

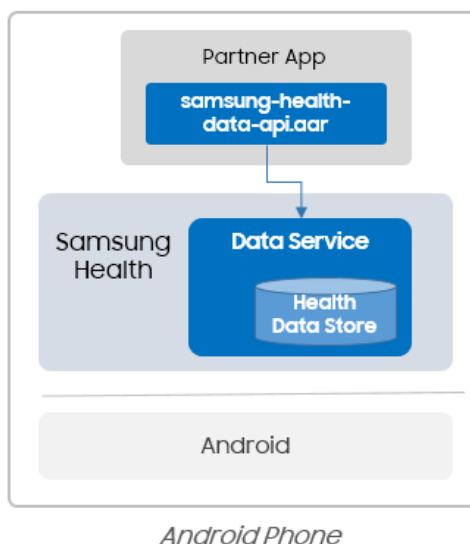
# Programming Guide

## for Samsung Health Data SDK

v1.0.0 Beta2

The [Samsung Health application](#) offers various features to measure user's health data. It lets the user automatically record a range of activities with a Galaxy Watch and any connected accessories. The user can check their daily steps, heart rate, sleep, nutrition, and many more of their lifestyle metrics. Managing a healthy lifestyle is easier and simpler with the Samsung Health.

Samsung Health Data SDK helps to access the health data in the Samsung Health application. An application importing the SDK library (`samsung-health-data-api.aar`) can access selected health data from the Samsung Health's data store. Health data in the Samsung Health application may come from different sources, like the Galaxy Watch, which automatically transfers its data to a paired mobile phone's Samsung Health application.



# Development Environment

## Target Device

The Samsung Health Data SDK runs on devices with Android 10 (API level 29) or above. It is available on all Samsung smartphones and non-Samsung Android smartphones.

## SDK Content

The SDK provides the following content:

Folder structure	Content description
/docs	Guide and API Reference
/libs	The Samsung Health Data SDK's library <ul style="list-style-type: none"><li>samsung-health-data-api.aar</li></ul>
/sample-code	Sample application codes <ul style="list-style-type: none"><li>HealthDiary</li></ul>
/tool	A test tool displaying saved data in the Samsung Health application <ul style="list-style-type: none"><li>DataViewer</li></ul>

## Limitations

- The Samsung Health Data SDK works with the Samsung Health application. The Samsung Health version 6.29 or higher is required.
- The emulator is not supported.
- Data obtained using the Samsung Health Data SDK is for fitness and wellness information only; it is not for the diagnosis or treatment of any medical condition.

# Developer Support

If issues occur while using the Samsung Health Data SDK, please visit:

Samsung Developer site > Support > [Developer Support](#)

After logging into a Samsung Account, submit a query with an event log and a detailed issue description. You can get the event log with the following instruction:

1. A Wi-Fi connection on your phone is recommended.
2. Go to the phone's Samsung Health > **Settings** > **About Samsung Health**.
3. Tap the Samsung Health logo quickly, more than 10 times until the "Send log report" button appears. Click the button.
4. Select an email application from the popup.
5. Save the email's attachment and copy the application information such as the Samsung Health application version and device model.
6. Share the obtained log report by attaching it to the support request.

# Device Settings

## Turn on the Phone's Developer Options

To develop and debug your application, enable the Developer options:

1. Go to the phone **Settings**.
2. Tap **About device** or **About phone**.
3. Tap **Software information**.
4. Tap **Build number** seven times.
5. Depending on your device and operating system, you may not need to enter your pattern, a PIN, or a password to enable the Developer options menu.
6. Go back to the phone's **Settings**. The **Developer options** menu is now available.

Depending on your device, it may appear under **Settings > General > Developer options**.

## Samsung Health's Developer Mode

The Samsung Health Data SDK works properly for registered partner applications in a Samsung Health system. The Health team registers the partner application information including an application package name and signature (sha256), with an available data type scope in the Samsung system. Moreover, the Samsung Health application allows the SDK's data access only if a running application's package name and signature are both matched with the registered information.

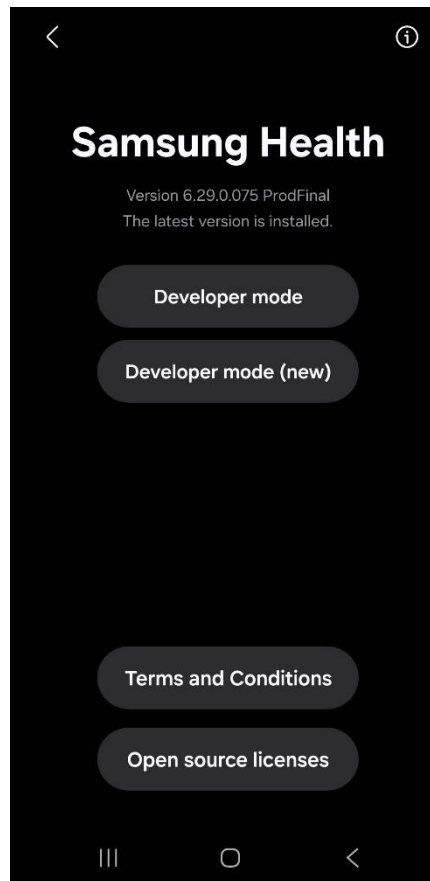
The application signature registered in the Samsung Health system should come from the application release key. If a partner application developer would like to test the application with a different application key, activate the Samsung Health's application Developer mode.

### Note

The Samsung Health Developer mode is ONLY intended for testing or debugging your application. It is NOT for application users. Do not provide a Developer mode guide to application users.

You can activate Developer mode with the following steps:

1. Select the ':' button of the Samsung Health in the top-right.
2. Select **Settings > About Samsung Health**.
3. Tap the version line region quickly 10 times or more.  
If you are successful, the **Developer mode (new)** button is shown.
4. Select **Developer mode (new)**.
5. Agree with **Notice** of usage of the developer mode.
6. For reading data from Samsung with the Samsung Health Data SDK, turn **Developer Mode for Data Read** on.



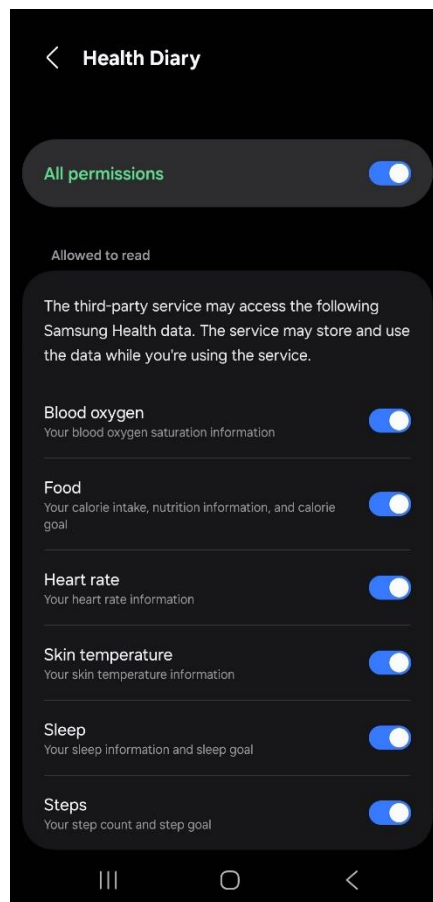
7. To write data to Samsung Health with the SDK, obtain the access code through a partner application program request. See the [process](#) page and submit the request. After the request is approved, an access code will be provided.
8. Enter the **Client ID** as the application's package name and the **Access Code** obtained.

# Main Features

## Data Permission

Accessing the user's health data in Samsung Health is available after getting the user's consent. The Samsung Health Data SDK provides a data permission request feature. An application using the SDK can check the granted permissions and request permissions if more permissions are required.

