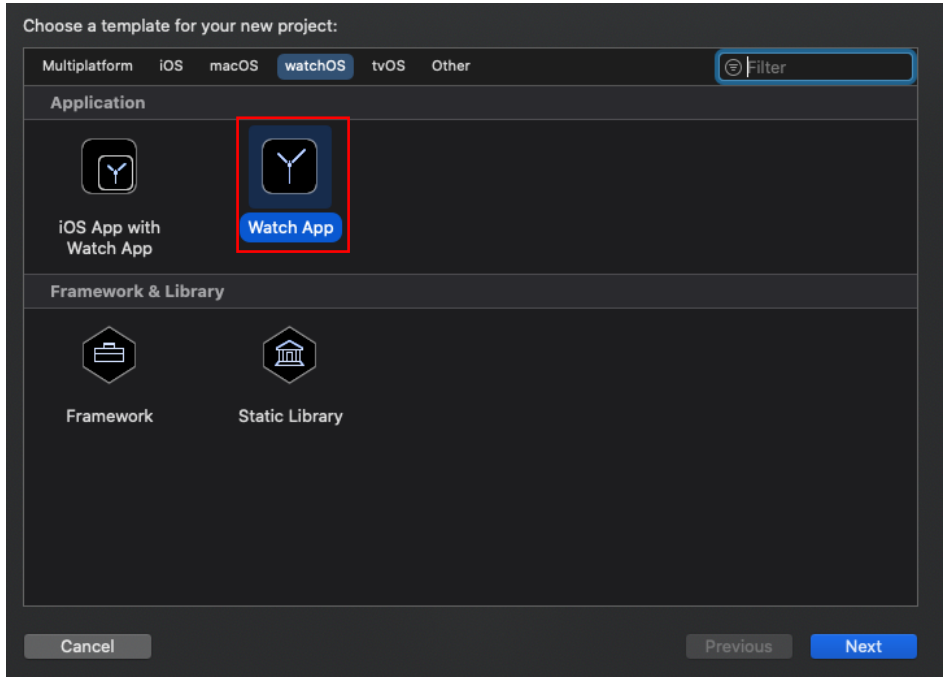
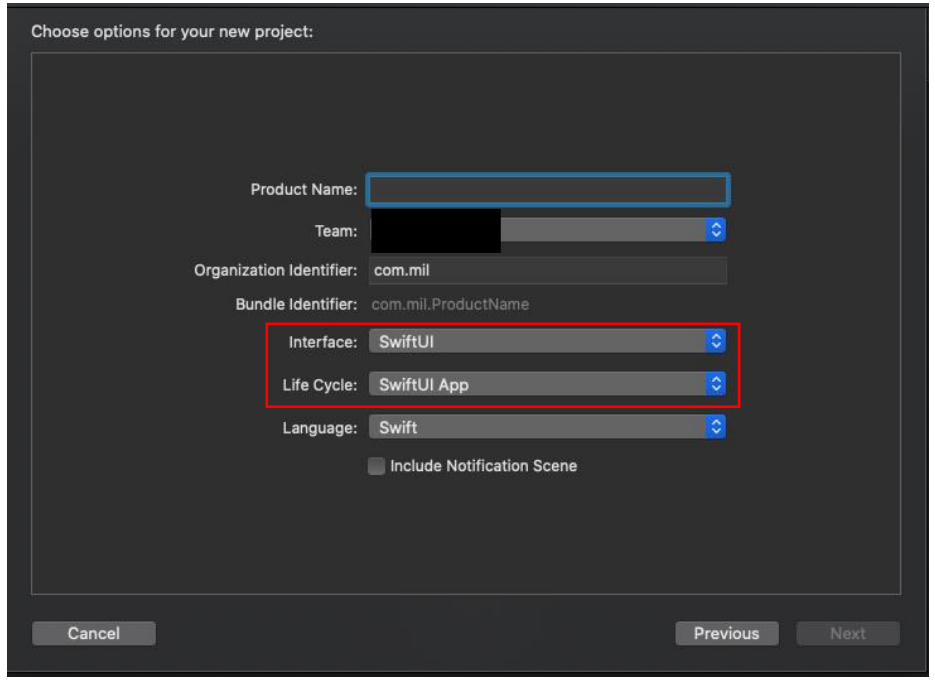
Display Steps of the last 15 Minutes on Apple Watch

