MODULE: Taking Your Apps to Production

WEEK-10 SESSION-1:

Event Programming

Sources:

Professional iOS Programming [Chapter 13] [9781118661130]

MODULE Objectives

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

- Work with calendars and reminders
- Programmatically access the event store where the calendars and reminders are stored

Session Objectives

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

- Use the Event Kit framework to add and modify events in the Calendar database
- Programmatically access the Calendar database to add and modify events
- Work with reminders

<H1> Introduction

The Calendar app in iOS is the default way to find appointments and events. Suppose you have created an application that needs to access the Calendar database of a user to interface with his/her events and reminders. To do so, you need to know how to ask the user to authorize your application to access the Calendar database.

In this session, you will learn how to create and edit events using the user interface elements of the Event Kit framework as well as programmatically in the Calendar database. You will also learn to work with reminders.

<H1> Introduction to the Event Kit Framework

Source: [Professional iOS Programming] [Chapter 13] [Page No. 385]

The **Event Kit framework** provides you access to the user's Calendar and Reminders information. The native iOS applications Calendar and Reminders use the same database for storing their data, which is called the **Calendar database**. In programming terms, this is known as an **event store**. Events refer to calendar items and reminders.

The Event Kit technology consists of the following two parts:

- **The Event Kit Framework**: This enables you to read the user's Calendar database and also modify it by creating or editing events. To stay synchronized between applications, your application can create an observer to receive notifications from the Event Kit framework, in case the event store is modified by another application.

- **The Event Kit UI Framework**: This provides user interface elements for manipulating events.

Big Picture

Event programming is useful for an application, as you can get day-to-day events and holiday updates by accessing the calendar. The important feature of calendar events and programming is storing events and reminders to the calendar, which you can create, edit, and delete.

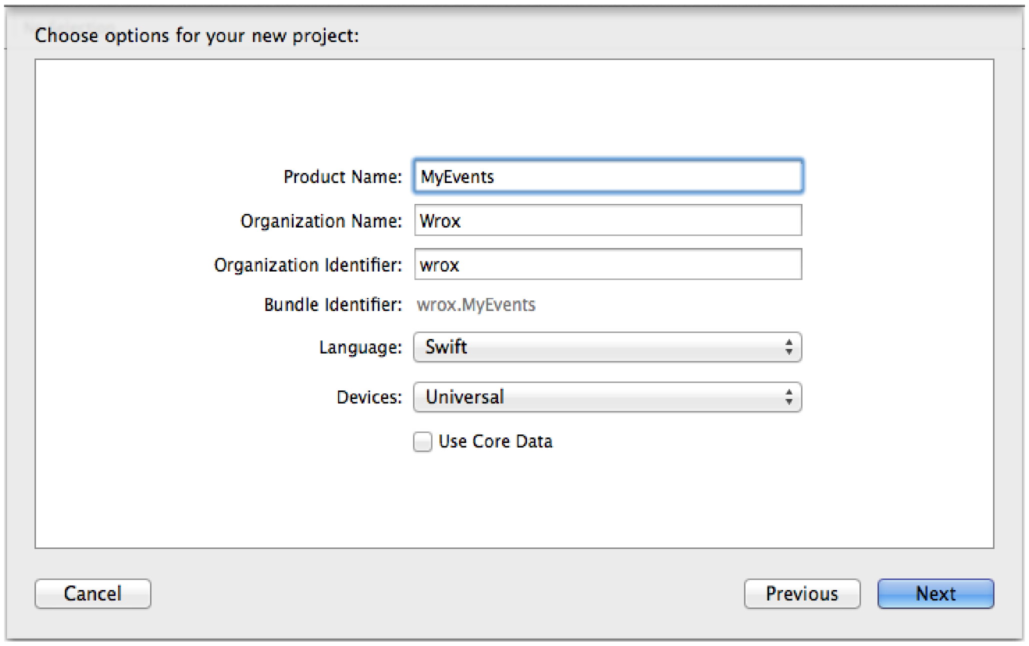
<H2> Using the Event Kit UI Framework

Source: [Professional iOS Programming] [Chapter 13] [Page No. 386]

The Event Kit UI framework provides the following user interface elements to work with events:

- **`EKCalendarChooser`**: This interface element enables the user to select one or more calendars.
- **`EKEventViewController`**: This interface element enables the user to create or edit an existing event.

To work with events in the Event Kit UI framework, start Xcode and create a new project using the **Single View Application** template. Name the project as **MyEvents**. You can choose **Objective-C** or **Swift** as the language, as shown in **Figure 1**.



Choose options for your new project:

Product Name: MyEvents
Organization Name: Wrox
Organization Identifier: wrox
Bundle Identifier: wrox.MyEvents
Language: Swift
Devices: Universal
Use Core Data

Figure 1: Create a Project, MyEvents, in Xcode

Quick Tip

It is important to realize that a user can have multiple calendars. In the device's settings, a user can set a default calendar by following the **Settings** ➪ Mail, Contacts, Calendars

<H2>Requesting Access Permission

Source: [Professional iOS Programming] [Chapter 13] [Page No. 386-387]

Apple introduced calendar access in iOS 4 and reminder access in iOS 6. Subsequently, Apple improved security access in iOS 7. If you run an application, a `UIAlertView` will appear that explicitly asks for user authorization to access the Calendar database. You deal with this requesting access permission just as you did with the Address Book database.

The following code snippet shows how to prompt the user to authorize your application to access the Calendar database.

```
-(void)requestAccessToCalendar
{
      self.myEventStore = [[EKEventStore alloc] init];
      __block BOOL accessGranted = NO;
      [self.myEventStore requestAccessToEntityType:EKEntityTypeEvent
            completion:^(BOOL granted, NSError *error)
      {
            // handle access here
            accessGranted = granted;
            dispatch_async(dispatch_get_main_queue(),^{
                  accessGranted=YES;
            });
      }];
}
```

The above code creates and initializes an instance of the `EKEventStore`. The `requestAccessToEntityTpe:completion:` method of the `EKEventStore` object is called in a block. You can pass the following two constants to the `requestAccessToEntityType:` method:

- **`EKEntityTypeReminder`** **–** This constant asks for authorization to access the reminders in the Calendar database.