Request for permissions is available with a set of permission list composed of a data type and access type. If `requestPermissions()` or `requestPermissionsAsync()` is called, the following data permission popup is shown. For example, developer will have access to heart rate data if the user of the application grants permission for **Heart Rate** in the **Allowed to read** section of the permission popup.



Applications using the Samsung Health Data SDK are listed up in Samsung Health > **Settings** > **Apps**.

### Note

For the user data privacy, set a menu so that the user can change the data allowance state anytime in your application.

## Data Types

The Samsung Health Data SDK supports various data types:

Data type	Access scope
Active calories burned goal	Aggregate
Active time goal	Aggregate
Activity summary	Aggregate
Blood glucose	Read Read changes Insert Update Delete
Blood oxygen	Read Read changes
Blood pressure	Read Read changes Insert Update Delete
Body composition	Read Read changes Insert Update Delete
Energy score	Read Read changes
Exercise	Read Aggregate (for specific properties) Read changes
Floor climbed	Aggregate
Heart rate	Read Aggregate (for specific properties) Read changes Insert Update Delete
Nutrition	Read Aggregate (for a specific property) Read changes

	Insert Update Delete
Nutrition goal	Aggregate
Skin temperature	Read Read changes
Sleep	Read Aggregate (for a specific property) Read associated data Read changes
Sleep goal	Aggregate
Steps	Aggregate
Step goal	Aggregate
Water intake	Read Aggregate (for a specific property) Read changes Insert Update Delete
Water intake goal	Aggregate

## User Profile Data

The user profile stores gender, date of birth, height, weight, and nickname information. It's read-only.

- User profile (read-only)

## Data Access

The Samsung Health Data SDK provides various operation types, allowing developers to access Samsung Health data. Available operations may vary, depending on the data type. Operations like reading data, reading associated data and aggregating data return a list of data points, representing values stored in the Samsung Health, such as a list of heart rate values from a specific time range.

Each data point includes a data ID and data source information representing the **application** and the **device**, which initially measured the data.



## Read Data

A reading operation is a basic query to get a set of data from the Samsung Health. Users can set conditions such as data types, time range, or data source.

To read data, check the following property in each data type:

- `readDataRequestBuilder`

## Aggregate Data

The Aggregate operation is helpful to get summarized and processed data that is frequently used. It provides an aggregated value with a simple code. Conditions including a local time filter can be set to the aggregate operation.

To use the aggregate operation, check the properties of each data type from the below list:

- `TOTAL` of `StepType`
- `TOTAL_ACTIVE_TIME` in `ActivitySummaryType`
- `LAST` of `ActiveTimeGoalType`
- `MIN` and `MAX` in `HeartRateType`

## Read Changes

Reading changes gives data change information for a given data type and conditions. It returns a change list. Each change item includes a change type as `UPSERT` and `DELETE` and a change time. In case of updated data, a data point is retrieved. For deleted data, it gives the data's UID. This feature helps keep the application current by reading only the changes made since the last synchronization with the Samsung Health.

To read changed data, check the following property in each data type:

- `changedDataRequestBuilder`

## Read Associated Data

Reading associated data is useful to get multiple data types which are related to each other. For example, the Samsung Health records oxygen saturation and skin temperature during the user's sleep. This operation helps to make a simple query to get sleep data (base data) with oxygen saturation and skin temperature data (associated data).

To read associated data, check the following property in each data type:

- `associatedReadRequestBuilder`

## Insert Data

An insertion operation is a basic query to write sets of data into the Samsung Health. You can specify the data type, corresponding values for each field and device used to record the data. Optionally, you may assign a unique identifier, referred to as the client data ID. For more details, please refer to `HealthDataPoint.clientDataId` in the SDK's documentation.

To insert data, check the following property in each data type:

- `insertDataRequestBuilder`

## Update Data

Updating data is available for the data inserted into the Samsung Health by the application. To update data, obtain the desired data's `uid` or `clientId` and specify the data's relevant fields, then update the data.

To update data, check the following property in each data type:

- `updateDataRequestBuilder`

## Delete Data

Data deletion from the Samsung Health is available using the Samsung Health Data SDK. The data that can be deleted is restricted to what was inserted by the application. To build a request for data deletion, one may include conditions like the data's ID or time filters.

To delete data, check the following property in each data type:

- `deleteDataRequestBuilder`

### Note

You may perform the inserting, updating or deleting operation on multiple data at once. If a performed operation fails for even just one of the data, then none of the data from that request will be processed. So, check for an exception to the request.

## Device Manager

Saved health data in the Samsung Health can come from various connected devices like a Galaxy Watch, Galaxy Ring and a weight scale. `DeviceManager` provides source device information of saved health data in the Samsung Health's data store.

For instance, when Galaxy Watch measures blood oxygen level, the watch's measured data will be synchronized to a paired Android smartphone's Samsung Health. And if the user measures weight with a connected weight scale device, that weight data will be synchronized too. In this case, if the `DeviceManager.getDevices()` API is called with `DeviceGroup.WATCH`, the Galaxy Watch's `Device` object will be retrieved.

## Device Information

The `Device` object includes:

- Device ID
- Device type like `DeviceGroup.WATCH` or `DeviceGroup.ACCESSORY`  
In case of the `ACCESSORY` device group, `AccessoryType` provides an additional detailed type.
- Device's manufacturer name

- Device model name
- Device name costumed by the user

You can get device information for a specific device ID or a device type.

## Device Registration

`DeviceManager` can also be used to register custom device and use it as a data source when performing data writing operations. `DeviceRegistrationRequest.registerDevice()` API, where you provide your own `Device` object and a device seed, is used to create a request. Device seed is a unique string for the device – it can be, for example, its serial number. This request can be sent to `DeviceManager.registerDevice()` API in order to add a new device – the API will return the ID of newly registered device. It is used when performing queries such as data insertion.

Later, the `Device` object can be retrieved in various ways, for example by using `DeviceManager.getDeviceBySeed()`, where the unique seed used for registration is provided, or by `DeviceManager.getOwnDevices()`, which will return a list of devices registered by the package.

## Filters

Filters help to limit a query range to get data that are more precise. The Samsung Health Data SDK provides time, source, and ID filters. Available filters are different for each data operation.

For instance:

- `IdFilter`, `ReadSourceFilter`, and `TimeFilter` are available for **reading data**.
- `AggregateSourceFilter` and `TimeFilter` are available for **aggregating data**.
- `IdFilter` and `TimeFilter` are available for **deleting data**.

## Time Filters

A built request with a **start time** and **end time** gives all records that are inside this time range. For that, you can use the following filter types:

- `LocalTimeFilter`
- `LocalDateFilter`
- `InstantTimeFilter`

## Source Filters

All health data stored in Samsung Health has the following source information:

- Source application ID: an application package name which identifies the application that inserted the data to the Samsung Health.
- Source device ID: representing a device which measures or creates the data.

`ReadSourceFilter` supports the following functions:

## Samsung Health Data SDK

- `fromApplicationId`: an application package name
- `fromDeviceType`: a device type such as `MOBILE` or `WATCH`
- `fromLocalDevice`: a device with the Samsung Health installed
- `fromPlatform`: measured or inserted by the Samsung Health application

`AggregateSourceFilter` supports the `fromPlatform` function.

## ID Filter

This filter is for setting Data UUIDs or Client Data IDs. Client Data ID is a unique identifier that you may have set while inserting data.

## Grouping Retrieved Data

Retrieved health data can be grouped into time segments with:

- `InstantTimeGroup`: `MINUTELY`, `HOURLY`, and `DAILY`
- `LocalDateGroup`: `DAILY`, `WEEKLY`, `MONTHLY`, and `YEARLY`
- `LocalTimeGroup`: `MINUTELY`, `HOURLY`, and `DAILY`

# Hello Data SDK

This is a tutorial on how to use the Samsung Health Data SDK.