StepByStepExample with HealthKit and SwiftUi

https://github.com/lucy237/HealthKitStatisticsCollectionQueryExample

Step 1: Create a New Project on Xcode

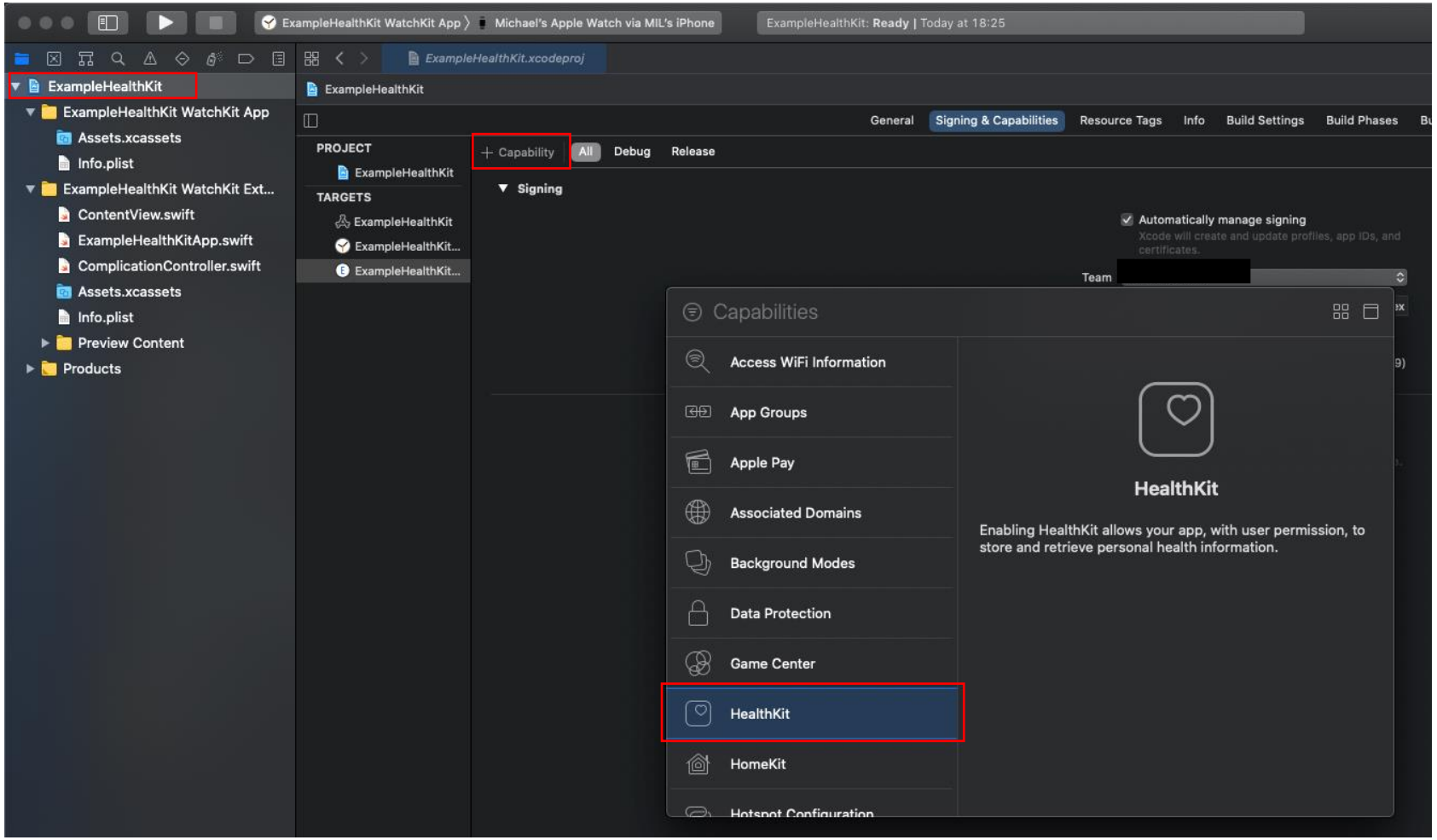




Create a new project in Xcode with selecting a Watch App and SwiftUI Framework

Step 2: Enable HealthKit

To use HealthKit in your project it is necessary to add the HealtKit Capability to your project as shown in the screenshots

