MODULE Deep Dive into iOS App Development
SESSION [5]
SESSION TITLE: [Understanding the Address Book]

Sources:

1. [Professional iOS Programming] [Chapter 12][ISBN No 9781118661130]
2. [Wiley Book Name] [Chapter Number][ISBN No]
3. [Wiley Book Name] [Chapter Number][ISBN No]
4. [Wiley Book Name] [Chapter Number][ISBN No]

Example: [Cloud Computing for Dummies] [Chapter 5] [978-0-470-48470-8]

MODULE Objective

At the end of this module, you will be able to:

- Describe the Address Book Framework

Session Objectives

At the end of this session, you will be able to:

- Explain the Address Book
- Access the Address Book programmatically

<H1> Introduction

The built-in Contacts application in your iOS device contains the list of contacts you have saved on your device. The Address Book is the way you can read or modify a user's contacts from your apps (the same contacts that show up in the Contacts app).

This session describes the Address Book technology. You will learn how to access the Address Book programmatically.

Exit from Week 9 Session 4: Sending Emails and SMS, and Dialing a Phone

- Send email from your application
- Send SMS or text messages from your application
- Dial phone numbers from your application

Entry to Week 9 Session 5: Understanding the Address Book

- Explain the Address Book
- Access the Address Book programmatically

<H1> [Introduction to the Address Book Framework]

Source: [Professional iOS Programming] [Chapter 12] [363]

The Address Book technology provides a way to store contacts and other personal information in a centralized database in an iOS device. You can share this information between applications such as mail and text by using the Address Book framework.

The Address Book technology consists of the following parts:

➢ **Address Book Framework**: This framework provides access to the centralized database with contact information.

➢ **Address Book UI Framework**: This framework provides user interface elements to present the information.

➢ **Address Book Database:** This database is used for storing the contact information on the iOS device.

➢ **Contacts Application**: This is a complete iOS application delivered by Apple to access the contact information.

In this session, a **contact** from the Address Book is also referred to as a **person**.

Quick Tip

You can find the complete Address Book framework reference at *http://developer.apple.com/library/ios/#documentation/AddressBook/Reference/AddressBook_iPhoneOS_Framework/_index.html*.

You can find the Address Book programming guide for iOS at *https://developer.apple.com/library/ios/documentation/ContactData/Conceptual/AddressBookProgrammingGuideforiPhone/Introduction.html.*

<H1> [Accessing the Address Book]

--

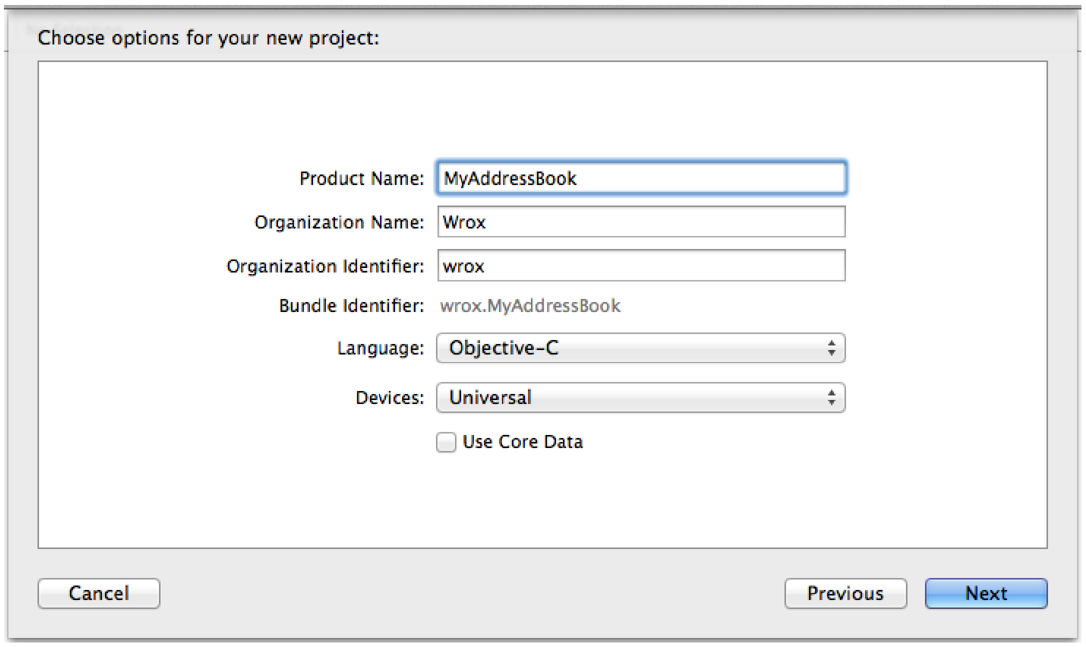
Source: [Professional iOS Programming] [Chapter 12] [364]

In this section, you will create an application that shows the main functionalities required to access the Address Book. You will learn how to:

- Select a contact
- Request access permission
- Display and edit a contact
- Create a contact
- Delete a contact

<H2> [Selecting a Contact]

Start Xcode and create a new project using the **Single View Application** template, and name it **MyAddressBook** using the options as shown in **Figure 1**.

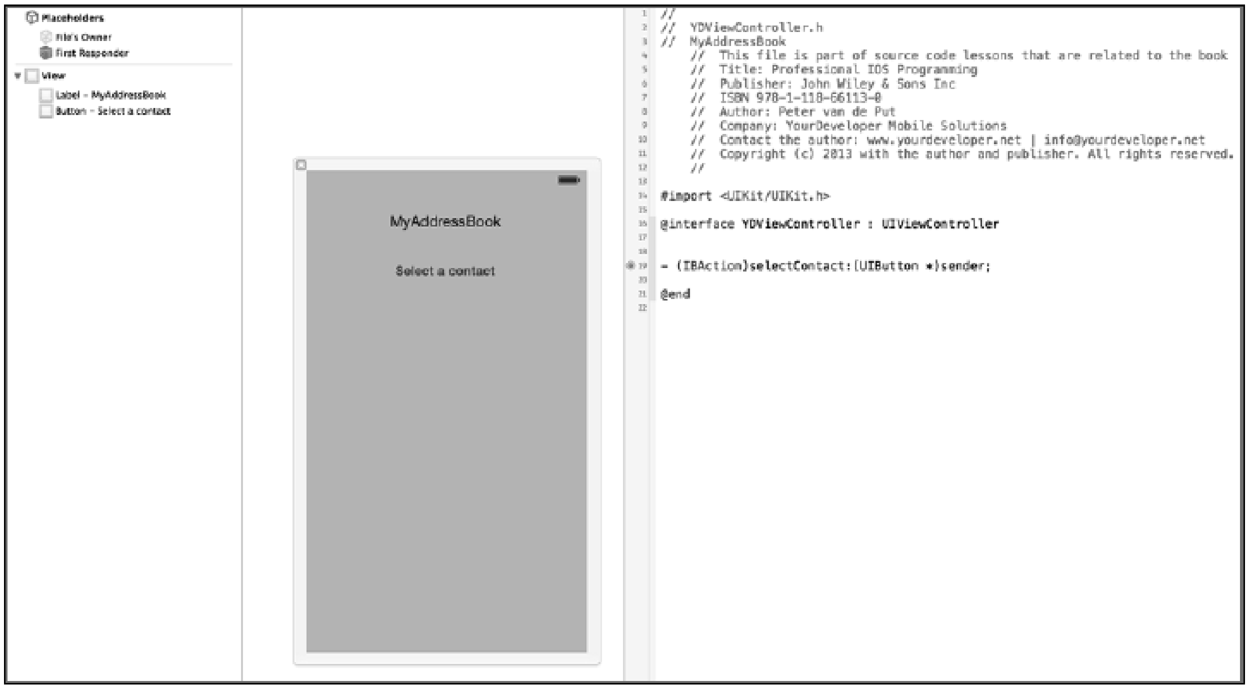


Choose options for your new project:

Product Name:	MyAddressBook
Organization Name:	Wrox
Organization Identifier:	wrox
Bundle Identifier:	wrox.MyAddressBook
Language:	Objective-C
Devices:	Universal
☐ Use Core Data	

Figure 1: Create a New Project, MyAddressBook, in Xcode

To access the Address Book, add the `AddressBook` and the `AddressBookUI` frameworks to your project. Use Interface Builder and the Assistant Editor to create a user interface for the `YDViewController.xib` file, as shown in **Figure 2**.



```
//
//  YDViewController.h
//  MyAddressBook
    //  This file is part of source code lessons that are related to the book
    //  Title: Professional IOS Programming
    //  Publisher: John Wiley & Sons Inc
    //  ISBN 978-1-118-66113-0
    //  Author: Peter van de Put
    //  Company: YourDeveloper Mobile Solutions
    //  Contact the author: www.yourdeveloper.net | info@yourdeveloper.net
    //  Copyright (c) 2013 with the author and publisher. All rights reserved.
    //

#import <UIKit/UIKit.h>

@interface YDViewController : UIViewController


- (IBAction)selectContact:(UIButton *)sender;

@end
```

Figure 2: Create a User Interface for the MyAddressBook Project

Open the `YDViewController.m` file in Objective-C, and if you are working in **Swift**, you can implement this in `ViewController.`**`Swift`**. In both languages, you need to import the `AddressBookUI` header file.

