New Ways to Work with Workouts

Session 707

Niharika Bedekar, Fitness Software Engineer
Karim Benhmida, Health Software Engineer
200 Million
Health and Fitness App Store Category Downloads
HealthKit

Activity

Health Data
- Activity
- Mindfulness
- Nutrition
- Sleep

Body Measurements
Health Records
Heart
Reproductive Health
Results

Today
Sources
Medical ID
Agenda

Privacy and Authorization
Agenda

Privacy and Authorization

New Workout API
Agenda

Privacy and Authorization

New Workout API

New Quantity Series API
Privacy and Authorization
Privacy and Authorization
Privacy and Authorization
Privacy and Authorization
Privacy and Authorization
Proportional Collection
HealthKit authorization can change
Privacy and Authorization

Ask for...

Only what you need

Only when you need it

Every time you need it
Privacy and Authorization
let typesToShare: Set = [ HKSampleType.workoutType() ]
Privacy and Authorization

```swift
let typesToShare: Set = [
    HKSampleType.workoutType()
]

let typesToRead: Set = [
    HKQuantityType.quantityType(forIdentifier: .heartRate)!,
    HKQuantityType.quantityType(forIdentifier: .activeEnergyBurned)!,
    HKQuantityType.quantityType(forIdentifier: .distanceWalkingRunning)!
]
```
let typesToShare: Set = [
    HKSampleType.workoutType()
]

let typesToRead: Set = [
    HKQuantityType.quantityType(forIdentifier: .heartRate)!,
    HKQuantityType.quantityType(forIdentifier: .activeEnergyBurned)!,
    HKQuantityType.quantityType(forIdentifier: .distanceWalkingRunning)!
]

healthStore.requestAuthorization(toShare: typesToShare,
                                  read: typesToRead) { (success, error) in

    // Handle errors
}


New Workout API

Karim Benhmida, Health Software Engineer
Workout App Lifecycle
Workout App Lifecycle

Setup
Workout App Lifecycle

Setup → Start
Workout App Lifecycle

Setup -> Start -> Active Workout
Workout App Lifecycle

- Setup
- Start
- Active Workout
- End
Workout App Lifecycle

Setup → Start → Active Workout → End → Save
Workout App Lifecycle

Setup -> Start -> User Interface -> End -> Save
Workout App Lifecycle

Setup → Start → User Interface → Collect Data → End → Save
Workout App Lifecycle

Setup → Start → User Interface → Control State → Collect Data → End → Save
Workout App Lifecycle
HKWorkoutSession (watchOS)
Workout App Lifecycle
HKWorkoutSession (watchOS)

Prepares sensors for data collection
Workout App Lifecycle
HKWorkoutSession (watchOS)

Prepares sensors for data collection
Allows background running
Workout App Lifecycle
HKWorkoutSession (watchOS)

- Prepares sensors for data collection
- Allows background running
- Controls workout state
Workout App Lifecycle
HKWorkoutSession (watchOS)

Prepares sensors for data collection
Allows background running
Controls workout state
Generates events
Workout App Lifecycle
Workout App Lifecycle

Collect data generated by the device
Workout App Lifecycle

Collect data generated by the device

Save a workout
Collect Data
Collect Data

HKWorkoutBuilder
Collect Data

HKWorkoutBuilder
Creates and saves an HKWorkout
HKWorkoutBuilder

Creates and saves an HKWorkout
Add samples, events, and metadata
Collect Data

HKWorkoutBuilder

Creates and saves an HKWorkout

Add samples, events, and metadata
Collect Data

• HKWorkoutBuilder
  Creates and saves an HKWorkout
  Add samples, events, and metadata

• HKLiveWorkoutBuilder
Collect Data

**HKWorkoutBuilder**
- Creates and saves an HKWorkout
- Add samples, events, and metadata

**HKLiveWorkoutBuilder**
- watchOS only
Collect Data

HKWorkoutBuilder
Creates and saves an HKWorkout
Add samples, events, and metadata

HKLiveWorkoutBuilder
watchOS only
Subclass of HKWorkoutBuilder
Collect Data

HKWorkoutBuilder
- Creates and saves an HKWorkout
- Add samples, events, and metadata