- **`EKEntityTypeEvent`** – This constant asks for authorization to access the events in the Calendar database.

Before you interact with the `EKEventStore`, you need to check if the user has indeed granted your application access to the Calendar database. You can perform a simple check by calling the `authorizationStatusForEntityType:` method of the `EKEventStore` class, as shown below:

```
EKAuthorizationStatus status = [EKEventStore
            authorizationStatusForEntityType:EKEntityTypeEvent];
if (status == EKAuthorizationStatusDenied ||
      status == EKAuthorizationStatusRestricted) {

      return;
}
```

The `authorizationStatusForEntityType:` method returns one of the following four constants:

- `EKAuthorizationStatusNotDetermined`
- `EKAuthorizationStatusRestricted`
- `EKAuthorizationStatusDenied`
- `EKAuthorizationStatusAuthorized`

You can set access permission to the Calendar database in **Swift** using the following code:

```
let eventStore = EKEventStore()
```

```
        switch
EKEventStore.authorizationStatusForEntityType(EKEntityTypeEvent) {

        case .Authorized:
            readEventsFromCalendar(eventStore)

        case .Denied:
            displayAccessDenied()

        case .NotDetermined:
            eventStore.requestAccessToEntityType(EKEntityTypeEvent,
completion:
                {[weak self](granted: Bool, error: NSError!) -> Void in

                    if granted {
                        self!.insertEventIntoStore(eventStore)

                    } else {
                        self!.displayAccessDenied()
                    }
            })
        case .Restricted:
            displayAccessRestricted()

        }
    }
```

Quick Tip

A user can always change the access permission to an application by following the **Settings** ➪ Privacy ➪ **Calendars** path or the **Settings** ➪ Privacy ➪ Reminders path on the device.

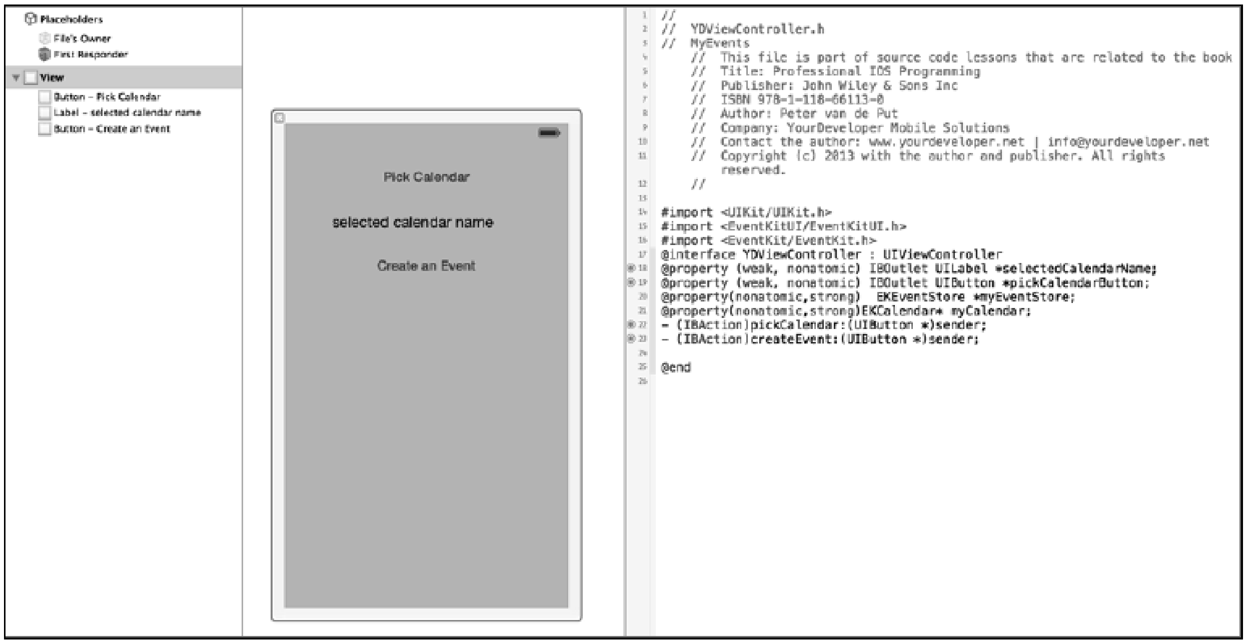
<H2> Accessing a Calendar

Source: [Professional iOS Programming][Chapter 13][Page No. 388]

Once your application has been authorized to access the user's Calendar database, you can start creating and editing events. It is important to realize that the Calendar database can contain **multiple calendars**. A user can decide to create a **Personal calendar** for personal events and a **Work calendar** for all work-related events.

The first step, therefore, is to either choose the **default calendar** or ask the user explicitly to allow your application to select the calendar with which you want to work.

Open the **YDViewController.xib** file using Interface Builder and the Assistant Editor to create a simple user interface with a `UIButton` and a `UILabel`, as shown in **Figure 2**.



```
//
//  YDViewController.h
//  MyEvents
    //  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>
#import <EventKitUI/EventKitUI.h>
#import <EventKit/EventKit.h>
@interface YDViewController : UIViewController
@property (weak, nonatomic) IBOutlet UILabel *selectedCalendarName;
@property (weak, nonatomic) IBOutlet UIButton *pickCalendarButton;
@property(nonatomic,strong)  EKEventStore *myEventStore;
@property(nonatomic,strong)EKCalendar* myCalendar;
- (IBAction)pickCalendar:(UIButton *)sender;
- (IBAction)createEvent:(UIButton *)sender;

@end
```

Figure 2: User Interface with a `UIButton` and a `UILabel`

Open the `YDViewController.h` file and create a strong property of type `EKEventStore` named `myEventstore`. To hold the selected calendar chosen by the user, create a strong property of type `EKCalendar` named `myCalendar` as shown below:

```
#import <UIKit/UIKit.h>
#import <EventKitUI/EventKitUI.h>
#import <EventKit/EventKit.h>
@interface YDViewController : UIViewController
@property (weak, nonatomic) IBOutlet UILabel *selectedCalendarName;
@property (weak, nonatomic) IBOutlet UIButton *pickCalendarButton;
@property(nonatomic,strong) EKEventStore *myEventStore;
@property(nonatomic,strong)EKCalendar* myCalendar;
- (IBAction)pickCalendar:(UIButton *)sender;
@end
```

Now open the `YDViewController.m` file and subscribe to the `EKCalendarChooserDelegate` protocol. Next in the `viewDidLoad:` method, call the `requestToAccessCalendar` method to ask for authorization.

