iPhone OS Address Bookプログラミングガイド

iPhone

2008-07-08

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
U.S.A.

アップルジャパン株式会社
〒163-1450 東京都新宿区西新宿
3 丁目20 番2 号
東京オペラシティタワー
http://www.apple.com/jp/

目次

図

はじめに

iPhone OSのAddress Bookテクノロジーは、連絡先情報、および人々に関するその他の個人情報を集中データベース内に格納し、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。
- Contacts（連絡先）アプリケーションは、連絡先リストを操作する手段をユーザに提供します。

この文書では、Address Bookテクノロジーの重要ないくつかの考え方を取り上げ、実行が可能な基本的な操作を説明します。このテクノロジーを使用するすべてのアプリケーションが情報を共有します。アプリケーションにおいてこのテクノロジーを使用することで、「メール(Mail)」や「SMS (Text)」などのほかのアプリケーションと同じ連絡先情報にアクセスすることができます。この文書では、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
- 連絡先情報の入力をユーザに求める方法
- 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

XcodeおよびiPhoneのCocoa開発の基礎をすでによく理解していることを前提とします。この文書を最大限に利用するには、Navigation Controller、View ControllerおよびiPhoneユーザインターフェイスの基礎も理解する必要があります。

Mac OS上のAddress Bookテクノロジーに精通しているデベロッパのための備考：Address Bookテクノロジーは、両方のプラットホーム上で同じデータを対象としますが、APIが異なります。Mac OSでこのテクノロジーを使用したことがあれば、Address Bookデータベースとのやり取りの方法はすでに理解しているかもしれませんが、iPhone OSにおける具体的な関数呼び出しは、知っているものとは異なるでしょう。

この書類の構成

この文書は、次の各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」（9 ページ）では、ユーザに連絡先を選択するように求め、その連絡先に関する情報を表示する簡単なアプリケーションを作成する方法を、最初の一歩として示します。

- 「Address Bookオブジェクトの操作」（15 ページ）では、Address Bookオブジェクトのインスタンスの作成方法、PersonオブジェクトとGroupオブジェクトの作成方法、およびそれらのプロパティの取得方法と設定方法を説明します。
- 「UI Controllerを使用したやり取り」（21 ページ）では、Address Book UIフレームワークのピッカーを使用した連絡先の表示方法、連絡先の選択方法、新しい連絡先の作成方法、および連絡先の編集方法を説明します。
- 「レコードとの直接のやり取り」（27 ページ）では、アプリケーションがPersonレコードおよびGroupレコードと直接やり取りする方法を説明します。

関連項目

- *Address Book Framework Reference*
- *Address Book UI Framework Reference*

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、選択された人物の姓名を表示する簡単なアプリケーションを作成します。

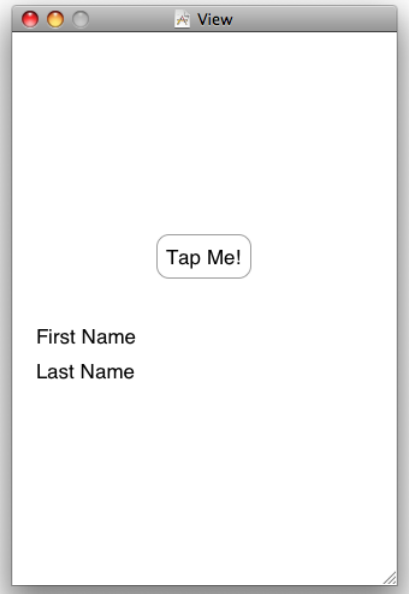
プロジェクトの作成

Xcodeで、View Based Applicationテンプレートから新規プロジェクトを作成します。プロジェクトをQuickStartという名前で保存します。次のステップで、必要なフレームワークを追加します。「ターゲット(Targets)」グループのプロジェクトウインドウには、`QuickStart`という名前のターゲットがあります。その情報パネル（「ファイル(File)」＞「情報を見る(Get Info)」）を開くと、「一般(General)」タブに、リンクされたライブラリがあります。プラス記号の付いたボタンをクリックして、リストの中からAddress Bookフレームワークおよび Address Book UIフレームワークを選択して、それらを追加します。