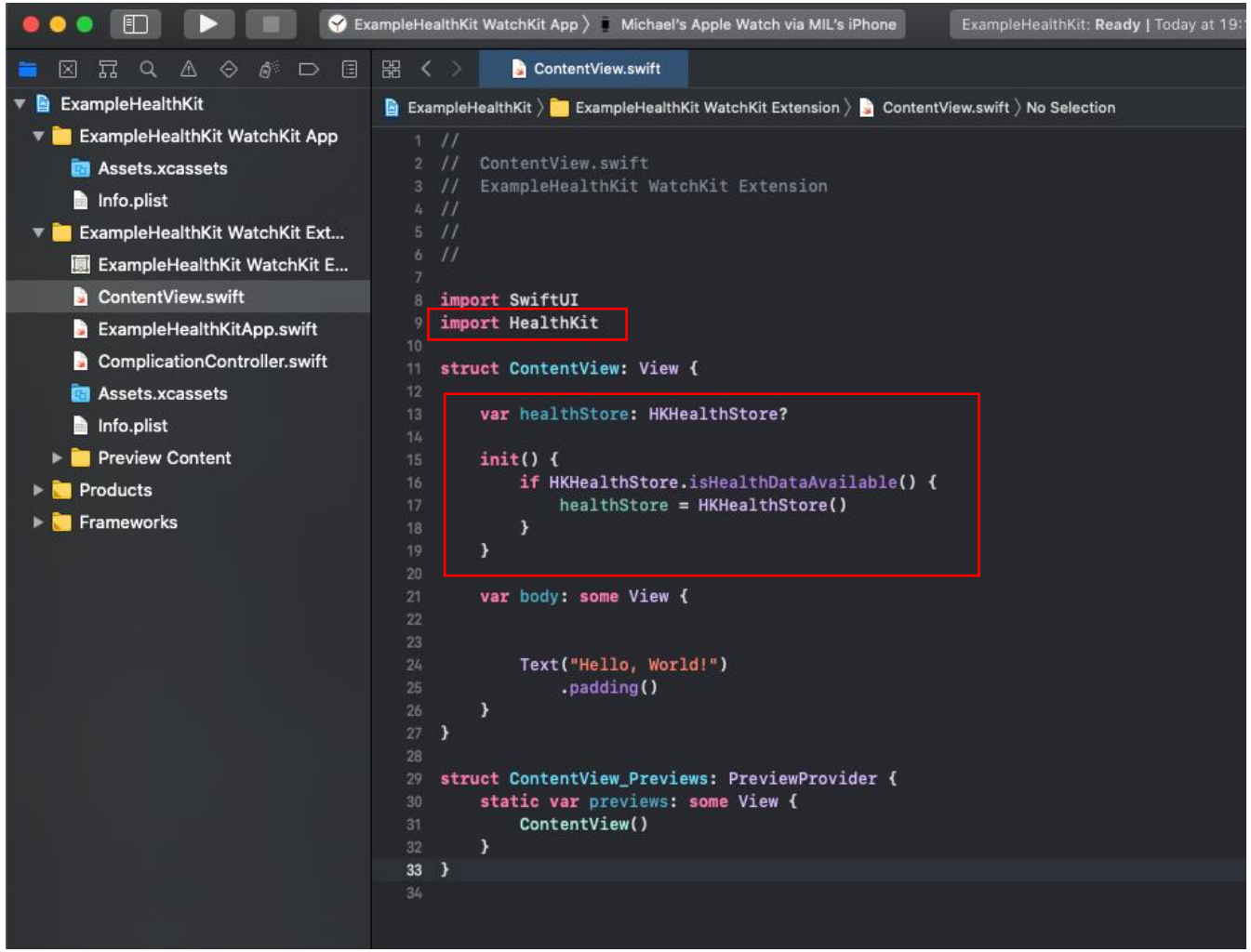


Find more information about HealthKit here: https://developer.apple.com/documentation/healthkit
Find more information about setting up HealthKit here: https://developer.apple.com/documentation/healthkit/setting_up_healthkit

Step 3: Initialize HKHealthStore

HKHealthStore provides access to all data related to health

- Before initializing you need to import the HealthKit Framework
- Also a check for availability of HealthData is necessary – otherwise you might get initialization error



