Sample Code

Accessing Calendar using EventKit and EventKitUI

Choose and implement the appropriate Calendar access level in your app.

Download

iOS 16.4+ | iPadOS 16.4+ | Xcode 15.0+

Overview

Prior to iOS 17, your app needs to include the `NSCalendarsUsageDescription` key in its `Info.plist` and request authorization from the user before it can access the user's calendar data. `NSCalendarsUsageDescription` indicates how your app intends to use calendar data. If the user approves the request, the app gets full access to all events on all the user's calendars, including the ones the app didn't create. If the user denies the request, the app gets no access to the user's data.

Starting in iOS 17, your app should only request the specific level of access it requires to complete its calendar data tasks. The iOS 17 SDK introduces new calendar usage description strings, the ability to add events to Calendar without prompting the user for access, and a new write-only access. See `Accessing the event store` for details.

This sample consists of three targets that illustrate how to implement Calendar access level using EventKit and EventKitUI. The `DropInLessons` target builds an app that saves events to Calendar without prompting the user for authorization. The `RepeatingLessons` target, which implements the write-only access feature, builds an app that saves events directly to Calendar with user permission. The `MonthlyEvents` target, which illustrates the full-access feature, builds an app that fetches and displays all events occuring within a month in all the user's calendars.

Note

This sample code project is associated with WWDC23 session 10052: Discover Calendar and EventKit.

Configure the sample code project

Before you run the sample code project in Xcode:

- Open the sample with Xcode 15 or later.
- Select the top-level Calendar Access project.
- For the three targets, choose your team from the Team menu in the Signing & Capabilities pane to let Xcode automatically manage your provisioning profile.
- Select the target you wish to build, then build and run it in the Simulator, in Mac Catalyst, or on a device.

Save events without prompting the user for access

In iOS 17, your app can add events to Calendar without prompting the user for access using `EKEventEditViewController`. If the purpose of your app is to create, configure, and present calendar events in an editor UI, consider saving events to Calendar without prompting the user for authorization in your app following these steps:

- Build your app with Xcode 15 and link against the iOS 17 SDK.
- If your app includes `NSCalendarsUsageDescription`, remove this key.
- If your app requests permission using `requestAccess(to:completion:)` or `requestAccess(to:)`, remove these instance methods from your source code.

The `DropInLessons` app writes data to Calendar without performing any other operations on the user's events. Because its workflow doesn't interact with the user's calendar data, the app isn't required to include any calendar usage strings or prompt the user for access. `EKEventStore` allows apps to request permission from the user, and read and write data to Calendar. `DropInLessons` creates an instance of the event store, `store`.

```
@State private var store = EKEventStore()
```

When the user schedules a lesson, `DropInLessons` creates a `selectedEvent`, then presents an event edit view controller.

```
    .sheet(isPresented: $showEventEditViewController,
           onDismiss: didDismissEventEditController, content: {
        EventEditViewController(event: $selectedEvent, eventStore: store)
})
```

The app creates `selectedEvent` in the event store, adds it to the default calendar for the store, then configures `selectedEvent` with the selected lesson's details. The view controller takes `selectedEvent` and `store` as parameters.

```
let controller = EKEventEditViewController()
controller.eventStore = eventStore
controller.event = event
controller.editViewDelegate = context.coordinator
```

`DropInLessons` relinquishes control once the editor is presented. Because the event edit view controller renders its content out of process, it has full access to all the user's calendars on the device, regardless of the access granted to the app. This allows the user to get a full-featured editing experience, such as choosing another calendar to save the selected lesson or changing presented information in the editor. However, the app isn't aware of any of these changes. When the user taps the Add button in the UI, the system saves the lesson to the user's selected or default calendar, then dismisses the editor.

```
func eventEditViewController(_ controller: EKEventEditViewController, didCompleteWit
    parent.presentationMode.wrappedValue.dismiss()
}
```

Because the calendar edits happen out of process, inspecting the properties of the dismissed `controller`, such as `event`, to determine what the user added to Calendar doesn't return any useful information. The app isn't aware of the changes, which naturally means it can't see them.

