Event Kit Programming Guide

Developer

Contents

Listings

Introduction

The Event Kit and Event Kit UI frameworks together allow iOS applications to access event information from a user's Calendar database. You can fetch events based on a date range or a unique identifier, receive notifications when event records change, and allow users to create and edit events for any of their calendars. Changes made to events in a user's Calendar database with Event Kit are automatically synced with the appropriate calendar (CalDAV, Exchange, and so on). This document describes Event Kit concepts and common programming tasks.

Who Should Read This Document?

You should read this document if you want to display calendar event data or allow users to edit their calendar event data in your iOS application. Event Kit provides limited access to a user's calendar information. It is not suitable for implementing a full-featured calendar application.

Organization of This Document

This document contains the following information:

- "Fetching Events" (page 6) explains how to fetch events from the Calendar database.
- "Using Event View Controllers" (page 9) explains how to display event view controllers to allow your users to create and edit events.
- "Creating and Editing Events Programmatically" (page 12) explains how to create and edit events programmatically.
- "Creating Recurring Events" (page 15) explains how to make an event a recurring event.
- "Observing Event Changes" (page 17) explains how to register for notifications about external changes to the Calendar database.

See Also

For an in-depth description of the Event Kit and Event Kit UI API, read:

- *Event Kit Framework Reference* provides an in-depth description of the Event Kit API.

- *Event Kit UI Framework Reference* provides an in-depth description of the Event Kit UI API.
- The *SimpleEKDemo* sample code provides a basic example of using the Event Kit and Event Kit UI frameworks to access and edit calendar data.

Fetching Events

You can fetch events from a user's Calendar database using the `EKEventStore` class. You can fetch a custom set of events that match a predicate you provide, or you can fetch an individual event by its unique identifier. After you fetch an event, you can access its associated calendar information with the properties of the `EKEvent` class.

Initializing an Event Store

You initialize an `EKEventStore` object with the default initializer:

```
EKEventStore *store = [[EKEventStore alloc] init];
```

An `EKEventStore` object requires a relatively large amount of time to initialize and release. Consequently, you should not initialize and release a separate event store for each event-related task. Instead, initialize a single event store when your app loads and use it repeatedly.

Fetching Events with a Predicate

It's common to fetch events that fall within a date range. The event store method `eventsMatchingPredicate:` fetches all events that fall within the date range specified in the predicate you provide. You must create the predicate for the `eventsMatchingPredicate:` method with the `EKEventStore` method `predicateForEventsWithStartDate:endDate:calendars:`. Listing 1-1 demonstrates fetching all events that occur between 30 days before the current date and 15 days after the current date.

Listing 1-1 Fetching events with a predicate

```
// Create the predicate's start and end dates.
CFGregorianDate gregorianStartDate, gregorianEndDate;
CFGregorianUnits startUnits = {0, 0, -30, 0, 0, 0};
CFGregorianUnits endUnits = {0, 0, 15, 0, 0, 0};
CFTimeZoneRef timeZone = CFTimeZoneCopySystem();
```

```
gregorianStartDate = CFAbsoluteTimeGetGregorianDate(
    CFAbsoluteTimeAddGregorianUnits(CFAbsoluteTimeGetCurrent(), timeZone,
startUnits),
    timeZone);
gregorianStartDate.hour = 0;
gregorianStartDate.minute = 0;
gregorianStartDate.second = 0;

gregorianEndDate = CFAbsoluteTimeGetGregorianDate(
   CFAbsoluteTimeAddGregorianUnits(CFAbsoluteTimeGetCurrent(), timeZone, endUnits),
    timeZone);
gregorianEndDate.hour = 0;
gregorianEndDate.minute = 0;
gregorianEndDate.second = 0;

NSDate* startDate =
    [NSDate
dateWithTimeIntervalSinceReferenceDate:CFGregorianDateGetAbsoluteTime(gregorianStartDate,
 timeZone)];
NSDate* endDate =
    [NSDate
dateWithTimeIntervalSinceReferenceDate:CFGregorianDateGetAbsoluteTime(gregorianEndDate,
 timeZone)];

CFRelease(timeZone);

// Create the predicate.
NSPredicate *predicate = [eventStore predicateForEventsWithStartDate:startDate
endDate:endDate calendars:nil]; // eventStore is an instance variable.

// Fetch all events that match the predicate.
NSArray *events = [eventStore eventsMatchingPredicate:predicate];
[self setEvents:events];
```

You can specify a subset of calendars to search by passing an array of `EKCalendar` objects as the `calendars` parameter of the `predicateForEventsWithStartDate:endDate:calendars:` method. You can get the user's calendars from the event store's `calendars` property. Passing `nil` tells the method to fetch from all of the user's calendars.