Next, subscribe to the `ABPeoplePickerNavigationControllerDelegate` protocol. The `ABPeoplePickerNavigationControllerDelegate` protocol is deprecated in iOS 8.

Now in the `selectContact:` method in Objective-C, create and initialize an instance of the `ABPeopleNavigationController` class named `peoplePicker`. Because you have subscribed to the `ABPeopleNavigationControllerDelegate` protocol, set the `peoplePickerDelegate` property to `self`. Optionally, you can use the `displayedProperties` property of the `ABPeoplePickerNavigationController` class to pass an `NSArray` with properties you want to display. These properties are wrapped as `NSNumber` objects, which are created from the constants whose names start with the `kABPerson` integer value.

All the available constants are defined in the `ABPerson.h` file. Some more APIs, which are added to `ABPerson.h` in iOS 8, include:

- `KABPersonAlternateBirthdayCalenderIdentifierKey`
- `KABPersonAlternateBirthdayDayKey`
- `KABPersonAlternateBirthdayEraKey`
- `KABPersonAlternateBirthdayIsLeapMonthKey`
- `KABPersonAlternateBirthdayMonthKey`
- `KABPersonAlternateBirthdayProperty`
- `KABPersonAlternateBirthdayYearKey`

The presented `peoplePicker` calls one of its delegate methods depending on the user's action:

- If the user taps Cancel, it calls the `peoplePickerNavigationControllerDidCancel:` method. In your implementation, you should dismiss the `peoplePicker`.

- If the user selects a person, it calls the `peoplePickerNavigationController:shouldContinueAfterSelectingPerson:` method to determine if the `peoplePicker` should continue. If you want to dismiss the `peoplePicker` and continue your application's logic with the selected person, you should return NO after dismissing the `peoplePicker`. However, if you want to display the selected person, you should return YES.

- If the user selects a property on the contact detail screen, it calls the `peoplePickerNavigationController:shouldContinueAfterSelectingPerson:property:identifier:` method. If you want to perform the default action, return YES; otherwise return NO.

The `peoplePickerNavigationController:shouldContinueAfterSelectingPerson:property:identifier:` method and the `peoplePickerNavigationController:shouldContinueAfterSelectingPerson:` method are deprecated in iOS 8.

When you run the application in the simulator and tap the Select a Contact button, the `ABPeoplePickerNavigationController` is presented, as shown in **Figure 3**.

If the `peoplePickerNavigationController:shouldContinueAfterSelectingPerson:` method returns YES, the details of the selected contact are shown, as in **Figure 4**.

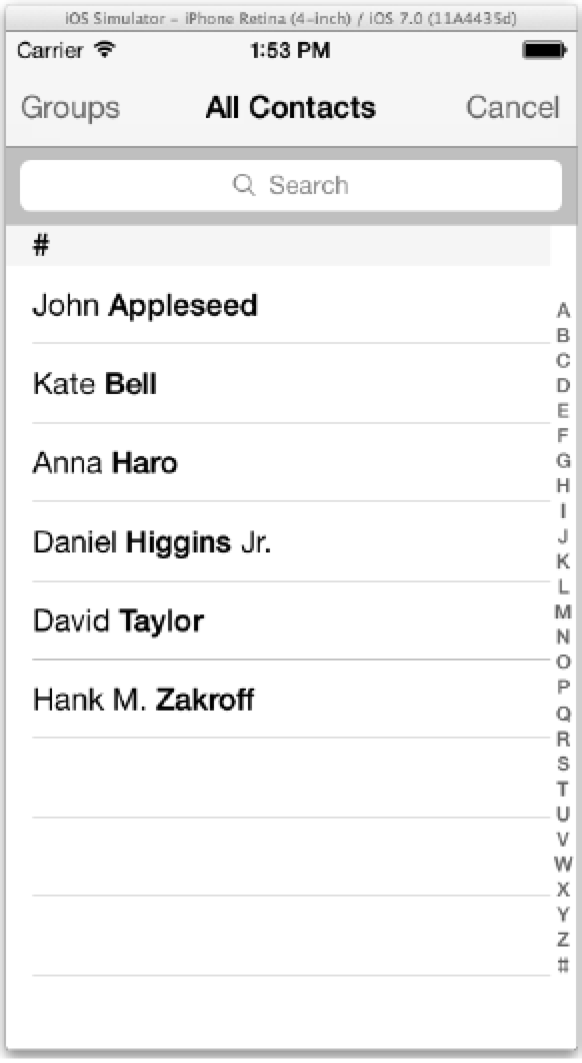
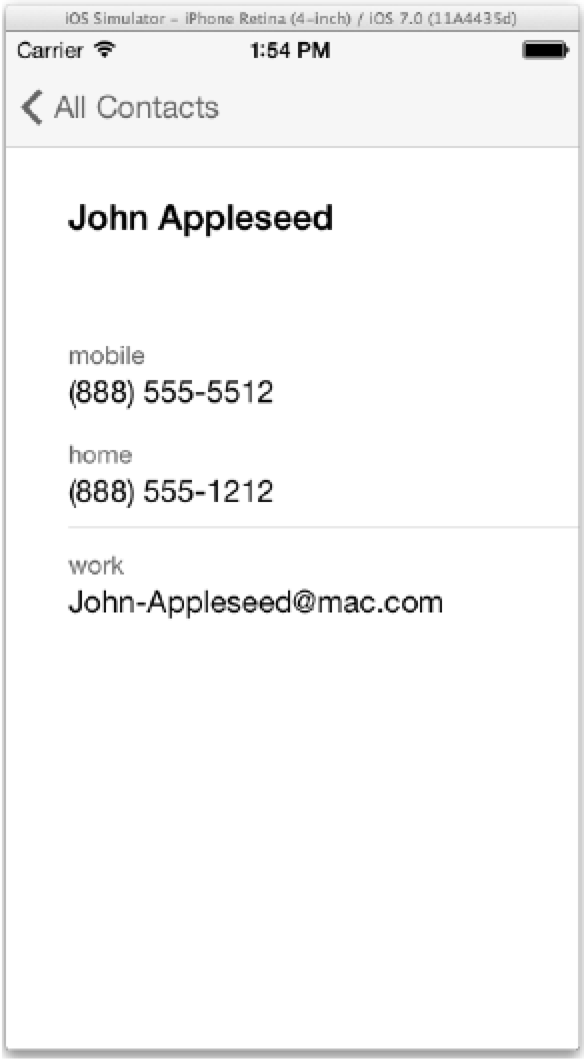


Figure 3: Presenting All Contacts in Simulator



John Appleseed

mobile
(888) 555-5512

home
(888) 555-1212

work
John-Appleseed@mac.com

Figure 4: Presenting Contact Details

If the `peoplePickerNavigationController:shouldContinueAfterSelectingPerson:property:identifier:` method returns YES and you tap a property, the default action is performed. When you tap the work address of John Appleseed, the address is shown on a map, as you can see in **Figure 5**.

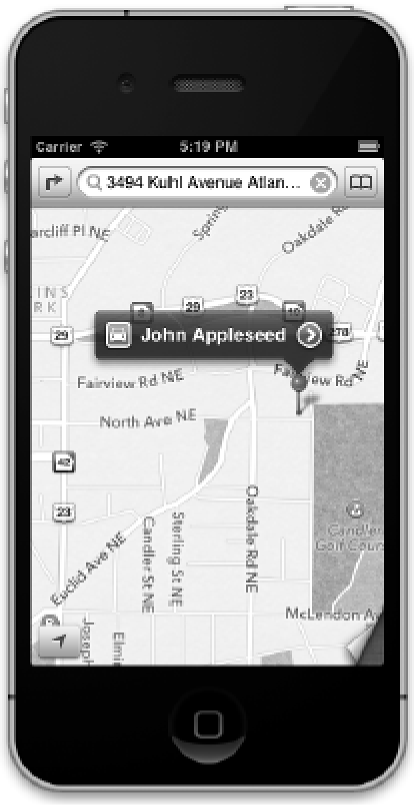


Figure 5: Presenting Work Address of John Appleseed

Before trying the given method to select a contact, enter a work address for John Appleseed in the Contacts database, since by default he does not have a work address.