## Importing the SDK Library and Build Setup

Import the `samsung-health-data-api.aar` library into your application project's "libs" folder and add it into the module level `build.gradle` of your application project. Include also Gson dependency.

```
dependencies {  
  
    implementation(fileTree(mapOf("dir" to "libs", "include" to listOf("*.aar"))))  
    implementation "com.google.code.gson:gson:2.9.0"  
}
```

Apply "kotlin-parcelize" plugin to your module level `build.gradle` file.

```
plugins {  
  
    id("kotlin-parcelize")  
}
```

## Getting HealthDataStore

To start utilizing Samsung Health Data SDK, get a `HealthDataStore` instance with:

- `HealthDataService.getStore()`

It helps to create the following requests:

- Checking granted data permissions and requesting data permissions.
- Reading data
- Reading changed data
- Inserting data
- Updating data
- Deleting data
- Getting `DeviceManager`

## Permission Request

### Granted Permission Check

Accessing health data requires the user's permissions for every data type. Check that the permissions are granted using:

- `HealthDataStore.getGrantedPermissions()`

Calling this method can result in throwing `HealthDataException`. There are several types of exception classes inheriting from the mentioned parent class: `AuthorizationException`, `InvalidRequestException`, `PlatformInternalException` and `ResolvablePlatformException`. The last one of them indicates that the Samsung Health platform is not ready to serve the specified operation. This kind of error can be resolved by the user at runtime in some cases by calling `resolve()` method.

`ResolvablePlatformException` can have one of the following error codes, which might help in applying appropriate action to the user:

- `ErrorCode.ERR_PLATFORM_NOT_INSTALLED`: Samsung Health is not installed.
- `ErrorCode.ERR_OLD_VERSION_PLATFORM`: The version of Samsung Health is old so it can't support this function of the SDK.
- `ErrorCode.ERR_PLATFORM_DISABLED`: Samsung Health has been installed but it is disabled.
- `ErrorCode.ERR_PLATFORM_NOT_INITIALIZED`: Samsung Health has been installed but the user didn't perform an initial process, such as agreeing to the Terms and Conditions.

For more details, please refer to `com.samsung.android.sdk.health.data.error` package of Samsung Health Data SDK's API Reference.

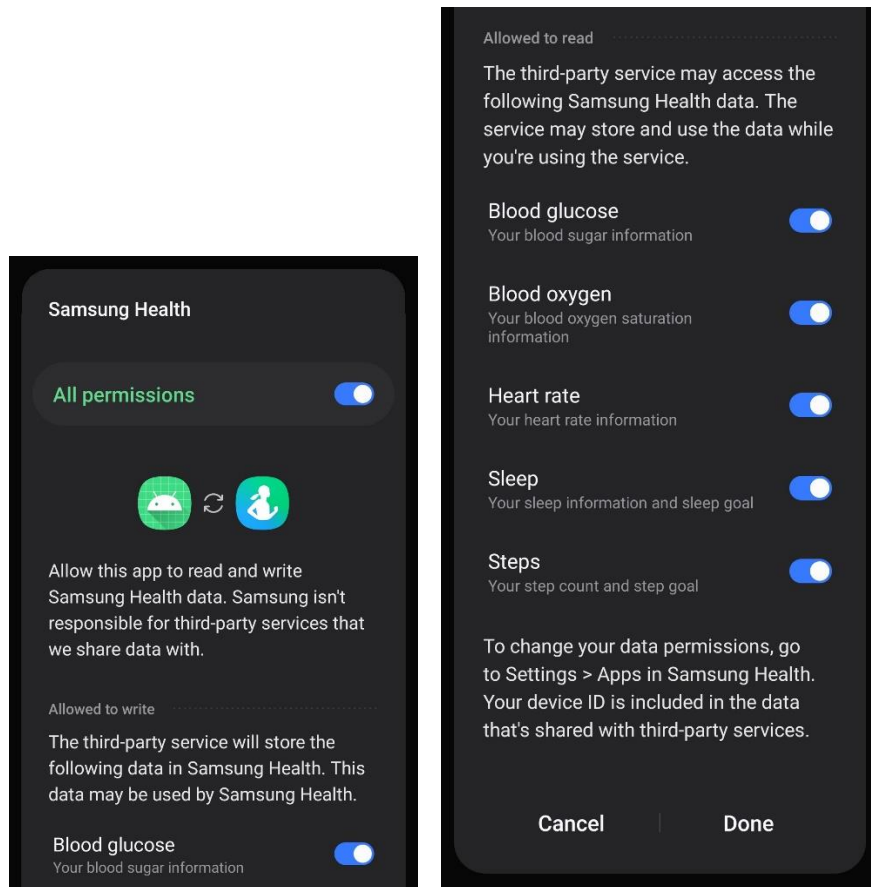
## Request for Permissions

If all permissions were not present, we need to request for permissions with:

- `HealthDataStore.requestPermissions()`

If this API call succeeds, a data permission popup will display on the application activity with a list of permissions to grant by the user. After the user allows permissions through the data permission popup, the API returns a set of granted permissions.

Below there's an example of permission popup that is shown to the user:



Below is an example code for permission checking, which covers all the topics discussed in this section:

```
suspend fun checkForPermissions(activity: Activity) {
    val permSet = mutableSetOf(
        Permission.of(DataTypes.HEART_RATE, AccessType.READ),
        Permission.of(DataTypes.STEPS, AccessType.READ),
        Permission.of(DataTypes.SLEEP_GOAL, AccessType.READ),
        Permission.of(DataTypes.SLEEP, AccessType.READ),
        Permission.of(DataTypes.BLOOD_OXYGEN, AccessType.READ),
        Permission.of(DataTypes.BLOOD_GLUCOSE, AccessType.READ),
        Permission.of(DataTypes.BLOOD_GLUCOSE, AccessType.WRITE)
    )

    try {
        val healthDataStore = HealthDataService.getStore(context)
        val grantedPermissions = healthDataStore.getGrantedPermissions(permSet)

        if (grantedPermissions.containsAll(permSet)) {
            // All Permissions already granted
        } else {
            // Partial or No permission, we need to request for the permission popup
            healthDataStore.requestPermissions(permSet, activity)
        }
    } catch (error: HealthDataException) {
        if (error is ResolvablePlatformException && error.hasResolution) {
            error.resolve(context)
        }
        // handle other types of HealthDataException
    }
}
```

## Read Data

You can now retrieve data from Samsung Health using:

- `HealthDataStore.readData()`

Apart from the data type, you can also set a time filter in the request.

The example below gives a data point list (representing heart rate values) from the specific time range.

```
suspend fun readHeartRateData(startDate: LocalDateTime, endDate: LocalDateTime) {

    val healthDataStore = HealthDataService.getStore(context)
    val localTimeFilter = LocalTimeFilter.of(startDate, endDate)
    val readRequest = DataTypes.HEART_RATE.readDataRequestBuilder
        .setLocalTimeFilter(localTimeFilter)
        .setOrdering(Ordering.DESC)
        .build()

    val heartRateList =
        healthDataStore.readData(readRequest).dataList
    // do data processing
}
```

The following example demonstrates how to retrieve heart rate data measured by a Galaxy Watch:

```
val localTimeFilter = LocalTimeFilter.of(startDate, endDate)
val readRequest = DataTypes.HEART_RATE.readDataRequestBuilder
    .setSourceFilter(ReadSourceFilter.fromDeviceType(DeviceGroup.WATCH))
    .setLocalTimeFilter(localTimeFilter)
    .setOrdering(Ordering.ASC)
    .build()
```

To get data created in the mobile device with the Samsung Health installed, set a source filter with a local device.

```
.setSourceFilter(ReadSourceFilter.fromLocalDevice())
```

## Aggregate Data Request

You can retrieve a data's total or minimum and maximum values or the last data entry by using the function:

- `HealthDataStore.aggregateData()`

### Total Steps

To get total steps, use `DataType.StepsType.TOTAL`.

