What’s New in Health

Session 221

Alexa VanHattum, iOS Software Engineer
Michael Ozeryansky, iOS Software Engineer
New HealthKit types
Workout API updates
Sync identifiers
Supporting diabetes management
New HealthKit types
Workout API updates
Sync identifiers
Supporting diabetes management
New HealthKit types
Workout API updates
Sync identifiers
Supporting diabetes management
New HealthKit types
Workout API updates
Sync identifiers
Supporting diabetes management
New HealthKit types
Workout API updates
Sync identifiers
Supporting diabetes management
New HealthKit Types
Sample Types
Sample Types

Workout route

• HKWorkoutRouteTypeIdentifier
Sample Types

Workout route
• HKWorkoutRouteTypeIdentifier

Waist circumference
• HKQuantityTypeIdentifierWaistCircumference
Sample Types

Workout route
• HKWorkoutRouteTypeIdentifier

Waist circumference
• HKQuantityTypeIdentifierWaistCircumference

VO₂ max
• HKQuantityTypeIdentifierVO2Max
Sample Types

Workout route
- HKWorkoutRouteTypeIdentifier

Waist circumference
- HKQuantityTypeIdentifierWaistCircumference

VO₂ max
- HKQuantityTypeIdentifierVO2Max

Insulin delivery
- HKQuantityTypeIdentifierInsulinDelivery
Workout Activity Types

Tai chi

- HKWorkoutRouteType.Taichi
Workout Activity Types

Tai chi
• HKWorkoutRouteType.Taichi

Mixed cardio
• HKWorkoutActivityType.MixedCardio
Tai chi
• HKWorkoutRouteType.Taichi

Mixed cardio
• HKWorkoutActivityType.MixedCardio

Hand cycling
• HKWorkoutActivityType.HandCycling
Workout API Updates
Swimming, segments, and pause/resume
Swimming
Tracking with Apple Watch

Support for pool and open water
Swimming
Tracking with Apple Watch

Support for pool and open water
Automatic swimming metrics
Swimming
Tracking with Apple Watch

Support for pool and open water

Automatic swimming metrics
• Swimming distance
Swimming
Tracking with Apple Watch

Support for pool and open water

Automatic swimming metrics
  • Swimming distance
  • Stroke count
Swimming
Tracking with Apple Watch

Support for pool and open water

Automatic swimming metrics
  • Swimming distance
  • Stroke count
  • Individual lap detection
Swimming
Tracking with Apple Watch

Support for pool and open water

Automatic swimming metrics
• Swimming distance
• Stroke count
• Individual lap detection
• Per-lap stroke style detection
Swimming
Tracking with Apple Watch

Support for pool and open water

Automatic swimming metrics
- Swimming distance
- Stroke count
- Individual lap detection
- Per-lap stroke style detection
- Set detection
Swimming
Tracking with Apple Watch

Support for pool and open water

Automatic swimming metrics
- Swimming distance
- Stroke count
- Individual lap detection
- Per-lap stroke style detection
- Set detection

Apps can enable water lock
public let HKMetadataKeySwimmingLocationType: String
public let HKMetadataKeySwimmingLocationType: String
public enum HKWorkoutSwimmingLocationType : Int {
    case Unknown
    case Pool
    case OpenWater
}
public let HKMetadataKeySwimmingStrokeStyle: String
public let HKMetadataKeySwimmingStrokeStyle: String
public enum HKSwimmingStrokeStyle : Int {
    case Unknown
    case Mixed
    case Freestyle
    case Backstroke
    case Breaststroke
    case Butterfly
}
Swimming
Workout configuration

```swift
let workoutConfiguration = HKWorkoutConfiguration()
```
let workoutConfiguration = HKWorkoutConfiguration()

workoutConfiguration.activityType = HKWorkoutActivityType.swimming
Swimming
Workout configuration

```swift
let workoutConfiguration = HKWorkoutConfiguration()

workoutConfiguration.activityType = HKWorkoutActivityType.swimming
workoutConfiguration.swimmingLocationType = HKWorkoutSwimmingLocationType.pool
```
let workoutConfiguration = HKWorkoutConfiguration()