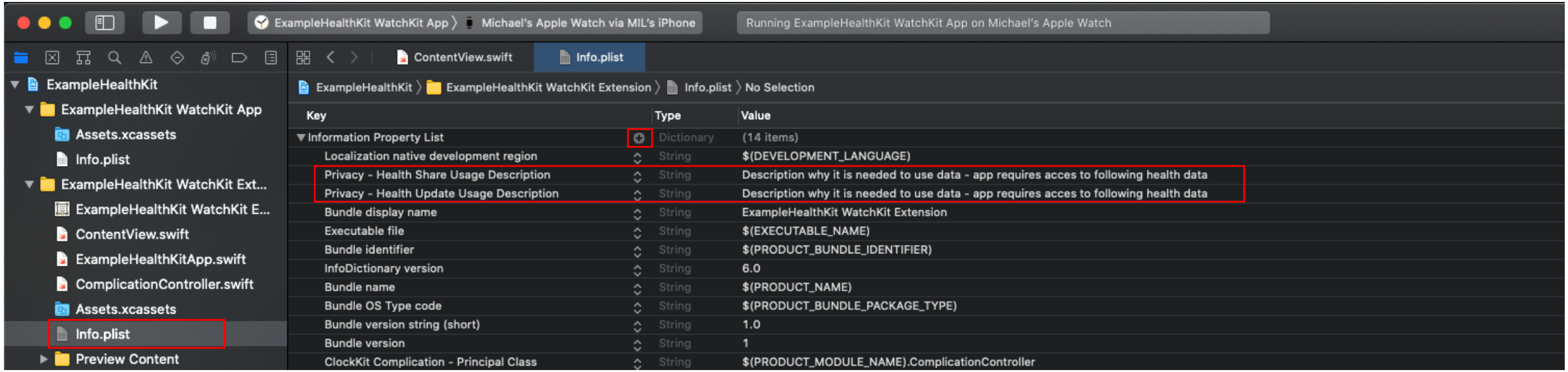
Find more information about HKHealthStore here: https://developer.apple.com/documentation/healthkit/hkhealthstore

Step 4: Request Access (1/2)

You need to provide custom messages for the permissions sheet to read and write HealthKit data.

- Add keys to info.plist as shown in the screenshot below
  the text in Information Property List will automatically change once pasting the keys below
  - **NSHealthShareUsageDescription**
  - **NSHealthUpdateUsageDescription**

  → Also add a description why the app needs to access your health data !



Find more information about Authorizing Access to health data here: https://developer.apple.com/documentation/healthkit/authorizing_access_to_health_data

Step 4: Request Access (2/2)

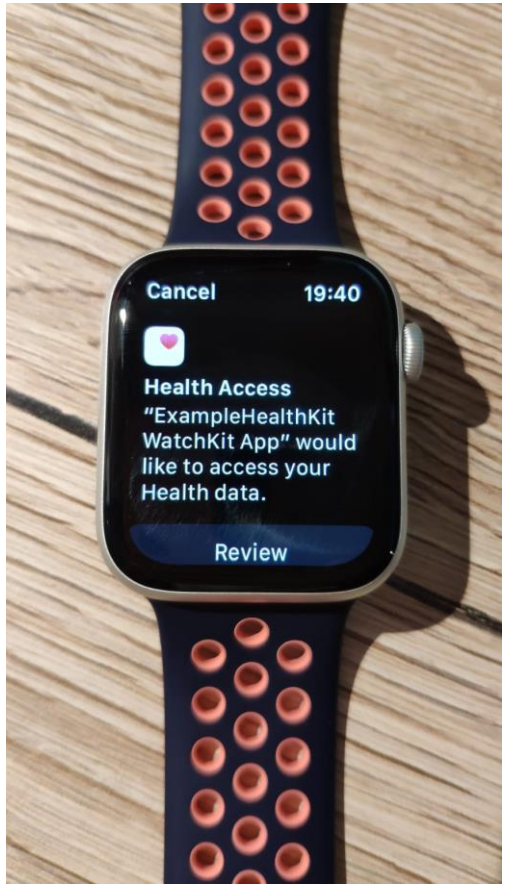
- In Content View add a Request Authorization Function
  execute as early as possible (like in this example .onAppear)
  - You can differentiate between toShare (like writing new data) and toRead access
- You have to define the types you want to access
  - .stepCount is the DataType, we want to use in this app
- find more data types here: https://developer.apple.com/documentation/healthkit/data_types