Viewのレイアウト

最初に、プロジェクトのインターフェイスをレイアウトします。プロジェクトウインドウの「Resources」フォルダ内で、`QuickStartViewController.xib`という名前のInterface Builderファイルを開きます。ライブラリからボタンとテキストラベルをドラッグして、Viewに1つのボタンと2つのテキストラベルを追加します。その後、次の図に示すようにそれらを配置します。

図 1-1 Interface BuilderでのViewのレイアウト



結び付きを設定する前に、ヘッダファイル内でInterface Builderのアウトレットを宣言する必要があります。最後のステップで戻ってきて、これらの結び付けを行います。ここでは、いったんファイルを保存してXcodeに戻ります。

ヘッダファイルの記述

View Controllerのヘッダファイル`QuickStartViewController.h`に次のコードを追加します。このコードは、Interface Builderで作成したばかりのボタンのラベルおよびアクションのアウトレットを宣言します。また、このView Controllerが`ABPeoplePickerNavigationControllerDelegate`プロトコルを採用することも宣言します。これについては、次で実装します。

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


@interface QuickStartViewController :UIViewController
<ABPeoplePickerNavigationControllerDelegate> {
    IBOutlet UILabel *firstName;
    IBOutlet UILabel *lastName;
}

@property (nonatomic, retain) UILabel *firstName;
@property (nonatomic, retain) UILabel *lastName;

- (IBAction)showPicker:(id)sender;

@end
```

アプリケーションデリゲートのヘッダファイルは、テンプレートによって作成されたままの状態で使用できます。

実装の記述ファイル

次に、View Controllerの実装ファイルQuickStartViewController.mに次のコードを追加します。このコードは、firstNameおよびlastNameのアクセサメソッドを合成し、showPicker:メソッドを実装します。このメソッドは、ユーザが「Tap Me!」ボタンをタップしたときに呼び出されるようにします。このメソッドは、新しいPeopleピッカーを作成し、View Controllerをそのデリゲート（delegate、委任）として設定し、作成したPeopleピッカーをModal View Controllerとして表示します。

```
#import "QuickStartViewController.h"

@implementation QuickStartViewController

@synthesize firstName;
@synthesize lastName;


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

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

同じファイルにコードの追加を続けながら、この次のセクションではデリゲートプロトコルの実装を開始します。最初の関数は、ユーザがPeopleピッカーを閉じるためにキャンセルをしたときに呼び出されます。2番目の関数は、ユーザが人物を選択すると呼び出されます。この関数は、その人物の名前と姓をラベルにコピーし、それからPeopleピッカーを閉じます。

この例では、ABRecordCopyValue関数が最も一般的な関数であるため、名前の取得にこの関数を使用しています。これは、PersonレコードまたはGroupレコードからプロパティを取得するためにも使用できます。しかし、指定された人物のフルネームを取得して表示する場合は、ABRecordCopyCompositeName関数の使用をお勧めします。これは、名前と姓をユーザの指定する順に表示します。

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


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

    NSString* name = (NSString *)ABRecordCopyValue(person,
                                        kABPersonFirstNameProperty);
    self.firstName.text = name;
    [name release];
```

```
    name = (NSString *)ABRecordCopyValue(person, kABPersonLastNameProperty);
    self.lastName.text = name;
    [name release];

    [self dismissModalViewControllerAnimated:YES];

    return NO;
}
```

デリゲートプロトコルを完全に実装するには、次の関数も追加する必要があります。この関数は、ユーザが、ピッカー内のある人物の特定のプロパティを選択すると呼び出されます。このアプリケーションでは、Peopleピッカーは、通常、人物が選択されるとすぐに閉じられます。そのため、ユーザはプロパティを選択することができません。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。

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

テンプレートからは、このほかに2つの関数があります。この2つの関数はそのまま使用できます。

```
- (BOOL)shouldAutorotateToInterfaceOrientation:
           (UIInterfaceOrientation)interfaceOrientation {
    // サポートされている方向に対してYESを返す
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}


- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning]; // スーパービューがなければ
                                     // ビューを解放する

    // キャッシュデータなどの重要でないものをすべて解放する
}
```

最後に、メモリリークをしないようにすべてを解放します。

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

@end
```

