# HealthKit via C# / Xamarin.iOS

Within a Xamarin.iOS project I needed to be able to access data stored in the Apple [Health](http://www.apple.com/ios/health/?cid=wwa-us-kwg-features) app.

The data stored in the default Apple Health app uses [HealthKit](https://developer.apple.com/library/ios/documentation/HealthKit/Reference/HealthKit_Framework/) as a backing store.

If you read through the HealthKit documentation, you can quickly become overwhelmed with the structure of the HealthKit API and the way in which data is stored and retrieved.

Let’s get you some resources and simplifications that should help pierce the obfuscation bubble of HealthKit.

In all the examples below I will be using C# / Xamarin.iOS HealthKit wrappers. Where possible I will point out the pure iOS implementation as well. In general, much of what is talked about below should help if you need Swift or Objective-C access to HealthKit stored data as well.

## Resources

Apple has a [great sample app named **Fit**](https://developer.apple.com/library/ios/samplecode/Fit/Introduction/Intro.html) coded in Objective-C that gives great examples against the HealthKit API surface.

* Most of my C# / Xamarin.iOS code was largely ported from the sample code present in the Fit sample app.

My [HealthKitStatelessSample](https://github.com/Batgar/HealthKitStatelessSample) out on Github

* Most of the code in this write up is out of a single file: [HealthKitDataStore.cs](https://github.com/Batgar/HealthKitStatelessSample/blob/master/HealthKitSample/DataStore/HealthKitDataStore.cs)

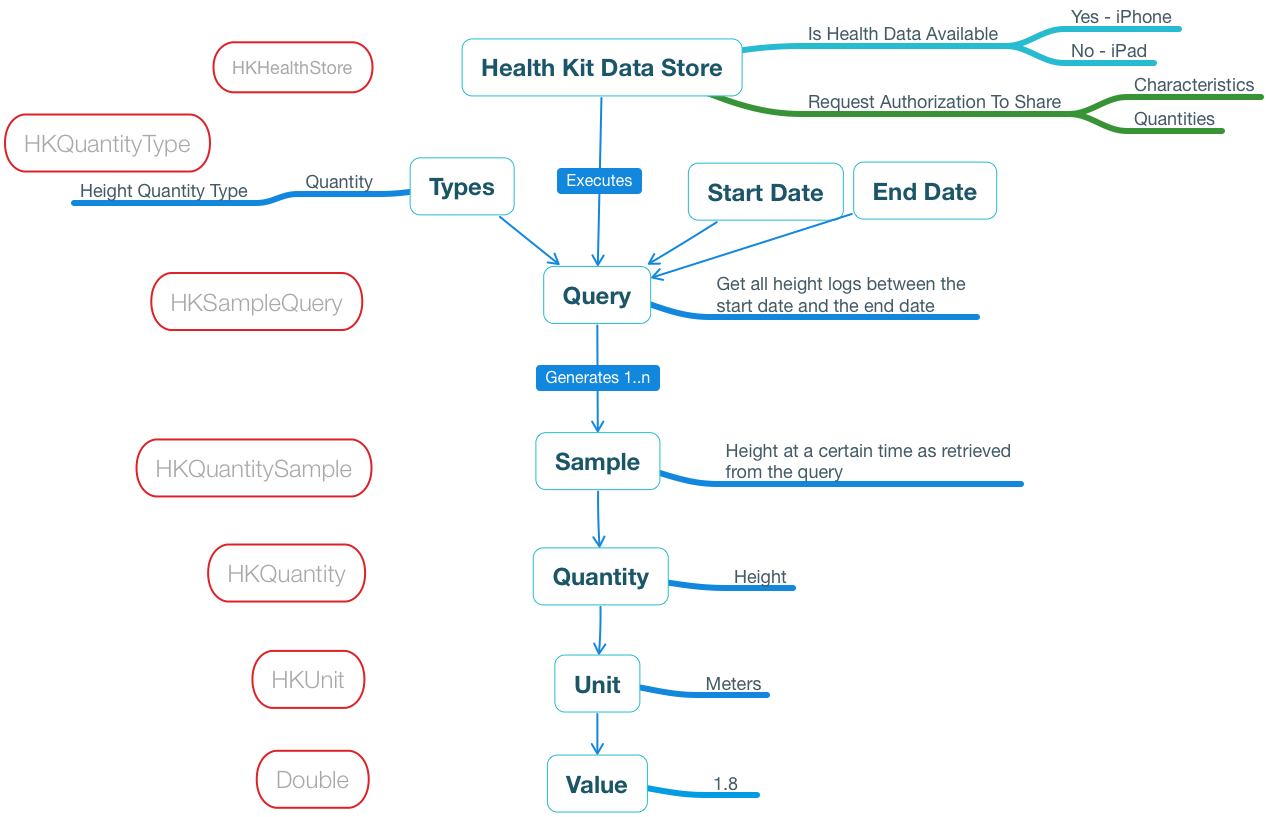
## General Health / HealthKit Structure

## 

* **HealthKit** – The primary API that all applications use to access shared health data.
* **App1, App2, Your App, …**
  + Each of the apps with purple borders above access data via HealthKit. Each of these apps has to request specific read and/or write access to each specific type of health data that is stored within HealthKit.
    - i.e. App 1 requests to read and write Blood Pressure.
  + Once an app has been granted access to a specific data point (i.e. Blood Pressure) then that app can access all the data points that all installed iOS applications have put into the central HealthKit store.
* **Apple Health App** – Is a highly privileged app against HealthKit. Health is where a user can:
  + Add and deny per-app permissions to stored data points by type.
  + Manually add and remove data for certain data types (i.e. Blood Glucose level) even if no app is present to automatically place data into the health store.

