, kABPersonFirstNameProperty, CFSTR("Katie"),
&anError);
if (!didSet) {/*ここでエラーを処理 */}


didSet = ABRecordSetValue(aRecord, kABPersonLastNameProperty, CFSTR("Bell"),
&anError);
if (!didSet) {/*ここでエラーを処理 */}


CFStringRef firstName, lastName;
firstName = ABRecordCopyValue(aRecord, kABPersonFirstNameProperty);
lastName  = ABRecordCopyValue(aRecord, kABPersonLastNameProperty);
```

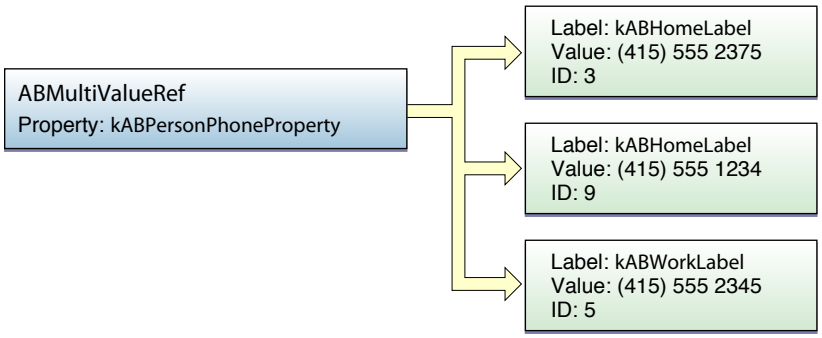
```
/* ... firstNameおよびlastNameを使って何かする ... */


CFRelease(aRecord);
CFRelease(firstName);
CFRelease(lastName);
```

多値プロパティ

多値プロパティは、値のリストで構成されます。各値は、1つのテキストラベルと、それに対応する1つの識別子を持ちます。複数の値が同じラベルを持つ場合もありますが、識別子は必ず一意です。一般的に使用されるいくつかのテキストラベルに対しては、定数が定義されています。「Generic Property Labels」 in *ABPerson Reference* を参照してください。

たとえば、図2-1は電話番号のプロパティを示しています。ここでは、1人の人物が複数の電話番号を持っています。それぞれの電話番号は、自宅や職場などのテキストラベルと識別子を持っています。この例では、自宅の電話番号が2つあります。これらはラベルは同じですが識別子は異なります。

図 2-1　　多値プロパティ



多値プロパティの個々の値は、内容に応じて、識別子またはインデックスで参照されます。`ABMultiValueGetIndexForIdentifier`および`ABMultiValueGetIdentifierAtIndex`関数を使用して、インデックスと多値の識別子の相互変換を行います。

多値プロパティの特定の値への参照を保持するには、その識別子を保存します。値が追加または削除されると、インデックスが変わります。識別子は、デバイスが変わる場合以外は、変わらないことが保証されています。

次の関数を使用すると、インデックスで指定する個々の値の内容を読み取ることができます。

- `ABMultiValueCopyLabelAtIndex`と`ABMultiValueCopyValueAtIndex`は、個々の値をコピーします。
- `ABMultiValueCopyArrayOfAllValues`は、すべての値を配列にコピーします。

可変の多値プロパティ

多値オブジェクトは不変です。多値オブジェクトに変更を加えるには、`ABMultiValueCreateMutableCopy`関数を使用して可変のコピーを作成する必要があります。`ABMultiValueCreateMutable`関数を使用して、可変の多値オブジェクトを新規に作成することもできます。

以下の関数を使用すると、可変の多値プロパティに変更を加えることができます。

- `ABMultiValueAddValueAndLabel`と`ABMultiValueInsertValueAndLabelAtIndex`は、値を追加します。
- `ABMultiValueReplaceValueAtIndex`と`ABMultiValueReplaceLabelAtIndex`は、値を変更します。
- `ABMultiValueRemoveValueAndLabelAtIndex`は、値を削除します。

以下に、多値プロパティを取得および設定するコードを示します。

```
ABMutableMultiValueRef multi =
        ABMultiValueCreateMutable(kABMultiStringPropertyType);
CFErrorRef anError = NULL;
ABMultiValueIdentifier multivalueIdentifier;
bool didAdd, didSet;

