Article

Accessing condensed workout samples

Read series data from condensed workouts.

Overview

To reduce the size of workout data, HealthKit can condense system generated samples associated with first party workouts. HealthKit enumerates the data associated with a workout and adds that data to one or more quantity series samples. After saving all the data to series objects, HealthKit deletes the originals. In addition, it converts older `HKQuantitySample` objects to `HKCumulativeQuantitySample` or `HKDiscreteQuantitySample` objects.

HealthKit condenses high-frequency first party workout data. This includes, but it's not limited to, the following data types:

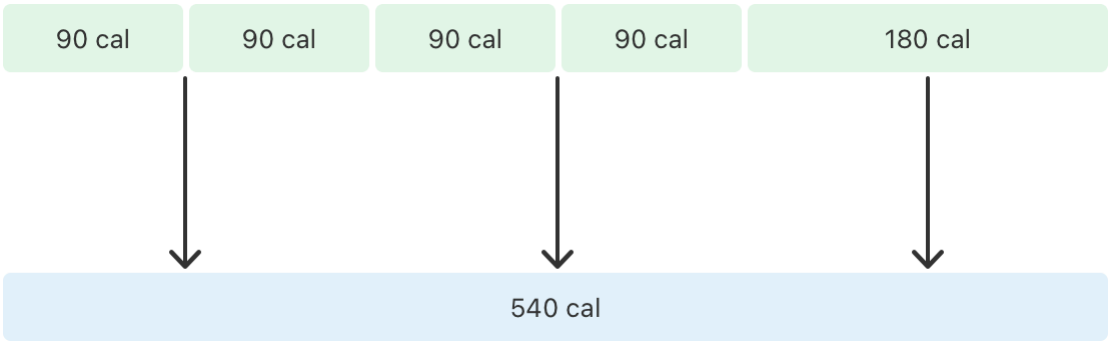
- `distanceWalkingRunning`
- `distanceCycling`
- `basalEnergyBurned`
- `activeEnergyBurned`
- `heartRate`

For these data types, HealthKit condenses samples associated with a workout that's at least a few months old and has a duration of at least a few minutes. HealthKit may condense samples for a given workout more than once. For example, if the system syncs new samples associated with the workout to the phone, or if the condenser algorithm changes, HealthKit may condense the workout again.

HealthKit merges quantities within each series to save space and improve query performance on workout data, while still preserving data resolution. HealthKit also coalesces cumulative types and heart rate samples. For cumulative types, such as active energy, HealthKit combines data that's consecutive in time and has the same value for rate over time. When combining quantities, the

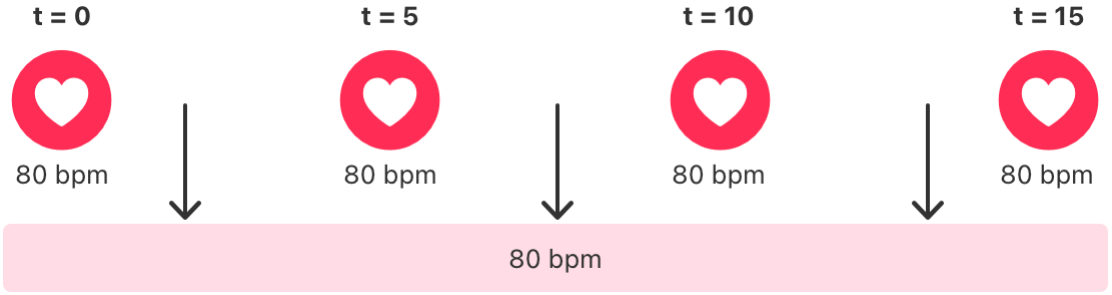
framework replaces the data with a new total that spans the combined timespan, with a value is the sum of the original quantities.

In the example below, several energy samples are consecutive in time, and the rates (calories per time unit) of these samples are identical. Thus, instead of storing five quantities in series, HealthKit combines all these into one. Since the rate over time is the same, coalescing doesn't alter the results of statistics.



For heart rate samples associated with first-party workouts, HealthKit coalesces quantities that have the same value and occurred close together. When coalescing, HealthKit replaces the quantities, which are instantaneous in time, with a new quantity that's the original value and a continuous interval that spans the combined timespan.

In the example below, HealthKit merges consecutive heart rate samples that have the same value and aren't far apart. Merging instantaneous heart rate quantities results in a continuous interval. HealthKit does this to preserve statistics.



The condensing and coalescing processes preserve all the data from the original workout and reduce the overhead needed to save that data to disk. However, your app may need to perform additional steps to read the condensed and coalesced data.

Query high-frequency data from a condensed workout

Because HealthKit condenses and coalesces older workout data, any samples associated with a workout may actually represent a series of higher-frequency data. In many cases, you don't need to operate on the `HKSamples` or the backing quantity series data. Instead, you should use `HKStatisticsQuery` and `HKStatisticsCollectionQuery` to transparently compute statistics for the underlying data.

However, if your app needs to access the underlying data directly, start by querying for all the samples associated with a workout:

```
// Create the workout predicate.
let forWorkout = HKQuery.predicateForObjects(from: workout)

// Create the heart-rate descriptor.
let heartRateDescriptor = HKQueryDescriptor(sampleType: myHeartRateType,
                                            predicate: forWorkout)

// Create the query.
let heartRateQuery = HKSampleQuery(queryDescriptors: [heartRateDescriptor],
                                   limit: HKObjectQueryNoLimit)
{ query, samples, error in
    // Process the samples.
}

// Run  the query.
myStore.execute(heartRateQuery)
```

Then, in the query's results handler, if a sample has a `count` greater than 1, it contains series data.

```
// Create the query.
let heartRateQuery = HKSampleQuery(queryDescriptors: [heartRateDescriptor],
                                   limit: HKObjectQueryNoLimit)
{ query, samples, error in

    // Start by checking for errors.
    guard let samples = samples else {
        // Handle the error.
        fatalError("*** An error occurred: \(error!.localizedDescription) ***")
    }

    // Iterate over all the samples.
    for sample in samples {

        guard let sample = sample as? HKDiscreteQuantitySample else {
            fatalError("*** Unexpected Sample Type ***")
        }

        // Check to see if the sample is a series.
        if sample.count == 1 {
```

```
            // This is a single sample.
            // Use the sample.
            myOutput.append("\(sample)\n")
        }
        else {
            // This is a series.
            // Get the detailed items for the series.
            myGetDetailedItems(for: sample)
        }
    }
}
```

Use an HKQuantitySeriesSampleQuery to access the detailed data from the series.

```
// Create the predicate.
let inSeriesSample = HKQuery.predicateForObject(with: series.uuid)

// Create the query.
let detailQuery = HKQuantitySeriesSampleQuery(quantityType: myHeartRateType,
                                              predicate: inSeriesSample)
{ query, quantity, dateInterval, HKSample, done, error in

    guard let quantity = quantity, let dateInterval = dateInterval else {
        fatalError("*** An error occurred: \(error!.localizedDescription) ***")
    }

    // Use the data.
    myOutput.append("\(quantity.doubleValue(for: HKUnit(from: "count/min"))): \(date
}

// Run the query.
myStore.execute(detailQuery)
```

See Also

Samples

Adding samples to a workout

Create associated samples that add details to a workout.

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.