workoutConfiguration.activityType = HKWorkoutActivityType.swimming
workoutConfiguration.swimmingLocationType = HKWorkoutSwimmingLocationType.pool
workoutConfiguration.lapLength = HKQuantity(unit: .yard(), doubleValue: 25)
let workoutConfiguration = HKWorkoutConfiguration()

workoutConfiguration.activityType = HKWorkoutActivityType.swimming
workoutConfiguration.swimmingLocationType = HKWorkoutSwimmingLocationType.pool
workoutConfiguration.lapLength = HKQuantity(unit: .yard(), doubleValue: 25)

do {
    let workoutSession = try HKWorkoutSession(configuration: workoutConfiguration)
}
catch let error {
    // Handle error...
}
let workoutConfiguration = HKWorkoutConfiguration()

workoutConfiguration.activityType = HKWorkoutActivityType.swimming
workoutConfiguration.swimmingLocationType = HKWorkoutSwimmingLocationType.pool
workoutConfiguration.lapLength = HKQuantity(unit: .yard(), doubleValue: 25)

do {
    let workoutSession = try HKWorkoutSession(configuration: workoutConfiguration)
    workoutSession.delegate = self
}

} catch let error {
    // Handle error...
}
let workoutConfiguration = HKWorkoutConfiguration()

workoutConfiguration.activityType = HKWorkoutActivityType.swimming
workoutConfiguration.swimmingLocationType = HKWorkoutSwimmingLocationType.pool
workoutConfiguration.lapLength = HKQuantity(unit: .yard(), doubleValue: 25)

do {
    let workoutSession = try HKWorkoutSession(configuration: workoutConfiguration)
    workoutSession.delegate = self
    healthStore.start(workoutSession)
    // ...
} catch let error {
    // Handle error...
}
func workoutSession(_ workoutSession: HKWorkoutSession,
    didChangeTo toState: HKWorkoutSessionState,
    from fromState: HKWorkoutSessionState,
    date: Date) {

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

    switch (fromState, toState) {

    }

}
func workoutSession(_ workoutSession: HKWorkoutSession, didChangeTo toState: HKWorkoutSessionState, from fromState: HKWorkoutSessionState, date: Date) {
    switch (fromState, toState) {
    case (.notStarted, .running):
    }
}
func workoutSession(_ workoutSession: HKWorkoutSession,
               didChangeTo toState: HKWorkoutSessionState,
               from fromState: HKWorkoutSessionState,
               date: Date) {

    switch (fromState, toState) {
    case (.notStarted, .running):
        let wkExtension = WKExtension.shared()
        wkExtension.enableWaterLock()

        // ...
    }
}
HKWorkoutEvent

Highlight a specific time in the workout
HKWorkoutEvent

Highlight a specific time in the workout

Used for pausing, resuming, laps, and markers
HKWorkoutEvent

Highlight a specific time in the workout

Used for pausing, resuming, laps, and markers

Created by HealthKit or your app
HKWorkoutEvent

Highlight a specific time in the workout

Used for pausing, resuming, laps, and markers

Created by HealthKit or your app

Save a list on HKWorkout
HKWorkoutEvent

Highlight a specific time in the workout

Used for pausing, resuming, laps, and markers

Created by HealthKit or your app

Save a list on HKWorkout

Affect the workout’s duration
Swimming
Observing lap events

// In your workout session’s delegate
func workoutSession(_ workoutSession: HKWorkoutSession, didGenerate event: HKWorkoutEvent) {

}
Swimming
Observing lap events

// In your workout session’s delegate
func workoutSession(_ workoutSession: HKWorkoutSession, didGenerate event: HKWorkoutEvent) {
    switch event.type {
    }
}
// In your workout session's delegate
func workoutSession(_ workoutSession: HKWorkoutSession, didGenerate event: HKWorkoutEvent) {
    switch event.type {
    case .lap:
    }
}
// In your workout session’s delegate
func workoutSession(_ workoutSession: HKWorkoutSession, didGenerate event: HKWorkoutEvent) {
    switch event.type {
    case .lap:
        lapCount += 1
    }
}
In your workout session’s delegate

```swift
func workoutSession(_ workoutSession: HKWorkoutSession, didGenerate event: HKWorkoutEvent) {
    switch event.type {
    case .lap:
        lapCount += 1
        if let strokeStyle = event.metadata?[HKMetadataKeySwimmingStrokeStyle] {
            self.displayCurrentStrokeStyle(strokeStyle)
        }
    // ...
    }
}
```
New Workout Events
New Workout Events