Implement the `pickCalendar:` method as shown in the following code:

```
- (IBAction)pickCalendar:(UIButton *)sender
{
      EKAuthorizationStatus status = [EKEventStore
            authorizationStatusForEntityType:EKEntityTypeEvent];
      if (status == EKAuthorizationStatusDenied ||
            status == EKAuthorizationStatusRestricted) {

            return;
      }
      EKCalendarChooser *calendarChooser = [[EKCalendarChooser alloc]
            initWithSelectionStyle:EKCalendarChooserSelectionStyleSingle
            displayStyle:EKCalendarChooserDisplayAllCalendars
            eventStore:self.myEventStore];
      //if you don't set the selectedCalendars to an initialized NSSet it
      always returns nil

      calendarChooser.selectedCalendars = [[NSSet alloc] init];
      calendarChooser.showsDoneButton = YES;
      calendarChooser.showsCancelButton = YES;
      calendarChooser.delegate = self;

      UINavigationController* newNavController = [[UINavigationController
      alloc] initWithRootViewController:calendarChooser];
      [self presentViewController:newNavController animated:YES
      completion:nil];
}
```

The above method starts by checking whether the application has been authorized by the user to access the event store. Notice that the `authorizationStatusForEntityType:` method of the `EKEventStore` class is a static method, so call `[EKEventStore authorizationStatusForEntityType:EKEntityTypeEvent]` instead of `[self.myEventStoreauthorizationStatusForEntityType:EKEntityTypeEvent]`.

Next, create and initialize an instance of the `EKCalendarChooser` object named `calendarChooser` and set the properties. Finally, create a `UINavigationController`, which you use to present the `calendarChooser`.

Because you have subscribed to the `EKCalendarChooserDelegate` protocol, you need to implement the following three delegate methods both in **Swift** and **Objective-C**:

- ➢ `calendarChooserSelectionDidChange:`
- ➢ `calendarChooserDidFinish:`
- ➢ `calendarChooserDidCancel:`

You can implement these delegates in **Swift** with the `func` keyword.

The `calendarChooserDidFinish:` and `calendarChooserDidCancel:` methods are called if the user selects the Done or Cancel button. Whether they appear or not depends on the value of the `showsDoneButton` and `showsCancelButton` properties of the `calendarChooser` instance. The **Done** and **Cancel** buttons only show if the respective properties are set to `true` and the selection style has been set to `EKCalendarChooserSelectionStyleMultiple`. If you set the selection style to `EKCalendarChooserSelectionStyleSingle`, tapping a calendar from the presented list automatically calls the `calendarChooserSelectionDidChange:` method of the `EKCalendarChooser` class.

<H2> Creating and Editing a Calendar Event

--

Source: [Professional iOS Programming][Chapter 13][Page No. 390]

To create or edit a calendar event, you use the `EKEventEditViewController` object.

Open the `YDviewController.xib` file using Interface Builder and the Assistant Editor.

Add a new `UIButton` to the user interface with the Label **Create an Event** and an `IBAction createEvent:` (see **Figure 2**).

Now open the `YDViewController.m` file, subscribe to the `EKEventEditViewDelegate` protocol, and implement the `createEvent:` method.

Create and initialize an instance of `EKEventEditViewController` named `addEventController`. Set the `addEventController's eventStore` property to `myEventStore` and the `editViewDelegate` property to `self`.

Present the `addEventController` controller. If you want to edit an existing event, you need to pass an instance of the event to the event property, otherwise set the event property to nil.

The `createEvent:` method is shown as follows:

```
- (IBAction)createEvent:(UIButton *)sender
{
      EKEventEditViewController *addEventController =
            [[EKEventEditViewController alloc]
            initWithNibName:nil bundle:nil];
      // set the addController's event store to the current event store.
      addEventController.eventStore = self.myEventStore;
      addEventController.editViewDelegate = self;
      //If you have an existing event named myEvent and want to edit it pass
      it to the event property
      //addEventController.event = myEvent;
      // present EventsAddViewController as a modal view controller
      [self presentViewController:addEventController animated:YES
      completion:nil];
}
```

The `EKEventEditViewController` delegate protocol has one required delegate method named `eventEditViewController:didCompleteWithAction:` and one optional method named `eventEditViewControllerDefaultCalendarForNewEvents:`.

You need to implement the mandatory `eventEditViewController:didCompleteWithAction:` delegate method. This method returns one of the following `EKEventEditViewAction` constant values to inform you if the event edit has been canceled, deleted, or saved by the user:

- `EKEventEditViewActionCanceled`
- `EKEventEditViewActionSaved`
- `EKEventEditViewActionDeleted`

You are responsible for dismissing the presented `ViewController` by calling `[self dismissViewControllerAnimated:YES completion:nil]`.

For creating events in **Swift**, you can use the following code:

```
func AddEvent(title: String, startDate: NSDate, endDate: NSDate, inCalendar:
EKCalendar, inEventStore: EKEventStore, notes: String) -> Bool{

            if inCalendar.allowsContentModifications == false
            {
                println("Calendar does not allow modification")
                return false
            }
            //event created
            var EventSet = EKEvent(eventStore: inEventStore)
            EventSet.calendar = inCalendar
            EventSet.title = title      //set title
            EventSet.notes = notes      //set note
            EventSet.startDate = startDate  //set start date
            EventSet.endDate = endDate      //set end date

            var error:NSError?
            // save event into the calendar
            let EventSetResult = inEventStore.saveEvent(EventSet, span:
EKSpanThisEvent, error: &error)
```

```
                if EventSetResult == false
                {
                    if let CalError = error
                    {
                        println("An error occurred \(CalError)")
                 }
                    }

            return EventSetResult
    }
```

Technical Stuff

If your application requires access to both the reminders and the events, you have to call `requestAccessToEntityType:completion:` twice, once for each entity type.

