Sample Code

Scheduling an alarm with AlarmKit

Create prominent alerts at specified dates for your iOS app.

Download

iOS 26.0+ | iPadOS 26.0+ | Xcode 26.0+

Overview

An alarm is an alert that presents at a pre-determined time based on a schedule or after a countdown. It overrides both a device's focus and silent mode, if necessary.

This sample project uses AlarmKit to create and manage different types of alarms. In this app people can create and manage:

- **One-time alarms** which alert only once at a specified time in the future.
- **Repeating alarms** which alert with a weekly cadence.
- **Timers** which alert after a countdown, and start immediately.

This project also includes a widget extension for setting up the custom countdown Live Activity associated with an alarm.

> Note
>
> This sample code project is associated with WWDC25 session 230: Wake up to the AlarmKit API.

Authorize the app to schedule alarms

This sample prompts people to authorize the app to allow AlarmKit to schedule alarms and create alerts by calling `requestAuthorization()` on `AlarmManager`. Otherwise, when a person adds their first alarm, AlarmKit automatically requests this authorization on behalf of the app, before scheduling the alarm. If this sample doesn't get this authorization, then any alarm created by the app isn't scheduled and subsequently doesn't alert.

```
do {
    let state = try await alarmManager.requestAuthorization()
    return state == .authorized
} catch {
    print("Error occurred while requesting authorization: \(error)")
    return false
}
```

The sample includes the `NSAlarmKitUsageDescription` key in the app's `Info.plist` with a descriptive string explaining why it schedules alarms. This string appears in the system prompt when requesting authorization, in this sample the string is:

```
We'll schedule alerts for alarms you create within our app.
```

If the `NSAlarmKitUsageDescription` key is missing or its value is an empty string, apps can't schedule alarms with AlarmKit.

Create the alarm schedule

The sample app creates an alarm with either, or both, a countdown duration and a schedule, based on the options a person sets.

`Alarm.CountdownDuration` uses the selected `TimeInterval` for the pre-alert countdown, which displays the alert when the countdown reaches 0.

`Alarm.Schedule` enables people to set a one-time alarm, or configure a weekly schedule. For single-occurrence alarms, the `repeats` property is set to `Alarm.Schedule.Relative.Recurrence.never`. For recurring alarms, the `repeats` property is set to `Alarm.Schedule.Relative.Recurrence.weekly(_:)` with an associated array `Locale.Weekday`, indicating the days of the week the alarm alerts.

```
let time = Alarm.Schedule.Relative.Time(hour: hour, minute: minute)
return .relative(.init(
    time: time,
    repeats: weekdays.isEmpty ? .never : .weekly(Array(weekdays))
))
```

Configure the alarm’s UI attributes

AlarmKit provides a presentation for each of the three alarm states - `AlarmPresentation.Alert`, `AlarmPresentation.Countdown`, and `AlarmPresentation.Paused`. Because `Countdown` and `Paused` are optional presentations, this sample doesn’t use them if the alarm only has an `Alert` state.

```
let alertContent = AlarmPresentation.Alert(title: userInput.localizedLabel,
        stopButton: .stopButton,
        secondaryButton: secondaryButton,
        secondaryButtonBehavior: secondaryButtonBehavior)

guard userInput.countdownDuration != nil else {
    // An alarm without countdown specifies only an alert state.
    return AlarmPresentation(alert: alertContent)
}

// With countdown enabled, provide a presentation for both a countdown and paused st
let countdownContent = AlarmPresentation.Countdown(title: userInput.localizedLabel,
        pauseButton: .pauseButton)

let pausedContent = AlarmPresentation.Paused(title: "Paused",
        resumeButton: .resumeButton)

return AlarmPresentation(alert: alertContent, countdown: countdownContent, paused: p
```

Alongside the `stopButton`, the sample includes another action button in the alerting UI. This action depends on `secondaryButton` and `secondaryButtonBehavior`.

```
var secondaryButtonBehavior: AlarmPresentation.Alert.SecondaryButtonBehavior? {
    switch selectedSecondaryButton {
    case .none: nil
    case .countdown: .countdown
    case .openApp: .custom
    }
}
```

When the `secondaryButtonBehavior` property is set to `AlarmPresentation.Alert.SecondaryButtonBehavior.countdown`, the secondary button is a `Repeat` action, which re-triggers the alarm after a certain `TimeInterval`, as specified in `postAlert`. If the `secondaryButtonBehavior` is set to `AlarmPresentation.Alert.SecondaryButtonBehavior.custom`, the alarm alert displays an `Open` action to launch the app.

```
let secondaryButton: AlarmButton? = switch secondaryButtonBehavior {
    case .countdown: .repeatButton
    case .custom: .openAppButton
    default: nil
}
```

Note

The system forwards the alert presentation to a paired watch (if any) to notify people when an alarm is alerting.

The content for these presentations is wrapped into `ActivityAttributes`, along with `tintColor`, and `metadata`. The tint color associates the alarms with the sample app and also differentiates them from other app's alarms on the person's device.

```
let attributes = AlarmAttributes(presentation: alarmPresentation(with: userInput),
        metadata: CookingData(),
        tintColor: Color.blue)
```

Schedule the configured alarm

The sample uses a unique identifier to track alarms registered with AlarmKit. The sample manages and updates alarm states, such as `pause(id:)` and `cancel(id:)`, using this identifier.

When a person taps the button in the alerting UI, the `AlarmManager` automatically handles stop or countdown functionalities, depending on the button type.

Tip

You can add additional actions for each button type using App Intents, which you can configure using `AlarmManager.AlarmConfiguration`.

```
let id = UUID()
let alarmConfiguration = AlarmConfiguration(countdownDuration: userInput.countdownDu
        schedule: userInput.schedule,
        attributes: attributes,
        stopIntent: StopIntent(alarmID: id.uuidString),
        secondaryIntent: secondaryIntent(alarmID: id, userInput: userInput))
```

This sample creates the alarm ID and `AlarmManager.AlarmConfiguration` and schedules the alarm with `AlarmManager`.

```
let alarm = try await alarmManager.schedule(id: id, configuration: alarmConfiguratio
```

Observe state changes on the alarms

At initialization, the `ViewModel` subscribes to alarm events from `shared`. This enables the sample app to have the latest state of an alarm even if the alarm state updated while the sample app isn't running.

```
Task {
    for await incomingAlarms in alarmManager.alarmUpdates {
        updateAlarmState(with: incomingAlarms)
    }
}
```

> **Note**
>
> An `Alarm` that's not included in the `alarmUpdates` asynchronous stream is no longer scheduled with AlarmKit.

Create a Widget Extension for Live Activities

The sample app adds a widget extension target to customize non-alerting presentations in the Dynamic Island, Lock Screen, and StandBy. The widget extension receives the same `AlarmAttributes` structure that you provide to `shared` when scheduling alarms. It includes the metadata provided in the Configure the alarm's UI attributes section above.

Important

AlarmKit expects a widget extension if an app supports a countdown presentation. Otherwise, the system may unexpectedly dismiss alarms and fail to alert. For more information, see ActivityKit.

See Also

Alarm management

`class AlarmManager`

An object that exposes functions to work with alarms: scheduling, snoozing, cancelling.

`struct Alarm`

An object that describes an alarm that can alert once or on a repeating schedule.