Request write-only access

In iOS 17, an app with write-only access can create and save events to Calendar, display events using `EKEventEditViewController`, and allow the user to select another calendar using `EKCalendarChooser`. If your app needs to write data directly, consider implementing write-only access in your app following these steps:

- Build your app with Xcode 15 and link against the iOS 17 SDK.
- Add the `NSCalendarsWriteOnlyAccessUsageDescription` key to the `Info.plist` file of the target building your app.

- To request write-only access to events, use `requestWriteOnlyAccessToEvents(completion:)` or `requestWriteOnlyAccessToEvents()`.

> **Note**
>
> `EKEventEditViewController` and `EKCalendarChooser` require write-only or full access. `EKEventEditViewController` doesn't require any user permission.

`RepeatingLessons` displays a list of recurring lessons and a "Select calendar" button in the toolbar. The app offers the lessons on specific dates and times and doesn't fetch any events from the user's calendars. `RepeatingLessons` can't let the user or the system make any changes to these events. Because of these reasons, the app requires write-only access so it can control the date and time of every event added to Calendar. When the user selects a lesson, then taps the booking button, the app first checks whether it has authorization to access the user's calendar data. If the authorization status is `.notDetermined`, the app uses an instance of `EKEventStore`, `eventStore`, to prompt the user for write-only access.

```
return try await eventStore.requestWriteOnlyAccessToEvents()
```

`RepeatingLessons` includes `NSCalendarsWriteOnlyAccessUsageDescription` in its `Info.plist` file and uses its value when showing an alert. The alert prompts the user for write-only acess to save repeating lessons to a calendar that the user chooses. If the user grants the request, the app receives a `.writeOnly` authorization status, creates a recurring event using the selected lesson's details, then saves it to Calendar without the user making any changes to this event.

```
try self.eventStore.save(newEvent, span: .futureEvents)
```

The "Select calendar" button in the toolbar allows the user to choose another calendar to save the recurring events using `EKCalendarChooser`. The app turns off the button by default. The app turns it on when the user grants write-only or full access to the app. When the user taps the button, `RepeatingLessons` presents a calendar chooser with an instance of `EKCalendar`, `calendar`, which keeps track of calendars the user chooses in the view controller.

```
.sheet(isPresented: $showCalendarChooser) {
    CalendarChooser(calendar: $calendar)
}
```

The `displayStyle` property of `EKCalendarChooser` specifies whether to display writable calendars only or all calendars. In write-only access apps, the calendar chooser ignores the value of the `displayStyle` setting and this setting always behaves as if it's set to `.writableCalendarsOnly`. As a result, the app only allows the user to select a single writable calendar from the list presented in the calendar chooser.

```
// Initializes a calendar chooser that allows the user to select a single calendar f
let calendarChooser = EKCalendarChooser(selectionStyle: .single,
                                        displayStyle: .writableCalendarsOnly,
                                        entityType: .event,
                                        eventStore: storeManager.store)
```

The app sets the `selectedCalendars` property of `EKCalendarChooser` to `calendar`, which is empty when the user hasn't selected a calendar.

```
/*
    Set up the selected calendars property. If the user previously selected a calend
    Otherwise, update selected calendars with an empty set.
*/
if let calendar = calendar {
    let selectedCalendar: Set<EKCalendar> = [calendar]
    calendarChooser.selectedCalendars = selectedCalendar
} else {
    calendarChooser.selectedCalendars = []
}
```

`RepeatingLessons` configures the chooser to show the Done and Cancel buttons.

```
calendarChooser.delegate = context.coordinator
// Configure the chooser to display Done and Cancel buttons.
calendarChooser.showsDoneButton = true
calendarChooser.showsCancelButton = true
return UINavigationController(rootViewController: calendarChooser)
```

If the user chooses a calendar from the view controller, `RepeatingLessons` adds recurring events to that calendar. If the user doesn't make any selection, the app saves the events to the user's default calendar.

Request full access

In iOS 17, an app with full access can create, edit, save, delete, and fetch all events on all the user's calendars. Additionally, the app can display events using `EKEventViewController` and allow the user to select another calendar using EKCalendarChooser. Implement full access if your app needs to read and write data to Calendar. If your app only needs to write data directly to Calendar, implement write-only access instead. If your app only uses EventKit APIs to create and set up events, consider saving events to Calendar without prompting the user for authorization.