Big Picture

You can manage the Calendar database by creating your own calendar. You can create events and reminders and work according to them to easily manage day-to-day tasks. The calendar enables you to get data of the previous day events also, by accessing the history of events.

An advanced feature of the calendar is that it enables you to connect your contact list with the calendar.

Additional Knowhow

Some additional class references used in event programming are `EKRecurrenceEnd`, `EKStructuredLocations`, `EKParticipant`, `EKRecurrenceRule`. These are the classes of Event Kit, which are used for manipulating calendar events and reminders.

<H1> Programmatically Accessing the Calendar Database

# Professional iOS Programming, Chapter 13 and Page No. 390#

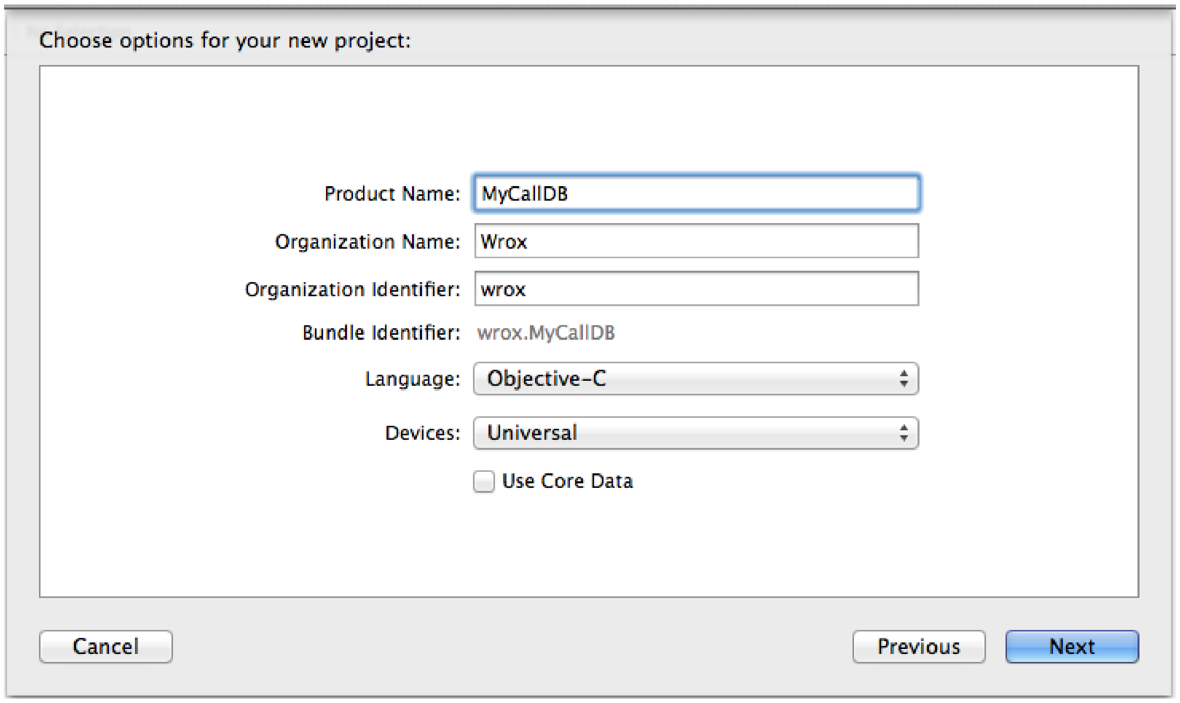
In the previous section, you learned about the basics of interacting with the Calendar database using the Event Kit UI framework. In this section, you will learn how to programmatically access the Calendar database for the following purposes:

- Create an event
- Edit an event
- Delete an event
- Stay synchronized

<H2>Creating an Event

Source: [Professional iOS Programming][Chapter 13][Page No. 392]

Start **Xcode** and create a new project using the **Single View Application** template, and name it **MyCalDB** using the options shown in **Figure 3**.



Choose options for your new project:

Product Name: MyCallDB

Organization Name: Wrox

Organization Identifier: wrox

Bundle Identifier: wrox.MyCallDB

Language: Objective-C

Devices: Universal

Use Core Data: unchecked

Figure 3: Create a New Project, MyCalDB, in Xcode

Add the Event Kit and the Event Kit UI frameworks to your project.

Open your `YDAppDelegate.h` file and create a strong property of type `UINavigationController` named `navController`.

