Article

Running workout sessions

Track a workout on Apple Watch.

Overview

Use workout sessions to track a user's activity on Apple Watch. While the session runs, the system fine-tunes the watch's sensors for the specified activity. For example, all workout sessions generate high-frequency heart rate samples; however, an outdoor cycling activity generates accurate location data, while an indoor cycling activity doesn't.

To track workouts on Apple Watch, you must set up, start, and save a workout session. Make starting and stopping workouts on Apple Watch as easy and obvious as possible. The app must clearly indicate when a workout session is in progress, and either automatically save workout data for the user, or provide a clear option to explicitly save or discard workout data. Finally, the watchOS app should provide clear feedback when a workout is successfully saved. For more information, see Health and Fitness in watchOS Human Interface Guidelines.

Set up the app

Before creating a workout session, you must set up HealthKit and request permission to read and share any health data your app intends to use. For step-by-step instructions, see Setting up HealthKit.

For workout sessions, you must request permission to share workout types. You may also want to read any data types automatically recorded by Apple Watch as part of the session.

```
// The quantity type to write to the health store.
let typesToShare: Set = [
    HKQuantityType.workoutType()
]
```

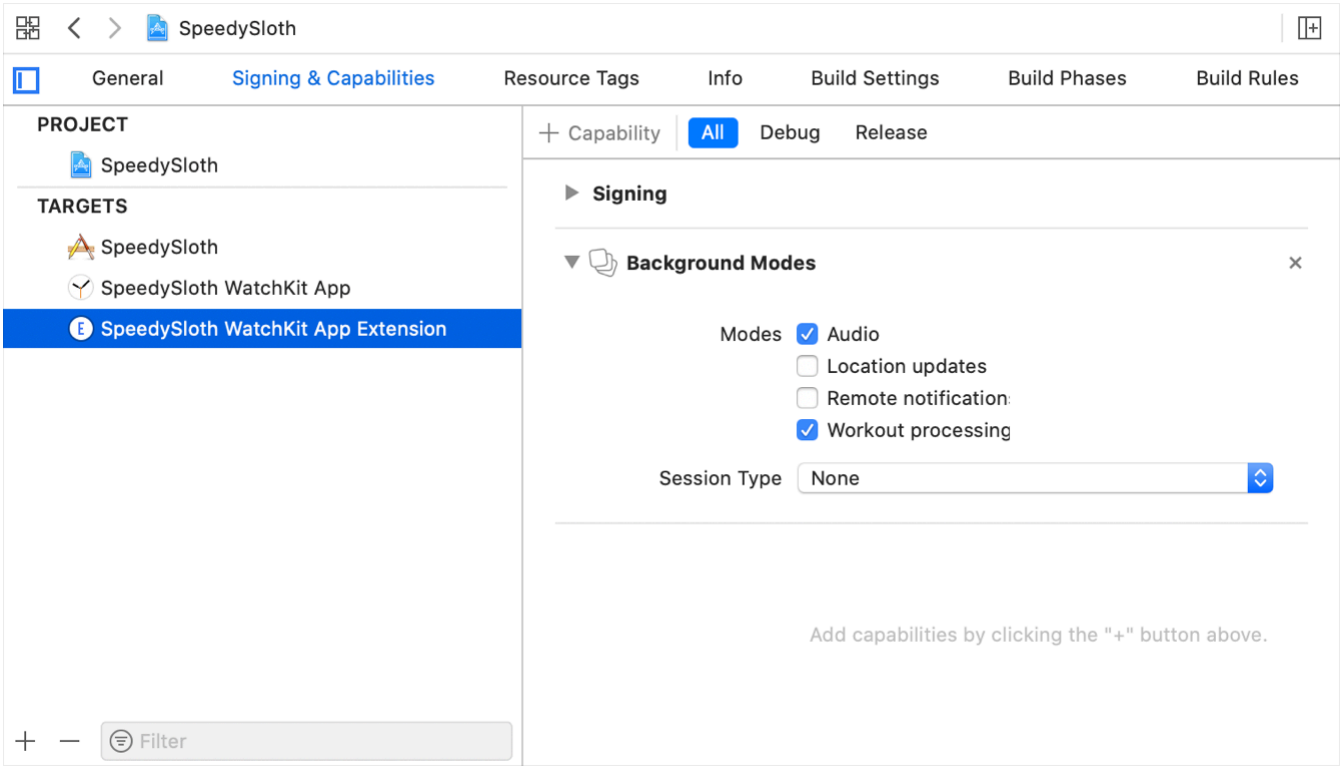
```
// The quantity types to read from the health store.
let typesToRead: Set = [
    HKQuantityType.quantityType(forIdentifier: .heartRate)!,
    HKQuantityType.quantityType(forIdentifier: .activeEnergyBurned)!,
    HKQuantityType.quantityType(forIdentifier: .distanceWalkingRunning)!,
    HKObjectType.activitySummaryType(),
    HKCharacteristicType(.activityMoveMode)
]

// Request authorization for those quantity types.
healthStore.requestAuthorization(toShare: typesToShare, read: typesToRead) { (succes
    // Handle errors here.
}
```

Note

In watchOS 6 and later, users can authorize reading and sharing data on Apple Watch. As a result, you must add usage descriptions to your WatchKit App Extension. Use the `NSHealthShareUsageDescription` key to describe why your app needs to read the requested data. Use `NSHealthUpdateUsageDescription` for the data your app intends to write. For projects created using Xcode 13 or later, set these keys in the Target Properties list on the app's Info tab. For projects created with Xcode 12 or earlier, set these keys in the apps `Info.plist` file. For more information, see `Information Property List`.