HKLiveWorkoutBuilder
- watchOS only
- Subclass of HKWorkoutBuilder
- Automatic sample and event collection
Setup and Start
HKWorkoutBuilder
// HKWorkoutBuilder

let builder = HKWorkoutBuilder(healthStore: healthStore,
                               configuration: workoutConfiguration,
                               device: nil)
let builder = HKWorkoutBuilder(healthStore: healthStore, configuration: workoutConfiguration, device: nil)

builder.beginCollection(withStart: Date(), completion: {
    (success, error) in
        // Handle error
    })
Setup and Start
HKLiveWorkoutBuilder (watchOS)
// Create session
let session = try HKWorkoutSession(healthStore: healthStore,
    configuration: workoutConfiguration)
// Create session
let session = try HKWorkoutSession(healthStore: healthStore,
    configuration: workoutConfiguration)

// Retrieve builder
let builder = session.associatedWorkoutBuilder()
Setup and Start
HKLiveWorkoutBuilder (watchOS)

// Create session
let session = try HKWorkoutSession(healthStore: healthStore,
        configuration: workoutConfiguration)

// Retrieve builder
let builder = session.associatedWorkoutBuilder()

// Start session and builder
session.startActivity()

builder.beginCollection(withStart: Date(), completion: { (success, error) in
    // Handle error
})
Workout App Lifecycle

Setup → Start → User Interface → Collect Data → Control State → End → Save
Collect Data

Samples
let samples: [HKSample] = ...

builder.add(samples) { (success, error) in
    // Handle error
}
Collect Data
Events
Collect Data

Events

```swift
let events: [HKWorkoutEvent] = ...

builder.addWorkoutEvents(events) { (success, error) in
    // Handle error
}
```
Collect Data
Metadata
let metadata: [String: Any] = ...

builder.addMetadata(metadata) { (success, error) in
    // Handle error
}
Collect Data

HKLiveWorkoutDataSource (watchOS)
Collect Data
HKLiveWorkoutDataSource (watchOS)

Automatically collects data specific to the workout
Collect Data
HKLiveWorkoutDataSource (watchOS)

Automatically collects data specific to the workout

Data types to collect can be customized
Collect Data
HKLiveWorkoutDataSource (watchOS)
// Create a data source
let dataSource = HKLiveWorkoutDataSource(healthStore: healthStore,
workoutConfiguration: workoutConfiguration)
Collect Data
HKLiveWorkoutDataSource (watchOS)

// Create a data source
let dataSource = HKLiveWorkoutDataSource(healthStore: healthStore,
workoutConfiguration: workoutConfiguration);

// Set data source
workoutBuilder.dataSource = dataSource
Collect Data
HKLiveWorkoutDataSource (watchOS)

// Create a data source
let dataSource = HKLiveWorkoutDataSource(healthStore: healthStore,
                                       workoutConfiguration: workoutConfiguration);

// Set data source
workoutBuilder.dataSource = dataSource

// Optionally add other types
let quantityType: HKQuantityType = ...
let optionalPredicate: NSPredicate = ...

dataSource.collectStatistics(for: quantityType, predicate: optionalPredicate)
User Interface
Update displayed statistics
User Interface