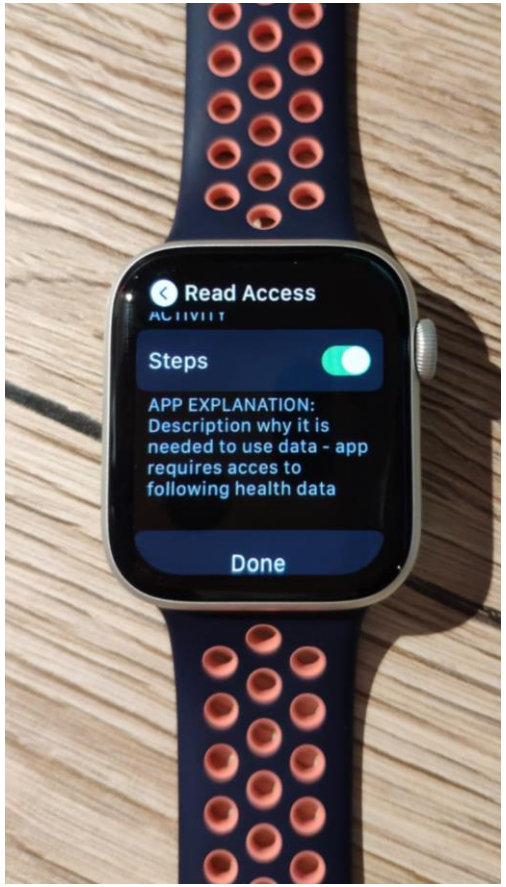
```
    // MARK: - Check if Data is available and initialize HealthStore
    init() {
        if HKHealthStore.isHealthDataAvailable() {
            healthStore = HKHealthStore()
        }
    }

    var body: some View {

        Text("Hello, World")
            .padding()

            .onAppear(){
                // MARK: - Execute Request Authorization
                requestAuthorization{ success in

                }
            }
    }


    // MARK: - Func Request Authorization
    func requestAuthorization(completion: @escaping (Bool) -> Void) {
        let typesToRead: Set = [
            HKQuantityType.quantityType(forIdentifier: .stepCount)!
        ]

        healthStore?.requestAuthorization(toShare: nil, read: typesToRead) { (success, error) in
            completion(success)
        }
    }
}
```

Find more information about Authorizing Access to health data here: https://developer.apple.com/documentation/healthkit/authorizing_access_to_health_data

Step 5: Check Authorization Code

- Execute code
  - → It is necessary to run the code on a real device in order to work
- Your display should show something similar to the images on the right
  - → This is where your message defined in info.plist will be displayed





Step 6: Create Query (1/4)

- For out app we want to have the steps for every minute within the last 15 minutes. So the query we are looking for is

  HKStatisticsCollection Query:
  a query that performs multiple statistics queries over a series of fixed-length time intervals, and returns the results.

- On the next pages, let's put this query together

Creating Statistics Collection Objects

```
init(quantityType: HKQuantityType, quantitySamplePredicate:
NSPredicate?, options: HKStatisticsOptions, anchorDate: Date,
intervalComponents: DateComponents)
```

Initializes a statistics collection query to perform the specified calculations over a set of time intervals.

Find more information about reading data from HealthKit here: https://developer.apple.com/documentation/healthkit/reading_data_from_healthkit
Find more information about Query Types and their usage here: https://developer.apple.com/documentation/healthkit/queries

Step 6: Create Query (2/4)

- Let‘s create the variables for anchor date and start date, which will be part of the quantitySamplePredicate first
- The anchor date defines, when our week begins. Therefore we want to extend Date with a function that defines the anchorDate to midnight on Monday
- As we will reuse the startDate later in the process, we will define it within the ContentView Scope
    - We use the current date and time and add -15 minutes to it

```
init(quantityType: HKQuantityType, quantitySamplePredicate:
NSPredicate?, options: HKStatisticsOptions, anchorDate: Date,
intervalComponents: DateComponents)
```

Initializes a statistics collection query to perform the specified calculations over a set of time intervals.

```swift
import HealthKit

extension Date {
    static func mondayAt12AM() -> Date {
        return Calendar(identifier: .iso8601).date(from: Calendar(identifier: .iso8601)
                                                   .dateComponents([.yearForWeekOfYear, .weekOfYear], from: Date()))!
    }
}

struct ContentView: View {

    public var healthStore: HKHealthStore?
    let startDate = Calendar.current.date(byAdding: .minute, value: -15, to: Date())!
```