Because the `eventsMatchingPredicate:` method is synchronous, you may not want to run it on your application's main thread. For asynchronous behavior, run the method on another thread with the `dispatch_async` function or with an `NSOperation` object.

Fetching Individual Events with an Identifier

If you want to fetch an individual event and you know the event's unique identifier from fetching it previously with a predicate, use the EKEventStore method `eventWithIdentifier:` to fetch the event. You can get an event's unique identifier with the `eventIdentifier` property.

Sorting Events by Start Date

Applications often want to display event data to the user that is sorted by start date. To sort an array of `EKEvent` objects by date, call `sortedArrayUsingSelector:` on the array, providing the selector for the `compareStartDateWithEvent:` method.

Using Event View Controllers

The Event Kit UI framework provides two types of view controllers for manipulating events:

- Use the `EKEventViewController` class if you have an existing event you want to display or allow the user to edit.
- Use the `EKEventEditViewController` class if you allow the user to create, edit, or delete events.

Displaying and Editing Events

You must have an existing event you obtain from an event store to use the `EKEventViewController` class. You need to set the `event` property and any other display options before presenting this type of view controller. Listing 2-1 shows how to create an event view controller and add it to a navigation controller assuming `myEvent` already exists. If you don't allow the user to edit the event, set the `allowsEditing` property to `NO`.

Listing 2-1 Editing an existing event

```
    EKEventViewController *eventViewController = [[EKEventViewController alloc]
init];
    eventViewController.event = myEvent;
    eventViewController.allowsEditing = YES;
    navigationController = [[UINavigationController alloc]
                                initWithRootViewController:eventViewController];
    [eventViewController release];
```

You need to assign a delegate to an event view controller to receive a notification when the user finishes viewing the event. The delegate conforms to the `EKEventViewDelegate` protocol and must implement the `eventViewController:didCompleteWithAction:` method.

Creating and Editing Events

To allow the user to create, edit, or delete events, use the `EKEventEditViewController` class and the `EKEventEditViewDelegate` protocol. You create an event edit view controller similar to an event view controller except that you must set the `eventStore` property and setting the `event` property is optional.

- If the event property is `nil` when you present the view controller, the user creates a new event in the default calendar and saves it to the specified event store.
- If you set the `event` property, the user edits an existing event. The event must reside in the specified event store or an exception is raised.

Instances of the `EKEventEditViewController` class are designed to be presented modally, as shown in Listing 2-2. In this code fragment, `self` is the top view controller of a navigation controller. For details on modal view controllers, read "Presenting a View Controller Modally" in *View Controller Programming Guide for iOS*.

Listing 2-2 Presenting an event edit view controller modally

```
    EKEventEditViewController* controller = [[EKEventEditViewController alloc]
init];
    controller.eventStore = myEventStore;
    controller.editViewDelegate = self;
    [self presentModalViewController: controller animated:YES];
    [controller release];
```

You must also specify a delegate to receive notification when the user finishes editing the event. The delegate conforms to the `EKEventEditViewDelegate` protocol and must implement the `eventEditViewController:didCompleteWithAction:` method to dismiss the modal view controller as shown in Listing 2-3. In general, the object that presents a view controller modally is responsible for dismissing it.

Listing 2-3 The delegate dismisses the modal view

```
- (void)eventEditViewController:(EKEventEditViewController *)controller
didCompleteWithAction:(EKEventEditViewAction)action {
    [self dismissModalViewControllerAnimated:YES];
}
```

The delegate is also passed the action that the user took when finishing the edit. The user can either cancel the changes, save the event, or delete the event. If you need to take further action, implement the `eventEditViewController:didCompleteWithAction:` delegate method.

Creating and Editing Events Programmatically

You can use the Event Kit framework to allow users to create new events and edit existing events in their Calendar database.

> **Note** The recommended method for allowing users to modify event data is with the event view controllers provided in the Event Kit UI framework. For information on how to use these event view controllers, see "Using Event View Controllers" (page 9). Use the techniques described in this chapter only if event view controllers are not appropriate for your application.