Update displayed statistics

```swift
//MARK: HKLiveWorkoutBuilderDelegate
func workoutBuilder(_ workoutBuilder: HKLiveWorkoutBuilder,
    didCollectDataOf collectedTypes: Set<HKSampleType>) {
    let heartRateType = HKQuantityType.quantityType(forIdentifier: .heartRate)!

    if collectedTypes.contains(heartRateType) {
        let updatedStatistics = workoutBuilder.statistics(for: heartRateType)
        updateHeartRateLabel(updatedStatistics)
    }
}
```
User Interface
Update displayed statistics

```swift
//MARK: HKLiveWorkoutBuilderDelegate
func workoutBuilder(_ workoutBuilder: HKLiveWorkoutBuilder,
didCollectDataOf collectedTypes: Set<HKSampleType>) {
    let heartRateType = HKQuantityType.quantityType(forIdentifier: .heartRate)!

    if collectedTypes.contains(heartRateType) {
        let updatedStatistics = workoutBuilder.statistics(for: heartRateType)
        updateHeartRateLabel(updatedStatistics)
    }
}
```
User Interface
Update displayed statistics

```swift
//MARK: HKLiveWorkoutBuilderDelegate
func workoutBuilder(_ workoutBuilder: HKLiveWorkoutBuilder,
didCollectDataOf collectedTypes: Set<HKSampleType>) {
    let heartRateType = HKQuantityType.quantityType(forIdentifier: .heartRate)!

    if collectedTypes.contains(heartRateType) {
        let updatedStatistics = workoutBuilder.statistics(for: heartRateType)
        updateHeartRateLabel(updatedStatistics)
    }
}
```
User Interface
Update displayed statistics

```swift
//MARK: HKLiveWorkoutBuilderDelegate
func workoutBuilder(_ workoutBuilder: HKLiveWorkoutBuilder,
    didCollectDataOf collectedTypes: Set<HKSampleType>){
    let heartRateType = HKQuantityType.quantityType(forIdentifier: .heartRate)!

    if collectedTypes.contains(heartRateType) {
        let updatedStatistics = workoutBuilder.statistics(for: heartRateType)
        updateHeartRateLabel(updatedStatistics)
    }
}
```
User Interface
Track elapsed time
User Interface
Track elapsed time

```swift
//MARK: HKLiveWorkoutBuilderDelegate
func workoutBuilderDidCollectEvents(_ workoutBuilder: HKLiveWorkoutBuilder) {
    let elapsedTime = builder.elapsedTime
    updateTimer(with: elapsedTime)
}
```
User Interface
Track elapsed time

```swift
//MARK: HKLiveWorkoutBuilderDelegate
func workoutBuilderDidCollectEvents(_ workoutBuilder: HKLiveWorkoutBuilder) {
    let elapsedTime = builder.elapsedTime
    updateTimer(with: elapsedTime)
}
```
Control State
HKWorkoutSession (watchOS)
Control State
HKWorkoutSession (watchOS)

Not Started

NEW
Control State
HKWorkoutSession (watchOS)

Not Started ➔ Prepared
Control State
HKWorkoutSession (watchOS)

Not Started → Prepared → Running
Control State
HKWorkoutSession (watchOS)

- Not Started
- Prepared
- Running
- Paused
Control State
HKWorkoutSession (watchOS)

Not Started → Prepared → Running → Stopped
Paused
Control State
HKWorkoutSession (watchOS)

Not Started → Prepared → Running → Stopped → Ended

Paused → Running → Paused
Control State
HKWorkoutSession (watchOS)
Control State
HKWorkoutSession (watchOS)

```swift
session.prepare()
session.startActivity()
session.pause()
session.resume()
session.stopActivity()
session.end()
```
Workout App Lifecycle

Setup → Start → User Interface → Collect Data → Control State → End → Save
End a Workout
End a Workout

// End session
session.end()
// End session
session.end()

// End builder
builder.endCollection(withEnd: Date(), completion: { (success, error) in
    // Handle error
})
Save a Workout
Save a Workout

// Save
builder.finishWorkout { (workout, error) in
    // Handle error
}

NEW
Demo

Building a Workout App for Apple Watch
Workout Recovery (watchOS)
Workout Recovery (watchOS)

App automatically relaunched if it crashes during a workout
Workout Recovery (watchOS)

App automatically relaunched if it crashes during a workout

Session and builder restored in their previous state
Workout Recovery (watchOS)

App automatically relaunched if it crashes during a workout

Session and builder restored in their previous state

Data source must be setup again
Workout Recovery (watchOS)
class ExtensionDelegate: NSObject, WKExtensionDelegate {

    func handleActiveWorkoutRecovery() {
        let healthStore = HKHealthStore()
        healthStore.recoverActiveWorkoutSession { (session, error) in
            // Handle error
        }
    }
}
class ExtensionDelegate: NSObject, WKExtensionDelegate {

    func handleActiveWorkoutRecovery() {
        let healthStore = HKHealthStore()
        healthStore.recoverActiveWorkoutSession { (session, error) in
            // Handle error
        }
    }
}
class ExtensionDelegate: NSObject, WKExtensionDelegate {

    func handleActiveWorkoutRecovery() {
        let healthStore = HKHealthStore()
        healthStore.recoverActiveWorkoutSession { (session, error) in
            // Handle error
        }
    }
}
class ExtensionDelegate: NSObject, WKExtensionDelegate {

    func handleActiveWorkoutRecovery() {
        let healthStore = HKHealthStore()
        healthStore.recoverActiveWorkoutSession { (session, error) in
            // Handle error
        }
    }
}
New Quantity Series API