Find more information about reading data from HealthKit here: https://developer.apple.com/documentation/healthkit/reading_data_from_healthkit
Find more information about Query Types and their usage here: https://developer.apple.com/documentation/healthkit/queries

Step 6: Create Query (3/4)

- As a type, as defined in the authorization part as well we want to use the quantityType: .stepCount
- With the now created extension to Date wen can also create the anchor date
- For the interval we define 1 minute, as we want to have the data for each minute within the last 15 minutes
- Therefore the Predicate is defined by the time from startDate till Date(), which is now
- .strictStartDate states, that we only want data within the defined time frame

```
    var body: some View {

        Text("Hello, World")
            .padding()

            .onAppear(){
                // MARK: - Execute Request Authorization
                requestAuthorization{ success in

                }
            }
    }

    // MARK: - Func Calculate Steps
    func calculateSteps(completion: @escaping (HKStatisticsCollection?) -> Void){
        let type = HKQuantityType.quantityType(forIdentifier: HKQuantityTypeIdentifier.stepCount)!
        let anchorDate = Date.mondayAt12AM()
        let interval = DateComponents(minute: 1)
        let predicate = HKQuery.predicateForSamples(withStart: startDate, end: Date(), options: .strictStartDate)

        let query = HKStatisticsCollectionQuery(quantityType: type, quantitySamplePredicate: predicate,
                                                options: .cumulativeSum, anchorDate: anchorDate, intervalComponents: interval)

        query.initialResultsHandler = { query, statisticsCollection, error in
            completion(statisticsCollection)
        }

        if let healthStore = self.healthStore{
            healthStore.execute(query)
        }
    }
```

Find more information about reading data from HealthKit here: https://developer.apple.com/documentation/healthkit/reading_data_from_healthkit
Find more information about Query Types and their usage here: https://developer.apple.com/documentation/healthkit/queries

Step 6: Create Query (4/4)

- In The query itself, we define that we want the option: .cummulativeSum, which means, we want data from all devices (phone and watch)
- Next we have to define the callback handler initialResultHandler – gets fired whenever the query is executed
  - Returns the query and statisticsCollection and Error in case there is one
  - Create completion and passing in statisticsCollection
- Finally create code to execute query

```swift
    var body: some View {

        Text("Hello, World")
            .padding()

            .onAppear(){
                // MARK: - Execute Request Authorization
                requestAuthorization{ success in

                }
            }
    }

    // MARK: - Func Calculate Steps
    func calculateSteps(completion: @escaping (HKStatisticsCollection?) -> Void){
        let type = HKQuantityType.quantityType(forIdentifier: HKQuantityTypeIdentifier.stepCount)!
        let anchorDate = Date.mondayAt12AM()
        let interval = DateComponents(minute: 1)
        let predicate = HKQuery.predicateForSamples(withStart: startDate, end: Date(), options: .strictStartDate)

        let query = HKStatisticsCollectionQuery(quantityType: type, quantitySamplePredicate: predicate,
                                                options: .cumulativeSum, anchorDate: anchorDate, intervalComponents: interval)

        query.initialResultsHandler = { query, statisticsCollection, error in
            completion(statisticsCollection)
        }

        if let healthStore = self.healthStore{
            healthStore.execute(query)
        }
    }
```

Find more information about reading data from HealthKit here: https://developer.apple.com/documentation/healthkit/reading_data_from_healthkit
Find more information about Query Types and their usage here: https://developer.apple.com/documentation/healthkit/queries

Step 7: Run Query to see if it‘s working

- We will add the execution code to the part where we made sure to be authorized access to health data as seen on the right.
- For now we only print the returned statistics collection in the output to see if everything works
- Therefore run the code and see if you get an output in your console in order to see if the code is working
  - Again, as before you have to run on a real device
  - This is not gonna display anything on the screen but only in the output on Xcode.
  - We just use it to get an indication that there is something within statisticsCollection

```
25          }
26      }
27
28      var body: some View {
29
30          Text("Hello, World")
31              .padding()
32
33              .onAppear(){
34                  // MARK: - Execute Request Authorization
35                  requestAuthorization{ success in
36                      // execute calculate steps
37                      calculateSteps{ statisticsCollection in
38                          if let statisticsCollection = statisticsCollection {
39                              // UpdateUI here
40                              print(statisticsCollection)
41                          }
42                      }
43                  }
44              }
45      }
46
47      // MARK: - Func Calculate Steps
48      func calculateSteps(completion: @escaping (HKStatisticsCollection?) -> Void){
49          let type = HKQuantityType.quantityType(forIdentifier: HKQuantityTypeIdentifier.stepCount)!
```