Always Notify the User

If your application modifies a user's Calendar database programmatically, *it must get confirmation from the user before doing so*. An application should never modify the Calendar database without specific instruction from the user.

Creating and Editing Events

Create a new event with the `eventWithEventStore:` method of the `EKEvent` class.

To edit the details of a new event or an event you have fetched from the Calendar database, set the corresponding properties of the event. The details you can edit include:

- The event's title
- The event's start and end dates
- The calendar the event is associated with

  You can get an array of the user's calendars with the event store property `calendars`.
- The event's recurrence rule, if it is a repeating event
- The alarms associated with the event

Adding and Removing Alarms

You can add an alarm to an event with the `addAlarm:` method. Alarms can be created with an absolute date or with an offset relative to the start date of the event. Alarms created with a relative offset must occur before or at the start date of the event. You can remove an alarm from an event with the `removeAlarm:` method.

Saving Events

Changes you make to an event are not permanent until you save them. Save your changes to the Calendar database with the `EKEventStore` method `saveEvent:span:error:`. Doing so automatically syncs your changes with the calendar the event belongs to (CalDAV, Exchange, and so on).

If you are saving a recurring event, you can have your changes apply to all occurrences of the event by specifying `EKSpanFutureEvents` for the `span` parameter of the `saveEvent:span:error:` method.

Removing Events

Permanently remove an event from the Calendar database with the `EKEventStore` method `removeEvent:span:error:`.

If you are removing a recurring event, you can remove all occurrences of the event by specifying `EKSpanFutureEvents` for the `span` parameter of the `removeEvent:span:error:` method.

Processing Events with a Predicate

You can perform an operation on all events that match a provided predicate with the `EKEventStore` method `enumerateEventsMatchingPredicate:usingBlock:`. You must create the predicate for this method with the `EKEventStore` method `predicateForEventsWithStartDate:endDate:calendars:`. The operation you provide is a block of type `EKEventSearchCallback`.

```
typedef void (^EKEventSearchCallback)(EKEvent *event, BOOL *stop);
```

The block is passed two parameters:

`event`
: This is the event that is currently being operated on.

`stop`

You can set the value of this parameter to `YES` to tell the `enumerateEventsMatchingPredicate:usingBlock:` method to stop processing events when this block returns. Any events that match the predicate but have not yet been processed remain unprocessed.

Keep in mind that using this method can result in significant changes to the user's Calendar database. Make sure the user is fully informed of the actions you are about to perform when you request user confirmation.

Because the `enumerateEventsMatchingPredicate:usingBlock:` method is synchronous, you may not want to run it on your application's main thread. For asynchronous behavior, run the method on another thread with the `dispatch_async` function or with an `NSOperation` object.

Creating Recurring Events

Recurring events are events that repeat daily, weekly, monthly, or yearly. The pattern that repeats can be complex. For example, you might schedule an event every Tuesday and Thursday of the first and second week of every month of the year.

You make an event a recurring event by assigning it a recurrence rule, which describes when the event occurs. Recurrence rules are represented by instances of the `EKRecurrenceRule` class.

Creating a Basic Recurrence Rule

You can create a recurrence rule with a simple daily, weekly, monthly, or yearly pattern with the `initRecurrenceWithFrequency:interval:end:` method. You provide three values to this method:

- **The recurrence frequency**. This is a value of type `EKRecurrenceFrequency` that indicates whether the recurrence rule is daily, weekly, monthly, or yearly.
- **The recurrence interval**. This is an integer greater than 0 that specifies how often a pattern repeats. For example, if the recurrence rule is a weekly recurrence rule and its interval is 1, then the pattern repeats every week. If the recurrence rule is a monthly recurrence rule and its interval is 3, then the pattern repeats every three months.
- **The recurrence end**. This optional parameter is an instance of the `EKRecurrenceEnd` class, which indicates when the recurrence rule ends. The recurrence end can be based on a specific end date or on a number of occurrences.

  If you don't want to specify an end for the recurrence rule, pass `nil`.

Creating a Complex Recurrence Rule

You can create a recurrence rule with a complex pattern with the `initRecurrenceWithFrequency:interval:daysOfTheWeek:daysOfTheMonth:monthsOfTheYear:weeksOfTheYear:daysOfTheYear:setPositions:end:` method. As you do for a basic recurrence rule, you provide a frequency, an interval, and an optional end for the recurrence. In addition, you can provide the following values:

- **Days of the week**. For all recurrence rules besides daily recurrence rules, you can provide an array of `EKRecurrenceDayOfWeek` objects that indicate the days of the week on which the event occurs.