Quick Tip

A very common mistake is to set the delegate property of the `peoplePicker` instance to `self`. If you do that, none of the delegate methods will be called, since you need to set the `peoplePickerDelegate` property to `self` instead.

<H2> [Requesting Access Permission]

With the introduction of iOS 8, Apple has improved security. So now if you launch the application on an iOS device running iOS 8.x or higher, a `UIAlertView` will appear when you tap the **Select a Contact** button. **Figure 6** shows the presented `UIAlertView`. The application is explicitly requesting you to give permission to the application to access your contact database.

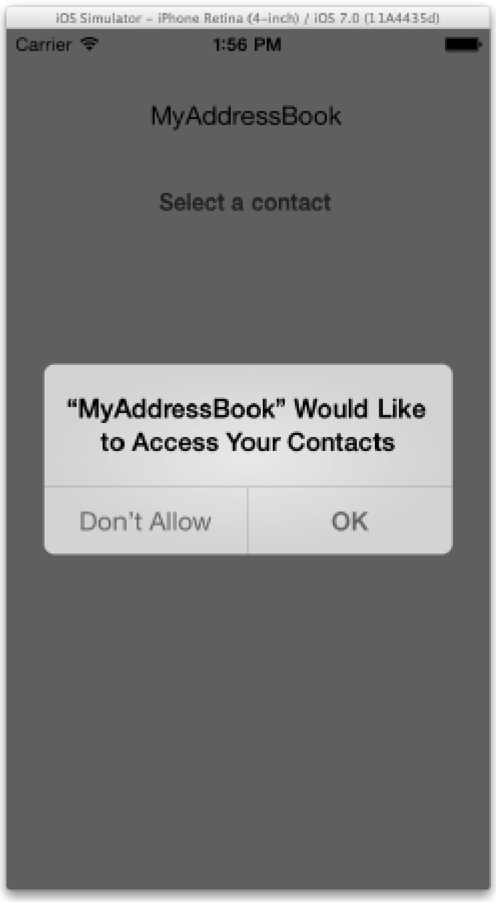
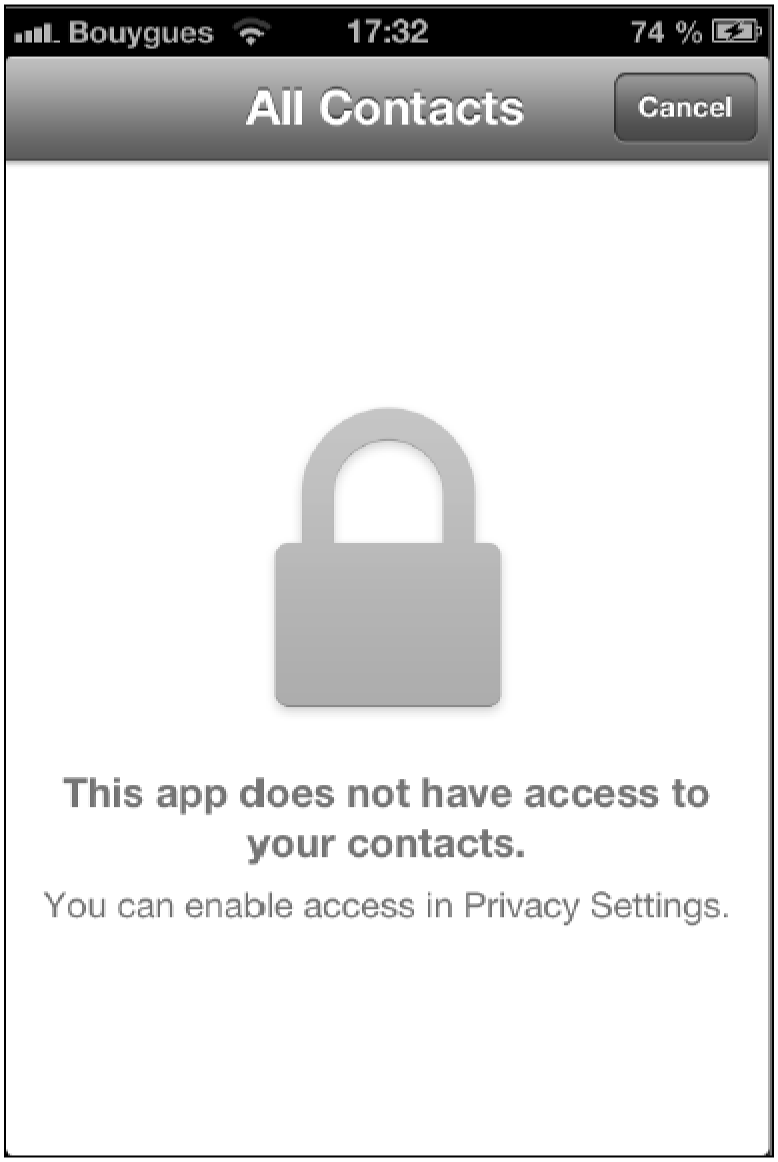

iOS Simulator – iPhone Retina (4-inch) / iOS 7.0 (11A4435d)
Carrier
1:56 PM
MyAddressBook
Select a contact
"MyAddressBook" Would Like to Access Your Contacts
Don't Allow
OK

Figure 6: Requesting Access Permission

If you select OK from the prompted `UIAlertView`, your application will run without any problem. If you do not allow access, the application will not run and a default screen is displayed, as shown in **Figure 7**.



All Contacts

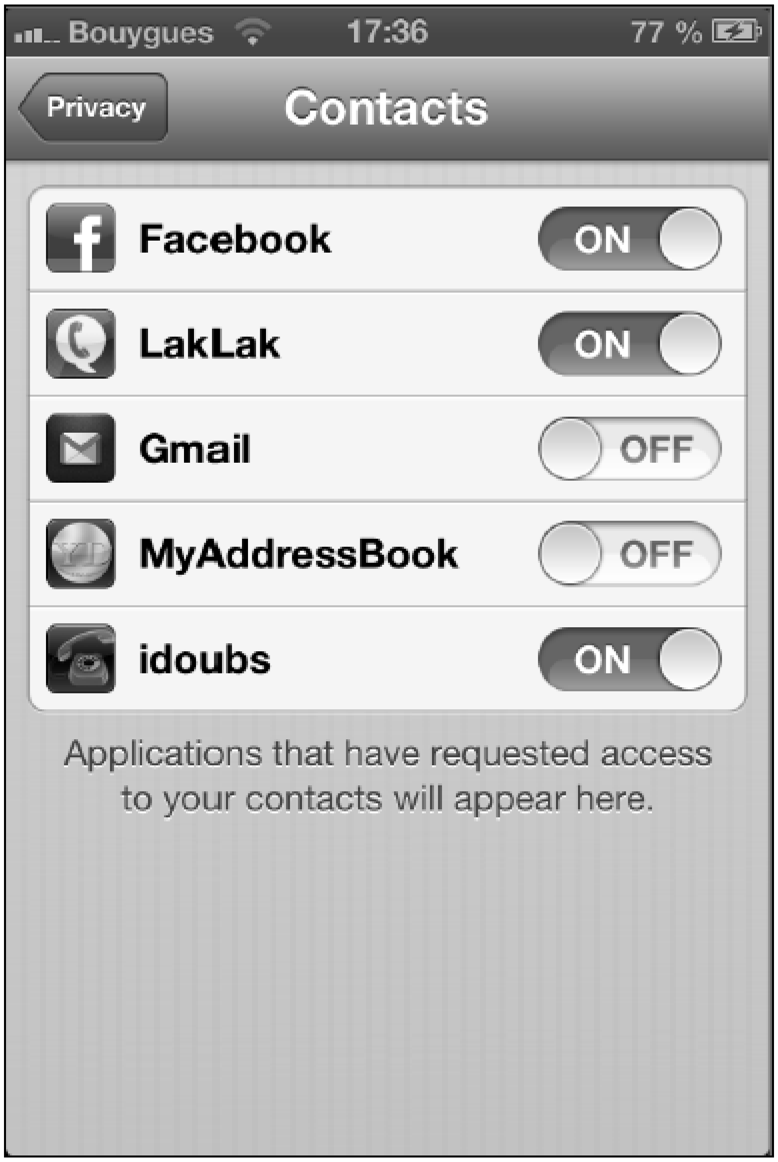
This app does not have access to your contacts.

