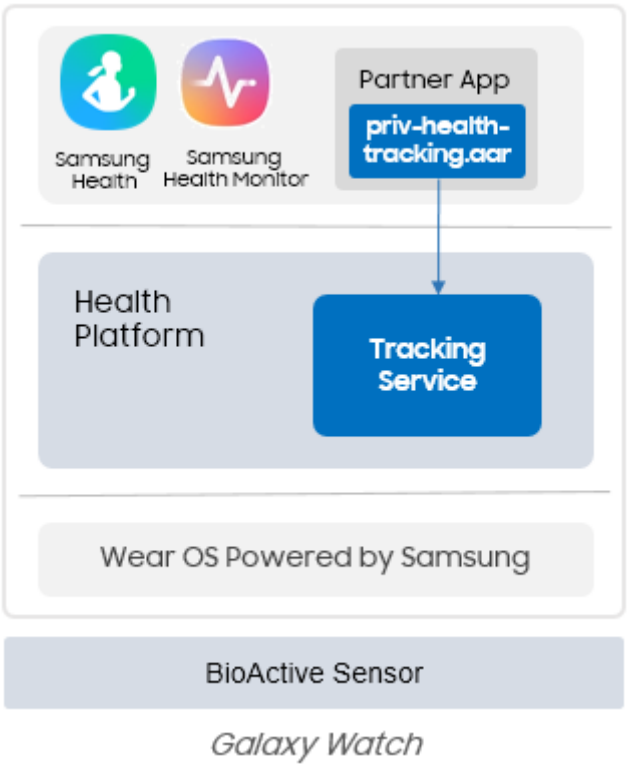
Programming Guide
- Samsung Privileged Health SDK | Tracking Service

v1.1.0

Health Platform provides tracking health data and access the health data storage's data. Its tracking service gives raw and processed sensor data like accelerometer, PPG and body composition data that BioActive sensor sent. The new Galaxy Watch's BioActive sensor precisely runs three powerful health sensors such as PPG, ECG, BIA, sweat loss, and SpO2.

Samsung Privileged Health SDK's Tracking APIs enable a partner app to track health related sensor data. An application import the SDK's tracking library (`priv-health-tracking.aar`) and sets a tracking event listener to receive data events. Tracking data is transferred to the application with one time or periodical events until removing the event listener. A number of events and an event frequency is different depending on a tracking type.



Main features of Tracking APIs are:

Capabilities

It gives available tracking type list whether the watch supports.

Tracking Watch's Sensor Data

A partner app can track Galaxy watch's health related sensor data with Samsung Privileged Health SDK's Tracking APIs. Supported tracking types are:

[Batching]

- Accelerometer
- Heart rate including IBI
- PPG Green

[On-demand]

- BIA
- ECG
- PPG IR
- PPG Red
- SpO2 (oxygen saturation)
- Sweat loss

Glossary

Glossary	Description
BIA	Bioelectrical Impedance analysis
BioActive Sensor	A new 3-in-1 sensor uses a single chip embedded in the new Galaxy Watch.
ECG	Electrocardiogram
Health Platform	It indicates a Samsung's health platform that provides tracking and access the user's health data.
IBI	Inter-beat Interval
PPG	Photoplethysmogram
Samsung Privileged Health SDK	It provides a tracking package to track a watch's raw or processed sensor data. A partner app imports the SDK's library 'priv-health-tracking.aar' and can track the watch's sensor data.
Tracking Service	It is included in Health Platform. It tracks a watch's raw or processed sensor data.

Data Specifications

> **Note**
>
> Measured data by Samsung Privileged Health SDK is for fitness and wellness only, not for the diagnosis or treatment of any medical condition.

Batching Types

An application can retrieved sensor data with a batching event. The event will be retrieved every specified period with collected sensor data.

Tracking Type	Event Type	Description
ACCELEROMETER	Batching	In 300 `AccelerometerSet` data points, every 12 seconds (25 hz).
HEART_RATE	Batching	In 600 `HeartRateSet` data points, every 10 minutes (1 hz).
	Batching / Watch display-on	1 `HeartRateSet` data point, every 1 second (1 hz).
PPG_GREEN	Batching	In 300 `PpgGreenSet` data points, every 12 seconds (25 hz).

On-demand Types

An on-demand tracking type is not for continuous measurement. Please track on-demand type sensors in 30 seconds.