To obtain the aggregated total steps for each hour of the day, call the following function:

- `setLocalTimeFilterWithGroup()`



It takes two parameters - `LocalTimeFilter` and `LocalTimeGroup`. If the `LocalTimeFilter` is set from the beginning of today to the current time and the `LocalTimeGroup` of an hour interval is used, the result `dataList` will contain aggregated total steps for every hour of the day.

You can also sum of all elements of the `dataList` and verify that the resulting total matches the step count for the entire day as displayed in the Samsung Health Pedometer tracker.

```
private suspend fun readStepCount(startDate: LocalDateTime, endDate: LocalDateTime) {
    val healthDataStore = HealthDataService.getStore(context)
    val localTimeFilter = LocalTimeFilter.of(startDate, endDate)

    val readRequest = DataType.StepsType.TOTAL.requestBuilder
        .setLocalTimeFilterWithGroup(
            localTimeFilter,
            LocalTimeGroup.of(LocalTimeGroupUnit.HOURLY, 1))
        .setOrdering(Ordering.ASC)
        .build()

    val dataList = healthDataStore.aggregateData(readRequest).dataList
    dataList.forEach {
        val hourlyStepCount = it.value
    }
    val dailyStepCount = dataList.sumOf { it.value as Long }
}
```

## Minimum and Maximum Heart Rate Values

You can retrieve minimum and maximum heart rate data from a specific time range using the following example:

```
suspend fun minHeartRateAggregateRequest(startDate: LocalDate, endDate : LocalDate) {

    val healthDataStore = HealthDataService.getStore(context)
    val localDateFilter = LocalDateFilter.of(startDate, endDate)
    val minHeartRateAggregate = DataType.HeartRateType.MIN.requestBuilder
        .setLocalDateFilter(localDateFilter)
        .build()

    val dataList = healthDataStore.aggregateData(minHeartRateAggregate).dataList
    dataList.forEach {
        val minHR = it.value
    }
}

suspend fun maxHeartRateAggregateRequest(startDate: LocalDate, endDate : LocalDate) {

    val healthDataStore = HealthDataService.getStore(context)
    val localDateFilter = LocalDateFilter.of(startDate, endDate)
    val maxHeartRateAggregate = DataType.HeartRateType.MAX.requestBuilder
        .setLocalDateFilter(localDateFilter)
        .build()

    val dataList = healthDataStore.aggregateData(maxHeartRateAggregate).dataList
    dataList.forEach {
        val maxHR = it.value
    }
}
```

## Last Bedtime of Sleep Goal Data

The aggregate operation LAST refers to the most recent values that have been saved or updated in Samsung Health.

To get the last bedtime goal, see the example below. This value is the same as that in the **Samsung Health > Sleep Tracker > Set target**.

```
suspend fun lastBedTimeAggregateRequest(startDate: LocalDate, endDate : LocalDate) {

    val healthDataStore = HealthDataService.getStore(context)
    val localDateFilter = LocalDateFilter.of(startDate, endDate)
    val lastBedTimeAggregate = DataType.SleepGoalType.LAST_BED_TIME.requestBuilder
        .setLocalDateFilter(localDateFilter)
        .build()

    val dataList = healthDataStore.aggregateData(lastBedTimeAggregate).dataList
    dataList.forEach {
        val lastBedTime = it.value
    }
}
```

## Read Data Changes

Health data in the Samsung Health can be changed or deleted. If you need to get data changes for a specific data type, use:

- `HealthDataStore.readChanges()`

To read sleep data change, see the following example. If the user changed the sleep data in the Samsung Health application, the `readChanges()` gives a changed data result. The result's changed item includes a changed time, a change type with `ChangeType.UPSERT` and the updated data point. You can retrieve deleted sleep data using `ChangeType.DELETE`, it includes the ID of the deleted data record.

```
suspend fun getChangedSleepGoal(lastSyncTime: Instant, endTime: Instant) {

    val healthDataStore = HealthDataService.getStore(context)
    val changeRequest = DataTypes.SLEEP.changedDataRequestBuilder
        .setChangeTimeFilter(InstantTimeFilter.of(lastSyncTime, endTime))
        .build()

    val changes = healthDataStore.readChanges(changeRequest).dataList
    changes.forEach { change ->
        if (change.changeType == ChangeType.DELETE) {
            val deletedDataUid = change.deletedDataUid
        } else if (change.changeType == ChangeType.UPSERT) {
            val upsertDataPoint = change.upsertDataPoint
        }
    }
}
```

## Read Associated Data

You can read data for a specific data with associated data for one or more types.

- `HealthDataStore.readAssociateData()`

The Samsung Health measures the user's oxygen saturation and skin temperature during sleep, if the user slept with a wearable device (like Galaxy Watch). Such data is present in the Samsung Health's sleep tracker.

After getting a specific sleep data point, reading an associated oxygen saturation data is available as the following example.

```
private suspend fun readAssociatedOxygenSaturationData(sleepData: HealthDataPoint) {
    val healthDataStore = HealthDataService.getStore(context)
    val sleepSessionUid = sleepData.uid
    val readAssociateRequest = DataTypes.SLEEP.associatedReadRequestBuilder
        .setIdFilter(IdFilter.fromDataUid(sleepSessionUid))
        .addAssociatedDataType(DataType.SleepType.Associates.BLOOD_OXYGEN)
        .build()
    val oxygenList = healthDataStore.readAssociatedData(readAssociateRequest).dataList
    for (oxygenData in oxygenList) {
        val oxygenDataPoints = oxygenData.getDataPointOf(DataTypes.BLOOD_OXYGEN)
        oxygenDataPoints?.let {
            for (oxygenPoint in it) {
                val oxygen =
                    oxygenPoint.getValue(DataType.BloodOxygenType.OXYGEN_SATURATION)
            }
        }
    }
}
```

## Insert Data

To insert data into the Samsung Health, use the following API and create a HealthDataPoint for a specific data type you want to add:

- HealthDataBuilder.builder()

Then, you can set the data's properties. Properties of HealthDataPoint may vary depending on data type. Be careful to set mandatory properties.

## Set the Source Device

If the measured health data is from a specific device like a weight scale or blood glucose meter, set the data's source device. Otherwise, the source device will be the current local device installing the Samsung Health application.

Find the device information in the registered devices. If no proper device, register a new device information. See:

- Read Devices Registered in Samsung Health
- Register Your Devices into Samsung Health

## Read Devices Registered in Samsung Health

To get a list of the devices registered in the Samsung Health, you should use DeviceManager. The example below shows how to get information about a local device (an Android device in which the Samsung Health application is installed) and a list of **watches** that have stored their data in the Samsung Health:

```
val deviceManager = healthDataStore.getDeviceManager()

val localDevice = deviceManager.getLocalDevice()
Log.i(TAG, "LOCAL DEVICE: id: ${localDevice.id}, type: ${localDevice.deviceType},
manufacturer: ${localDevice.manufacturer}, model: ${localDevice.model}, name:
```

```

${localDevice.name}")

val devices = deviceManager.getDevices(DeviceGroup.WATCH)
devices.forEach { device ->

    Log.i(TAG, "DEVICE: id: ${device.id}, type: ${device.deviceType}, manufacturer:
${device.manufacturer}, model: ${device.model}, name: ${device.name}")
}

```

## Register Your Devices into Samsung Health

As mentioned above, to set the source of inserted data, it is necessary to have your device registered in Samsung Health. To do this, you need to create a DeviceRegistrationRequest. See the following example:

```

fun getSeed(): String {
    return "01:A2:B3:11:C4:D5" // example MAC address used for device seed
}

fun createDevice(): Device {
    return Device
        .accessoryBuilder()
        .addType(AccessoryType.BLOOD_GLUCOSE_METER)
        // add more types to this device, if required
        .build()
}

suspend fun deviceRegistration() {
    val deviceRegistrationRequest =
        DeviceRegistrationRequest.registerDevice(createDevice(), getSeed())

    val healthDataStore = HealthDataService.getStore(context)
    val deviceManager = healthDataStore.getDeviceManager()
    deviceManager.registerDevice(deviceRegistrationRequest)
}

```

## Create a HealthDataPoint Object

Having our device registered, we can obtain its id at any time using provided seed. This can be done in the following way:

```

suspend fun getDeviceId(): String {
    val healthDataStore = HealthDataService.getStore(context)
    // get device based on MAC address used in registration
    val device = healthDataStore.getDeviceManager().getDeviceBySeed(getSeed())
    // return id of registered device
    return device?.id ?: ""
}

```

We can now use the obtained id to assign it while creating our new HealthDataPoint.