You can enable access in Privacy Settings.

Figure 7: Screen Displayed after the App is Denied Access to Contacts

The user of your application always has the option to change access to the Address Book for your application. If you do not explicitly check if the application has been authorized, your application might crash—something you need to avoid under all circumstances.

The user can change the access permission to an application by using the **Settings** ⇨ Privacy ⇨ **Contacts** path on the device, as shown in **Figure 8**.



Facebook	ON
LakLak	ON
Gmail	OFF
MyAddressBook	OFF
idoubs	ON

Applications that have requested access to your contacts will appear here.

Figure 8: Changing Access Permission to an Application

The complete `YDViewController.m` implementation is shown below:

```
#import "YDViewController.h"
#import <AddressBookUI/AddressBookUI.h>

@interface YDViewController ()<ABPeoplePickerNavigationControllerDelegate>

@end
```

```
@implementation YDViewController

- (void)viewDidLoad
{
      [super viewDidLoad];
      // Do any additional setup after loading the view, typically from a
      nib.
}
- (IBAction)selectContact:(UIButton *)sender
{
      ABPeoplePickerNavigationController *peoplePicker =
            [[ABPeoplePickerNavigationController alloc] init];
            peoplePicker.peoplePickerDelegate=self;
      //optional to limit the number of properties displayed
      NSArray *displayedItems = [NSArray arrayWithObjects:
            [NSNumber numberWithInt:kABPersonPhoneProperty],
            [NSNumber numberWithInt:kABPersonEmailProperty],
            [NSNumber numberWithInt:kABPersonFirstNameProperty], nil];
      peoplePicker.displayedProperties = displayedItems;

      [self presentViewController: peoplePicker animated:NO completion:nil];
}
#pragma mark delegates
// Called after the user has pressed cancel
// The delegate is responsible for dismissing the peoplePicker
- (void)peoplePickerNavigationControllerDidCancel:
      (ABPeoplePickerNavigationController *)peoplePicker
{
      [self dismissViewControllerAnimated:NO completion:nil];
}

// Called after a person has been selected by the user.
// Return YES if you want the person to be displayed.
// Return NO to do nothing (the delegate is responsible for dismissing the
peoplePicker).
-   (BOOL)peoplePickerNavigationController:(ABPeoplePickerNavigationController
*)
     peoplePicker shouldContinueAfterSelectingPerson:(ABRecordRef)person
{
      //[self dismissViewControllerAnimated:NO completion:nil];
      return YES;
}
```

```
// Called after a value has been selected by the user.
// Return YES if you want default action to be performed.
// Return NO to do nothing (the delegate is responsible for dismissing the
peoplePicker).
-   (BOOL)peoplePickerNavigationController:(ABPeoplePickerNavigationController
*)
      peoplePicker shouldContinueAfterSelectingPerson:(ABRecordRef)person
      property:(ABPropertyID)property
      identifier:(ABMultiValueIdentifier)identifier
{
      return YES;
}

- (void)didReceiveMemoryWarning
{
      [super didReceiveMemoryWarning];
      // Dispose of any resources that can be recreated.
}

@end
```

You can display, edit, and delete contacts by using the `ABPersonViewController`. It is important to know that the `ABPersonViewController` object must be used with a `UINavigationController` in order to function properly in both **Swift** and Objective-C.

In **Swift**, you deal with a different code format. Some code format in `AppDelegate.`**`Swift`** and `ViewController.`**`Swift`** will be as follows:

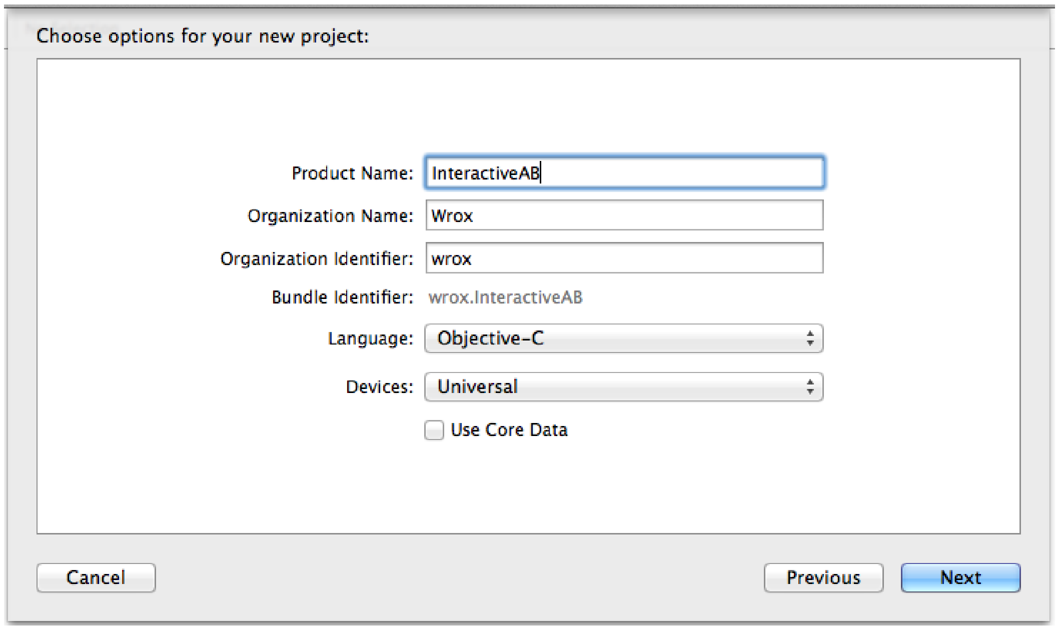
```
// importing addressBookUI
import AddressBook
import AddressBookUI

class ViewController: UIViewController,
ABPeoplePickerNavigationControllerDelegate{
      let peoplePicker: ABPeoplePickerNavigationController required
init(coder: aDecoder: NSCoder) {
            peoplePicker = ABPeoplePickerNavigationController()
      super.init(coder: aDecoder)
      peoplePicker.peoplePickerDelegate = self
      }


}
```

WILEY

```
// --- selecting contact and displaying some properties---
      @IBAction func selectContact(sender : UIButton) {
            self.presentViewController    (peoplePicker,    animated:    false,
      completion: nil)
                  }


       func peoplePickerNavigationControllerDidCancel(
            peoplePicker: ABPeoplePickerNavigationController!)

//---selected person by user---
       func peoplePickerNavigationController(
            peoplePicker: ABPeoplePickerNavigationController!,
            didSelectPerson person: ABRecordRef!)
//---Displaying items of the selected contact as ---
      let email: ABMultiValueRef = ABRecordCopyValue(person,
                  kABPersonEmailProperty).takeRetainedValue()               as
ABMultiValueRef
                  println(email)
                  }


      let FirstName: ABMultiValueRef = ABRecordCopyValue(person,
            kABPersonFirstNameProperty).takeRetainedValue()                 as
ABMultiValueRef
                  println(FirstName) }
```

<H2>[Displaying and Editing a Contact]

Start Xcode and create a new project using the **Single View Application** template, and name it **InteractiveAB** using the options shown in **Figure 9**.