アプリケーションデリゲートの実装ファイルは、テンプレートによって作成されたままの状態で使用できます。

Interface Builderでの結び付け

Interface Builderに戻り、「Viewのレイアウト」（9 ページ）で作成を開始したファイルを完成させます。必要な結び付きがいくつかあります。まず、File's Ownerのクラス識別情報は、「Inspector」の「Identity」で`QuickStartViewController`に設定されていることに注目してください。これは、プロジェクトテンプレートによって設定されました。File's Ownerの`firstName`および`lastName`のアウトレットをラベルに結び付けます。次に、ボタンの「Touch Up Inside」アウトレットをFile's Ownerに結び付け、`showPicker`メソッドを選択します。これで、アプリケーションのビルドおよび実行の準備が整いました。

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、最初に表示されるビューには、1つのボタンと2つの空のテキストラベルがあります。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ラベルに表示されます。

これは、概念的にかなり簡単な作業であり、2つのAddress Bookフレームワークを使用して、極めて簡単にできます。Peopleピッカーを表示し、`ABPeoplePickerNavigationControllerDelegate`プロトコルを採用するためには、テンプレートにコードを追加する必要がありました。ユーザがPeopleピッカーを使用して選ぶ連絡先情報を、アプリケーションで有意義に活用する方法を想像するのは難しいことではないでしょう。

Address Bookオブジェクトの操作

Address Bookデータベースと本格的にやり取りをするために理解しておく必要のある4つの基本的なオブジェクトがあります。

アドレスブック	Address Bookデータベースとやり取りをし、Address Bookデータベースへの変更を保存できます。レコードで構成されます。
レコード	人物またはグループを表します。プロパティで構成されます。
単一値プロパティ	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	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。値で構成されます。

Address Book

典型的なワークフローには、アドレスブックの読み取り、変更、Address Bookデータベースへの変更の保存があります。アドレスブックを使用するには、`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。

重要： `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。

アドレスブックが用意できたら、アプリケーションは、そのアドレスブックのデータを読み取ったり、アドレスブックに変更を保存したりできます。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。

以下は、Address Bookデータベースへの変更および変更の保存のための一般的なコードパターンを示すサンプルコードです。

```
ABAddressBookRef addressBook;
CFErrorRef error = NULL;
BOOL wantToSaveChanges = YES;

addressBook = ABAddressBookCreate();

/* ... Address Bookを対象とした操作 ... */

if (ABAddressBookHasUnsavedChanges(addressBook)) {
    if (wantToSaveChanges) {
        ABAddressBookSave(addressBook, &error);
    } else {
        ABAddressBookRevert(addressBook);
```

```
    }
}

/*... エラーが発生したら処理する ...*/

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。`ABAddressBookRegisterExternalChangeCallback`関数を使用して、`ABExternalChangeCallback`プロトタイプの関数を登録します。`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、コードでは単純にアドレスブック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、変更が統合できずに保存が失敗した場合に、適切なアクションを取れるよう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、人物とグループは`ABRecordRef`オブジェクトとして格納されます。`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フレームワークをよく理解しているデベロッパであれば、これらが`ABPersonRef`オブジェクトまたは`ABGroupRef`オブジェクトでないことに気付いたのではないでしょうか。Personオブジェクトと Groupオブジェクトはどちらも、`ABRecordRef`のインスタンスです。

重要： 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」（27 ページ）を参照してください。

レコードは、通常、アドレスブックの一部を構成しますが、独立に存在することもできます。これは、アプリケーションが操作の対象とする連絡先情報を格納する便利な方法になります。

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。`ABRecordSetValue`関数および`ABRecordCopyValue`関数は、それぞれプロパティの設定および取得を行います。`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レコードには、単一値および多値の2種類のプロパティがあります。姓、名、ニックネームなど、人物が1つしか持つことができないプロパティは、単一値プロパティです。住所、電話番号、および電子メールアドレスなどのほかのプロパティは、多値プロパティです。Personレコードのプロパティは、『*ABPerson Reference*』の「Constants」の複数のセクションで示しています。

Personレコードとの直接のやり取りに関連する関数の詳細については、「レコード識別子の使用」（27 ページ）を参照してください。

単一値プロパティ

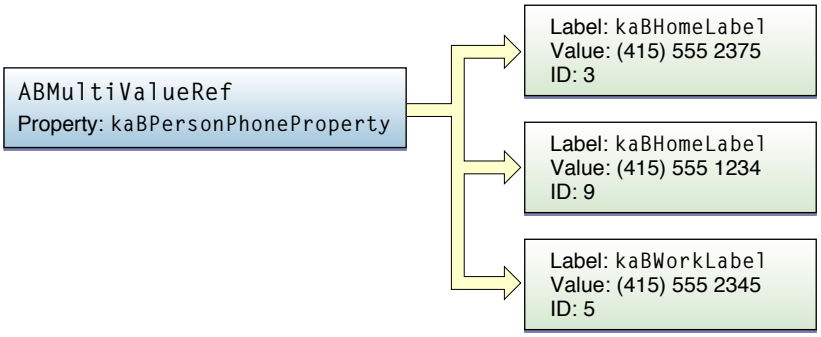
以下は、単一値プロパティの取得および設定の方法を示すサンプルコードです。