To implement full access in your app, follow these steps:

- Build your app with Xcode 15 and link against the iOS 17 SDK.
- Add the `NSCalendarsFullAccessUsageDescription` key to the `Info.plist` file of the target building your app.
- To request full access to events, use `requestFullAccessToEvents(completion:)` or `requestFullAccessToEvents()`.

Upon its first launch, the `MonthlyEvents` app registers for EKEventStoreChanged notifications to listen for any changes to the event store.

```
let center = NotificationCenter.default
let notifications = center.notifications(named: .EKEventStoreChanged).map({ (notific
for await _ in notifications {
    guard await dataStore.isFullAccessAuthorized else { return }
    await self.fetchLatestEvents()
}
```

Then, the app checks whether it's authorized to access the user's calendar data.

```
let status = EKEventStore.authorizationStatus(for: .event)
```

If the authorization status is `.notDetermined`, the app uses an instance of `EKEventStore`, `eventStore`, to prompt the user for full access.

```
return try await eventStore.requestFullAccessToEvents()
```

`MonthlyEvents` includes `NSCalendarsFullAccessUsageDescription` in its `Info.plist` file and uses its value when showing an alert. The alert prompts the user for full access to fetch events in all the user's calendars and delete the ones the user selects in the app. If the user grants the request, the app receives a `.fullAccess` authorization status.

```
EKEventStore.authorizationStatus(for: .event) == .fullAccess
```

Then, the app fetches and displays all events occuring within a month in all the user's calendars sorted by start date in ascending order.

```
let start = Date.now
let end = start.oneMonthOut
let predicate = eventStore.predicateForEvents(withStart: start, end: end, calendars:
return eventStore.events(matching: predicate).sortedEventByAscendingDate()
```

If the user denies the request, the app does nothing. In subsequent launches, the app displays a message prompting the user to grant the app full access in Settings on their device.

Because the user authorized the app for full access, the user can additionally select and delete one or more events in `MonthlyEvents`. The app iterates through an array of events that the user chose to delete. It calls and sets the `commit` parameter of the `remove(_:span:commit:)` function to `false` to batch the deletion of each event in the array.

```
try self.eventStore.remove(event, span: .thisEvent, commit: false)
```

Then, the app commits the changes once it's done iterating through the array.

```
try eventStore.commit()
```

When you assign `true` to `commit` to immediately save or remove the event in your app, the event store automatically rolls back any changes if the commit operation fails. However, if you set `commit` to `false` and your app successfully removes some events and fails removing others, this can result in a later commit failing. Every subsequent commit fails until you roll back the changes. Call `reset()` to manually roll back the changes.

```
eventStore.reset()
```

Run apps on operating system earlier than iOS 17

If you build your app with Xcode 15, link it against the iOS 17 SDK, and need to run it on systems earlier than iOS 17:

- Add `NSCalendarsUsageDescription` to the `Info.plist` file of the target building your app. If your app that's linked on iOS 10 through iOS 16 doesn't include `NSCalendarsUsageDescription`, your app crashes.

- To request access to events, use `requestAccess(to:completion:)` or `requestAccess(to: .event)`.
- To determine whether your app is authorized to access the user's calendar data, confirm that `authorizationStatus(for:)` is set to `.authorized`.

Note

The new request methods are unavailable on systems earlier than iOS 17, which may cause your app to crash when running on these versions. Check that these methods are available in the iOS version that you wish to run your app on before calling them in your app. See Declaration Attributes for details.

The `DropInLessons`, `MonthlyEvents`, and `RepeatingLessons` targets in the sample project have a deployment target of iOS 16.4, meaning their apps can run on devices running iOS 16.4 and later. These apps include `NSCalendarsUsageDescription` in their `Info.plist` and use `requestAccess(to: .event)` when requesting permission from the user.

```
// Fall back on earlier versions.
return try await eventStore.requestAccess(to: .event)
```

Important

In iOS 17, calling `requestAccess(to: .event)` or `requestAccess(to:completion:)` doesn't prompt the user for access and throws an error.

`MonthlyEvents` and `RepeatingLessons` confirm that they have an `.authorized` authorization status.

```
// Fall back on earlier versions.
EKEventStore.authorizationStatus(for: .event) == .authorized
```

See Also

Essentials

Accessing the event store

Request access to a person's calendar data through the event store.

```
class EKEventStore
```

An object that accesses a person’s calendar events and reminders and supports the scheduling of new events.