Article

# About the HealthKit framework

Learn about the architecture and design of the HealthKit framework.

## Overview

Share health and fitness data between apps using the HealthKit framework. Rather than developers creating custom data types and units, HealthKit constrains data types and units to a predefined list. This ensures that all apps understand what the data means and how they can use it.

Additionally, the framework uses a large number of subclasses, producing deep hierarchies of similar classes. Often, these classes have subtle but important differences between them. For example, you use an `HKQuantitySample` object to store data with a numeric value and an `HKCategorySample` object to store a value selected from an enumeration.

HealthKit also uses pairs of closely related classes that you need to use together. For example, the `HKObject` and `HKObjectType` abstract classes have largely parallel hierarchies of concrete subclasses. When working with objects and object types, you must use matching subclasses.

## HealthKit data

HealthKit saves a variety of data types in the HealthKit Store:

Characteristic data
: Characteristics that typically don't change, such as the user's birthdate, blood type, biological sex, and skin type. You can read this data directly from the HealthKit store, using the `dateOfBirth()`, `bloodType()`, `biologicalSex()`, and `fitzpatrickSkinType()` methods. Your application can't save characteristic data. The user must enter or modify this data using the Health app.

Sample data

Samples that represent a measurement at a particular point in time. All sample classes are subclasses of the `HKSample` class, which is a subclass of the `HKObject` class. For more information, see Samples.

Workout data
: Samples that store information about fitness and exercise activities. While `HKWorkout` is a subclass of `HKSample`, it behaves somewhat differently than other sample subclasses. For more information, see Workout data.

Source data
: Information about a sample's source. The `HKSourceRevision` object contains information about the app or device that saved the sample. The `HKDevice` object contains information about the hardware device that generated the data.

Deleted objects
: An object that represents a sample after something deletes it from the HealthKit store. HealthKit uses an `HKDeletedObject` instance to temporarily store the UUID of deleted samples. You can use deleted objects to respond when the user or another app deletes an object. For more information, see `HKAnchoredObjectQuery` and `HKDeletedObject`.

## Properties of objects and samples

The `HKObject` class is the superclass of all HealthKit sample types. All `HKObject` subclasses are immutable. Each object has the following properties:

UUID
: A unique identifier for that particular entry.

Metadata
: A dictionary containing additional information about the entry. The metadata can contain both predefined and custom keys. The predefined keys facilitate the sharing of data between apps. Custom keys help extend a given HealthKit object type, adding app-specific data to the entry.

Source Revision
: The source of the sample. The source can be a device that directly saves data into HealthKit or an app. HealthKit automatically records each object's source and version when it saves the data to the HealthKit store. This property is available only on objects retrieved from the store.

Device
: The hardware device that generated the data stored in this sample.

The `HKSample` class is a subclass of `HKObject`. Sample objects represent data at a particular point in time, and all sample objects are subclasses of the `HKSample` class. They have the following properties:

Type
: The sample type, such as a sleep analysis sample, a height sample, or a step count sample.

Start date
: The sample's start time.

End date
: The sample's end time. If the sample represents a single point in time, the end time should equal the start time. If the sample represents data collected over a time interval, the end time should occur after the start time.

Samples are further divided into four concrete subclasses:

Category samples
: Data that can you can classify into a finite set of categories. See `HKCategorySample`.

Quantity samples
: Data that you can store as numeric values. Quantity samples are the most common data types in HealthKit. These include the user's height and weight, as well as other data such as the number of steps taken, the user's temperature, and their pulse rate. See `HKQuantitySample`.

Correlations
: Composite data containing one or more samples. HealthKit uses correlations to represent food and blood pressure. You should always use a correlation when creating food or blood pressure data. See `HKCorrelation`.

Workouts
: Data representing a physical activity, like running, swimming, or even play. Workouts often have *type*, *duration*, *distance*, and *energy burned* properties. You can also associate a workout with additional, fine-grained samples. Unlike correlations, the workout doesn't contain these samples; however, you can query for them using the workout. For more information, see `HKWorkout`.

# Threading

The HealthKit store is thread-safe, and most HealthKit objects are immutable. In general, you can use HealthKit safely in a multithreaded environment.

> **Note**
>
> All the HealthKit API's completion handlers execute on private background queues. You typically dispatch this data back to the main queue before updating your user interface or changing any other resources that you can only safely modify from the main thread.

For more information about multithreading and concurrent programming, see Concurrency Programming Guide.

## Syncing data between devices

iPhone, Apple Watch, and visionOS each have their own HealthKit store. iPadOS 17 and later also has its own HealthKit store. It is also available on iPadOS apps running on Vision Pro. HealthKit automatically syncs data between these devices. To save space, old data is periodically purged from Apple Watch. Use `earliestPermittedSampleDate()` to determine the earliest samples available on Apple Watch.

While the HealthKit framework is available on iPadOS 16 and earlier and on MacOS 13 and later, these devices don't have a copy of the HealthKit store. This means you can include HealthKit code in apps running on these devices, simplifying the creation of multiplatform apps. However, they can't read or write HealthKit data, and calls to `isHealthDataAvailable()` return `false`.

# See Also

## Essentials

- Setting up HealthKit
  Set up and configure your HealthKit store.
- Authorizing access to health data
  Request permission to read and share data in your app.
- Protecting user privacy
  Respect and safeguard your user's privacy.
- HealthKit updates
  Learn about important changes to HealthKit.
- HealthKitUI
  Display user interface that enables a person to view and interact with their health data.