```
ABRecordRef aRecord = ABPersonCreate();
CFErrorRef  anError = NULL;
CFStringRef firstName, lastName;

ABRecordSetValue(aRecord, kABPersonFirstNameProperty,
                 CFSTR("Katie"), &anError);
ABRecordSetValue(aRecord, kABPersonLastNameProperty,
                 CFSTR("Bell"), &anError);
if (anError != NULL) { /* ... エラー処理 ... */}

/* ... */

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

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。それぞれの値は、テキストラベルと、それに対応する識別子を持ちます。たとえば、以下の図は電話番号のプロパティを示します。図にあるように、一人の人物が複数の電話番号を持つ場合があります。それぞれの電話番号には、家や仕事などのテキストラベル、および識別子があります。複数の値が同じラベルの場合もあり得ますが、識別子は常に一意です。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。『*ABPerson Reference*』の「Generic Property Labels」を参照してください。

図 2-1 多値プロパティ



次の関数により、多値プロパティの内容を読み取ることができます。

- `ABMultiValueCopyLabelAtIndex`および`ABMultiValueCopyLabelAtIndex`は、個々の値をコピーします。
- `ABMultiValueCopyArrayOfAllValues`は、すべての値をCFArrayにコピーします。
- `ABMultiValueGetIndexForIdentifier`および`ABMultiValueGetIdentifierAtIndex`は、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取っておくには、その識別子を保存します。一般的に、インデックスの保存は避けるべきです。これは、値が追加されたり削除されたりするとインデックスが変わるからです。識別子は、デバイスが変わる場合を除いては、同じままであることが保証されています。

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、新しい可変の多値オブジェクトを作成することもできます。