Choose options for your new project:

Product Name: InteractiveAB
Organization Name: Wrox
Organization Identifier: wrox
Bundle Identifier: wrox.InteractiveAB
Language: Objective-C
Devices: Universal
Use Core Data

Figure 9: Create a New Project, InteractiveAB, in Xcode

Remove the `YDViewController` header, implementation, and Interface Builder files from your project. Copy these three files from the `MyAddressBook` project into the `InteractiveAB` project. Because the `ABPersonViewController` works properly only with a `UINavigationController`, modify the `YDAppDelegate` class and add the strong property of type `UINavigationController`, as shown below:

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

@class YDViewController;

@interface YDAppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;
@property (nonatomic, strong) UINavigationController *navController;
@property (strong, nonatomic) YDViewController *viewController;

@end
```

In **Swift**, you will define the `AppDelegate` class in the following format:

```
class AppDelegate: UIResponder, UIApplicationDelegate {
      var window: UIwindow?
```

In **Objective-C,** open the `YDViewController.m` file and replace the `application:didFinishLaunchingWith Options:` method with the implementation shown below:

```
- (BOOL)application:(UIApplication *)application
 didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
      self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen]
      bounds]];
      self.viewController = [[YDViewController alloc]
      initWithNibName:@"YDViewController" bundle:nil];
      self.navController = [[UINavigationController alloc]
                        initWithRootViewController:self.viewController];
      self.window.rootViewController = self.navController;
      self.window.backgroundColor = [UIColor clearColor];
      [self.window makeKeyAndVisible];
      return YES;
}
```

The `didFinishLaunchingWithOption:` method will be implemented as a function in **Swift** in the `AppDelegate.`**`Swift`** file. The format will be as follows:

```
      func application(application: UIApplication!,
            didFinishLaunchingWithOptions launchOptions: [NSObject :
AnyObject]?) -> Bool {
```

Now, to subscribe the `ABPersonViewControllerDelegate` protocol, edit the `YDViewController.m` file in Objective-C. In **Swift**, you will add code in `ViewController.`**`Swift`**.

Follow these steps to display and edit a contact by using the `ABPersonViewController`:

1. Create and initialize an instance of the `ABPersonViewController` class.
2. Set the `personViewDelegate` property to `self`.
3. Set the `displayedPerson` property to the person record you want to display.
4. Optionally, set the `displayedProperties` as you did in the `MyAddressBook` example.
5. Display the `ABPersonViewController` using the `pushViewController:animated:` method of your `UINavigationController`.

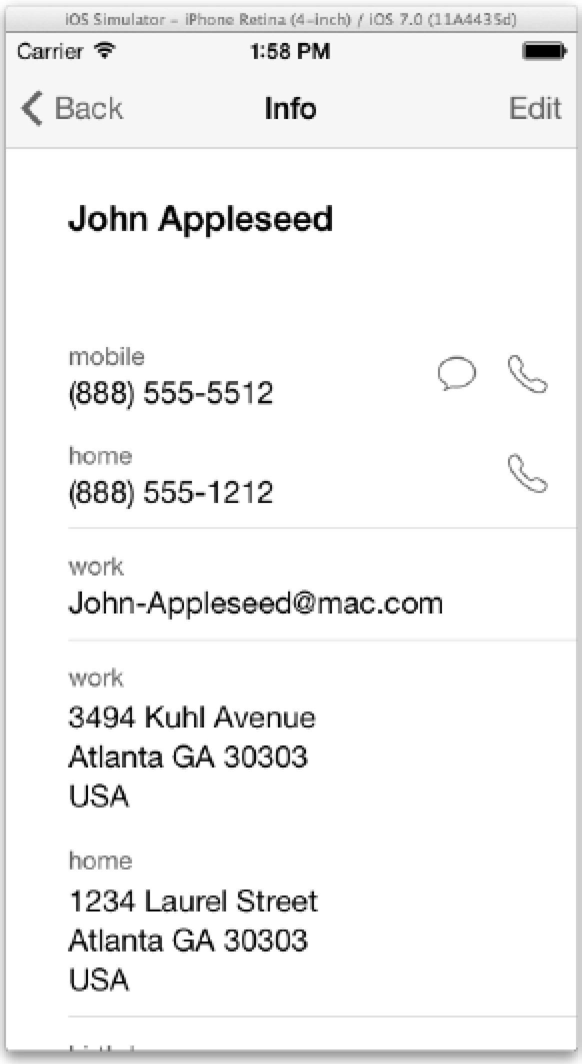
Implement a method named `showPersonViewController:` in which you create an `ABPersonViewController` instance named `pvController`. Then, set the personViewDelegate to self and set the allowsEditing property. Finally, display the pvController by using the pushViewController:animated: method of your UINavigationController as shown below:

```
- (void) showPersonViewController: (ABRecordRef)person
{
      ABPersonViewController *pvController =
      [ [ABPersonViewController alloc] init];
      pvController.personViewDelegate = self;
      pvController.displayedPerson = person;
      //Allow users to edit the information
      pvController.allowsEditing = YES;
      [self.navigationController        pushViewController.pvController
      animated:YES];
}
```

In the peoplePickerNavigationController:shouldContinueAfterSelectingPerson: method, you first call the dismissViewController:animated: method to remove the ABPeoplePickerNavigationController from the view stack. You can call the showPersonViewcontroller:method and pass the selected person record as shown below:

```
-(BOOL)peoplePickerNavigationController:(ABPeoplePickerNavigationController
*)
peoplePicker shouldContinueAfterSelectingPerson: (ABRecordRef) person
{
      [self dismissViewControllerAnimated:NO completion:nil];
      [self showPersonViewController:person];
      return NO;
}
```

When you launch the application and tap the Select a Contact button, the ABPeoplePickerNavigationController is presented. When you select a contact from the list, the showPersonViewController: method is called with the selected person, and the ABPersonViewController is presented, as shown in **Figure 10**.



John Appleseed

mobile
(888) 555-5512

home
(888) 555-1212

work
John-Appleseed@mac.com

work
3494 Kuhl Avenue
Atlanta GA 30303
USA

home
1234 Laurel Street
Atlanta GA 30303
USA

Figure 10: Displaying the Contact Details

Tap the **Edit** button to make the changes you want to make, and then tap the **Done** button to save these changes to the Address Book database.

<H2>[Creating a Contact]

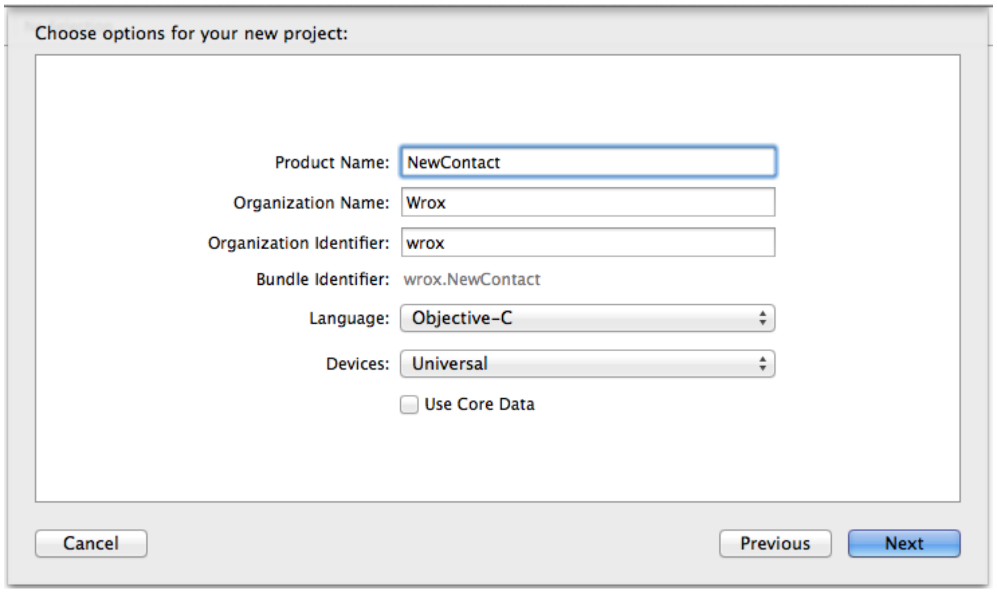
To create a new contact in the Address Book, you can use the `ABNewPersonViewController` class, which allows users to create a new contact. The steps for creating a contact are similar to the steps for displaying and editing a contact. These steps are as follows:

1. Subscribe to the `ABNewPersonViewControllerDelegate` protocol.
2. Create and initialize an instance of the `ABNewPersonViewController` class named `newController`.
3. Set the `newPersonViewDelegate` property to `self`.

4. Create and initialize a new `UINavigationController` named `newNavController` and set its root view controller to the `newController` object you just created.

5. Present the `newNavController` as a modal view using the `presentModalViewController:animated:` method.

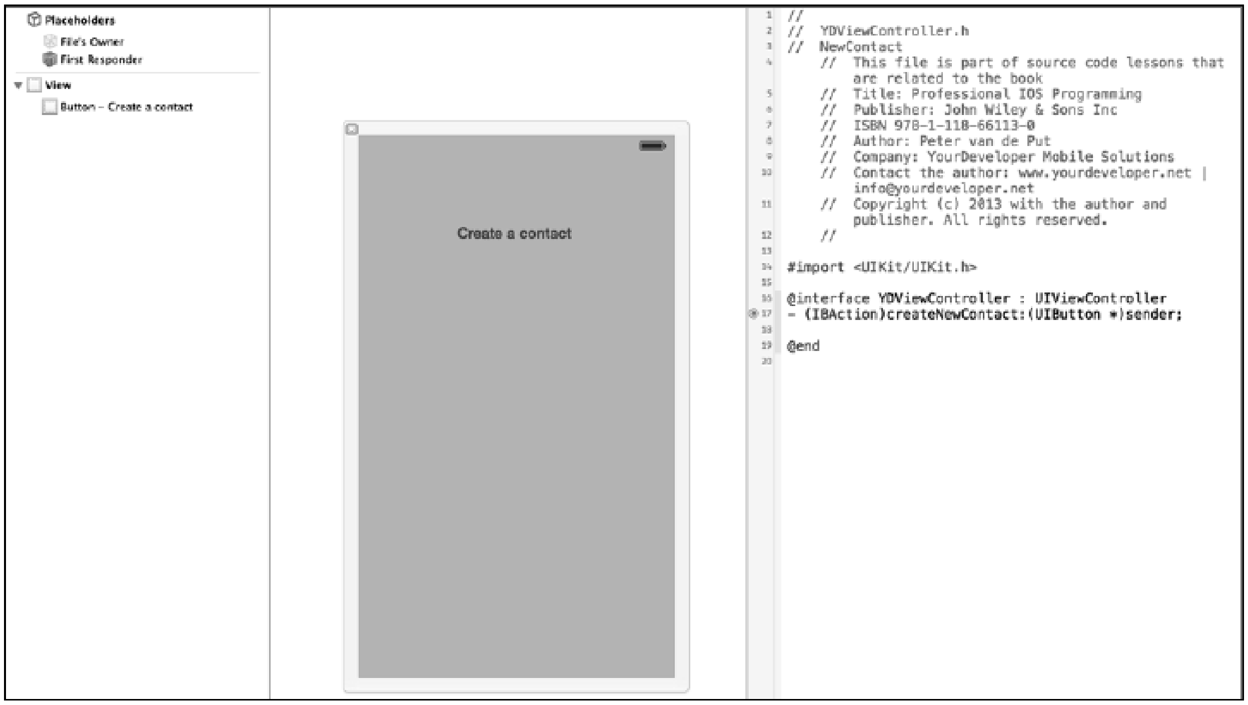
Start Xcode and create a new project using the **Single View Application** template. Name the new project as **NewContact** using the options shown in **Figure 11**.


Choose options for your new project:

Product Name: NewContact
Organization Name: Wrox
Organization Identifier: wrox
Bundle Identifier: wrox.NewContact
Language: Objective-C
Devices: Universal
Use Core Data

Figure 11: Create a New Project, NewContact, in Xcode

Open the `YDViewController.xib` file using Interface Builder and the Assistant Editor and create a user interface as shown in **Figure 12**.