Niharika Bedekar, Fitness Software Engineer
HKCumulativeQuantitySeriesSample
HKQuantitySample

HKCumulativeQuantitySeriesSample
Quantity Series API

Use cases
Quantity Series API

Use cases

Data Visualization
Quantity Series API

Use cases

Data Visualization

HKStatisticsQuery
HKStatisticsCollectionQuery
Quantity Series API
Use cases

Data Visualization
HKStatisticsQuery
HKStatisticsCollectionQuery

Data Analysis
Quantity Series API

Use cases

Data Visualization

HKStatisticsQuery
HKStatisticsCollectionQuery

Data Analysis

HKQuantitySeriesSampleQuery
Quantity Series API

Use cases

- Data Visualization
  - HKStatisticsQuery
  - HKStatisticsCollectionQuery

- Data Analysis
  - HKQuantitySeriesSampleQuery

- High Frequency Data Storage
Quantity Series API

Use cases

Data Visualization

Data Analysis

High Frequency Data Storage

HKStatisticsQuery
HKStatisticsCollectionQuery

HKQuantitySeriesSampleQuery

HKQuantitySeriesSampleBuilder
HKQuantitySeriesSampleQuery
HKQuantitySeriesSampleQuery
Implementation
HKQuantitySeriesSampleQuery
Implementation

// Step 1: Create array to hold HKQuantity objects
var quantities = [HKQuantity]()
// Step 1: Create array to hold HKQuantity objects
var quantities = [HKQuantity]()

// Step 2: Initialize your query and gather data in handler
let seriesQuery = HKQuantitySeriesSampleQuery(quantitySample: seriesSample) {
    (query, quantity, date, success, error) in
    analyzeQuantity(quantity: quantity)
}

// Step 1: Create array to hold HKQuantity objects
var quantities = [HKQuantity]()

// Step 2: Initialize your query and gather data in handler
let seriesQuery = HKQuantitySeriesSampleQuery(quantitySample: seriesSample) {
    (query, quantity, date, success, error) in
    analyzeQuantity(quantity: quantity)
}

// Step 3: Execute the query
self.healthStore.execute(seriesQuery)
HKQuantitySeriesSampleBuilder
HKQuantitySeriesSampleBuilder

Implementation
HKQuantitySeriesSampleBuilder
Implementation

// Step 1: Create an HKQuantitySeriesSampleBuilder
let seriesBuilder = HKQuantitySeriesSampleBuilder(healthStore: healthStore, quantityType: quantityType, startDate: Date(), device: nil)
// Step 1: Create an HKQuantitySeriesSampleBuilder
let seriesBuilder = HKQuantitySeriesSampleBuilder(healthStore: healthStore, quantityType: quantityType, startDate: Date(), device: nil)

// Step 2: Insert HKQuantity objects
while shouldInsert {
    let quantity = HKQuantity(unit: .meter(), doubleValue: lastDistance)
    seriesBuilder.insert(quantity) { (success, error) in
        // Handle errors...
    }
}
// Step 1: Create an HKQuantitySeriesSampleBuilder
let seriesBuilder = HKQuantitySeriesSampleBuilder(healthStore: healthStore, quantityType: quantityType, startDate: Date(), device: nil)

// Step 2: Insert HKQuantity objects
while shouldInsert {
    let quantity = HKQuantity(unit: .meter(), doubleValue: lastDistance)
    seriesBuilder.insert(quantity) { (success, error) in
        // Handle errors…
    }
}

// Step 3: Complete the series
seriesBuilder.finishSeries(metadata: nil) { (quantity, error) in
    // Handle errors…
}
Summary
Summary

Respect User Privacy
Summary

Respect User Privacy

Quickly Build Robust Workout Apps
Summary

Respect User Privacy

Quickly Build Robust Workout Apps

Save Data Efficiently
More Information


<table>
<thead>
<tr>
<th>Event</th>
<th>Location</th>
<th>Date, Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Health and Fitness Technologies Lab</td>
<td>Technology Lab 4</td>
<td>Wednesday, 1:00PM</td>
</tr>
<tr>
<td>Health, Fitness, and Research Get-Together</td>
<td>Market Terrace</td>
<td>Wednesday, 6:15PM</td>
</tr>
<tr>
<td>Accessing Health Records with HealthKit</td>
<td></td>
<td>WWDC 2018</td>
</tr>
</tbody>
</table>