次の関数により、可変の多値オブジェクトに変更を加えることができます。

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

以下は、多値プロパティの取得および設定方法を示すサンプルコードです。

```
ABMutableMultiValueRef multi =
        ABMultiValueCreateMutable(kABMultiStringPropertyType);
CFErrorRef anError = NULL;

// この例では、新しい値の多値識別子は必要ないため、
// “返される識別子”にはNULL値が入る
```

```
bool didAdd = ABMultiValueAddValueAndLabel(multi, @"(555) 555-1234",
                        kABPersonPhoneMobileLabel, NULL)
           && ABMultiValueAddValueAndLabel(multi, @"(555) 555-2345",
                        kABPersonPhoneMainLabel, NULL);
if (didAdd != YES) { /* ... エラー処理 ... */}

ABRecordRef aRecord = ABPersonCreate();
ABRecordSetValue(aRecord, kABPersonPhoneProperty, multi, &anError);
if (anError != NULL) { /* ... エラー処理 ... */}
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を使ってなにかする ... */

    CFRelease(phoneNumberLabel);
    CFRelease(phoneNumber);
}

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

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。アプリケーションではグループを利用して、ユーザがアドレスブックの複数の人々を対象に同時に特定のアクションを実行するようにできます。

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。グループ内のすべての人々を取得するには、`ABGroupCopyArrayOfAllMembersWithSortOrdering`関数または`ABGroupCopyArrayOfAllMembers`関数を使用します。これらは、`ABRecordRef`オブジェクトの`CFArray`を、それぞれ並べ替えありと並べ替えなしで返します。

Groupレコード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」（28ページ）を参照してください。

UI Controllerを使用したやり取り

Address Book UIフレームワークは、アドレスブックの使用に関連する一般的なタスクのためのView ControllerおよびNavigation Controllerを提供します。独自にコントローラを作成するよりもこれらのコントローラを使用することで、必要な作業を減らし、使用しない場合よりも一貫性のある体験をユーザに提供できます。この章では、提供されているコントローラとそれらの使用方法について学びます。

Address Book UIフレームワークが提供するコントローラは4つあります。

`ABPeoplePicker-NavigationController`	ユーザに、アドレスブックから人物を選択するよう求めます。
`ABPersonViewController`	ユーザに人物を提示し、必要に応じて編集できるようにします。
`ABNewPersonViewController`	ユーザに新しい人物を作成するよう求めます。
`ABUnknownPerson-ViewController`	ユーザに特定の人物の未完成のレコードを完成するように求め、必要に応じてそのレコードをアドレスブックに追加できるようにします。

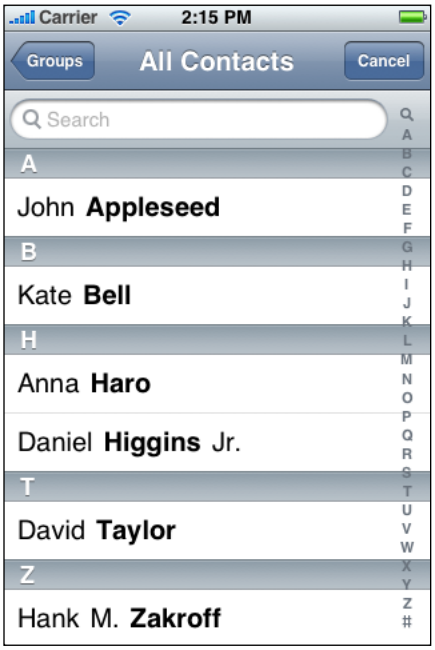
各コントローラには、対応するデリゲートプロトコルがあります。これらのプロトコルは、コントローラを閉じ、その結果を使用するアプリケーション部分に結果を渡す役割を担います。このデリゲートがコントローラを閉じることができなかった場合、ユーザはアプリケーションを終了する以外にコントローラを閉じる方法がありません。

注： これらのコントローラをサブクラス化する必要はありません。コントローラの動作を変更する方法として想定されているのは、コントローラのデリゲートを実装することです。