// HKWorkout.h

public enum HKWorkoutEventType : Int {
    case pause
    case resume
    case lap
    case marker
    case motionPaused
    case motionResumed
}

New Workout Events

```swift
// HKWorkout.h

public enum HKWorkoutEventType : Int {
    case pause
    case resume
    case lap
    case marker
    case motionPaused
    case motionResumed
    case segment
    case pauseOrResumeRequest
}
```
public enum HKWorkoutEventType : Int {
    case pause
    case resume
    case lap
    case marker
    case motionPaused
    case motionResumed
    case segment
    case pauseOrResumeRequest
}
open class HKWorkoutEvent : NSObject, NSSecureCoding, NSCopying {

    open var date: Date { get }

    public convenience init(type: HKWorkoutEventType, date: Date, metadata: [String : Any])

}
open class HKWorkoutEvent : NSObject, NSSecureCoding, NSCopying {

    open var date: Date { get }
    open var dateInterval: DateInterval { get }

    public convenience init(type: HKWorkoutEventType, date: Date, metadata: [String : Any])
    public convenience init(type: HKWorkoutEventType, dateInterval: DateInterval, metadata: [String : Any]?)
}

On HKWorkoutEvent: date → dateInterval
Start workout
Start workout

.type
date interval

{metadata}
Start workout

.type
2:00
{freestyle}

.date interval
{metadata}
Start workout

- **Lap 1:**
  - Time: 2:00
  - Type: freestyle

- **Lap 2:**
  - Time: 3:00
  - Type: butterfly
Start workout

.lap 3:00 { .freestyle }

.lap 3:00 { .butterfly }

.pause 3:30

{}
Start workout

**.type**
date interval
{metadata}

**.segment**
(0, 3:30)
{}

**.lap**
2:00
{.freestyle}

3:00
{.butterfly}

**.pause**
3:30
{}

**.resume**
4:30
{ }
Start workout

Pause
3:30

Resume
4:30

Lap 2:00
{freestyle}

Lap 3:00
{butterfly}

Lap 6:30
{freestyle}

Segment
(0, 3:30)
{metadata}

type
date interval
{metadata}
Start workout

1. pause 3:30
2. pause
3. resume 4:30
4. resume

1. lap 2:00 {freestyle}
2. lap 3:00 {butterfly}
3. lap 6:30 {freestyle}

1. segment (0, 3:30) {}
2. segment (4:30, 6:30) {}

.type
date interval
{metadata}
Workout Request Pause/Resume

New gesture for pausing and resuming workouts
Workout Request Pause/Resume

New gesture for pausing and resuming workouts
Quick press of the Digital Crown and side button
Workout Request Pause/Resume

New gesture for pausing and resuming workouts
Quick press of the Digital Crown and side button
Does work in water lock
Workout Request Pause/Resume

New gesture for pausing and resuming workouts
Quick press of the Digital Crown and side button
Does work in water lock
Handle in your workout session delegate
User
Presses Digital Crown and side button

HealthKit

Your app
<table>
<thead>
<tr>
<th>User</th>
<th>HealthKit</th>
<th>Your app</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presses Digital Crown and side button</td>
<td>Generates pauseOrResumeRequest</td>
<td></td>
</tr>
</tbody>
</table>
User