// ここでは、multivalueIdentifierは例を示すためだけに使用し、
// 後でリストで使用しない。実際のコードでは、この識別子を使用して
// 新規に追加された値を参照する。
didAdd = ABMultiValueAddValueAndLabel(multi, @"(555) 555-1234",
                        kABPersonPhoneMobileLabel, &multivalueIdentifier);
if (!didAdd) {/*ここでエラーを処理 */}

didAdd = ABMultiValueAddValueAndLabel(multi, @"(555) 555-2345",
                        kABPersonPhoneMainLabel, &multivalueIdentifier);
if (!didAdd) {/*ここでエラーを処理 */}

ABRecordRef aRecord = ABPersonCreate();
didSet = ABRecordSetValue(aRecord, kABPersonPhoneProperty, multi, &anError);
if (!didSet) {/*ここでエラーを処理 */}
```

```
CFRelease(multi);

/* ... */

CFStringRef phoneNumber, phoneNumberLabel;
multi = ABRecordCopyValue(aRecord, kABPersonPhoneProperty);

for (CFIndex i = 0; i < ABMultiValueGetCount(multi); i++) {
    phoneNumberLabel = ABMultiValueCopyLabelAtIndex(multi, i);
    phoneNumber      = ABMultiValueCopyValueAtIndex(multi, i);

    /* ... phoneNumberLabelおよびphoneNumberを使って何かする ... */

    CFRelease(phoneNumberLabel);
    CFRelease(phoneNumber);
}

CFRelease(aRecord);
CFRelease(multi);
```

住所

住所は、辞書による多値プロパティとして表現されます。多値プロパティについての上の説明は、住所にも当てはまります。各値は、自宅や職場などのラベルを1つ持ちます（「Generic Property Labels」in *ABPerson Reference* を参照）。多値プロパティ内の各値は、辞書として格納された1つの住所です。この値の内部では、辞書が住所のさまざまな部分のキーを保持します。これらは、「Address Property」in *ABPerson Reference* に示されています。

以下のコードリストは、住所を設定したり表示する方法を示します。

```
ABMutableMultiValueRef address =
        ABMultiValueCreateMutable(kABDictionaryPropertyType);

// 辞書のキーと値をセットアップする
CFStringRef keys[5];
CFStringRef values[5];
keys[0] = kABPersonAddressStreetKey;
```

```
keys[1] = kABPersonAddressCityKey;
keys[2] = kABPersonAddressStateKey;
keys[3] = kABPersonAddressZIPKey;
keys[4] = kABPersonAddressCountryKey;
values[0] = CFSTR("1234 Laurel Street");
values[1] = CFSTR("Atlanta");
values[2] = CFSTR("GA");
values[3] = CFSTR("30303");
values[4] = CFSTR("USA");

CFDictionaryRef aDict = CFDictionaryCreate(
        kCFAllocatorDefault,
        (void *)keys,
        (void *)values,
        5,
        &kCFCopyStringDictionaryKeyCallBacks,
        &kCFTypeDictionaryValueCallBacks
);

// 住所を多値プロパティに追加する
ABMultiValueIdentifier identifier;
bool didAdd;
didAdd = ABMultiValueAddValueAndLabel(address, aDict, kABHomeLabel, &identifier);
if (!didAdd) {/*ここでエラーを処理 */}
CFRelease(aDict);