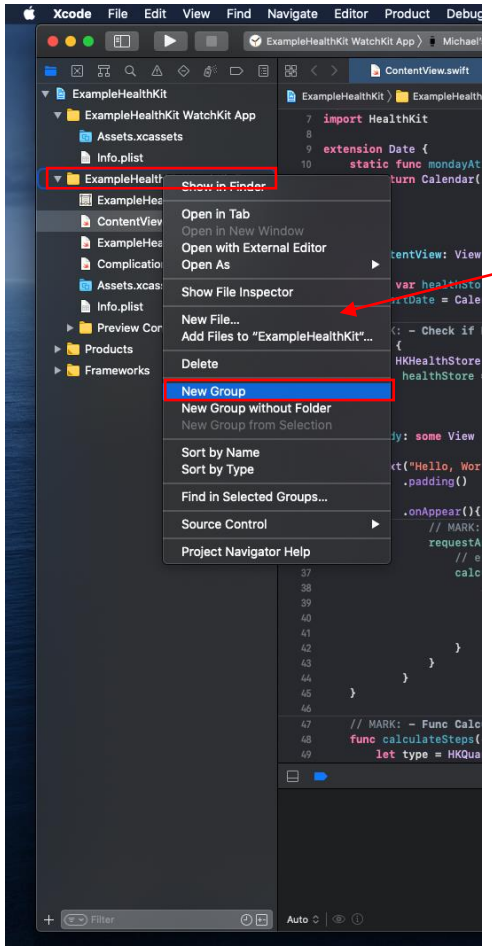
ExampleHealthKit WatchKit App

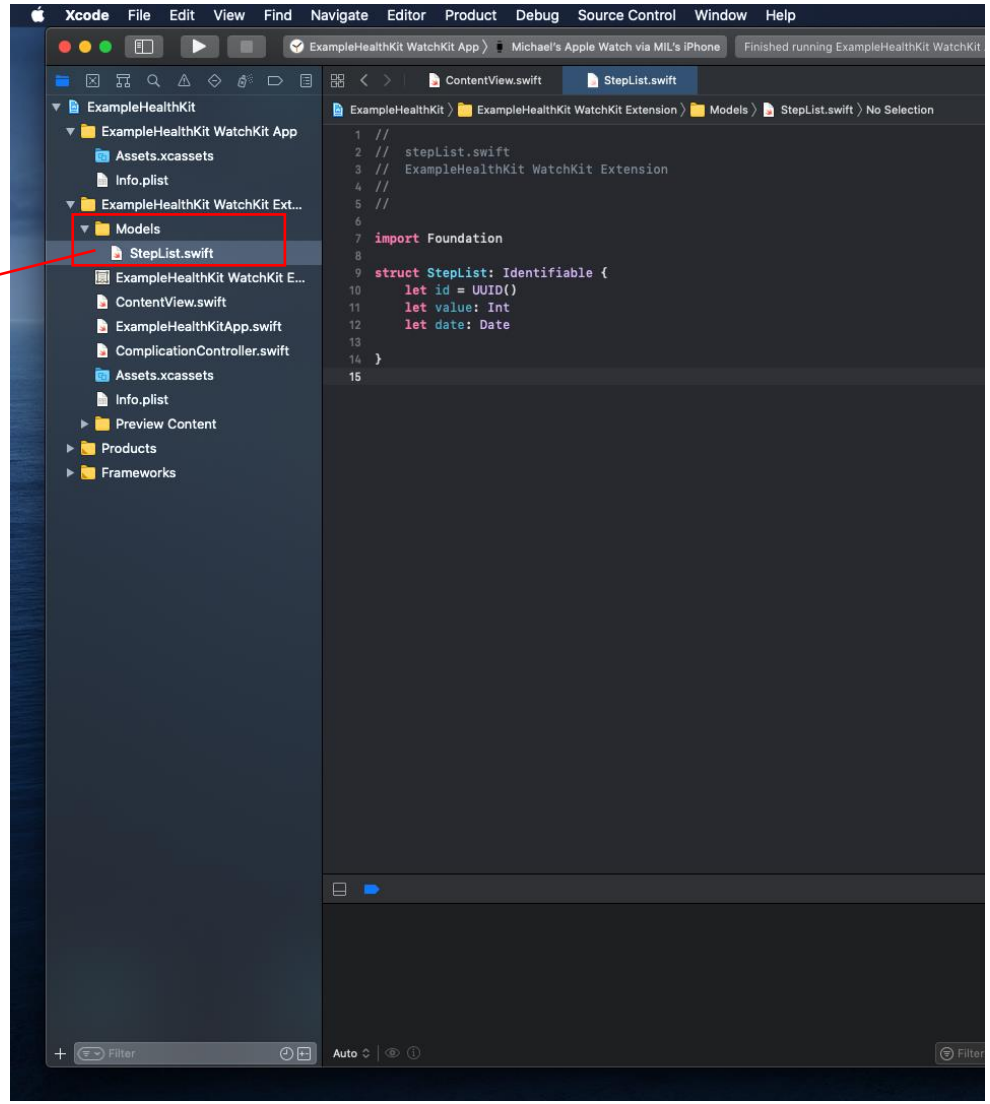
```
<HKStatisticsCollection: 0x13b1cb90>
```