Presses Digital Crown and side button

---

HealthKit

Generates pauseOrResumeRequest

---

Your app

Receives request event in workout session delegate
User

Presses Digital Crown and side button

HealthKit

Generates pauseOrResumeRequest

Your app

Receives request event in workout session delegate

Based on state, calls pause or resume on health store
User

Presses Digital Crown and side button

Generates pauseOrResumeRequest

Receives request event in workout session delegate

Based on state, calls pause or resume on health store

Generates pause event or resume event

Your app
<table>
<thead>
<tr>
<th>User</th>
<th>HealthKit</th>
<th>Your app</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presses Digital Crown and side button</td>
<td>Generates <code>pauseOrResumeRequest</code></td>
<td>Receives request event in workout session delegate</td>
</tr>
<tr>
<td>Generates pause event or resume event</td>
<td>Based on state, calls pause or resume on health store</td>
<td>Receives pause event or resume event in delegate</td>
</tr>
</tbody>
</table>
Workout Routes
Reading Workout Routes

New data type

HKWorkoutRouteType
Reading Workout Routes

New data type

HKWorkoutRouteType

Requires additional authorization
Reading Workout Routes

New data type

HKWorkoutRouteType

Requires additional authorization

Modeled as an array of CLLocations
Reading Workout Routes
New data type

**HKWorkoutRouteType**
Requires additional authorization
Modeled as an array of CLLocations
Datasets can be large
Reading Workout Routes

New data type

HKWorkoutRouteType

Requires additional authorization

Modeled as an array of CLLocations

Datasets can be large

• New HKWorkoutRouteQuery
Reading Workout Routes
New data type

**HKWorkoutRouteType**
Requires additional authorization
Modeled as an array of CLLocations
Datasets can be large
• New **HKWorkoutRouteQuery**
  • Returns location data in batches
// Step 1: Query for samples of type HKWorkoutRoute associated to your workout
// Step 1: Query for samples of type HKWorkoutRoute associated to your workout
let workoutRouteType = HKSeriesType.workoutRoute()
let workoutPredicate = HKQuery.predicateForObjects(from: workout)
// Step 1: Query for samples of type HKWorkoutRoute associated to your workout
let workoutRouteType = HKSeriesType.workoutRoute()
let workoutPredicate = HKQuery.predicateForObjects(from: workout)

let workoutRoutesQuery = HKSampleQuery(sampleType: workoutRouteType,
predicate: workoutPredicate, limit: HKObjectQueryNoLimit, sortDescriptors: nil)
{ (query, samples, error) in
}

}
// Step 1: Query for samples of type HKWorkoutRoute associated to your workout
let workoutRouteType = HKSeriesType.workoutRoute()
let workoutPredicate = HKQuery.predicateForObjects(from: workout)

let workoutRoutesQuery = HKSampleQuery(sampleType: workoutRouteType, predicate: workoutPredicate, limit: HKObjectQueryNoLimit, sortDescriptors: nil)
{(query, samples, error) in
guard let routeSamples = samples as? [HKWorkoutRoute] else { return }
}
// Step 1: Query for samples of type HKWorkoutRoute associated to your workout
let workoutRouteType = HKSeriesType.workoutRoute()
let workoutPredicate = HKQuery.predicateForObjects(from: workout)

let workoutRoutesQuery = HKSampleQuery(sampleType: workoutRouteType,
                                        predicate: workoutPredicate, limit: HKObjectQueryNoLimit, sortDescriptors: nil)
{ (query, samples, error) in
  guard let routeSamples = samples as? [HKWorkoutRoute] else { return }

  // Step 2: Query for location data from the routes
  for routeSample in routeSamples {
    
  }
}

// Step 1: Query for samples of type HKWorkoutRoute associated to your workout
let workoutRouteType = HKSeriesType.workoutRoute()
let workoutPredicate = HKQuery.predicateForObjects(from: workout)

let workoutRoutesQuery = HKSampleQuery(sampleType: workoutRouteType,
predicate: workoutPredicate, limit: HKObjectQueryNoLimit, sortDescriptors: nil)
{ (query, samples, error) in
  guard let routeSamples = samples as? [HKWorkoutRoute] else { return }

  // Step 2: Query for location data from the routes
  for routeSample in routeSamples {
    let locationQuery = HKWorkoutRouteQuery(route: routeSample) {
      (routeQuery, locations, done, error) in