/* ... 多値を使って、Personレコードに追加するなどの処理をする ...*/

CFRelease(address);
```

直接のやり取り：データの選択と表示

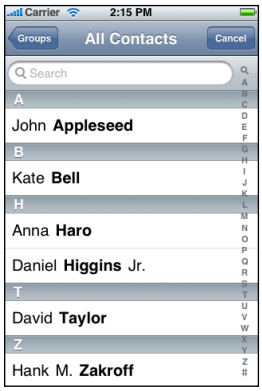
Address Book UIフレームワークは、Address Bookデータベースと連絡先情報の操作に関連する一般的なタスクのために、3つのView Controllerと1つのNavigation Controllerを提供しています。独自のコントローラを作成するよりもこれらを使用したほうが、必要な作業が減り、より一貫性のある体験をユーザに提供できます。

この章には、出発点として使用できる短いコードリストがいくつか含まれています。完全に動作するサンプルコードについては、『*QuickContacts*』を参照してください。

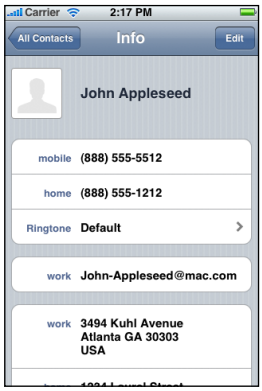
利用可能なコントローラ

Address Book UIフレームワークは以下の4つのコントローラを提供しています。

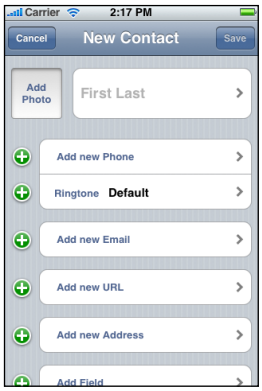
- `ABPeoplePickerNavigationController`。ユーザにAddress BookからPersonレコードを選択させることができます。
- `ABPersonViewController`。ユーザにPersonレコードを提示して、必要に応じてユーザに編集させることできます。
- `ABNewPersonViewController`。ユーザに新規のPersonレコードを作成させることができます。
- `ABUnknownPersonViewController`。ユーザに未完成のPersonレコードを完成させて、必要に応じてそのレコードをAddress Bookに追加させることができます。



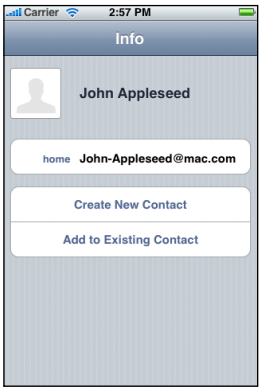
People picker



Person View Controller



New-Person View Controller



Unknown-Person View Controller

これらのコントローラを使用するには、これらにデリゲートを1つ設定して、適切なデリゲートプロトコルを実装しなければなりません。これらのコントローラのサブクラスを作成する必要はありません。コントローラの動作を変更するために想定されているのは、デリゲートの実装による方法です。この章では、これらのコントローラの詳細とその使用方法について学びます。

デリゲートの詳細については、「Delegates and Data Sources」 in *Cocoa Fundamentals Guide* を参照してください。プロトコルの詳細については、「Protocols」 in *The Objective-C Programming Language* を参照してください。

ユーザにPersonレコードを選ばせる

`ABPeoplePickerNavigationController`クラスは、ユーザによる連絡先リストの参照と、人物、および必要に応じた人物の特定のプロパティの選択を可能にします。Peopleピッカーを使用するには、以下の手順を実行します。

1. このクラスのインスタンスを作成して初期化します。
2. デリゲートを設定します。このデリゲートは、`ABPeoplePickerNavigationControllerDelegate`プロトコルを採用しなければなりません。
3. 必要であれば、表示したいプロパティの配列に`displayedProperties`を設定します。関連する定数は整数として定義されています。これらの定数を`NSNumber`オブジェクトでラップし、`numberWithInt:`メソッドを使用して、配列に入れることのできるオブジェクトを取得します。
4. `presentModalViewController:animated:`メソッドを使用して、Peopleピッカーをモーダル View Controllerとして表示します。表示にはアニメーションを使用することをお勧めします。

以下のコードは、`ABPeoplePickerNavigationControllerDelegate`プロトコルを実装したView ControllerがPeopleピッカーを表示する方法を示しています。