Auto | Filter | All Output

Step 8: Create Steps Array

- We will now create a struct to hold our result
- Therefore create a new group, by right click on the extension folder, as shown on the first screenshot and call it Models
- The same way, create a new swift file and call it StepList
- Create a struct and make sure to make it Identifiable, so we can use UUID() to create a unique identifier
- Also we want to add variables to save the value (in our case step count) and the date of the sample





Step 9: Populate to StepList Instance

- First create an instance steps of Step List where we can later append our entries to
- To fill the array we will create a function updateUiFromStatistics
  - Pass in statisticsCollection
- To calculate number of steps we can use .enumerateStatistics by providing from and to date
  - We can use the startDate defined and the date now
  - In case of steps we want to get the sumQuantity over the timeframe
    (eg for heartRate it would be the .averageQuantity)
  - Append data to steps

```
19  let startDate = Calendar.current.date(byAdding: .minute, value: -15, to: Date())!
20  @State private var steps: [StepList] = [StepList]()
21
22
23  // MARK: - Check if Data is available and initialize HealthStore
24  init() {
25      if HKHealthStore.isHealthDataAvailable() {
26          healthStore = HKHealthStore()
27      }
28  }
29
30  var body: some View {
31
32      Text("Hello, World")
33          .padding()
34
35          .onAppear(){
36              // MARK: - Execute Request Authorization
37              requestAuthorization{ success in
38                  // execute calculate steps
39                  calculateSteps{ statisticsCollection in
40                      if let statisticsCollection = statisticsCollection {
41                          // UpdateUI
42                          updateUiFromStatistics(statisticsCollection)
43                      }
44                  }
45              }
46          }
47  }
48
49  // MARK: - Function Update UI
50  func updateUiFromStatistics(_ statisticsCollection: HKStatisticsCollection){
51
52      statisticsCollection.enumerateStatistics(from: startDate, to: Date()){statistics, stop in
53          let value = statistics.sumQuantity()?.doubleValue(for: .count())
54          let stepEntry = StepList(value: Int(value ?? 0), date: statistics.startDate)
55
56          steps.append(stepEntry)
57      }
58  }
59
60  // MARK: - Func Calculate Steps
```

Step 10: Display Data on UI

The only thing we have left to do is to display the result on our screen

- Therefore we replace the hello world with a list view in which we loop over out step array and display the stepCount value as well as the time of the entry

```
20      @State private var steps: [StepList] = [StepList]()
21
22
23      // MARK: - Check if Data is available and initialize HealthStore
24      init() {
25          if HKHealthStore.isHealthDataAvailable() {
26              healthStore = HKHealthStore()
27          }
28      }
29
30      var body: some View {
31
32          List(steps, id: \.id) { entry in
33              VStack{
34                  Text("\(entry.value)")
35                  Text(entry.date, style: .time)
36                      .opacity(0.5)
37              }
38          }
39
40              .onAppear(){
41                  // MARK: - Execute Request Authorization
42                  requestAuthorization{ success in
43                      // execute calculate steps
44                      calculateSteps{ statisticsCollection in
45                          if let statisticsCollection = statisticsCollection {
46                              // UpdateUI
47                              updateUiFromStatistics(statisticsCollection)
48                          }
49                      }
50                  }
51              }
```

Result