See the following example for adding new Continuous Blood Glucose (CGM) data into the Samsung Health. Be careful to set timestamps of each BloodGlucose sample correctly – series data must be enclosed between specified `startTime` and `endTime` of created `HealthDataPoint`. Otherwise, you will receive an error and you will not be able to insert the data. For example, you could consider time period of 15 minutes for the `HealthDataPoint` and add the first `BloodGlucose` entry after first 5 minutes and the second `BloodGlucose` entry after 10 minutes since given `startTime`.

Such approach is presented in the code snippet below:

```
suspend fun createBloodGlucoseDataPoint(
    startTime: Instant,
    endTime: Instant
): HealthDataPoint {
    val FIVE_MINUTES_AS_SECONDS = 5L * 60
    val TEN_MINUTES_AS_SECONDS = 2 * FIVE_MINUTES_AS_SECONDS
    val values = listOf(3.90F, 3.95F)
    val bloodGlucose = BloodGlucose.of(
        values[0],
        startTime.plusSeconds(FIVE_MINUTES_AS_SECONDS))
    val bloodGlucoseTwo = BloodGlucose.of(
        values[1],
        startTime.plusSeconds(TEN_MINUTES_AS_SECONDS))
    val seriesData = listOf(bloodGlucose, bloodGlucoseTwo)

    return HealthDataPoint.builder()
        .setStartTime(startTime)
        .setEndTime(endTime)
        .setDeviceId(getDeviceId())
        .setClientDataId("My ID")
        // set mandatory fields
        .addFieldData(
            DataType.BloodGlucoseType.GLUCOSE_LEVEL,
            values.average().toFloat())
        .addFieldData(
            DataType.BloodGlucoseType.MEASUREMENT_TYPE,
            DataType.BloodGlucoseType.MeasurementType.WHOLE_BLOOD)
        .addFieldData(
            DataType.BloodGlucoseType.MEAL_STATUS,
            DataType.BloodGlucoseType.MealStatus.GENERAL)
        .addFieldData(
            DataType.BloodGlucoseType.SERIES_DATA,
            seriesData)
        // set other fields
        .build()
}
```

You can now write created data into the Samsung Health using:

- `HealthDataStore.insertData()`

The example code is presented below:

```
suspend fun insertBloodGlucoseData(data: HealthDataPoint) {
    val healthDataStore = HealthDataService.getStore(context)
    val insertRequest = DataTypes.BLOOD_GLUCOSE.insertDataRequestBuilder
        .addData(data)
        // add more if needed
        .build()

    try {
        healthDataStore.insertData(insertRequest)
    }
    // handle possible exceptions
}
```

## Update Data

You can also update your application's data in the Samsung Health with:

- `HealthDataStore.updateData()`

After reading data, select the data's ID to update. You can use:

- `uid` – The data's ID managed in the Samsung Health. It is automatically assigned by the Samsung Health when a new data record is created. You can obtain it by making standard read request and accessing returned `HealthDataPoint`'s `uid` field.
- `clientId` – It indicates the ID managed in your application. It works only when the `clientId` has been set at the time of data insertion.

In this example, to update `BloodGlucose` data with the new value for each of element of `seriesData`, let's utilize the `clientId` we assigned while initially creating the `BloodGlucose HealthDataPoint`.

```
suspend fun updateBloodGlucoseData (
    startTime: Instant,
    endTime: Instant
) {
    val healthDataStore = HealthDataService.getStore(context)

    val FIVE_MINUTES_AS_SECONDS = 5L * 60
    val TEN_MINUTES_AS_SECONDS = 2 * FIVE_MINUTES_AS_SECONDS
    val newValues = listOf(5.4F, 5.5F)
    val bloodGlucose = BloodGlucose.of(
        newValues[0],
        startTime.plusSeconds(FIVE_MINUTES_AS_SECONDS))
    val bloodGlucoseTwo = BloodGlucose.of(
        newValues[1],
        startTime.plusSeconds(TEN_MINUTES_AS_SECONDS))
    val seriesData = listOf(bloodGlucose, bloodGlucoseTwo)

    val updatedData = HealthDataPoint.builder()
        .setStartTime(startTime)
        .setEndTime(endTime)
        .addFieldData(
            DataType.BloodGlucoseType.GLUKOSE_LEVEL,
            newValues.average().toFloat())
        .addFieldData(
            DataType.BloodGlucoseType.SERIES_DATA,
            seriesData)
        .build()

    val updateRequest = DataTypes.BLOOD_GLUCOSE.updateDataRequestBuilder
        // updating with custom clientId assigned by user
        .addDataWithClientId(clientId = "My ID", data = updatedData)
        // you may update more data entries in the request
        .build()

    try {
        healthDataStore.updateData(updateRequest)
    }
    // handle possible exceptions
}
```

## Delete Data

You can also delete data, inserted by you into the Samsung Health, using:

- `HealthDataStore.deleteData()`

To build a delete request, it is necessary to specify the data type of the data to be deleted with it.

Remember, that you can delete only the data inserted by you. Otherwise, when trying to delete the data from another application source (e.g. entered manually in the Samsung Health application), you will get an `AuthorizationException` with an error code 2002: `ERR_NO_OWNERSHIP_TO_WRITE`.

For more details regarding exceptions and error handling please refer to `com.samsung.android.sdk.health.data.error` package of the SDK's API Reference.

In order to get uid of the last entered BloodGlucose data, first we need to make a standard read request. We set `LocalTimeFilter` with given `startTime` and `endTime` and apply ascending `Ordering`. Please make sure to set the `LocalTimeFilter` correctly to include all of your Continuous Blood Glucose (CGM) data, which might concern significant amount of time. We also need to apply `ReadSourceFilter` in order to make sure the last inserted data originated from our application.

If the result list of the request is not empty (meaning there were some BloodGlucose inserts before), we simply take the last element of it and access its uid.

Below, there is an example of deleting the last inserted Continuous Blood Glucose (CGM) data.

```
suspend fun readLastBloodGlucoseUid(
    startTime: LocalDateTime,
    endTime: LocalDateTime,
    appName: String
): String {
    val healthDataStore = HealthDataService.getStore(context)
    var lastBloodGlucoseUid = ""
    val localTimeFilter = LocalTimeFilter.of(startTime, endTime)
    val sourceFilter = ReadSourceFilter.fromApplicationId(appName)

    val readRequest = DataTypes.BLOOD_GLUCOSE.readDataRequestBuilder
        .setLocalTimeFilter(localTimeFilter)
        .setSourceFilter(sourceFilter)
        .setOrdering(Ordering.ASC)
        .build()

    CoroutineScope(Dispatchers.IO).async {
        try {
            val readResponse = healthDataStore.readData(readRequest)
            val responseDataList = readResponse.dataList

            if (responseDataList.isNotEmpty()) {
                lastBloodGlucoseUid = responseDataList.last().uid
            }
        }
        // handle possible exceptions
    }.await()

    return lastBloodGlucoseUid
}
```