```
ABPeoplePickerNavigationController *picker =
        [[ABPeoplePickerNavigationController alloc] init];
picker.peoplePickerDelegate = self;
[self presentModalViewController:picker animated:YES];
```

このPeopleピッカーは、ユーザのアクションに応じて、以下のデリゲートメソッドのいずれかを呼び出します。

- ユーザが操作をキャンセルした場合、Peopleピッカーはデリゲートの`peoplePickerNavigationControllerDidCancel:`メソッドを呼び出します。このメソッドは、Peopleピッカーを消去します。

- ユーザが人物を選択した場合、Peopleピッカーはデリゲートの`peoplePickerNavigationController:shouldContinueAfterSelectingPerson:`メソッドを呼び出し、Peopleピッカーが処理を続行するかどうかを決定します。選択中の人物の特定のプロパティをユーザに選択させるには、`YES`を返します。それ以外の場合は、`NO`を返して、ピッカーを消去します。
- ユーザがプロパティを選択した場合、Peopleピッカーはデリゲートの`peoplePickerNavigationController:shouldContinueAfterSelectingPerson:property:identifier:`メソッドを呼び出し、Peopleピッカーが処理を続行するかどうかを決定します。選択中のプロパティに対してデフォルトのアクション（電話番号に電話をかける、新規メールを起動するなど）を実行するには、`YES`を返します。それ以外の場合は、`NO`を返し、`dismissModalViewControllerAnimated:`メソッドを使用してピッカーを消去します。Peopleピッカーの消去には、アニメーションを使用することをお勧めします。

Personレコードの表示と編集

`ABPersonViewController`クラスは、ユーザにレコードを表示します。このコントローラを使用するには、以下の手順を実行します。

1. このクラスのインスタンスを作成して初期化します。
2. デリゲートを設定します。このデリゲートは、`ABPersonViewControllerDelegate`プロトコルを採用しなければなりません。ユーザにレコードの編集を許可するには、`allowsEditing`を`YES`に設定します。
3. `displayedPerson`プロパティを、表示したいPersonレコードに設定します。
4. 必要であれば、表示したいプロパティの配列に`displayedProperties`を設定します。関連する定数は整数として定義されています。これらの定数を`NSNumber`オブジェクトでラップし、`numberWithInt:`メソッドを使用して、配列に入れることのできるオブジェクトを取得します。
5. 現在のNavigation Controllerの`pushViewController:animated:`メソッドを使用して、このPerson View Controllerを表示します。表示にはアニメーションを使用することをお勧めします。

> **Important**　Person View Controllerが正しく機能するには、Navigation Controllerと組み合わせて使用する必要があります。

以下のコードは、Navigation ControllerがPerson View Controllerを表示する方法を示しています。

```
ABPersonViewController *view = [[ABPersonViewController alloc] init];


view.personViewDelegate = self;
```

```
view.displayedPerson = person; // personはすでに定義されている


[self.navigationController pushViewController:view animated:YES];
```

ユーザがビューのプロパティをタップすると、Person View Controllerは、デリゲートの`personViewController:shouldPerformDefaultActionForPerson:property:identifier:`メソッドを呼び出して、そのプロパティに対するデフォルトのアクションを実行するかどうかを決定します。選択中のプロパティに対してデフォルトのアクション（電話番号に電話をかける、新規メールを作成するなど）を実行するには、`YES`を返します。それ以外の場合は`NO`を返します。

ユーザに新規Personレコードを作成させる

`ABNewPersonViewController`クラスは、ユーザによる新しい人物の作成を可能にします。これを使用するには、以下の手順を実行します。

1. このクラスのインスタンスを作成して初期化します。
2. デリゲートを設定します。このデリゲートは、`ABNewPersonViewControllerDelegate`プロトコルを採用しなければなりません。フィールドを読み込むには、`displayedPerson`の値を設定します。特定のグループに新規の人物を登録するには、`parentGroup`を設定します。
3. 新規のNavigation Controllerを作成して初期化します。次に、そのルートView ControllerをNew Person View Controllerに設定します。
4. `presentModalViewController:animated:`メソッドを使用して、このNavigation Controllerをモーダルview Controllerとして表示します。表示にはアニメーションを使用することをお勧めします。

Important　New Person View Controllerが正しく機能するには、Navigation Controllerと組み合わせて使用する必要があります。New Person View Controllerはモーダルモードで表示することをお勧めします。

以下のコードは、Navigation ControllerがNew Person View Controllerを表示する方法を示しています。

```
ABNewPersonViewController *view = [[ABNewPersonViewController alloc] init];
view.newPersonViewDelegate = self;


UINavigationController *newNavigationController = [[UINavigationController alloc]
                                                   initWithRootViewController:view];
```