```
//
//  YDViewController.h
//  NewContact
    //  This file is part of source code lessons that
        are related to the book
    //  Title: Professional IOS Programming
    //  Publisher: John Wiley & Sons Inc
    //  ISBN 978-1-118-66113-0
    //  Author: Peter van de Put
    //  Company: YourDeveloper Mobile Solutions
    //  Contact the author: www.yourdeveloper.net |
        info@yourdeveloper.net
    //  Copyright (c) 2013 with the author and
        publisher. All rights reserved.
    //

#import <UIKit/UIKit.h>

@interface YDViewController : UIViewController
- (IBAction)createNewContact:(UIButton *)sender;

@end
```

Figure 12: Create a User Interface for the NewContact Project

Import the `AddressBook` and the `AddressBookUI` frameworks in your project and open the `YDViewController.m` file. Import the `AddressBookUIheader` file and subscribe to the `ABNewPersonViewControllerDelegate` protocol.

Implement the `createNewContact:` method as shown in the following code. Here, you create the `ABNewPersonViewController` class, set the `newPersonViewDelegate` property, create a `UINavigationController`, and present it modally.

```
- (IBAction) createNewContact : (UIButton *) sender
{
     ABNewPersonViewController* newPersonController =
     [ [ABNewPersonViewController alloc] init];
     newPersonController.newPersonViewDelegate=self;
     UINavigationController* newNavController =
          [ [UINavigationController alloc]
          initWithRootViewController:newPersonController];
     [self     presentViewController:newNavController     animated:YES
     completion:nil];
}
```

When the user taps the Save or Cancel button, the `newPersonController` calls the `newPersonViewController:didCompleteWithNewPerson:` method, which you need to implement.

Dismiss the `ViewController` by calling the `dismissViewControllerAnimated:completion:` method. If the user has tapped the **Cancel** button, the value of person is `NULL`; otherwise, it will represent the person record just created.

When you launch the application and tap the Create a Contact button, the application will display the `ABNewPersonViewController` class, as shown in **Figure 13**.

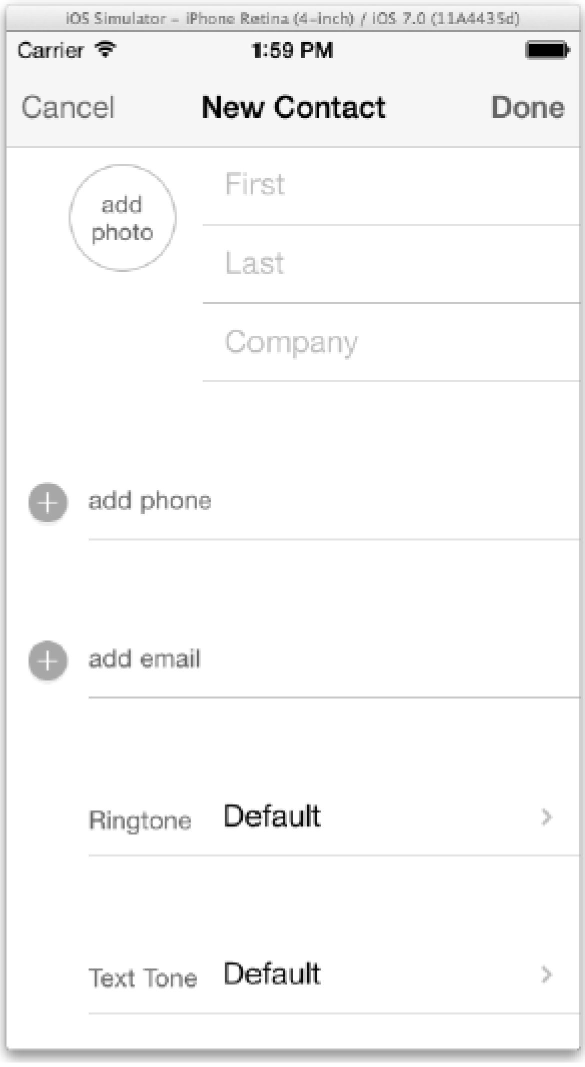


Figure 13: The New Contact Interface

You can modify the above syntax in **Swift** as follows:
```
   class ViewController: UIViewController, ABNewPersonViewControllerDelegate {