Now, when we successfully obtained uid, we can use it to delete the data by making a delete request with applied `IdFilter`:

```
suspend fun deleteLastBloodGlucoseData(lastBloodGlucoseUid: String) {
    val healthDataStore = HealthDataService.getStore(context)
    val idFilter = IdFilter.fromDataUid(lastBloodGlucoseUid)

    val deleteRequest = DataTypes.BLOOD_GLUCOSE.deleteDataRequestBuilder
        .setIdFilter(idFilter)
```

```
        .build()

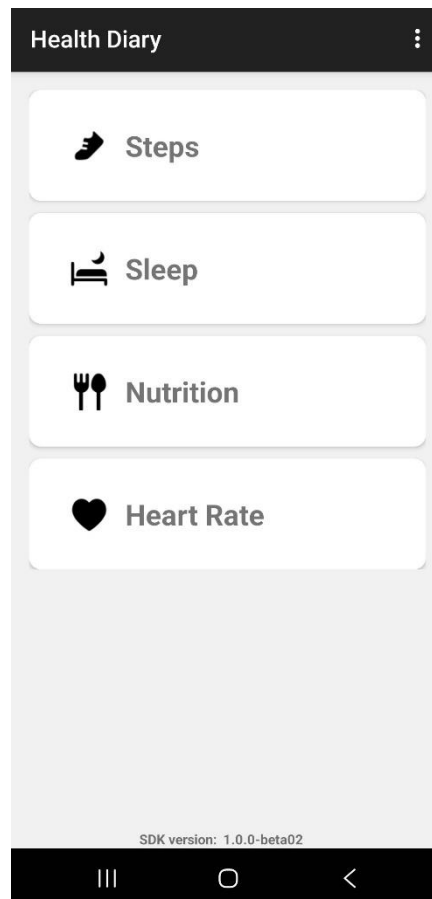
    try {
        healthDataStore.deleteData(deleteRequest)
    }
    // handle possible exceptions
}
```



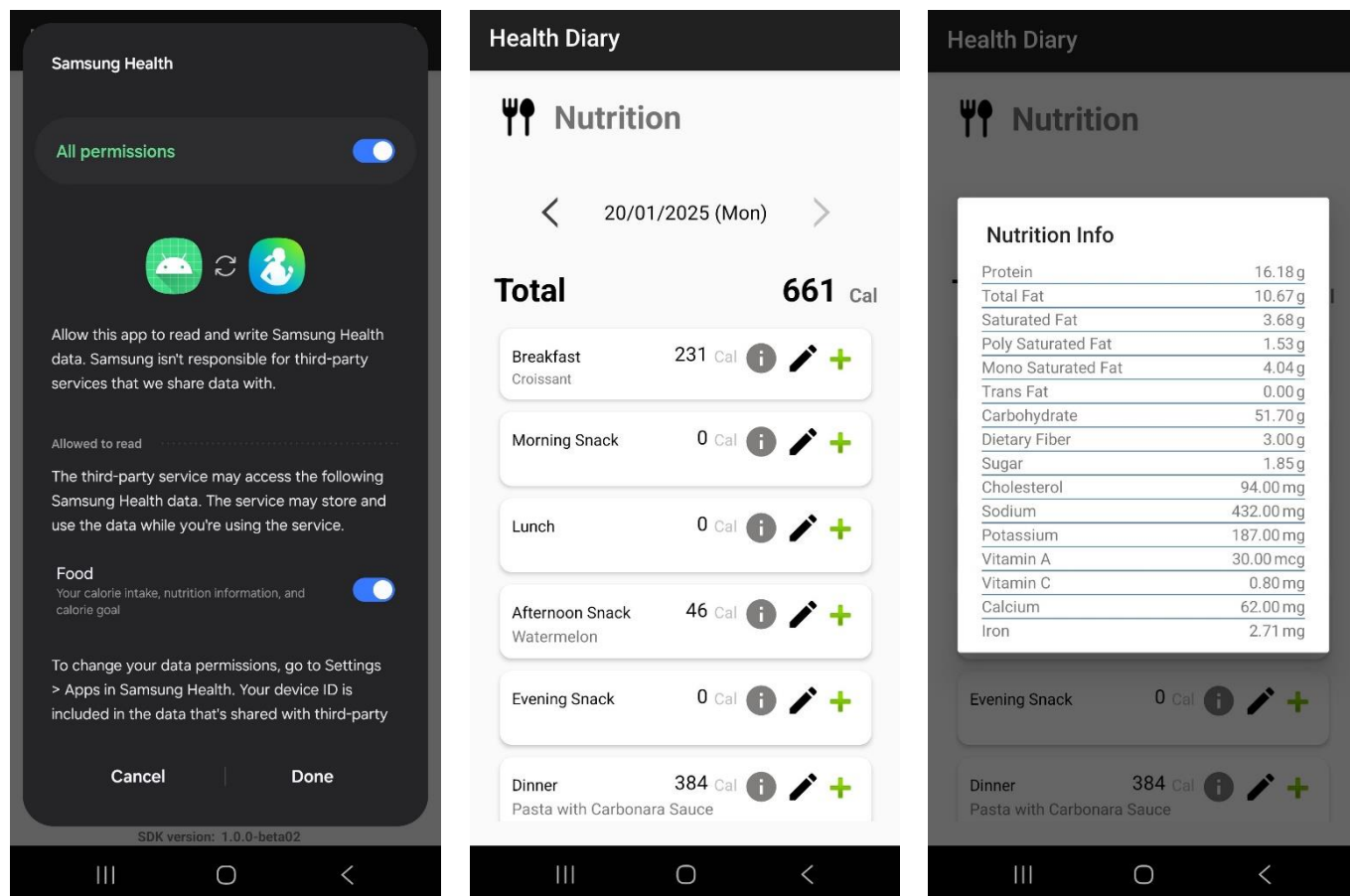
## Sample Application – HealthDiary

The HealthDiary application enables you to learn an entire data access flow. It includes the data permission request and representative data request operations. Requests for steps, sleep, nutrition, and heart rate data are processed on a daily basis.

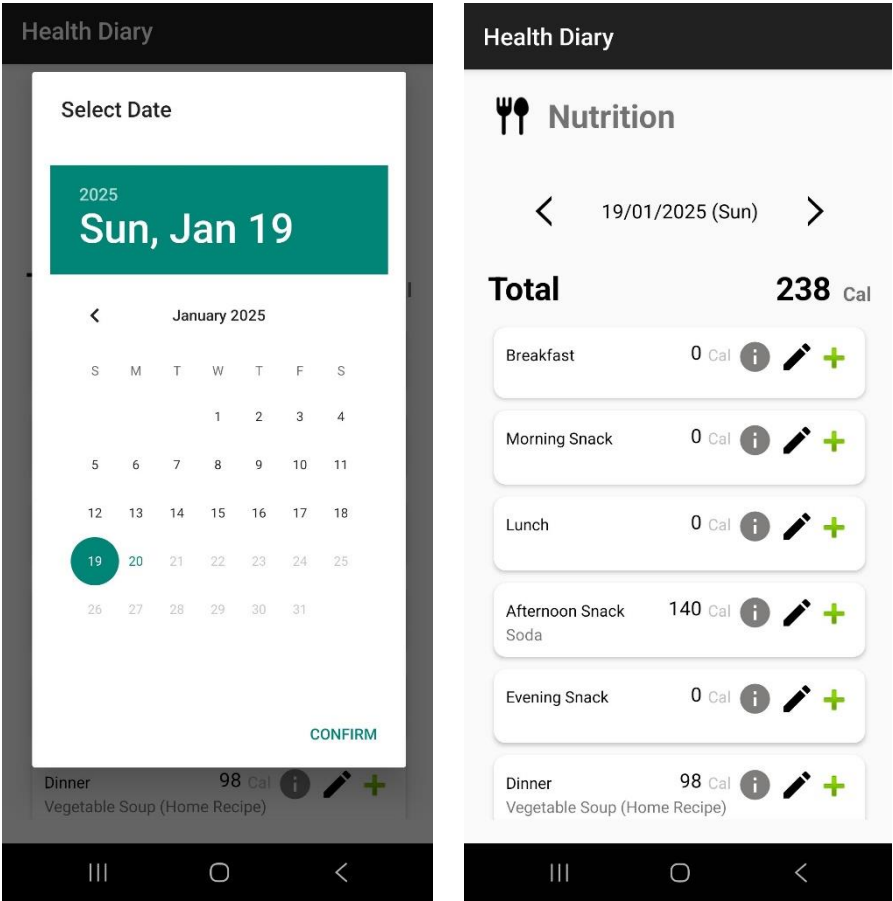
Open the HealthDiary application project and see the detailed code. If you build and run the HealthDiary application, the following main activity is shown.