```
[self presentModalViewController:newNavigationController
                        animated:YES];
```

ユーザが「保存(Save)」ボタンまたは「キャンセル(Cancel)」ボタンをタップすると、New Person View Controllerはデリゲートの`newPersonViewController:didCompleteWithNewPerson:`メソッドを呼び出して、作成されたPersonレコードを渡します。ユーザがレコードを保存した場合は、まずこの新規レコードがAddress Bookに追加されます。ユーザがキャンセルした場合は、`person`の値が`NULL`になります。デリゲートは、Navigation Controllerの`dismissModalViewControllerAnimated:`メソッドを使用して、New Person View Controllerを消去します。消去にはアニメーションを使用することをお勧めします。

ユーザに既存のデータから新規Personレコードを作成させる

`ABUnknownPersonViewController`クラスを利用すると、ユーザに既存のPersonレコードにデータを追加させたり、そのデータ用から新規Personレコードを作成させることができます。これを使用するには、以下の手順を実行します。

1. このクラスのインスタンスを作成して初期化します。
2. 新規のPersonレコードを作成し、表示するプロパティを読み込みます。
3. 前の手順で作成した新規Personレコードに`displayedPerson`を設定します。
4. デリゲートを設定します。このデリゲートは、`ABUnknownPersonViewControllerDelegate`プロトコルを採用しなければなりません。
5. ユーザが、UnknownPerson View Controllerによって表示される情報を既存の連絡先に追加できるようにしたり、それを利用して新規の連絡先を作成できるようにするには、`allowsAddingToAddressBook`を`YES`に設定します。
6. このNavigation Controllerの`pushViewController:animated:`メソッドを使用して、Unknown Person View Controllerを表示します。表示にはアニメーションを使用することをお勧めします。

Important Unknown Person View Controllerが正しく機能するには、Navigation Controllerと組み合わせて使用する必要があります。

以下のコードは、Unknown Person View Controllerを表示する方法を示しています。

```
ABUnknownPersonViewController *view = [[ABUnknownPersonViewController alloc] init];
```

```
view.unknownPersonViewDelegate = self;
view.displayedPerson = person; // personはすでに定義されている
view.allowsAddingToAddressBook = YES;

[self.navigationController pushViewController:view animated:YES];
```

ユーザが新規連絡先の作成や、既存の連絡先へのプロパティの追加を終了すると、Unknown Person View Controllerは、デリゲートの`unknownPersonViewController:didResolveToPerson:`メソッドを呼び出して、作成されたPersonレコードを渡します。ユーザがキャンセルした場合は、`person`の値が`NULL`になります。

直接のやり取り：プログラムによるデータベースへのアクセス

Address Bookデータベースの一般的なタスクの多くは、ユーザとのやり取りに応じて実行されますが、アプリケーションがAddress Bookデータベースと直接やり取りする必要が生じる場合もあります。Address Bookフレームワークには、この機能を提供する関数がいくつかあります。

一貫したユーザ体験を提供するために、適切なときにのみ、これらの関数を使用することが重要です。プログラムでは、これらの関数を使用して新規にView ControllerやNavigation Controllerを作成するよりも、できる限り、あらかじめ用意されているView ControllerやNavigation Controllerを呼び出すべきです。詳細については、“直接のやり取り：データの選択と表示”（19ページ）を参照してください。

Address Bookデータベースは、結局のところユーザが所有していることを忘れないでください。したがって、アプリケーション側でAddress Bookデータベースに予期しない変更を加えないように注意する必要があります。一般的に、変更はユーザが行い、ユーザが確認するべきです。このことは、グループについて特に当てはまります。なぜなら、ユーザがグループを管理したり、アプリケーションによる変更を取り消したりするためのインターフェイスがデバイス上にはないからです。

レコード識別子の使用

Address Bookデータベースの各レコードには、一意のレコード識別子があります。この識別子は、レコードが削除されたり、MobileMeの同期データがリセットされたりしない限り、常に同じレコードを指します。レコード識別子は、スレッド間で安全に受け渡すことができます。デバイスをまたいで同じままであることは保証されません。

長期間、特定のレコードを保持し続けるために推奨される方法は、識別子のほかに姓名そのもの、または姓名のハッシュ値を保存することです。IDでレコードを検索するときに、そのレコードの名前と、保存されている名前を比較します。それらが一致しない場合は、保存されている名前を使用してレコードを検索し、そのレコード用に新しいIDを保存します。

レコードのレコード識別子を取得するには、`ABRecordGetRecordID`関数を使用します。識別子を使用してPersonレコードを検索するには、`ABAddressBookGetPersonWithRecordID`関数を使用します。識別子を使用してグループを検索するには、`ABAddressBookGetGroupWithRecordID`関数を使用します。名前を使用してPersonレコードを検索するには、`ABAddressBookCopyPeopleWithName`関数を使用します。

Personレコードを対象とした操作

ABAddressBookAddRecord関数とABAddressBookRemoveRecord関数を使用して、Address Bookデータベースにレコードを追加したり、データベースから削除できます。

Address BookデータベースからPersonレコードを検索するには、2つの方法があります。ABAddressBookCopyPeopleWithName関数を使用して名前で検索する方法と、ABAddressBookGetPersonWithRecordID関数を使用してレコード識別子で検索する方法です。他の種類の検索に対応するには、ABAddressBookCopyArrayOfAllPeople関数を使用してから、NSArrayのfilteredArrayUsingPredicate:メソッドで検索結果をフィルタリングします。

Peopleレコードの配列を並べ替えるには、CFArraySortValues関数を使用し、比較関数にはABPersonComparePeopleByName関数を指定し、コンテキストタイプにはABPersonSortOrderingを指定します。ABPersonGetSortOrderingから返されるユーザの望む並べ替え順が、一般的には望ましいコンテキストです。

以下のコードは、Address Bookデータベース全体を並べ替える例を示しています。