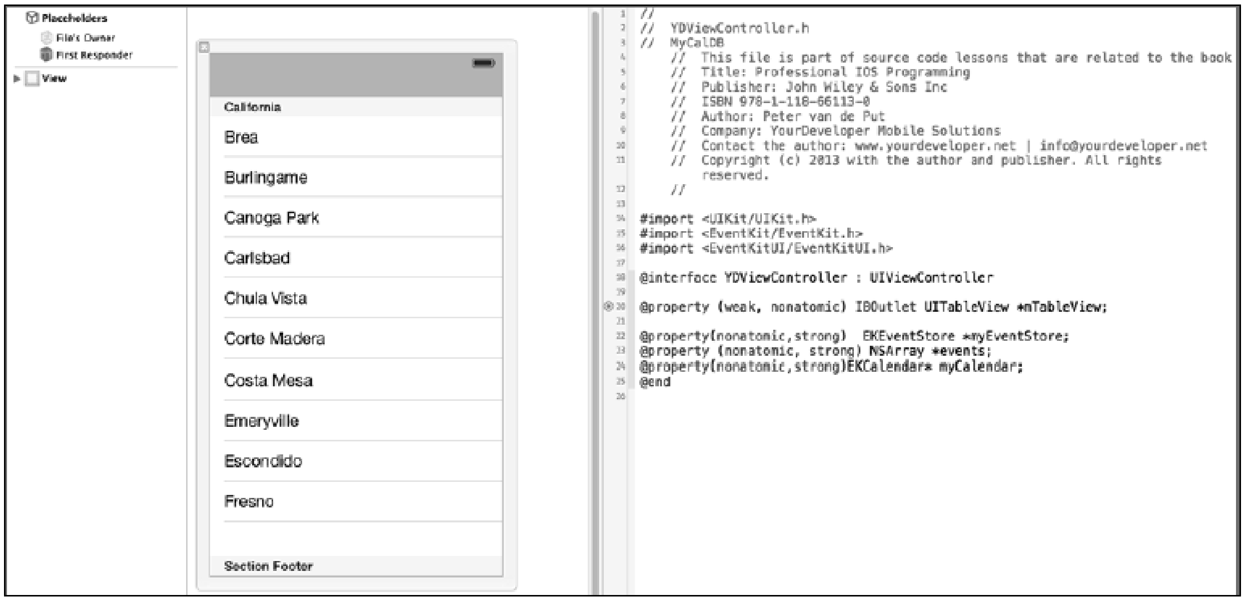
Open your `YDAppDelegate.m` file and change the `application:didFinishWithLaunchingOptions:` method, as shown in the following code:

```
      - (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 **Swift** format for the `didFinishLaunchingWithOption` method is as follows:

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

        window = UIWindow(frame: UIScreen.mainScreen().bounds)
            if let Display = window
            {
                    Display.backgroundColor = UIColor.whiteColor()
                    Display.rootViewController = ViewController()
                    Display.makeKeyAndVisible()
            }
    return true
            }
```

Now open the `YDViewController.xib` file using Interface Builder and the Assistant Editor and set up a user interface with a `UITableView` as shown in **Figure 4**.



```
//
//  YDViewController.h
//  MyCalDB
    //  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>
#import <EventKit/EventKit.h>
#import <EventKitUI/EventKitUI.h>

@interface YDViewController : UIViewController

@property (weak, nonatomic) IBOutlet UITableView *mTableView;

@property(nonatomic,strong)  EKEventStore *myEventStore;
@property (nonatomic, strong) NSArray *events;
@property(nonatomic,strong)EKCalendar* myCalendar;
@end
```

Figure 4: Set a User Interface with a `UITableView`

Open the YDViewController.h file and import the EventKit and the EventKitUI header files. Create strong properties as shown in the following code:

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

@interface YDViewController : UIViewController

@property (weak, nonatomic) IBOutlet UITableView *mTableView;

@property(nonatomic,strong) EKEventStore *myEventStore;
@property (nonatomic, strong) NSArray *events;
@property(nonatomic,strong)EKCalendar* myCalendar;
@end
```

Now open the YDViewController.m file and implement the requestAccessCalendar method as used in the previous section. Create a loadEvents method that uses an NSPredicate to query the myEventStore for events between now and 24 hours from now.

Implement the addEvent: method, which programmatically creates an event in the default calendar for myEventStore. Create and initialize an EKEvent instance named newEvent and set the properties for the title, notes, calendar, start date, and end date, and call the saveEvent:span:commit: method of the myEventStore instance.

The complete implementation is shown below:

```
#import "YDViewController.h"

@interface YDViewController ()<UITableViewDataSource,UITableViewDelegate>

@end

@implementation YDViewController

- (void)viewDidLoad
{
      [super viewDidLoad];
      [self requestAccessToCalendar];
      //Create an Add Event button
      UIBarButtonItem *addEventButtonItem =
            [[UIBarButtonItem alloc] initWithBarButtonSystemItem:
            UIBarButtonSystemItemAdd
            target:self
            action:@selector(addEvent:)];
      self.navigationItem.rightBarButtonItem = addEventButtonItem;
}
-(void)requestAccessToCalendar
{
      self.myEventStore = [[EKEventStore alloc] init];
      __block BOOL accessGranted = NO;
      [self.myEventStore requestAccessToEntityType:EKEntityTypeEvent
            completion:^(BOOL granted,
                        NSError *error) {
            // handle access here
            accessGranted = granted;
            dispatch_async(dispatch_get_main_queue(),^{
                  accessGranted=YES;
                  //call loadEvents here now the user has granted access
                  [self loadEvents];
            });
      }];
}
-(void)loadEvents
{
      EKAuthorizationStatus status = [EKEventStore
            authorizationStatusForEntityType:EKEntityTypeEvent];
      if (status == EKAuthorizationStatusDenied ||
      status == EKAuthorizationStatusRestricted) {

      return;
      }
      if (self.events)
            self.events=nil;

      self.myCalendar = [self.myEventStore defaultCalendarForNewEvents];

NSDate *startDate = [NSDate date];
NSDate *endDate = [NSDate dateWithTimeIntervalSinceNow:86400];
// Create the predicate. Pass it the default calendar.
```

```
NSArray *calendarArray = [NSArray arrayWithObject:self.myCalendar];
NSPredicate *predicate = [self.myEventStore
                        predicateForEventsWithStartDate:startDate
                        endDate:endDate
                        calendars:calendarArray];

// Fetch all events that match the predicate and store in self.events
self.events = [[NSArray alloc] initWithArray:
            [self.myEventStore eventsMatchingPredicate:predicate]];
[self.mTableView reloadData];
}
- (void)addEvent:(id)sender
{
      //Add an event without UI
      EKEvent *newEvent = [EKEvent eventWithEventStore:self.myEventStore];
      newEvent.calendar = self.myCalendar;
      newEvent.title = @"Finalize chapter 13";
      newEvent.notes = @"Finish module 10 of the course iOS programming";
      newEvent.startDate = [NSDate date];
      newEvent.endDate =[[NSDate date]
                        initWithTimeInterval:600
                        sinceDate:newEvent.startDate];
      NSError *err=nil;
      [self.myEventStore saveEvent:newEvent span:EKSpanThisEvent
            commit:YES error:&err];
      if (err)
      {
            //Handle errors here
      }
      [self loadEvents];
}
#pragma mark Table View

- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
      return [self.events count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
      static NSString *CellIdentifier = @"Cell";
      UITableViewCellAccessoryType editableCellAccessoryType
            =UITableViewCellAccessoryDisclosureIndicator;
      UITableViewCell *cell = [tableView
      dequeueReusableCellWithIdentifier:CellIdentifier];
      if (cell == nil) {
            cell             =            [[UITableViewCell            alloc]
            initWithStyle:UITableViewCellStyleDefault
            reuseIdentifier:CellIdentifier];
      }
      cell.accessoryType = editableCellAccessoryType;
      // Get the event at the row selected and display its title
      cell.textLabel.text    =    [[self.events    objectAtIndex:indexPath.row]
      title];
      return cell;
}
-(void)viewDidAppear:(BOOL)animated
{
      [self loadEvents];
}
#pragma mark UITableView delegates


- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:
(NSIndexPath *)indexPath {
      // Upon selecting an event, create an EKEventViewController to display
      the event.
      EKEventViewController* editController =
      [[EKEventViewController alloc] initWithNibName:nil bundle:nil];
      editController.event = [self.events objectAtIndex:indexPath.row];
      // Allow event editing.
      editController.allowsEditing = YES;
      [self.navigationController             pushViewController:editController
      animated:YES];
}
- (void)didReceiveMemoryWarning
{
      [super didReceiveMemoryWarning];
      // Dispose of any resources that can be recreated.
}

@end
```

<H2> Editing an Event

Source: [Professional iOS Programming][Chapter 13][Page No. 396]

Using the `EKEventviewController` to edit an event is the easy way to use the user control elements of the Event Kit UI framework. If you want to edit an existing event, simply change the properties of the event and call the `saveEvent:span:commit:` method of the `myEventStore`.

A sample method implementation is shown below:

```
- (void) editEvent : (EKEvent* ) theEvent
{
     theEvent.calendar = self.myCalendar;
     reminder.calendar          =          [self.          myEventStore
defaultCalendarForNewReminders];
     NSError *err;
     [self.myEventStore saveReminder:reminder commit: YES error:&err];
     if (err)
            {
          //Handle errors here
             }
         [self loadReminders];
}
```

<H2> Deleting an Event

Source: [Professional iOS Programming][Chapter 13][Page No. 397]

You can delete an event by simply passing the event to the `removeEvent:span:commit:error:` method of the `myEventStore` instance.

<H2> Staying Synchronized

Source: [Professional iOS Programming][Chapter 13][Page No. 397]

Other applications can make changes to the Calendar database when running in the background. To stay synchronized, the Event Kit framework sends an `NSNotification` each time a change is applied to the Calendar database.

To stay synchronized, you can set up an observer to receive the notifications broadcasted by the event store using the following code in your `viewDidLoad` method:

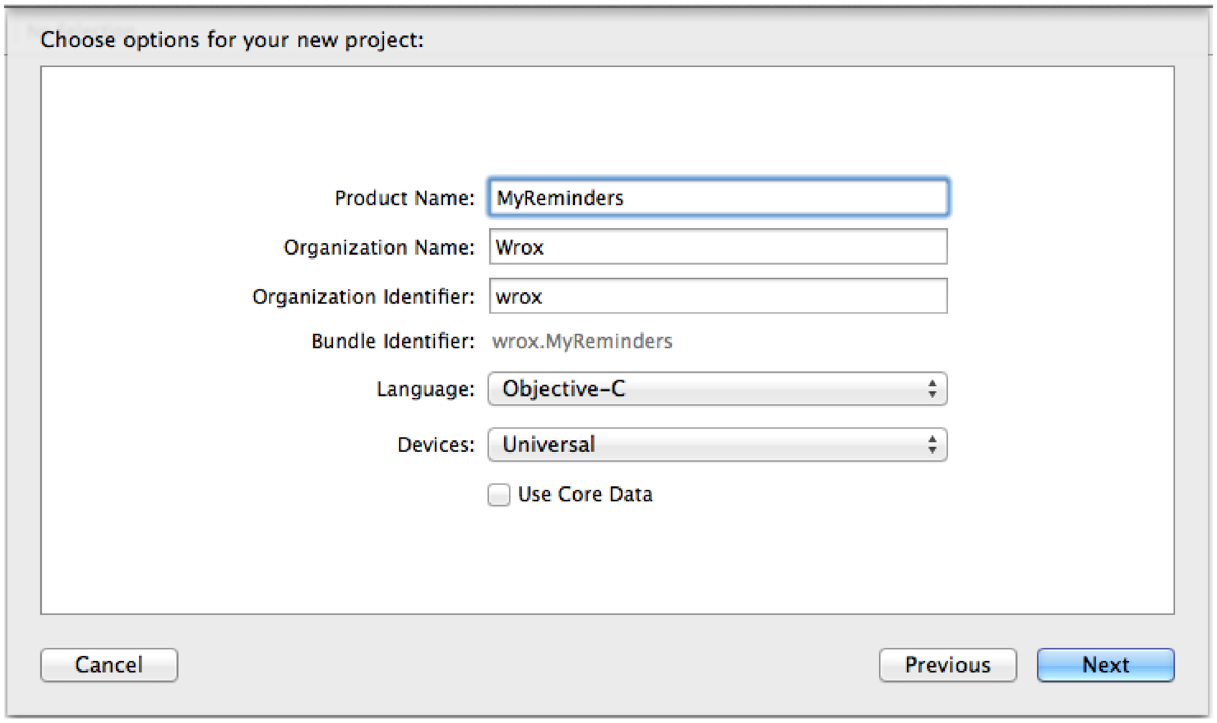
```
[[NSNotificationCenter defaultCenter] addObserver:self
                                          selector:@selector(storeChanged:)
                                          name:EKEventStoreChangedNotificatio
                                          n
                                          object:self.myEventStore];
```

<H1>Working with Reminders

# Professional iOS Programming, Chapter 13 and Page No. 397#

Working with reminders is very similar to working with events, because they both use the Calendar database to store their information. This means that you can reuse a lot of the code you have already written, such as for requesting authorization to access the Calendar database and querying the event store.

Start **Xcode** and create a new project using the **Single View Application** template, and name it **MyReminders** using the options shown in **Figure 5**.


Choose options for your new project:

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

Figure 5: Create a New Project, MyReminders, in Xcode

Add `EKEventEditViewController` in the `ViewController.h` file. The following code shows how the `EKEventEditViewController` looks like:

```
-  (IBAction)createEvent:(UIButton *)sender
   {
         EKEventEditViewController *addEventController =
               [[EKEventEditViewController alloc]
                   initWithNibName:nil bundle:nil];

         //set the addController’s event store to the current event store.
         addEventController.eventStore = self.myEventStore;
         addEventController.editViewDelegate  = self;
         //If you have an existing event named myEvent and want to edit it
         pass it to the event property
         //addEventController.event = myEvent;
         //present EventsAddViewController as a modal view controller
         [self    presentViewController:addEventController    animated:YES
         completion:nil];
   }
```

Real Life Connect

Adding reminders and creating events is now a basic part of our daily activities and helps make our day-to-day plans. For example, you want to remember the birthdays and anniversaries of your friends and family. You can set reminders to receive notifications on these dates so that you can wish them.

<H2> Creating a Reminder

Source: [Professional iOS Programming][Chapter 13][Page No. 398-399]

Create a reminder by creating and initializing an instance of the `EKReminder` object, named **reminder**, by calling the static `reminderWithEventStore:` method.

Set the different properties on your reminder instance, such as the title, notes, start date, due date, or completion date. To save the reminder, call the `saveReminder:commit:error:` method of the `myEventStore` object. A sample `addReminder:` implementation is shown below:

```
- (void)addReminder:(id)sender
{
      EKReminder              *reminder              =              [EKReminder
      reminderWithEventStore:self.myEventStore];
      [reminder setTitle:@"Pick up 2 pizzas"];
      [reminder setNotes:@"Calzone's Pizza Cucina. 430 Columbus Ave, San
                        Francisco"];
      reminder.calendar = [self.myEventStore defaultCalendarForNewReminders];
      NSError *err;
      [self.myEventStore saveReminder:reminder commit:YES error:&err];
      if (err)
            {
                  //Handle errors here
            }
      [self loadReminders];
}
```

The **Swift** format to create a reminder is as follows:

```
func   addReminder(title:   String,   inCalendar:EKCalendar,   inEventStore:
EKEventStore, notes:String) -> Bool {

    var reminder = EKReminder(eventStore: inEventStore)
    reminder.calendar = inCalendar
    reminder.title = Pick up 2 pizzas
    reminder.notes= Calzone's Pizza Cucina. 430 Columbus Ave, San Francisco0
    var error:NSError?

    let result = inEventStore.saveReminder(reminder,commit:true,error:&error)

        if result == false
        {
            //Handle error
        }
}
```

<H2> Editing a Reminder

Source: [Professional iOS Programming][Chapter 13][Page No. 399]

To edit an existing reminder, you simply make changes to the properties of your `EKReminder` object, call the `saveReminder:commit:error:` method of the `myEventStore` object, and the changes are saved.

<H2> Deleting a Reminder

Source: [Professional iOS Programming][Chapter 13][Page No. 399]

Deleing a reminder is very similar to deleting an event. Pass the reminder instance you want to the delete to the `removeReminder:commit:error` method of the `myEventStore` instance.

<H2> Working with Alarms

Source: [Professional iOS Programming][Chapter 13][Page No. 399]

It is possible to add an alarm to a reminder by creating and initializing an instance of the `EKAlarm` object. You can use the `EKAlarm` object in several ways—you can set an absolute date or you can set an `EKStructuredLocation` object.

An `EKStructuredLocation` object is an object with three properties named title, geoLocation, and radius. The `EKStructuredLocation` represents your home address, work address, or the address of the grocery store. If you set an `EKStructuredLocation` object for the `EKAlarm` instance, the alarm will fire if you enter or leave the `EKStructuredLocation` related to the alarm.

A practical implementation of an alarm with geofencing technology would be in an application where you set a reminder to pick up some pizzas before coming home from work. Once you leave your office, the reminder will notify you using the geographical information that you have left the office. The `addReminder:` method is shown below:

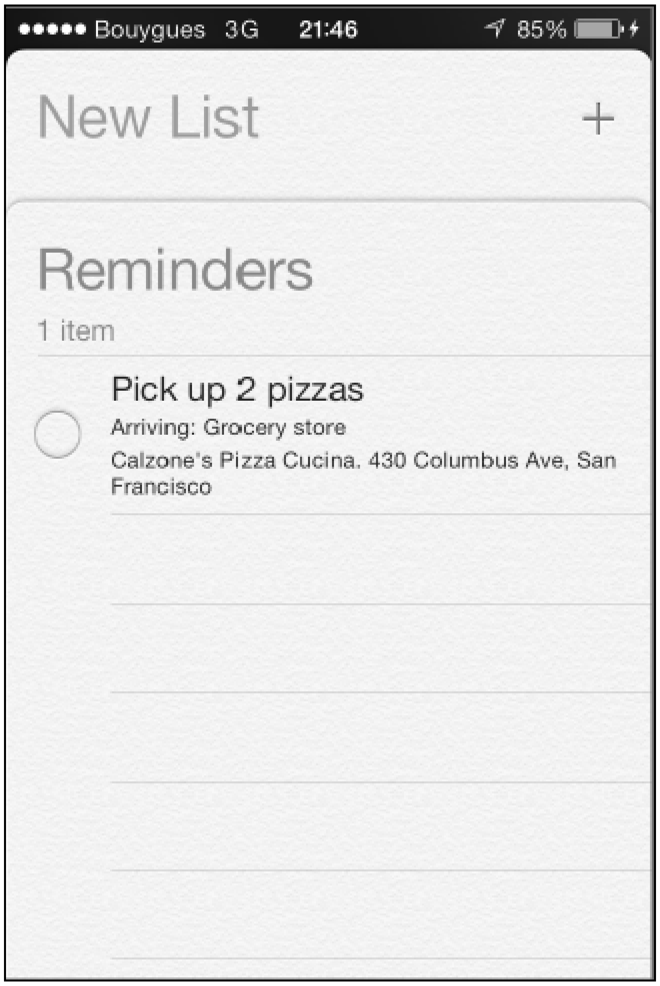
```
- (void)addReminder:(id)sender
{
     EKReminder           *reminder           =           [EKReminder
     reminderWithEventStore:self.myEventStore];
     [reminder setTitle:@"Pick up 2 pizzas"];
     [reminder setNotes:@"Calzone's Pizza Cucina. 430 Columbus Ave, San
                         Francisco"];
          //create the geofence alarm
     EKAlarm *enterAlarm = [[EKAlarm alloc] init];
     [enterAlarm setProximity:EKAlarmProximityEnter];
     EKStructuredLocation *enterLocation =
          [EKStructuredLocation locationWithTitle:@"Grocery store"];
     CLLocationDegrees lat = 37.799052;
     CLLocationDegrees lng = -122.408187;
     CLLocation *shopLocation = [[CLLocation alloc]
```