    }
  }
}
// Step 1: Query for samples of type HKWorkoutRoute associated to your workout

let workoutRouteType = HKSeriesType.workoutRoute()
let workoutPredicate = HKQuery.predicateForObjects(from: workout)

let workoutRoutesQuery = HKSampleQuery(sampleType: workoutRouteType, predicate: workoutPredicate, limit: HKObjectQueryNoLimit, sortDescriptors: nil) { (query, samples, error) in
  guard let routeSamples = samples as? [HKWorkoutRoute] else { return }

  // Step 2: Query for location data from the routes
  for routeSample in routeSamples {
    let locationQuery = HKWorkoutRouteQuery(route: routeSample) {
      (routeQuery, locations, done, error) in
      self.addLocationsToMapDisplay(locations)
    }
  }
}
// Step 1: Query for samples of type HKWorkoutRoute associated to your workout
let workoutRouteType = HKSeriesType.workoutRoute()
let workoutPredicate = HKQuery.predicateForObjects(from: workout)

let workoutRoutesQuery = HKSampleQuery(sampleType: workoutRouteType, predicate: workoutPredicate, limit: HKObjectQueryNoLimit, sortDescriptors: nil) { (query, samples, error) in
    guard let routeSamples = samples as? [HKWorkoutRoute] else { return }

    // Step 2: Query for location data from the routes
    for routeSample in routeSamples {
        let locationQuery = HKWorkoutRouteQuery(route: routeSample) {
            (routeQuery, locations, done, error) in
                self.addLocationsToMapDisplay(locations)
        }
        self.healthStore.execute(locationQuery)
    }
}

self.healthStore.execute(workoutRoutesQuery)
Building and Saving Workout Routes

Builder model—HKWorkoutRouteBuilder
Building and Saving Workout Routes

Builder model—HKWorkoutRouteBuilder

Location data is added asynchronously
Building and Saving Workout Routes

Builder model—HKWorkoutRouteBuilder

Location data is added asynchronously

Data is sorted by date when the series is finished
Building and Saving Workout Routes

Builder model—HKWorkoutRouteBuilder

Location data is added asynchronously

Data is sorted by date when the series is finished

The workout must be saved before the route
Create workout route builder
Create workout route builder
Create workout route builder → Start workout session
Create workout route builder → Start workout session → Active Workout
Create workout route builder → Start workout session → Active Workout

- Observe location data
Create workout route builder → Start workout session → Observe location data → Add locations to builder → End session, save workout → Finish workout route
// Step 1: Create a route builder and add locations
let builder = HKWorkoutRouteBuilder(healthStore: healthStore, device: nil)
// Step 1: Create a route builder and add locations
let builder = HKWorkoutRouteBuilder(healthStore: healthStore, device: nil)

// Step 2: Add locations as the workout is ongoing
let locations: [CLLocation] = self.fetchRecentLocations()
builder.insertRouteData(locations) { (success, error) in
    // Handle errors...
}
// Step 1: Create a route builder and add locations
let builder = HKWorkoutRouteBuilder(healthStore: healthStore, device: nil)

// Step 2: Add locations as the workout is ongoing
let locations: [CLLocation] = self.fetchRecentLocations()
builder.insertRouteData(locations) { (success, error) in
    // Handle errors...
}

// Step 3: After the workout is saved, save the route data
builder.finishRoute(with: workout, metadata: nil) { (workoutRoute, error) in
    // Handle errors...
}
Workout Route Demo
Incorporating routes into Speedy Sloth
HKObject Sync Identifiers

Michael Ozeryansky, iOS Software Engineer
HKObject Sync Identifiers
Identifiers and versioning

public let HKMetadataKeySyncIdentifier: String
public let HKMetadataKeySyncVersion: String
HKObject Sync Identifiers
Identifiers and versioning

public let HKMetadataKeySyncIdentifier: String
public let HKMetadataKeySyncVersion: String

Identifier can be any String
HKObject Sync Identifiers
Identifiers and versioning

public let HKMetadataKeySyncIdentifier: String
public let HKMetadataKeySyncVersion: String