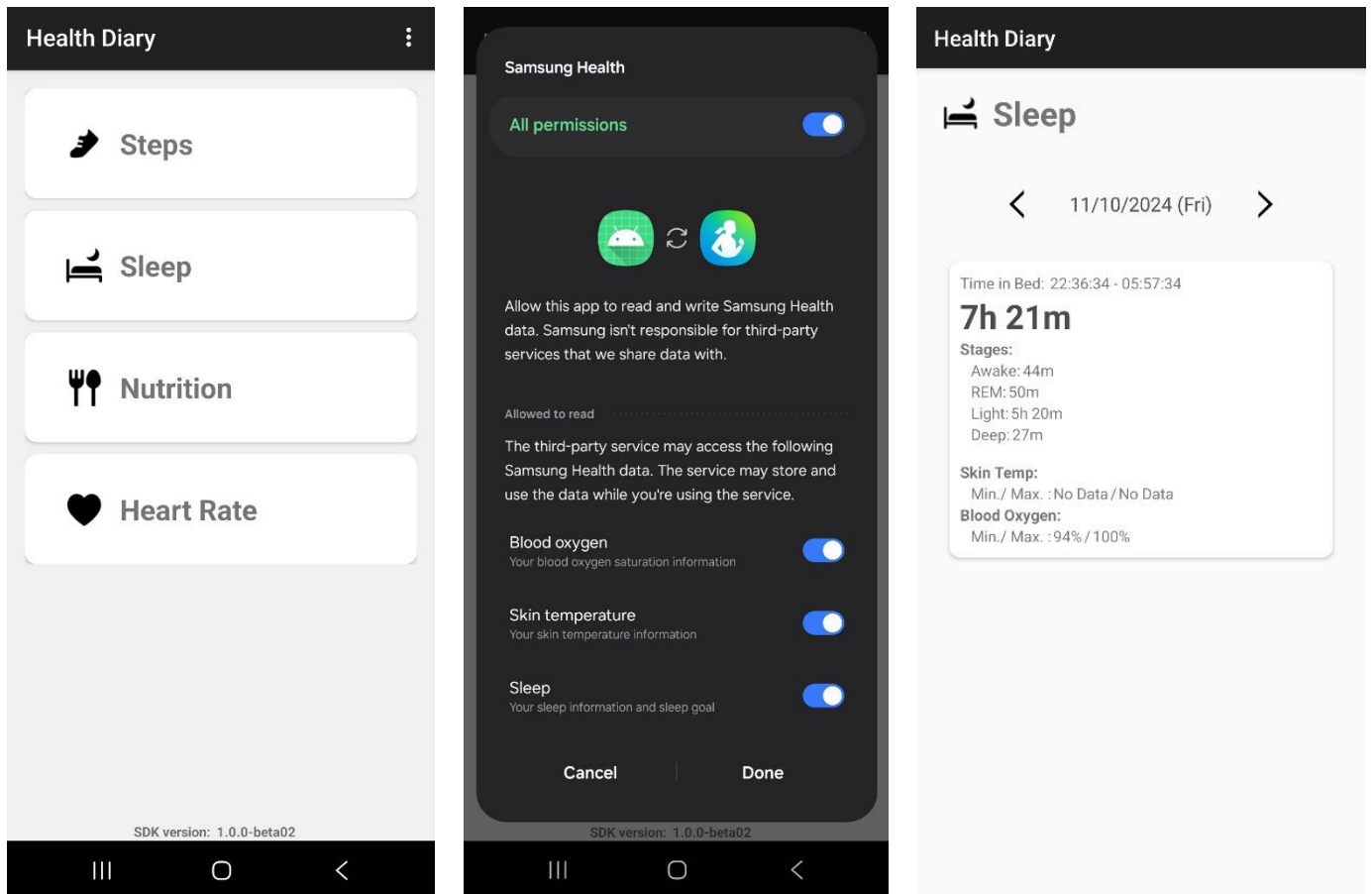
In the main activity display, select one of the data types. To acquire a data permission allowance, a data permission popup will be shown. If all permissions are allowed, the next page will show retrieved data from the Samsung Health. Click on the information icon to get the nutrition information of the meal.



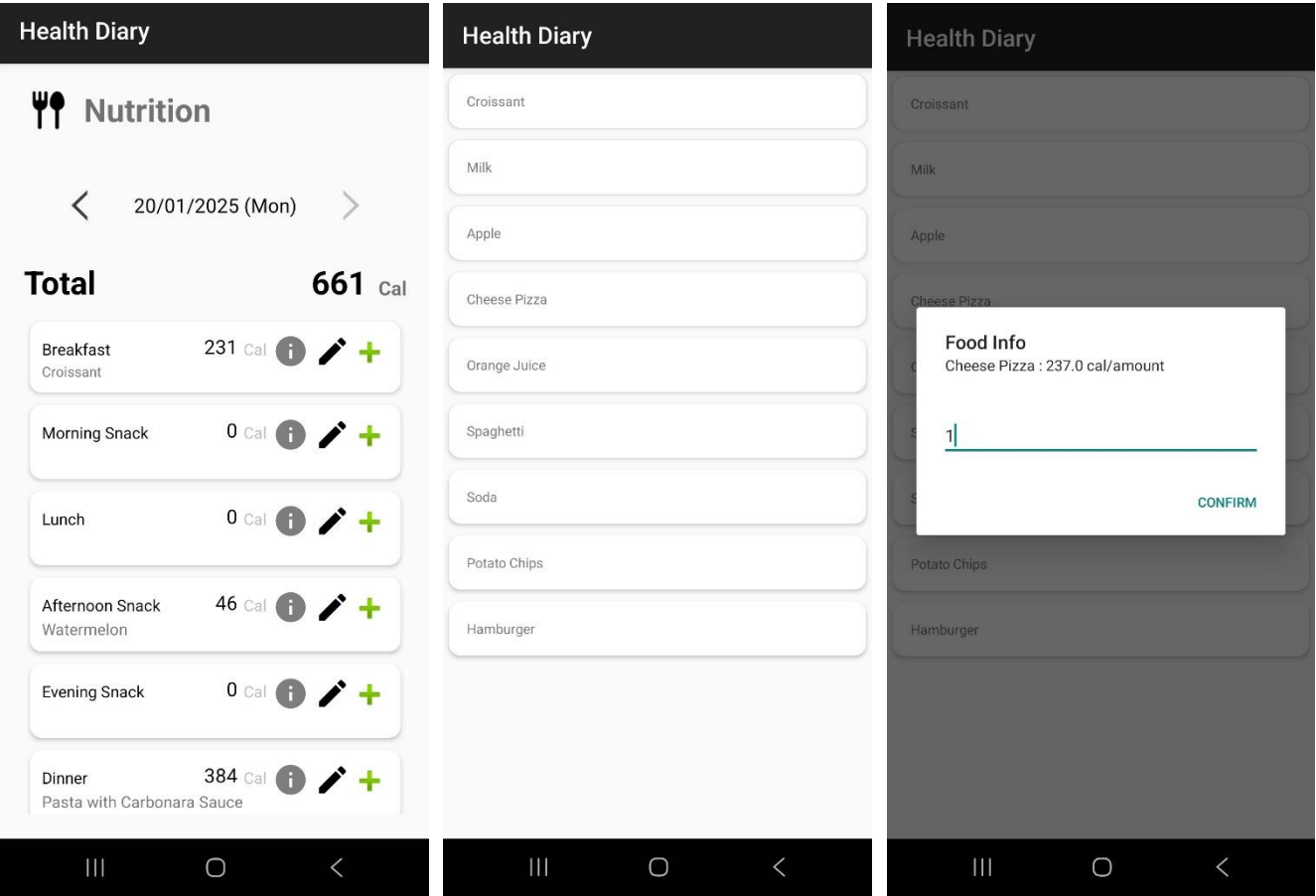
Changing to another date is available by selecting the date line. Whenever a date is changed, the sample application reads data for the given date and lists up the result data.