```
                initWithLatitude:lat longitude:lng];
        [enterLocation setGeoLocation:shopLocation];
        //set the radius in meters
        [enterLocation setRadius:200];
        reminder.calendar = [self.myEventStore defaultCalendarForNewReminders];
        [enterAlarm setStructuredLocation:enterLocation];
        [reminder addAlarm:enterAlarm];
        [reminder                                 setCalendar:[self.myEventStore
        defaultCalendarForNewReminders]];


        NSError *err;
        [self.myEventStore saveReminder:reminder commit:YES error:&err];
        if (err)
              {
                    //Handle errors here
              }
}
```

In **Swift**, you can set the alarm format as follows:

```
func enterAlarm(store: EKEventStore, cal: EKCalendar){

    let startDate = NSDate(timeIntervalSinceNow: 60.0)   // start alarm from
now

    let endDate = startDate.dateByAddingTimeInterval(20.0)

    let enterAlarm = EKEvent(eventStore: store)
    enterAlarm.calendar = cal
    enterAlarm.startDate = startDate
    enterAlarm.endDate = endDate

    let alarm = EKAlarm(relativeOffset: -5.0)     //End Alarm 5 sec before
event
```

When you create a reminder with an alarm, it shows as in **Figure 6**. When you create a reminder with a geofence, it shows like **Figure 7**.



Reminders

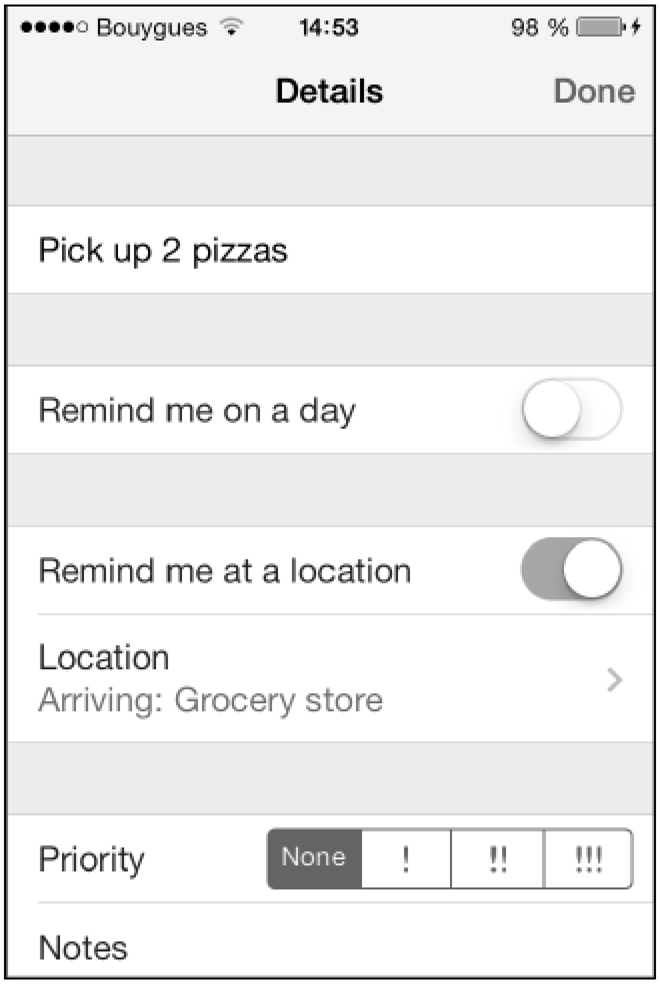
1 item

Pick up 2 pizzas

Arriving: Grocery store

Calzone's Pizza Cucina. 430 Columbus Ave, San Francisco

Figure 6: Reminder with an Alarm



●●●●○ Bouygues 14:53 98 %

Details Done

Pick up 2 pizzas

Remind me on a day

Remind me at a location

Location
Arriving: Grocery store

Priority [✓ None] ! !! !!!

Notes

Figure 7: Reminder with a Geofence

Quick Tip

When your use a notification in your application, make sure that the notifications are enabled in the phone settings. Even if you do not change settings in the phone, a popup message on the screen will ask your permission to do so when such an application runs. If you deny the permission, the application will not show notifications.

Additional Knowhow

A **geofence** is a virtual perimeter set around a global positioning system (GPS) coordinate. Geofencing in combination with reminders allows you to be notified when you enter or leave a virtual perimeter. For example, if you set a reminder with a geofence at your office's GPS location, you will be notified when you enter or leave the office.

Lab Connect

During the lab hour of this session, you will create calendars and reminders using the Event Kit framework.

Cheat Sheet

- An application needs to access the Calendar database to interact with events and reminders.
- The Event Kit UI framework is used for interacting with user interface elements.
- To set and modify calendar events and reminders, you can use the following class references:
    - `EKRecurrenceEnd`
    - `EKStructuredLocations`
    - `EKParticipant`
    - `EKRecurrenceRule`
- The `EKEntityTypeReminder` is a constant that asks for authorization to access the reminders in the Calendar database.
- The `EKEntityTypeEvent` is a constant that asks for authorization to access the events in the Calendar.
- The `EKCalendarChooser` is an interface element that enables a user to select one or more calendars.
- The `EKEventViewController` is an interface element that enables a user to create or edit an existing event.