Identifier can be any String
Version can be any Number
HKObject Sync Identifiers
Identifiers and versioning

public let HKMetadataKeySyncIdentifier: String
public let HKMetadataKeySyncVersion: String

Identifier can be any String
Version can be any Number
Use both keys together when saving an HKObject
HKObject Sync Identifiers
Identifiers and versioning

```swift
public let HKMetadataKeySyncIdentifier: String
public let HKMetadataKeySyncVersion: String
```

Identifier can be any String

Version can be any Number

Use both keys together when saving an HKObject

Restricted to your source
HKObject Sync Identifiers
Identifiers and versioning

NEW
HKObject Sync Identifiers
Identifiers and versioning

Sample uniqueness
HKObject Sync Identifiers
Identifiers and versioning

Sample uniqueness

Local versioning
HKObject Sync Identifiers
Identifiers and versioning

Sample uniqueness
Local versioning
Transaction safe
HKObject Sync Identifiers
Identifiers and versioning

Sample uniqueness
Local versioning
Transaction safe
Relationships are maintained
Demo

Updating samples using sync identifiers
Sample Source Information

Understanding the context and fidelity of HealthKit data
New Properties on HKSourceRevision
Operating system version and product type

```swift
open class HKSourceRevision : NSObject, NSSecureCoding, NSCopying {

    open var source: HKSource { get }
    open var version: NSString? { get }

}
```
New Properties on HKSourceRevision
Operating system version and product type

open class HKSourceRevision : NSObject, NSSecureCoding, NSCopying {

    open var source: HKSource { get }
    open var version: NSString? { get }
    open var productType: String? { get } // e.g. "watch2,4"

}
New Properties on HKSourceRevision
Operating system version and product type

```swift
open class HKSourceRevision : NSObject, NSSecureCoding, NSCopying {

    open var source: HKSource { get }
    open var version: NSString? { get }
    open var productType: String? { get } // e.g. "watch2,4"
    open var operatingSystemVersion: OperatingSystemVersion { get } // e.g. {4, 0, 0}
}
```
New Properties on HKSourceRevision
Operating system version and product type

```swift
open class HKSourceRevision : NSObject, NSSecureCoding, NSCopying {

    open var source: HKSource { get }
    open var version: NSString? { get }
    open var productType: String? { get } // e.g. “watch2,4”
    open var operatingSystemVersion: OperatingSystemVersion { get } // e.g. {4, 0, 0}
}

public let HKSourceRevisionAnyVersion: String
public let HKSourceRevisionAnyProductType: String
public let HKSourceRevisionAnyOperatingSystem: OperatingSystemVersion
```
Supporting Diabetes Management
Supporting Diabetes Management

What’s New in Core Bluetooth

Grand Ballroom B  Thursday 11:00AM
Supporting Diabetes Management

Blood glucose meal time

What's New in Core Bluetooth

Grand Ballroom B
Thursday 11:00AM
Supporting Diabetes Management

Blood glucose meal time

Insulin support

What's New in Core Bluetooth
Grand Ballroom B
Thursday 11:00AM
Supporting Diabetes Management

Blood glucose meal time

Insulin support

CoreBluetooth in watchOS 4
Supporting Diabetes Management
Blood glucose meal time

```swift
public let HKMetadataKeyBloodGlucoseMealTime: String
```
Supporting Diabetes Management
Blood glucose meal time

```swift
public let HKMetadataKeyBloodGlucoseMealTime: String

public enum HKBloodGlucoseMealTime: Int {
    case preprandial
    case postprandial
}
```
Supporting Diabetes Management
Blood glucose meal time

```swift
public let HKMetadataKeyBloodGlucoseMealTime: String

public enum HKBloodGlucoseMealTime: Int {
    case preprandial
    case postprandial
}
```

Time relative to a meal
Supporting Diabetes Management

Insulin delivery
Supporting Diabetes Management

Insulin delivery
Supporting Diabetes Management
Insulin delivery

```
public static let insulinDelivery: HKQuantityTypeIdentifier
```
Supporting Diabetes Management