            @IBAction func createNewContact( sender: AnyObject ) {
      var newPersonController = ABNewPersonViewController()
      newPersonController.newPersonViewDelegate = self
      let   newNavController   =   UINavigationController(rootViewController: 
controller)
      self.presentViewController(navigationController,   animated:   false   , 
completion: nil)
      }
```

<H2> [Deleting a Contact]

Deleting a contact from the Address Book database is not possible without programmatically accessing the Address Book. This is covered in the next section. So far you have been using classes and methods of the `ABAddressBookUI` framework to display, create, and edit contacts in the Address Book database. In the next section, you will learn how to programmatically access the Address Book without using the user interface element of the `ABAddressBookUI` framework.

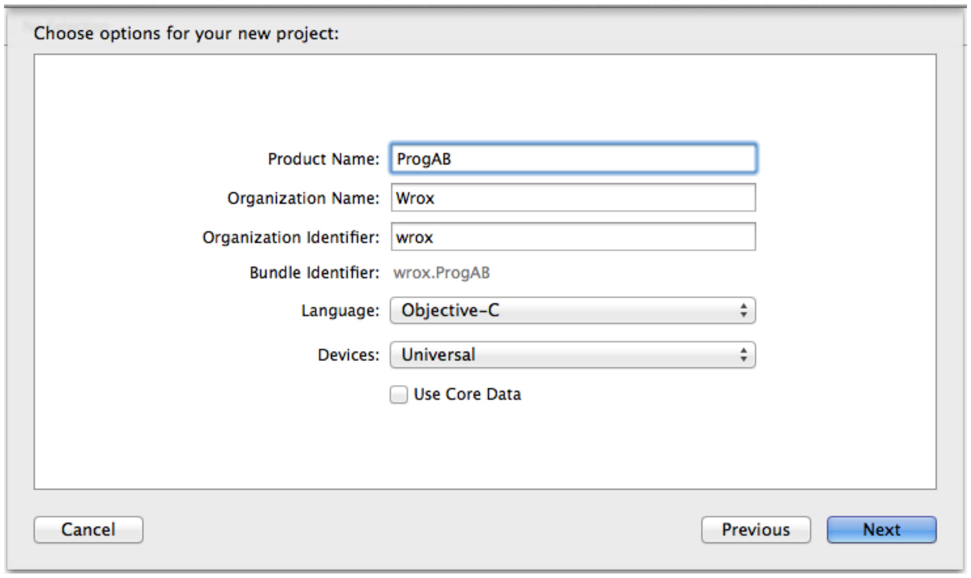
<H1> [Programmatically Accessing the Address Book]

Source: [Professional iOS Programming][Chapter 12][375]

Suppose you want your application to use the contacts from the Address Book database, but you want to use your custom `UIViewController` and `UIView` objects rather than the Address Book user interface components to display or capture the contact details. Another scenario is that you need to make use of the contacts in the Address Book database and want to store the details in Core Data with additional properties and/or relationships.

In those cases, you need to access the Address Book database programmatically and write custom logic to access the individual contact records and properties.

Start Xcode and create a new project using the **Single View Application** template, and name it **ProgAB** using the options shown in **Figure 14**.


Choose options for your new project:

Product Name: ProgAB
Organization Name: Wrox
Organization Identifier: wrox
Bundle Identifier: wrox.ProgAB
Language: Objective-C
Devices: Universal
Use Core Data

Figure 14: Create a New Project, ProgAB, in Xcode

Add the `AddressBook` and `AddressBookUI` frameworks to your project. There are four basic objects that are involved in the interaction with the Address Book:

- Address Books
- Records
- Single-value properties
- Multi-value properties

These four basic objects are explained in detail in the following sections.

<H2> [Address Books]

You use the `ABAddressBookRef` object to access the Address Book database. You need to create an instance of the `ABAddressBookRef` object using the `ABAddressBookCreateWithOptions` function, as shown below:

```
CFErrorRef erro = NULL;
ABAddressBookRef addressBook = ABAddressBookCreateWithOptions (NULL, &error);
```

Quick Tip

Instances of `ABAddressBookRef` cannot be used by multiple threads. Each thread needs to create its own instance of `ABAddressBookRef`.

<H3> [Requesting Access Permission]

As you now know, the user needs to give explicit permission to an application to access the Address Book. When you want to access the Address Book programmatically, you have a method containing your application's logic to select or manipulate records. To guarantee a flawless operation of this method, you need to ensure that the user has given access to the application to use the Address Book.

You can use the `ABAddressBookRequestAccessWithCompletionfunction`, which will prompt the user for access to his/her Address Book. You can use the `ABAddressBookGetAuthorizedStatus` function to check the authorization status for accessing the Address Book. This function returns four possible values:

- ➢ `kABAuthorizationStatusNotDetermined`
- ➢ `kABAuthorizationStatusDenied`
- ➢ `kABAuthorizationStatusAuthorized`
- ➢ `kABAuthorizationStatusRestricted`

When the user does not authorize access to his/her Address Book, you can use a `UIAlertView` to notify the user with an informational message about what will happen, such as the application cannot access the Address Book and therefore will not operate with all functionality.

The following code shows an implementation of the logic to request access and check the result using the `ABAddressBookGetAuthorizationStatus` function:

```
__block BOOL accessGranted = NO;
 if (ABAddressBookRequestAccessWithCompletion != NULL)
       {
       ABAddressBookRequestAccessWithCompletion(addressBook,
       ^(bool granted, CFErrorRef error)
       {
       accessGranted = granted;

      dispatch_async(dispatch_get_main_queue(),^{
       accessGranted=YES;
       });
 });
 }
 if (ABAddressBookGetAuthorizationStatus() ==
       kABAuthorizationStatusNotDetermined)
       {
       //Access is not determined
       }
 else if (ABAddressBookGetAuthorizationStatus() ==
       kABAuthorizationStatusDenied)
       {

       //Access is denied
       }
 else if (ABAddressBookGetAuthorizationStatus() ==
       kABAuthorizationStatusAuthorized)
       {
       //Access is authorized
       }
 else if (ABAddressBookGetAuthorizationStatus() ==
       kABAuthorizationStatusRestricted)
       {
       //Access is restricted
       }
```

In **Swift**, you can add the `if` condition in a function call as follows:

```
      func access(){
            if(ABAddressBookGetAuthorizationStatus()                    ==
ABAuthorizationStatus.NotDetermined) {
      //---Access is not determined---
      }
      else        if        (ABAddressBookGetAuthorizationStatus()        ==
ABAuthorizationStatusDenied){
            //Access is denied
      }
//--- similar to Objective-C contact get contact name---
      addressBook  =  !ABAddressBookCreateWithOption(emptyDictionary,  nil  )
ABAddressBookRequestAccessWithCompletion(addressBook,{(granted, error in )
      if granted{
            self.getContactName()
      }
      else{
      println("error")
      }
```

<H3> [Saving or Abandoning Address Book Changes]

After you have created the `ABAddressBookRef` instance and checked the authorization, you can read and write to the Address Book database. Once you are done with your operations, you can either save the changes you have made or abandon them. Using the `ABAddressBookHasUnsavedChanges` function you can check if changes have been made to the

Address Book from your application. If you want to abandon the changes, you call the `ABAddressBookRevert` function, and if you want to save the changes, you call the `ABAddressBookSave` function. Both functions are shown below:

```
if (ABAddressBookHasUnsavedChanges(addressBook))
       {
             //save changes
             BOOL didSave = ABAddressBookSave(addressBook, &error);
             if (!didSave)
                   {
                   //handle the error here
                   }
             //abandon changes
             ABAddressBookRevert(addressBook);
       }
```

<H3>Staying Synchronized

Once your application has been authorized by the user to access the Address Book, you can read and write to the Address Book database. Other applications also might have created an instance to the Address Book, and performed changes to the Address Book database (for example, in a background operation).

Use the `ABAddressBookRegisterExternalChangeCallback` function to register a function of the `prototypeABExternalChangeCallBack` that will handle changes in the Address Book from another application.

It is possible to create multiple callback functions by calling the `ABAddressBookRegisterExternalChangeCallback` function multiple times with a different function.

You can unregister the callback function by using the `ABAddressBookUnregisterExternalChangeCallback` function.

In your callback function you can decide to save your changes or to abandon your changes—this is completely up to your application's logic and requirements. A sample callback function is shown below:

```
void addressBookChanged(ABAddressBookRef reference,
                   CFDictionaryRef dictionary,
                   void *context)
{
       //Something has changed perform your action
}
```

In **Swift**, it is difficult to pass C pointers, so it is good to use **Swift** closure syntax.

<H2>[Understanding Records]

All the contact information in the Address Book is stored in records. A **record** in the Address Book database is of type `ABRecordRef` and contains a unique record identifier, which can be retrieved with the `ABRecordGetRecordID` function. It is not safe to pass records across threads; instead, you pass the record identifier. Each record represents a person or group. You can determine the type of record using the `ABRecordGetRecordType` function, because a person record will return the value `kABPersonType` and a group will return the value `kABGroupType`.

Quick Tip

The actual data of a record is stored in a collection of properties. The available properties for a person record are different from the available properties for a group record.

<H3>[Person Records]

A person record represents a person and is made up of two types of properties. Properties such as a first name or a last name (of which a person has only one) are stored as single-value properties. Properties like a phone number (of which a person can have more than one) are multi-value properties.

<H3>[Group Records]

A user can use group records to organize his/her Address Book database by creating groups. A group record only has one single-value property named `kABGroupNameProperty`, which holds the name of the group. To get all person objects related to a group record, use the `ABGroupCopyArrayOfAllMembers` function or the `ABGroupCopyArrayOfAllMembersWithSortOrdering` function. Both functions return a `CFArrayRef` of `ABRecordRef` objects.

<H2> [Single-Value Properties]

`ABRecord` objects contain a single-value property, which you can set, copy, or remove. There are three functions that can work with the properties of a record:

- **`ABRecordCopyValue`**: This is the getter for the property.
- **`ABRecordSetValue`**: This is the setter for the property.
- **`ABRecordRemoveValue`**: This is used to delete a property.

The following code shows how to access the first name property of a person record:

```
CFStringReffirstName; //assumption aRecord is a valid ABRecordRef object
```

```
firstName = ABRecordCopyValue(aRecord, kABPersonFirstNameProperty);
```

The following code sets the value Peter to the first name property of a person record:

```
NSString* firstname = @”Peter”;
ABRecordSetValue(aRecord, kABPersonFirstNameProperty,
 (__bridge CFTypeRef)(firstname), &error);
```

<H2>[Multi-Value Properties]

Multi-value properties consist of a list of values. Each value has an identifier and a text label.

There can be multiple values with the same label, but the identifier is always unique. Some generic property labels are defined in the `ABPerson.h` file, which are `kABWorkLabel`, `kABHomeLabel`, and `kABOtherLabel`.

Individual values of multi-value properties are accessible by identifier or by index. Use the functions `ABMultiValueGetIndexForIdentifier` and `ABMultiValueGetIdentifierAtIndex` to convert between indices and multi-value identifiers. To keep a reference to a particular value, store and use the identifier because the index might change if values are added or removed.

<H2>[Creating a Contact Programmatically]

To create a new contact programmatically, start by creating the `ABAddressBookRef` instance named `addressBook` and make sure you are granted access to the Address Book.

Create an `ABRecordRef` instance named `newPerson` using the `ABPersonCreate` function. Set the values for the first name and the last name properties.

To add a mobile phone number, create an instance of `ABMutableMultiValueRef` named `multiPhone` and set the property value for the `kAB-PersonPhoneMobileLabel`.

Finally, call the `ABAddressbookAddRecord` function to add the record to the Address Book. You must realize that although you have added the new record to the Address Book, it is not saved yet because you did not call the `ABAddressBookSave` function, which is the final thing to do.

A sample implementation is shown below:

```
CFErrorRef error = NULL;
ABAddressBookRef addressBook = ABAddressBookCreateWithOptions(NULL, &error);
 __block BOOL accessGranted = NO;
 if (ABAddressBookRequestAccessWithCompletion != NULL)
```

```
       {
       ABAddressBookRequestAccessWithCompletion(addressBook,
       ^(bool granted, CFErrorRef error)
       {
             accessGranted = granted;
             dispatch_async(dispatch_get_main_queue(),^{
             accessGranted=YES;
             });
       });
       } continues
ABRecordRef newPerson = ABPersonCreate();
       //add firstname
       ABRecordSetValue(newPerson, kABPersonFirstNameProperty,
       (__bridge CFTypeRef)(@"Peter"), &error);
       //add lastname
       ABRecordSetValue(newPerson, kABPersonLastNameProperty,
       (__bridge CFTypeRef)(@"van de Put"), &error);
       //add mobile phone number
       ABMutableMultiValueRef multiPhone =
             ABMultiValueCreateMutable(kABMultiStringPropertyType);
       ABMultiValueAddValueAndLabel(multiPhone,
             (__bridge CFTypeRef)(@"+3311111111"),
             kABPersonPhoneMobileLabel, NULL);
       ABRecordSetValue(newPerson, kABPersonPhoneProperty, multiPhone,nil);

      //Add the record to the address book
       ABAddressBookAddRecord(addressBook, newPerson, &error);
       //Save the changes
       ABAddressBookSave(addressBook, &error);
       if (error != NULL)
             {
                   //handle your error here
             }
```

<H2>[Selecting One or More Contacts]

To select all contacts from the Address Book, you can use the `ABAddressBookCopyArrayOfAllPeople` function, which returns a `CFArrayRef`. You can determine the number of records in the result by using the `ABAddressBookGetPersonCount` function.

Now, create a loop to create an `ABRecordRef` instance for each record in the array by using the `CFArrayGetValueAtIndex` function. A sample implementation is shown below:

```
-(void)loadAllPeople
{
 CFErrorRef error = NULL;
 ABAddressBookRef addressBook = ABAddressBookCreateWithOptions(NULL, &error);
 __block BOOL accessGranted = NO;
 if (ABAddressBookRequestAccessWithCompletion != NULL)
       {
       ABAddressBookRequestAccessWithCompletion(addressBook,
       ^(bool granted, CFErrorRef error)
                                  {
                                  accessGranted = granted;

                     dispatch_async(dispatch_get_main_queue(),^{
                                        accessGranted=YES;
                                  });
                                       });
       }
 CFArrayRef allPeople = ABAddressBookCopyArrayOfAllPeople(addressBook);
 CFIndex nPeople = ABAddressBookGetPersonCount(addressBook);
 for( int i = 0 ; i < nPeople ; i++ )
       {
       ABRecordRef ref = CFArrayGetValueAtIndex(allPeople, i );
       ABRecordType rectype = ABRecordGetRecordType(ref);
       //check if record is a person record
       if (rectype == kABPersonType)
             {
             //do what you want to do
             }
       }
 //release the addressBook
 CFRelease(addressBook);
}
```

To select a series of contacts, create an `NSArray` named `allContacts` by calling the `ABAddressBookCopyArrayOfAllPeople` function. Create an `NSPredicate` to filter this array using the `filteredArrayUsingPredicate:` method of the `NSArray` class, as shown below:

```
-(void)filterContacts
{
 CFErrorRef error = NULL;
 ABAddressBookRef addressBook = ABAddressBookCreateWithOptions(NULL, &error);
 __block BOOL accessGranted = NO;
 if (ABAddressBookRequestAccessWithCompletion != NULL)
```

```
        {
        ABAddressBookRequestAccessWithCompletion(addressBook,
        ^(bool granted, CFErrorRef error)
                                            {
                                            accessGranted = granted;
                          dispatch_async(dispatch_get_main_queue(),^{
                                                  accessGranted=YES;
                                            });
                                            });
        }
        //Create an NSArray containing all contacts
        NSArray* allContacts= (__bridge NSArray *)
                    (ABAddressBookCopyArrayOfAllPeople(addressBook));
        // Build a predicate that searches for contacts with at least one
       phone
        number starting with +33.
        NSPredicate* predicate = [NSPredicate predicateWithBlock: ^(id record,
              NSDictionary* bindings) {
              ABMultiValueRef phoneNumbers =
              ABRecordCopyValue( (__bridge ABRecordRef)record,
                                kABPersonPhoneProperty);

BOOL result = NO;
for (CFIndex i = 0; i < ABMultiValueGetCount(phoneNumbers); i++) {
NSString* phoneNumber = (__bridge_transfer NSString*)
ABMultiValueCopyValueAtIndex(phoneNumbers, i);
if ([phoneNumber hasPrefix:@"+33"]) {
result = YES;
break;
}
}
CFRelease(phoneNumbers);
return result;
}];
//release the addressBook
CFRelease(addressBook);
}
```

The code to fetch all contacts in **Swift** is as follows:

```
let loadAllPeople: NSArray
```

```
ABAddressBookCopyOfAllPeople(addressBook).takeRetainedValue()
for contactRef: ABRecordRef in loadAllPeople{
firstName                  =                   ABRecordCopyValue(contactRef,
kABPersonFirstNameProperty).takeUnretainedValue() as? NSString{
//---all properties with ABProperty can be fetched
```

<H2>[Deleting a Contact Programmatically]

To delete a contact from the Address Book, you simply use the `ABAddressBookRemoveRecord` function. Again, you must realize that although you have deleted the record from the Address Book, it is not yet physically removed from the Address Book database because you did not call the `ABAddressBookSave` function, which is the final thing to do.

The following example shows how to delete the `aPerson` record from the Address Book, assuming that `aPerson` is a valid initialized record:

```
ABAddressBookRemoveRecord(addressBook, aPerson, &error);//Save the changes


ABAddressBookSave(addressBook, &error);
if (error != NULL)
 {
 //handle your error here
}
```

Cheat Sheet

- The Address Book technology consists of the following parts:
    - Address Book framework
    - Address Book UI framework
    - Address Book database
    - Contacts application
- The Address Book UI framework provides controllers to facilitate select, edit, create and display address book records from the address book database
- The `ABPeoplePickerNavigationController` class is used to display views to pick contact information from an Address Book.
- The `peoplePickerNavigationController:shouldContinueAfterSelectingPerson:property:identifier:` method and the

`peoplePickerNavigationController:shouldContinueAfterSelectingPerson:` method are deprecated in iOS 8.
- In iOS 8 you use `UIAlertController` instead of `UIAlertView`. This is because `UIAlertView` and `UIAlertViewDelegate` are deprecated in iOS 8.
- The `ABPersonViewController` object is used with a `UINavigationController` to function properly in both **Swift** and Objective-C.