## Overall map for HealthKit quantity access

In the interest of starting with the end in mind, let’s dive right into the stack that we will use to retrieve all the user logged height values between 2 dates.



Along the left hand side there are the hard data types bordered in red from the Xamarin.iOS binding library to HealthKit.

Along the right side is a human readable stack on how you would retrieve all the height entries within HealthKit that the user has logged between 2 dates.

Let’s break down the above stack layer by layer.

**Health Kit Data Store –** This is the root **HKHealthStore** HealthKit data access object. It is queried to see if health data storage is possible on the device, and manages the user security choices between your app and the specific quantity types (i.e. Height). More on per-data type security later on.

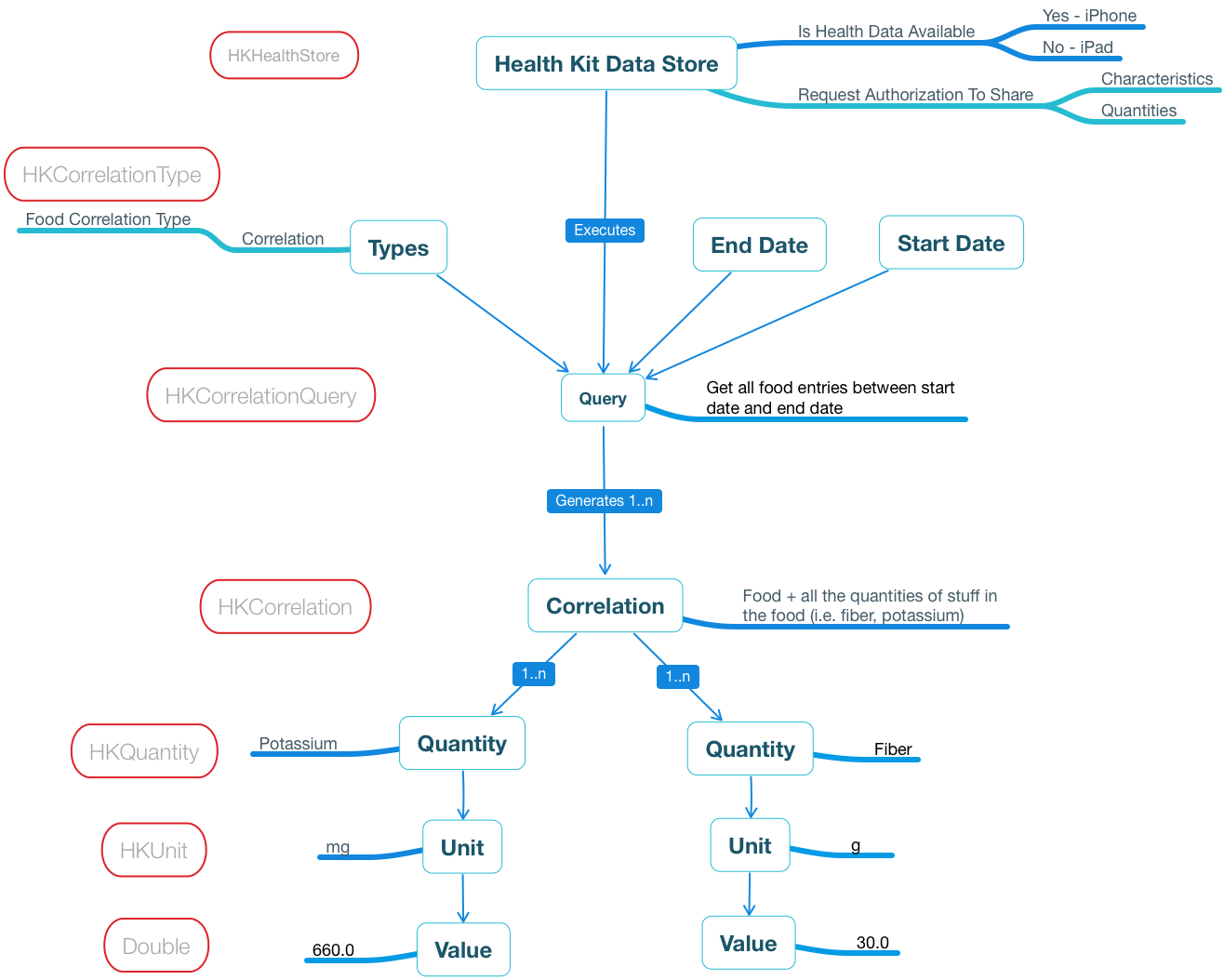
**Types – Quantity** – HealthKit has a way to designate each stored quantity type (i.e. height) via an object wrapper. The **HKQuantityType** is the special wrapper that HealthKit interacts with for queries and other calls to indicate what kind of data you are interested in accessing.