Insulin delivery

```swift
public static let insulinDelivery: HKQuantityTypeIdentifier

public let HKMetadataKeyInsulinDeliveryReason: String
```
public static let insulinDelivery: HKQuantityTypeIdentifier

public let HKMetadataKeyInsulinDeliveryReason: String

public enum HKInsulinDeliveryReason : Int {
    case basal
    case bolus
}
Supporting Diabetes Management
Insulin delivery

```swift
public static let insulinDelivery: HKQuantityTypeIdentifier

public let HKMetadataKeyInsulinDeliveryReason: String

public enum HKInsulinDeliveryReason : Int {
    case basal
    case bolus
}
```

Insulin that has been delivered
Supporting Diabetes Management

Insulin delivery

```swift
public static let insulinDelivery: HKQuantityTypeIdentifier

public let HKMetadataKeyInsulinDeliveryReason: String

public enum HKInsulinDeliveryReason : Int {
    case basal
    case bolus
}
```

Insulin that has been delivered

International unit
Supporting Diabetes Management

International unit

extension HKUnit {
    open class func internationalUnit() -> Self
}

NEW
extension HKUnit {
    open class func internationalUnit() -> Self
}

Biological effectiveness
Supporting Diabetes Management

International unit

```swift
extension HKUnit {
    open class func internationalUnit() -> Self
}
```

Biological effectiveness

Cannot be converted to other units
// Add Basal Insulin Sample From an Insulin Pump
// Add Basal Insulin Sample From an Insulin Pump

// Step 1: Create an insulin delivery quantity type
let quantityType = HKQuantityType.quantityType(forIdentifier: .insulinDelivery)!
// Add Basal Insulin Sample From an Insulin Pump

// Step 1: Create an insulin delivery quantity type
let quantityType = HKQuantityType.quantityType(forIdentifier: .insulinDelivery)!

// Step 2: Create a quantity of 0.825 units
let quantity = HKQuantity(unit: .internationalUnit(), doubleValue: 0.825)
// Add Basal Insulin Sample From an Insulin Pump

// Step 3: Create a quantity sample
let insulinSample = HKQuantitySample(
    type: quantityType,
    quantity: quantity,
    start: pumpDeliveryStartDate,
    end: pumpDeliveryEndDate,
    metadata: [
    ]
)
// Add Basal Insulin Sample From an Insulin Pump

// Step 3: Create a quantity sample
let insulinSample = HKQuantitySample(
    type: quantityType,
    quantity: quantity,
    start: pumpDeliveryStartDate,
    end: pumpDeliveryEndDate,
    metadata: [
        HKMetadataKeyInsulinDeliveryReason: HKInsulinDeliveryReason.basal.rawValue
    ]
)
// Add Basal Insulin Sample From an Insulin Pump

// Step 3: Create a quantity sample
let insulinSample = HKQuantitySample(
    type: quantityType,
    quantity: quantity,
    start: pumpDeliveryStartDate,
    end: pumpDeliveryEndDate,
    metadata: [
        HKMetadataKeyInsulinDeliveryReason: HKInsulinDeliveryReason.basal.rawValue
    ]
)

// Step 4: Save the new sample
healthStore.save(insulinSample) { success, error in }
// Statistics Query for Basal Samples
Statistics Query for Basal Samples

Step 1: Setup the query

```swift
let predicate = HKQuery.predicateForObjects(withMetadataKey: HKMetadataKeyInsulinDeliveryReason, allowedValues: [HKInsulinDeliveryReason.basal.rawValue])
```
// Statistics Query for Basal Samples

// Step 1: Setup the query
let predicate = HKQuery.predicateForObjects(withMetadataKey: HKMetadataKeyInsulinDeliveryReason,
allowedValues: [HKInsulinDeliveryReason.basal.rawValue])

let quantityType = HKQuantityType.quantityType(forIdentifier: .insulinDelivery)!
let predicate = HKQuery.predicateForObjects(withMetadataKey: HKMetadataKeyInsulinDeliveryReason, allowedValues: [HKInsulinDeliveryReason.basal.rawValue])

let quantityType = HKQuantityType.quantityType(forIdentifier: .insulinDelivery)!