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。このクラスを使用するには、まず、クラスのインスタンスを作成し初期化します。次に、デリゲートを設定します。デリゲートは、`ABPeoplePickerNavigationControllerDelegate`プロトコルを採用する必要があります。それから、現在のView Controllerの`presentModalViewController:animated:`メソッドを使用して、Modal View ControllerとしてPeopleピッカーを提示します。これにより、PeopleピッカーのNavigation Controllerが一時的に現在のView Controllerを覆います。

図 3-1 PeopleピッカーのNavigation Controller



ユーザがキャンセルすると、Peopleピッカーを閉じるためにデリゲートが呼び出されます。ユーザがある人物を選択すると、Peopleピッカーを閉じるために、またはPeopleピッカーに続行することを伝えるためにデリゲートが呼び出されます。選択した人物の特定のプロパティをユーザに選択してほしい場合には、続行を指示します。それ以外の場合は、続行せずにピッカーを閉じるように指示します。ユーザがプロパティを選択すると、デリゲートが再び呼び出されます。選択されたプロパティを対象にデフォルトのアクション（電話番号に電話をかける、新規メールを開始するなど）を実行するようにしたい場合は、再びPeopleピッカーに続行するよう指示します。それ以外の場合は、続行せずピッカーを閉じるよう指示します。

「クイックスタートチュートリアル」（9 ページ）で記述したアプリケーションは、このコントローラを使用しました。

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。`allowsEditing`を`YES`に設定すると、ユーザはレコードを編集することもできます。表示するプロパティは、`displayedProperties`プロパティを`ABPropertyType`値の適切な配列に設定することで変更できます。このコントローラを使用するには、クラスのインスタンスを作成および初期化し、その`displayedPerson`プロパティとデリゲートを設定します。デリゲートは、`ABPersonViewControllerDelegate`プロトコルを採用する必要があります。その後、View controllerを現在のNavigation Controllerのスタックにプッシュします。

重要： 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。

図 3-2 Person View Controller－編集が可能な状態での表示

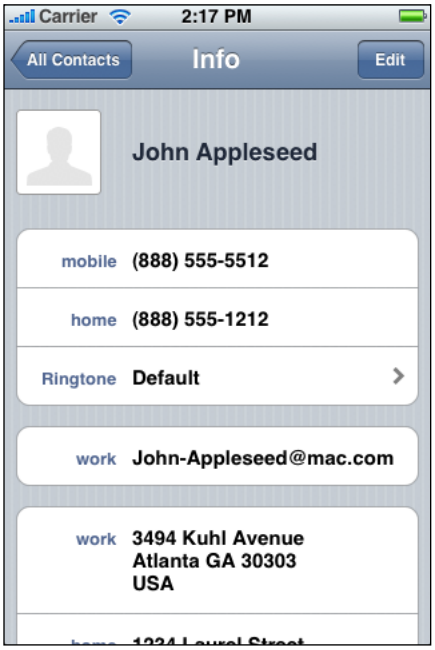
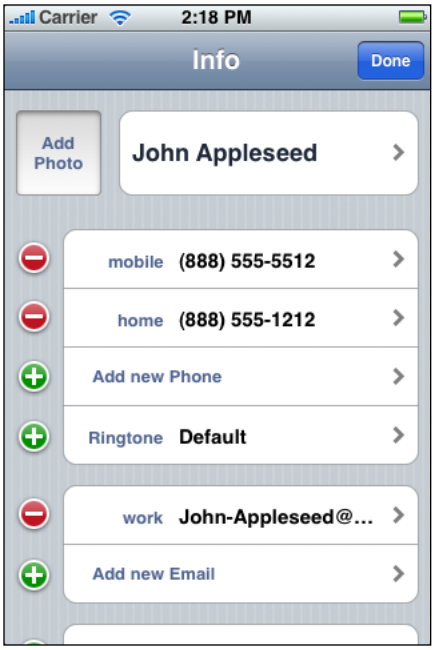


図 3-3 Person View Controller－編集



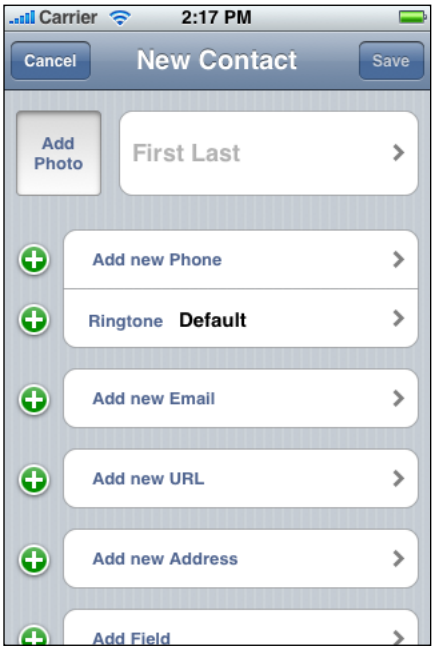
ユーザがビューのプロパティをタップすると、プロパティを対象としたデフォルトのアクションを決定するためにデリゲートが呼び出されます。

新規Personレコードの作成

`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。これは、Person View Controllerのように使用します。クラスの新しいインスタンスを作成および初期化します。プロパティを埋めるために`displayedPerson`の値を設定したり、人をグループに入れるために`parentGroup`の値を設定したりできます。これらはどちらも任意です。その後、View controllerを現在のNavigation Controllerのスタックにプッシュします。

重要： 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。

図 3-4 New Person View Controller



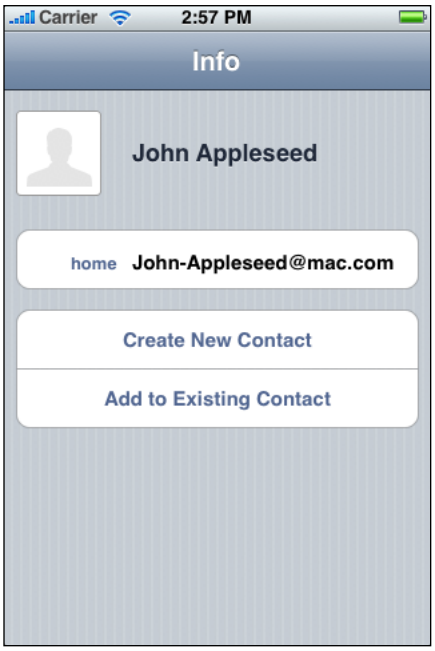
ユーザが保存をすると、デリゲートは自動的に新しいレコードをアドレスブックに追加します。ユーザが保存またはキャンセルした後には、デリゲートが呼び出され、結果のPersonレコードが渡されます。デリゲートは、`ABNewPersonViewControllerDelegate`プロトコルを採用する必要があります。

既存のデータからの新規Personレコードの作成

場合によっては、ユーザが、最近の発着信履歴にある友人の電話番号など、既存のPersonレコードに追加したい情報や、新しいPersonレコードの作成に使用したい情報を持っています。このような状況では、`ABUnknownPersonViewController`クラスを使用します。

重要： 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。

図 3-5 Unknown Person View Controller