Apps with an active workout session can run in the background, so you need to add the background modes capability to your WatchKit App Extension.

Workout sessions require the Workout processing background mode. If your app plays audio or provides haptic feedback during the workout session, you must also add the Audio background mode.



Use the `AVAudioPlayer` class to play short audio clips. To play long form audio in the background, see Playing Background Audio.

> **Note**
>
> Workout apps can use the AVFoundation framework to play short audio clips in the background, such as coaching or notifications. In order to play an audio clip, an active workout session must be running; any attempt to play background audio outside a workout session are invalid.

Start a session

To start a workout session, begin by creating a configuration object for the workout.

```
let configuration = HKWorkoutConfiguration()
configuration.activityType = .running
configuration.locationType = .outdoor
```

Use the configuration to set the type of activity and the location for the workout. Apple Watch optimizes both the sensors and the calorimetry based on the configuration.

For example, while the session runs, Apple Watch automatically saves active energy-burned samples to the HealthKit store. HealthKit provides optimized calorie calculations for some activities. These include, but are not limited to, run, walk, cycle, stair-climbing, elliptical, and

rowing activities. Furthermore, the calculations for activities may differ between indoor and outdoor locations. For all other activities, the system estimates calories based on the data from Apple Watch's sensors. Depending on the activity, this rate is either never lower than the brisk walk burn rate or never lower than the brisk walk burn rate when moving.

Next, use the configuration to create your workout session and get a reference to the session's `HKLiveWorkoutBuilder` object.

```
do {
    session = try HKWorkoutSession(healthStore: healthStore, configuration: configu
    builder = session.associatedWorkoutBuilder()
} catch {
    // Handle failure here.
    return
}
```

The `HKWorkoutSession` class's initializer can throw an exception if the configuration is invalid, so you need to wrap the initializer in a `do-catch` block.

Then, create an `HKLiveWorkoutDataSource` object and assign it to the workout builder.

```
builder.dataSource = HKLiveWorkoutDataSource(healthStore: healthStore,
                                             workoutConfiguration: configuration)
```

Use the same configuration object for the workout session and the live data source. While the session runs, Apple Watch automatically collects data about the workout, and saves samples to the HealthKit store. For example, an outdoor running session collects and saves `activeEnergyBurned`, `basalEnergyBurned`, `heartRate`, and `distanceWalkingRunning` samples.

You can assign delegates to monitor both the workout session and the workout builder.

```
session.delegate = self
builder.delegate = self
```

Your `HKWorkoutSessionDelegate` receives updates whenever the session's state changes, an event occurs, or the session fails due to an error. Your `HKLiveWorkoutBuilderDelegate` receives updates when either Apple Watch or your app adds a new sample or event to the builder.

By default, the workout session automatically forwards all events to the builder, so both delegates should receive the same set of events. However, you can set the builder's `shouldCollectWorkoutEvents` property to `false` if you want to control the events set to the builder.

Finally, start the session and the builder.

```
session.startActivity(with: Date())
builder.beginCollection(withStart: Date()) { (success, error) in

    guard success else {
        // Handle errors.
    }

    // Indicate that the session has started.
}
```

Your app's user interface should clearly indicate that a workout session is running. Use data passed to your `HKLiveWorkoutBuilderDelegate` to update information about the session.

Handle samples and events

While the session runs, Apple Watch automatically collects and adds samples and events based on the workout configuration. You can record additional information by adding your own events and samples to the workout. To add samples, call the builder's `add(_:completion:)` method. To add events, call the `addWorkoutEvents(_:completion:)` method.

Typically, you should update your app's user interface whenever the builder receives a new sample or event. To respond to new samples, implement your `HKLiveWorkoutBuilderDelegate` delegate's `workoutBuilder(_:didCollectDataOf:)` method.

```
func workoutBuilder(_ workoutBuilder: HKLiveWorkoutBuilder, didCollectDataOf collect
    for type in collectedTypes {
        guard let quantityType = type as? HKQuantityType else {
            return // Nothing to do.
        }

        // Calculate statistics for the type.
        let statistics = workoutBuilder.statistics(for: quantityType)
        let label = labelForQuantityType(quantityType)

        DispatchQueue.main.async() {
            // Update the user interface.
        }
    }
}
```

HealthKit calls this method whenever the builder receives a new sample. Your implementation should examine the samples and update your app's user interface—for example, by updating the current distance traveled, calories burned, and pace during a run. Use the workout builder's `statistics(for:)` method to calculate statistics, such as the total, average, minimum, or maximum, for any types that have new data.