let query = HKStatisticsCollectionQuery(quantityType: quantityType, quantitySamplePredicate: predicate, options: anchorDate: intervalComponents: )
let predicate = HKQuery.predicateForObjects(withMetadataKey: HKMetadataKeyInsulinDeliveryReason, allowedValues: [HKInsulinDeliveryReason.basal.rawValue])

let quantityType = HKQuantityType.quantityType(forIdentifier: .insulinDelivery)!

let query = HKStatisticsCollectionQuery(quantityType: quantityType, quantitySamplePredicate: predicate, options: [.cumulativeSum, .separateBySource], anchorDate: intervalComponents: )
// Statistics Query for Basal Samples

// Step 1: Setup the query
let predicate = HKQuery.predicateForObjects(withMetadataKey: HKMetadataKeyInsulinDeliveryReason,
                                          allowedValues: [HKInsulinDeliveryReason.basal.rawValue])

let quantityType = HKQuantityType.quantityType(forIdentifier: .insulinDelivery)!

let query = HKStatisticsCollectionQuery(quantityType: quantityType,
                                          quantitySamplePredicate: predicate,
                                          options: [.cumulativeSum, .separateBySource],
                                          anchorDate: Date.distantPast,
                                          intervalComponents: )
// Statistics Query for Basal Samples

// Step 1: Setup the query
let predicate = HKQuery.predicateForObjects(withMetadataKey: HKMetadataKeyInsulinDeliveryReason, allowedValues: [HKInsulinDeliveryReason.basal.rawValue])
let quantityType = HKQuantityType.quantityType(forIdentifier: .insulinDelivery)!

let query = HKStatisticsCollectionQuery(quantityType: quantityType, quantitySamplePredicate: predicate, options: [.cumulativeSum, .separateBySource], anchorDate: Date.distantPast, intervalComponents: DateComponents(hour: 1))
// Statistics Query for Basal Samples

// Step 2: Set the results handler
query.initialResultsHandler = { query, results, error in
    // Process statistics
}
// Statistics Query for Basal Samples

// Step 2: Set the results handler
query.initialResultsHandler = { query, results, error in
    // Process statistics
}

// Step 3: Execute the query
healthStore.execute(query)
Summary

Expand reach with new data types
Summary

Expand reach with new data types

Build engaging workout experiences
Summary

Expand reach with new data types

Build engaging workout experiences

Prevent data duplication with sync identifiers
Summary

Expand reach with new data types
Build engaging workout experiences
Prevent data duplication with sync identifiers
Support users managing diabetes
More Information

https://developer.apple.com/wwdc17/221
<table>
<thead>
<tr>
<th>Session Title</th>
<th>Location</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating Immersive Apps with Core Motion</td>
<td>Grand Ballroom B</td>
<td>Tuesday 4:10PM</td>
</tr>
<tr>
<td>What’s New in Core Bluetooth</td>
<td>Grand Ballroom B</td>
<td>Thursday 11:00AM</td>
</tr>
<tr>
<td>What’s New in Location Technologies</td>
<td>Executive Ballroom</td>
<td>Thursday 3:10PM</td>
</tr>
<tr>
<td>What’s New in CareKit and ResearchKit</td>
<td>Grand Ballroom A</td>
<td>Thursday 5:10PM</td>
</tr>
</tbody>
</table>
# Labs

<table>
<thead>
<tr>
<th>Lab</th>
<th>Location</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Health, Fitness, and Research Get-Together</td>
<td>Grand Ballroom A</td>
<td>Wed 6:30PM–7:45PM</td>
</tr>
<tr>
<td>HealthKit Lab</td>
<td>Technology Lab H</td>
<td>Thur 9:00AM–12:00PM</td>
</tr>
<tr>
<td>WatchConnectivity and WatchKit Lab</td>
<td>Technology Lab B</td>
<td>Fri 9:00AM–11:00AM</td>
</tr>
<tr>
<td>ResearchKit and CareKit Lab</td>
<td>Technology Lab H</td>
<td>Fri 11:00AM–1:00PM</td>
</tr>
</tbody>
</table>