Tracking Type	Event Type	Description
BIA	On-demand	1 `BiaSet` data point, every 1 second (1 hz).
ECG	On-demand	5 `EcgSet` data points, every 10 milliseconds (500 hz). Otherwise, 10 `EcgSet` data points, every 20 milliseconds (500 hz).
PPG_IR	On-demand	1 `PpgIrSet` data point, every 10 milliseconds (100 hz).
PPG_RED	On-demand	1 `PpgRedSet` data point, every 10 milliseconds (100 hz).
SPO2	On-demand	1 `Spo2Set` data point for a measurement. ** Tracking SpO2 is available with watch **Health Platform v1.3.0** or above.*
SWEAT_LOSS	On-demand	1 `SweatLossSet` after a running exercise. ** Tracking* sweat loss is available after the **watch software update in Feb, 2022**.

Development Environment

Device Target

Samsung Privileged Health SDK's Tracking APIs work on the following watch devices or above running Wear OS Powered by Samsung.

- Galaxy Watch4
- Galaxy Watch4 Classic

SDK Content

Folder structure	Content description
`/docs`	Guide and API Reference
`/libs`	SDK's tracking library. Import it in your app project. `- priv-health-tracking.aar`
`/sample-code`	App project including sample codes. It's project name is: `- TrackingSampleApp`

Restrictions

- Samsung Privileged Health SDK doesn't support an emulator.

Developer Support

If you meet a trouble to use Samsung Privileged Health SDK, please submit a query in Samsung developer site > Support > Developer Support with a bugreport log and a detailed issue description.

You can get the bugreport with:

1) Open Galaxy Wearable app on the phone and select More (Ξ) > Galaxy Wearable Info.
2) Touch [Galaxy Wearable] 10 times.
3) Select [Run Watch Dump] and wait.
4) Find and share MyFiles > Internal Storage > Download > log > GearLog > bundled-bugreport-xx.zip.

> **Note**
>
> For using Health Services APIs, create a new issue in Android developer site.

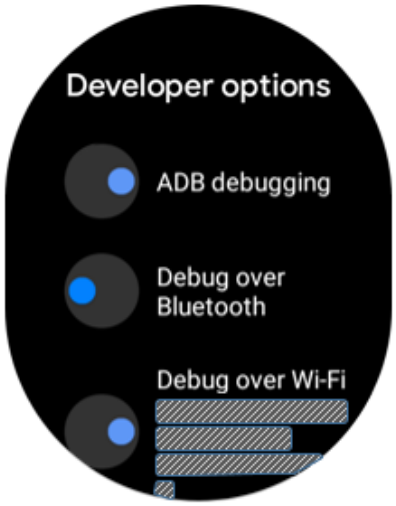
Device Settings

To debug your application, set a Galaxy Watch with following steps.

Turn on Watch's Developer Mode

If there is no menu in the watch **Settings > Developer options**, turn on the watch's developer mode.

1. Go to the watch's **Settings > About watch > Software** and tap 5 times on **Software version**.
2. In **Settings > Developer options**,
   A. Enable **ADB debugging**.
   B. Enable **Debug over Wi-Fi.**



Connect Watch to Your PC

> **Note**
>
> If you have a Galaxy Watch cradle, pass this step and place a watch on a watch cradle. And connect a 5 pin USB cable with your PC.

Enable the watch's Wi-Fi and connect to you PC.

1. Go to **Settings > Connection > Wi-Fi** and enable it.
2. Select your PC from the available Wi-Fi network list.
3. If it succeeds, the watch's IP address will display. Note the IP Address.

4. In your PC's Android Studio's **Terminal**,

   A. Connect the watch with the IP Address. If the IP address is `100.200.30.4:5555`:

   ```
   adb connect 100.200.30.4:5555
   ```

   B. Watch will display a popup. On the watch, select **Always allow from this computer** to allow debugging.

   C. If a connection succeeds, the following message displays on **Terminal**.

   ```
   connected to 100.200.30.4:5555
   ```

Now the watch is connected to the Android Studio's debugger.

Health Platform's Developer Mode

Samsung Privileged Health SDK's Tracking APIs work properly for registered partner apps in a Samsung health system. Health team registers the partner app's info including an app package name and signature (sha256) in the Samsung health system. And Health Platform on a watch device allows to track data only if a running app's package name and signature both are matched with the registered info. Otherwise, the SDK gives `SDK_POLICY ERROR`.

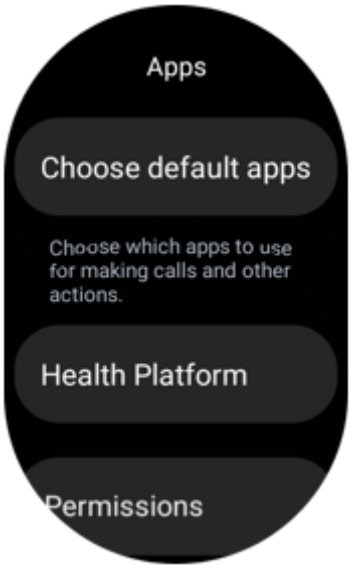
A registered app signature in the Samsung health system should be from the application's release key. If partner app developer would like to run or test the app with a different app key, use the Health Platform's developer mode.