```
ABAddressBookRef addressBook = ABAddressBookCreate();
CFArrayRef people = ABAddressBookCopyArrayOfAllPeople(addressBook);
CFMutableArrayRef peopleMutable = CFArrayCreateMutableCopy(
                                        kCFAllocatorDefault,
                                        CFArrayGetCount(people),
                                        people
                                    );



CFArraySortValues(
        peopleMutable,
        CFRangeMake(0, CFArrayGetCount(peopleMutable)),
        (CFComparatorFunction) ABPersonComparePeopleByName,
        (void*) ABPersonGetSortOrdering()
);

CFRelease(addressBook);
CFRelease(people);
CFRelease(peopleMutable);
```

Groupレコードを対象とした操作

レコード識別子に基づいて特定のグループを検索するには、`ABAddressBookGetGroupWithRecordID`関数を使用します。また、`ABAddressBookCopyArrayOfAllGroups`を使用して、アドレスブック内のすべてのグループの配列を取り出したり、`ABAddressBookGetGroupCount`関数を使用して、アドレスブック内のグループ数を取得したりすることができます。

グループのメンバはプログラムの中から変更できます。グループに人物を追加するには、`ABGroupAddMember`関数を使用します。グループから人物を削除するには、`ABGroupRemoveMember`関数を使用します。Personレコードをグループに追加する前に、そのレコードがAddress Bookデータベースに保存済みでなければなりません。新規のPersonレコードをグループとデータベースに同時に追加する必要がある場合は、まずPersonレコードをAddress Bookデータベースに追加してデータベースを保存してから、そのレコードをグループに追加しなければなりません。

書類の改訂履歴

この表は「*iOS Address Book プログラミングガイド*」の改訂履歴です。

日付	メモ
2012-02-16	ARCおよびストーリーボードを使うよう、内容を更新しました。
2010-12-22	コードリストの細かな誤りを訂正しました。
2010-11-15	コードリストの細かな誤りを訂正しました。全体を通して、その他の細かな変更を行いました。
2010-07-08	『iPhone OS Address Bookプログラミングガイド』からドキュメント名を変更しました。
2010-03-24	「UI Controllerを使用したやり取り」のサンプルコードを追加しました。
2009-10-05	コードリストの細かな変更を行いました。
2009-05-27	ABMultiValueCopyLabelAtIndexの戻り値についての説明を追加しました。AddressBookUI View Controllerの使用方法についての注記を訂正しました。
2009-05-06	レコード識別子についての説明で細かな訂正を行いました。表現上の細かな変更を行って、全体的にわかりやすくしました。
2009-02-04	細かな再編を行って、読みやすくしました。

日付	メモ
2008-10-15	住所の処理を行うサンプルコードを追加しました。全体を通して、その他の細かな変更を行いました。
2008-09-09	iOS 2.1用に細かな更新を行いました。
2008-07-31	表現上の細かな変更を行いました。誤字を訂正しました。「Address Bookオブジェクトの操作」の内容を再編しました。
2008-07-08	サンプルコードを更新しました。全体を通して、細かい編集上の変更と構成上の変更を行いました。
2008-06-06	Address Bookのレコードを処理するための方法、およびビューを使用して連絡先情報の表示や入力を求める方法について説明した新規ドキュメント。

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/