To respond to new events, implement your `HKLiveWorkoutBuilderDelegate` delegate's `workoutBuilderDidCollectEvent(_:)` method.

```
func workoutBuilderDidCollectEvent(_ workoutBuilder: HKLiveWorkoutBuilder) {

    let lastEvent = workoutBuilder.workoutEvents.last

    DispatchQueue.main.async() {
        // Update the user interface here.
    }
}
```

HealthKit calls this method whenever the builder receives a new event. Use the builder's `workoutEvents` property to examine the last event, and update your user interface as needed. For example, when a `HKWorkoutEventType.lap` event occurs, you can increment the number of laps shown.

Run in the background

When your app has an active workout session, it continues to run in the background. Background execution grants your app the following abilities:

- Your app continues to run throughout the entire workout session, even when the user lowers their wrist or interacts with a different app. When the user raises their wrist, your app reappears, letting the user quickly check their current progress and performance.
- If the user navigates back to the watch face, Apple Watch displays a small icon at the top of the screen, indicating that a workout session is running. Users can tap the icon to navigate back to your app.
- Your app continues to receive data from HealthKit and Apple Watch's sensors in the background. Use this data to keep your app up to date at all times. For example, your app could display the user's current heart rate, the distance that the user has cycled, and any other relevant statistics whenever the user raises their wrist.
- Your app can alert the user using audio or haptic feedback while running in the background.

To maintain high performance on Apple Watch, you must limit the amount of work your app performs in the background. If your app uses an excessive amount of CPU while in the background, watchOS may suspend it. Use Xcode's CPU report tool or the time profiler in Instruments to test your app's CPU usage. The system also generates a log with a backtrace whenever it terminates your app.

Coordinate with the companion app

If the watchOS app has an iOS companion, be sure to keep both apps in sync. This is particularly important if the user can start workouts on either device.

- If the user starts the workout in your watchOS app, and then tries to end it in the iOS companion, the iOS app should instruct the user to end the workout in your watchOS app. Otherwise, Apple Watch workout continues to run, which can lead to accidentally saving invalid data to the HealthKit store.
- If the user starts a workout in the iOS companion, and then opens your watchOS app, the watchOS app should automatically start a workout session for the workout in progress. If you save the workout in the watchOS app, you can incorporate data from the iOS workout. For example, you should set the workout's `startDate` to the iOS workout's start. Also, if your app calculates its own calories, you can retroactively give credit for the calories burned before the workout session began.
- If the user doesn't start and stop a workout session in your watchOS app, don't try to retroactively create a workout session on Apple Watch.

End a session

After the user finishes the workout, stop the session by calling `stopActivity(with:)`.

```
session.stopActivity(with: Date())
```

After the session has transitioned to the `.stopped` state, call the builder's `endCollection(withEnd:completion:)` and `finishWorkout(completion:)` methods. Finally, call `end()` to end the session.

```
func workoutSession(_ workoutSession: HKWorkoutSession, didChangeTo toState: HKWorko
                    from fromState: HKWorkoutSessionState, date: Date) {

    // Wait for the session to transition states before ending the builder.
    if toState == .stopped {
```

```
        builder.endCollection(withEnd: date) { (success, error) in

            guard success else {
                // Handle errors.
            }

            builder.finishWorkout { (workout, error) in

                guard workout != nil else {
                    // Handle errors.
                }

                session.end()

                DispatchQueue.main.async() {
                    // Update the user interface.
                }
            }
        }
    }
}
```

The `endCollection(withEnd:completion:)` method sets the workout's end date and deactivates the builder. The `finishWorkout(completion:)` method saves the workout and its associated data to the HealthKit store.

Recover from crashes

If your app crashes during a workout session, the system calls your extension delegate's `handleActiveWorkoutRecovery()` method when the app relaunches. In your implementation of this method, call the HealthKit store's `recoverActiveWorkoutSession(completion:)` method. HealthKit attempts to restore the previous workout session, returning either a new session object or an error to the completion block.

As soon as you receive the session object, you must access its builder and set up your data source and delegates again.

See Also

Sessions

{} Build a workout app for Apple Watch

Create your own workout app, quickly and easily, with HealthKit and SwiftUI.

{} Building a multidevice workout app

Mirror a workout from a watchOS app to its companion iOS app, and perform bidirectional communication between them.

{} Building a workout app for iPhone and iPad

Start a workout in iOS, control it from the Lock Screen with App Intents, and present the workout status with Live Activities.

`class` `HKWorkoutSession`

A session that tracks a person's workout.

`class` `HKWorkoutConfiguration`

An object that contains configuration information about a workout session.

`enum` `HKWorkoutSessionState`

A workout session's state.

`class` `HKLiveWorkoutBuilder`

A builder object that constructs a workout incrementally based on live data from an active workout session.

`class` `HKLiveWorkoutDataSource`

A data source that automatically provides live data from an active workout session.