**Start Date / End Date**  -- Standard NSDate objects that indicate the range you are interested in retrieving typed data entries between.

**Query** – HealthKit has a special query object that is populated with the exact criteria you want to use to retrieve values for. In this case we want to retrieve height quantity samples (hence HKSampleQuery, we will see other query types later on) between 2 dates.

**Sample / Quantity / Unit / Value –** The return from the query are all the HKQuantitySample objects you asked for from the query. From the HKQuantitySample object you can retrieve the specific **Quantity** as converted into any **Unit** and get the raw **Value** after the conversion.

## Correlation query to get all food + nutrition entries between 2 dates



This is similar to the quantity retrieval stack, but we inject the concept of a correlation.

A single food item has multiple things in it. In the example above we have a banana. – What you don’t see us asking for a banana and it arriving at the bottom? Let’s do a similar stack walkthrough.

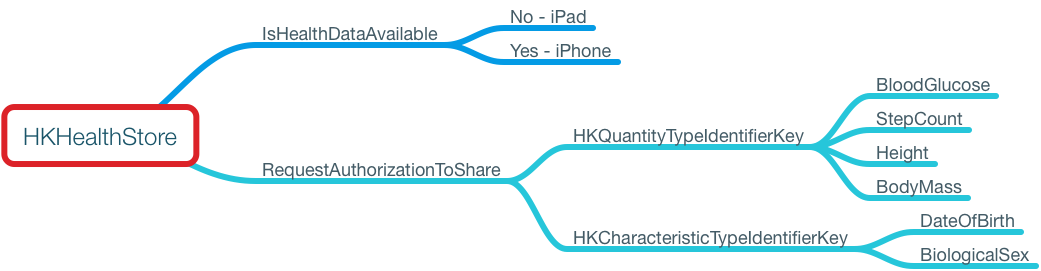
**Types** **/ EndDate / StartDate** – Notice that we create a special HKCorrelationType to indicate that we are going to ask for food. We take that type in addition to the NSDate for a start date and end date and feed it into a **Query** object.

**Query –** Notice that the the query type for food is an HKCorrelationQuery in this stack. It is not an HKSampleQuery. HealthKit has different types of queries depending on what types of values you are querying for.

**Correlation**  -- The return from the query are 1 to many different HKCorrelation objects. In this case each HKCorrelation object is an individual food that was logged. From the HKCorrelation we can get all the different nutrients / quantities that were in the logged food.

**Quantity / Unit / Value** – Each of the nutrients and quantities that were in the food can be asked for and retrieved. In this sample we pull mg of Potassium, and g of Fiber from the logged food entry.

## Getting Started accessing HealthKit data in your app



Before we can talk about accessing the data stored in HealthKit we have a series of considerations:

* You have to [add the HealthKit capability](https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/AddingCapabilities/AddingCapabilities.html) to your app via the entitlement editor.
* Access the static [**IsHealthDataAvailable**](https://developer.xamarin.com/api/property/HealthKit.HKHealthStore.IsHealthDataAvailable/) property off of **HKHealthStore** to determine if Health data can even be accessed on this iOS device.
  + If it returns false / No, then there is no Health app installed to the device (i.e. on iPad).
  + If it returns true / Yes, then a Health app is available.
* If Health is installed and HKHealthStore.IsHealthDataAvailable is true, then we need to request permissions to each type of data we want to access.
  + [Quantity Type Identifier](https://developer.apple.com/library/ios/documentation/HealthKit/Reference/HealthKit_Constants/index.html) – This is a data point that has many values which are logged over time.
  + [Characteristic Identifier](https://developer.apple.com/library/ios/documentation/HealthKit/Reference/HealthKit_Constants/index.html#//apple_ref/doc/constant_group/Characteristics_Identifiers) – This is a data point that usually only has 1 value and is not logged over time.
  + See examples of Quantity and Characteristic types from the Apple references above, or check out the [HKCharacteristicTypeIdentifierKey](https://developer.xamarin.com/api/type/MonoTouch.HealthKit.HKCharacteristicTypeIdentifierKey/) and [HKQuantityTypeIdentifierKey](https://developer.xamarin.com/api/type/MonoTouch.HealthKit.HKQuantityTypeIdentifierKey/) definitions from Xamarin.iOS documentation.
  + Each type permission asked for has to include a separate request to read and write the data for that type.
* So you now have your list of quantity types and characteristics by key that your app needs access to. Now we need to call into HKHealthStore. RequestAuthorizationToShare
  + We need to convert the NSString key values into HKQuantityType / HKCharacteristicType objects to feed them into the RequestAuthorizationToShare method.