For example, you can provide an array containing `EKRecurrenceDayOfWeek` objects with day of week values of `EKTuesday` and `EKFriday` to create a recurrence that occurs every Tuesday and Friday.

- **Days of the month**. For monthly recurrence rules only, you can provide an array of `NSNumber` objects that indicate the days of the month on which the event occurs. Values can be from 1 to 31, and from –1 to –31. Negative values indicate counting backward from the end of the month.

  For example, you can provide an array containing the values 1 and –1 to create a recurrence that occurs on the first and last day of every month.

- **Months of the year**. For yearly recurrence rules only, you can provide an array of `NSNumber` objects that indicate the months of the year in which the event occurs. Values can be from 1 to 12.

  For example, if your originating event occurs on January 10, you can provide an array containing the values 1 and 2 to create a recurrence that occurs every January 10 and February 10.

- **Weeks of the year**. For yearly recurrence rules only, you can provide an array of `NSNumber` objects that indicate the weeks of the year in which the event occurs. Values can be from 1 to 53, and from –1 to –53. Negative values indicate counting backward from the end of the year.

  For example, if your originating event occurs on a Wednesday, you can provide an array containing the values 1 and –1 to create a recurrence that occurs on the Wednesday of the first and last weeks of every year. If a specified week does not contain a Wednesday in the current year, as can be the case for the first or last week of a year, the event does not occur.

- **Days of the year**. For yearly recurrence rules only, you can provide an array of `NSNumber` objects that indicate the days of the year on which the event occurs. Values can be from 1 to 366, and from –1 to –366. Negative values indicate counting backward from the end of the year.

  For example, you can provide an array containing the values 1 and –1 to create a recurrence that occurs on the first and last day of every year.

- **Set positions**. For all recurrence rules besides daily recurrence rules, you can provide an array of `NSNumber` objects that filters which occurrences to include in the recurrence rule. This filter is applied to the set of occurrences determined from the other parameters you provide. Values can be from 1 to 366, and from –1 to –366. Negative values indicate counting backward from the end of the list of occurrences.

  For example, you can provide an array containing the values 1 and –1 to a yearly recurrence rule that has specified Monday through Friday as its value for days of the week, and the recurrence occurs only on the first and last weekday of every year.

You can provide values for any number of the above parameters (parameters that don't apply to a particular recurrence rule are ignored). If you provide a value for more than one of the above parameters, the recurrence occurs only on days that apply to all provided values.

Observing Event Changes

It's possible for a user's Calendar database to be modified by another process or application while your application is running. If your application fetches calendar events, you should register to be notified about changes to the Calendar database. By doing so, you ensure that the calendar information you display to the user is current.

Observing Notifications

An `EKEventStore` object posts an `EKEventStoreChangedNotification` notification whenever it detects changes to the Calendar database. Register for this notification if your application handles event data.

The following code registers for the `EKEventStoreChangedNotification` notification:

```
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(storeChanged:)
    name:EKEventStoreChangedNotification object:eventStore];
```

Responding to Notifications

When you receive an `EKEventStoreChangedNotification` notification, it's possible that changes have been made to `EKEvent` objects you fetched and retained. The effect of these changes depends on whether an event was added, modified, or deleted.

- If an event was added, it does not affect any of your retained events, but the added event may fall within the date range of events you are displaying to the user.
- If an event was modified or deleted, properties of `EKEvent` objects representing that event become out-of-date.

Because your local data is often invalidated or incomplete when a change occurs in the Calendar database, you should release and refetch your current date range of events whenever you receive an `EKEventStoreChangedNotification` notification. If you are currently modifying an event and you do not want to refetch it unless it is absolutely necessary to do so, you can call the `refresh` method on the event. If the method returns `YES`, you can continue to use the event; otherwise, you need to release and refetch it.

Events being modified in an event view controller are updated automatically when a change occurs in the Calendar database.

Document Revision History

This table describes the changes to *Event Kit Programming Guide*.

Date	Notes
2010-09-22	Made improvements throughout.
2010-08-03	Added a link to the SimpleEKDemo sample code.
2010-04-29	New document that explains how to access calendar data in iOS with the Event Kit framework.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010