Article

Adding samples to a workout

Create associated samples that add details to a workout.

Overview

When you create a workout sample, it only contains a general overview of the workout: the workout's duration, the total distance traveled, and the total energy burned. You can enhance the workout by creating samples that provide additional details, and associating these samples with the workout. To associate a sample with a workout use the HealthKit store's `add(_:to:completion:)` method. The workout must be saved to the HealthKit store before you add any samples. The samples don't need to be saved. Adding them to the workout automatically saves them.

Gather data about a workout

Associating samples with a workout provides fine-grain information about the workout. However, adding samples to the workout does not change any of the workout's properties. Specifically, adding distance samples won't change the quantity stored in the `totalDistance`, `startDate`, `endDate`, or `duration` properties. Likewise, adding active energy burned samples won't change the quantity stored in the `totalEnergyBurned` property. Therefore, adding associated samples leads to some duplication between the workout's properties and these samples.

Your app should always provide data for the workout's `duration`, `totalDistance`, and `totalEnergyBurned` properties when the data is both available and relevant to the workout. In addition, you should provide a set of associated samples that sum up to these totals. You can also provide additional associated samples, to help track how the intensity of the exercise changed during the course of the workout.

For example, an app that tracks runs might create a workout that includes the total distance, duration, and calories burned after the user finishes the run. The app could also save samples that

describe the distance, calories burned, step count, heart rate, flights climbed, and other data over much smaller time intervals.

Fine tune the exact length of your associated samples based on the type of workout and the needs of your app. Using five-minute intervals minimizes the amount of memory needed to store the workouts, while still providing a general sense of the change in intensity over the course of a long workout. Using five-second intervals provides a much more detailed view of the workout, but requires considerably more memory and processing.

The following code samples show how to associate distance, energy burned, and Heart Rate samples with a workout. For simplicity's sake, these samples use hard-coded values and perform all the operations inline. A real-world app would calculate these values from sensor data and break the operation up using helper methods.

Create and save the sample

Start by creating quantity objects for the total energy burned, and total distance traveled.

```
let energyBurned = HKQuantity(unit: HKUnit.largeCalorie(), doubleValue: 425.0)
let distance = HKQuantity(unit: HKUnit.mile(), doubleValue: 3.2)
```

Next create the workout sample.

```
let run = HKWorkout(activityType: HKWorkoutActivityType.running,
                    start: start,
                    end: end,
                    duration: 0,
                    totalEnergyBurned: energyBurned,
                    totalDistance: distance,
                    metadata: nil)
```

And save the sample to the HealthKit store.

```
store.save(run) { (success, error) -> Void in
    guard success else {
        // Perform proper error handling here.
        return
    }

    // Add detail samples here.
}
```

Add detailed samples

In the completion handler, you check to ensure that the save succeeded, and then add detailed samples to the workout. For example, you could split the workout into intervals and then calculate detailed information for each interval. The following code listing creates a sample for the distance covered in the first interval.

```
guard let distanceType =
    HKObjectType.quantityType(forIdentifier:
        HKQuantityTypeIdentifier.distanceWalkingRunning) else {
    fatalError("*** Unable to create a distance type ***")
}

let distancePerInterval = HKQuantity(unit: HKUnit.foot(),
                                     doubleValue: 165.0)

let distancePerIntervalSample = HKQuantitySample(type: distanceType,
                                                 quantity: distancePerInterval,
                                                 start: myIntervals[0],
                                                 end: myIntervals[1])

myDetailSamples.append(distancePerIntervalSample)
```

Then you create an energy-burned sample for that interval.

```
guard let energyBurnedType =
    HKObjectType.quantityType(forIdentifier:
        HKQuantityTypeIdentifier.activeEnergyBurned) else {
    fatalError("*** Unable to create an energy burned type ***")
}

let energyBurnedPerInterval = HKQuantity(unit: HKUnit.largeCalorie(),
                                         doubleValue: 15.5)

let energyBurnedPerIntervalSample =
    HKQuantitySample(type: energyBurnedType,
                     quantity: energyBurnedPerInterval,
                     start: myIntervals[0],
                     end: myIntervals[1])

myDetailSamples.append(energyBurnedPerIntervalSample)
```

However, you're not limited to just energy burned and distance. The following code creates a heart rate sample.

```
guard let heartRateType =
    HKObjectType.quantityType(forIdentifier:
        HKQuantityTypeIdentifier.heartRate) else {
    fatalError("*** Unable to create a heart rate type ***")
}

let heartRateForInterval = HKQuantity(unit: HKUnit(from: "count/min"),
                                      doubleValue: 95.0)

let heartRateForIntervalSample =
    HKQuantitySample(type: heartRateType,
                     quantity: heartRateForInterval,
                     start: myIntervals[0],
                     end: myIntervals[1])

myDetailSamples.append(heartRateForIntervalSample)
```

Continue to create all the samples you need for all of your intervals. Then use the HealthKit store to add these samples to the workout.

```
store.add(myDetailSamples, to: run) { (success, error) -> Void in
    guard success else {
        // Perform proper error handling here.
        return
    }
}
```

Read associated samples

To read a workout's fine-grain details, you need to create a query that returns only the samples associated with the workout. Use the `predicateForObjects(from:)` method to create a predicate object that matches only samples associated with the workout. You can then use that predicate to filter one or more queries. For example, the following sample code returns all the distance samples associated with the workout.

```
guard let distanceType =
    HKObjectType.quantityType(forIdentifier:
```

```
        HKQuantityTypeIdentifier.distanceWalkingRunning) else {
            fatalError("*** Unable to create a distance type ***")
}

let workoutPredicate = HKQuery.predicateForObjects(from: workout)

let startDateSort = NSSortDescriptor(key: HKSampleSortIdentifierStartDate, ascending

let query = HKSampleQuery(sampleType: distanceType,
                          predicate: workoutPredicate,
                          limit: 0,
                          sortDescriptors: [startDateSort]) { (sampleQuery, results,
                            guard let distanceSamples = results as? [HKQuantitySampl
                                // Perform proper error handling here.
                                return
                            }

                            // Use the workout's distance samples here.
}

store.execute(query)
```

You need to create a separate query for each data type associated with the workout. For example, to get the full details of the workout created earlier, you would need separate queries for distance, energy burned, and heart rate.

See Also

Samples

- Accessing condensed workout samples

  Read series data from condensed workouts.

- Dividing a HealthKit workout into activities

  Partition multisport and interval workouts into activities that represent the different parts of the workout.

`class` HKWorkout

A workout sample that stores information about a single physical activity.

`class` HKWorkoutActivity

An object that describes an activity within a longer workout.

`class HKWorkoutBuilder`

A builder object that incrementally constructs a workout.

`class HKWorkoutType`

A type that identifies samples that store information about a workout.

`let HKWorkoutTypeIdentifier: String`

The workout type identifier.

`enum HKWorkoutActivityType`

The type of activity performed during a workout.

`enum HKWorkoutSessionType`

The type of session.

`class HKWorkoutEvent`

An object representing an important event during a workout.