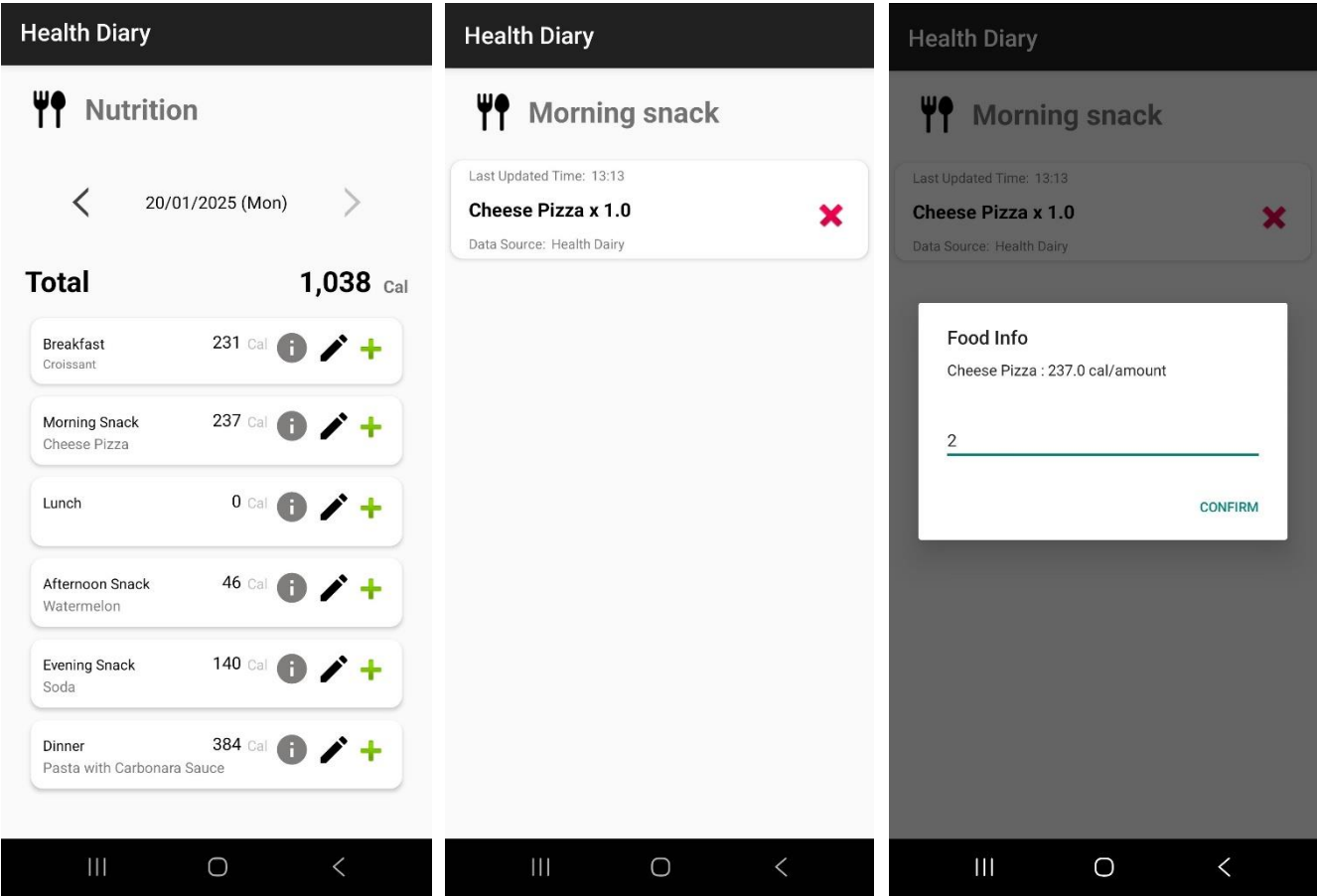
The sleep activity shows a reading associated data for a specific sleep data. The associated oxygen saturation and skin temperature are shown with an average aggregate operation.



Data insertion can be performed as follows. By clicking on the add button, an option to select food item is shown. Select an item, add its quantity and press confirm.



Update or delete can be performed in similar manner. Press the edit button. If you want to update the data, tap on the card and update the quantity. To delete, press the cross the button.



# Tool - DataViewer

The Samsung Health Data SDK provides the DataViewer tool, which allows for easy viewing of saved data in Samsung Health.

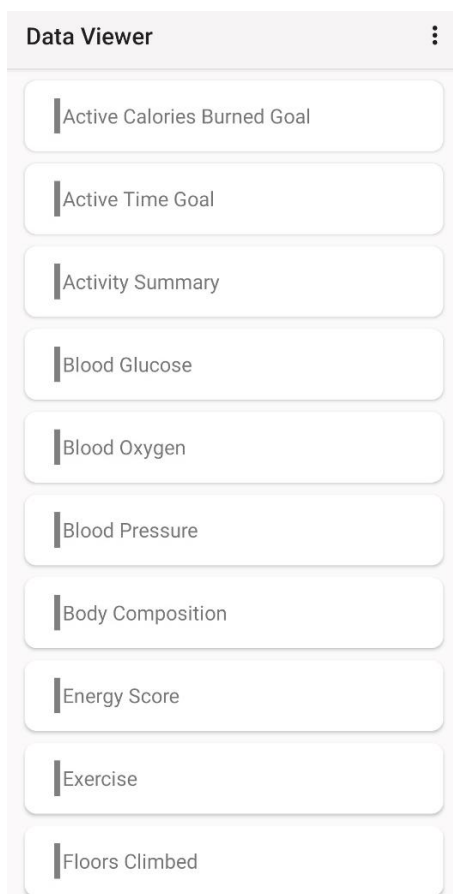
## Install DataViewer

Find the DataViewer APK file and install it on your smartphone.

```
> adb install dataviewer.apk
```

## Checking Saved Data




After installation, open the DataViewer, you can now check the saved data in the Samsung Health by selecting one of the data types in the figures below:



Each data type can be accessed after getting a data permission allowance. Whenever you select a data type, the data permission popup displays. In the saved data list, you can see data details by selecting each data.

Samsung Health

All permissions



Allow this app to read and write Samsung Health data. Samsung isn't responsible for third-party services that we share data with.

Allowed to read

The third-party service may access the following Samsung Health data. The service may store and use the data while you're using the service.

Steps

Your step count and step goal

To change your data permissions, go to Settings > Apps in Samsung Health. Your device ID is included in the data that's shared with third-party services.

Cancel | Done

Steps

2024-10-12T00:00

localStartTime 2024-10-12T00:00

localEndTime 2024-10-13T00:00

total\_steps 5562 [\(Click To View Details\)](#)

2024-10-11T00:00

2024-10-10T00:00

2024-10-09T00:00

2024-10-08T00:00

2024-10-07T00:00

2024-10-06T00:00

2024-10-05T00:00

2024-10-04T00:00

2024-10-03T00:00

2024-10-02T00:00

2024-10-01T00:00

2024-09-30T00:00

2024-09-29T00:00

2024-09-28T00:00

NEXT