これを使用するには、新しいPersonレコードを作成し、すでに分かっているプロパティを設定します。それから、Unknown Person View Controllerを作成し、作成した新しいPersonに`displayedPerson`を設定し、Unknown Person View ControllerをNavigation Controllerのスタックにプッシュします。`allowsAddingToAddressBook`を`YES`に設定すると、ユーザは、連絡先を追加したり、既存の連絡先に新しい情報を組み込んだりできます。

新しい連絡先の作成後、または既存の連絡先へのプロパティの追加後、デリゲートが呼び出され、結果のPersonレコードが渡されます。デリゲートは、`ABUnknownPersonViewControllerDelegate`プロトコルを採用する必要があります。

レコードとの直接のやり取り

アドレスブックに関わる多くの一般的なタスクは、ユーザとのやり取りをともないますが、ときには、プログラムがデータと直接やり取りすべき場合もあります。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。プログラムにおいては、これらの関数を使用してカスタムのView ControllerまたはNavigation Controllerを作成するよりも、可能な限り、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（「UI Controllerを使用したやり取り」（21 ページ）を参照してください）。

Address Bookデータベースは、最終的にはユーザが所有する点に留意することが重要です。そのため、アプリケーションが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データベースのすべてのレコードに、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ない限り、常に同じレコードを指します。このため、これはアプリケーションのデータにおいて人物またはグループへの参照を維持するのに適した方法です。またこれらは、スレッド間で安全に受け渡すこともできます。ただし、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。

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。

Personレコードを対象とした操作

Personレコードに関する基本的な情報、およびPersonレコードにどのようにデータが格納されるかについては、「Personレコード」（17 ページ）を参照してください。

`ABAddressBookAddRecord`関数および`ABAddressBookRemoveRecord`関数を使用して、アドレスブックからレコードを追加したり削除したりできます。アドレスブックの中で人物を検索するには、2つの方法があります。`ABAddressBookCopyPeopleWithName`関数を使用して名前に基づいて検索する方法と`ABAddressBookGetPersonWithRecordID`関数を使用してレコード識別子に基づいて検索する方法です。

Peopleの配列を並べ替えるには、CFArraySortValues関数を使用し、ABPersonComparePeopleByName関数を比較処理として、またABPersonSortOrderingをコンテキストタイプとして指定します。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、並べ替えの例を示します。

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

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

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

配列を並べ替えるためにCFArraySortValuesを呼び出す行は、Objective-C言語で記述することもできます。

```
[allPeopleMutable sortUsingFunction:ABPersonComparePeopleByName
                            context:(void*)sortOrdering];
```

Groupレコードを対象とした操作

Groupレコードに関する基本的な情報、およびGroupレコードにどのようにデータが格納されるかについては、「Groupレコード」（19 ページ）を参照してください。

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、グループから人物を削除するには、ABGroupRemoveMember関数を使用します。

書類の改訂履歴

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

日付	メモ
2008-07-08	サンプルコードを更新し、全体を通して、細かい編集上の変更と構成上の変更を加えました。
2008-06-06	Address Bookのレコードを対象とした作業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を説明する新規文書。

書類の改訂履歴