> **Note**
>
> The Health Platform's developer mode is ONLY for a test or debug your app. It is NOT for app users. Please do not provide a developer mode guide to app users.

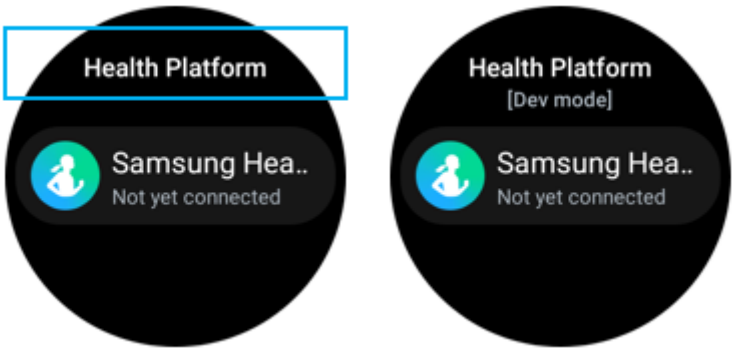
Developer Mode On

The developer mode can be enabled with:

1. Pull up the display and select the **Settings** icon.
2. Select **Apps** and swipe down. Select **Health Platform**.

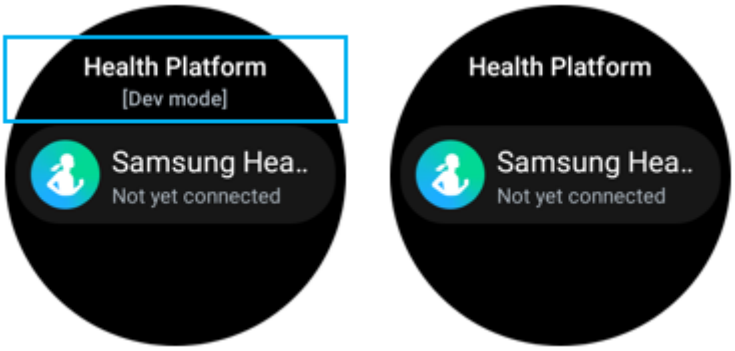


3. Tap the **Health Platform** title part quickly about 10 times.
4. **[Dev mode]** displays after the Health Platform title.
   It means the developer mode enabled.



Developer Mode Off

Tapping the **Health Platform** title makes the developer mode disabled.



Hello Tracking Service

Importing Tracking Library

Import the tracking library into your app project:

- `priv-health-tracking.aar`

Permission Declarations in Manifest

Add permissions in your app's manifest.

[AndroidManifest.xml]

```
<uses-permission android:name="android.permission.BODY_SENSORS"/>
<uses-permission android:name="android.permission.ACTIVITY_RECOGNITION"/>
```

Connecting with Health Platform

Connect to Health Platform with:

- `HealthTrackingService.connectService()`

```
import com.samsung.android.service.health.tracking.HealthTrackingService;
import com.samsung.android.service.health.tracking.ConnectionListener;
import com.samsung.android.service.health.tracking.data.HealthTrackerType;


public class MainActivity extends FragmentActivity {

    public HealthTrackingService healthTracking;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        // Connect to Health Platform.
        healthTracking = new HealthTrackingService(connectionListener, this);
        healthTracking.connectService();

        //...
    }

    @Override
    protected void onDestroy() {
        //.Disconnect the tracking service.
        healthTracking.disconnectService();
    }
```

Implement a tracer event listener with:

- `HealthTracker.TrackerEventListener`

If the connection failed, check `HealthTrackerException`. If the exception has a resolution, resolve it by calling `resolve()` API.

```
    private final ConnectionListener = new ConnectionListener(){

        @Override
        public void onConnectionSuccess() {
            // Connection success.

            // Capability Check
            // Tracking Data
        }

        @Override
        public void onConnectionEnded() {
            // Connection is ended.
        }

        @Override
        public void onConnectionFailed(HealthTrackerException e) {
            // Connection failed. Check the error and resolve.
            if(e.hasResolution()) {
                e.resolve(AccelerometerActivity.this);
            }
        }

    };
```

If the connection fails, check an error and resolve it with:

- `HealthTrackerException.getErrorCode()`
- `HealthTrackerException.hasResolution()`
- `HealthTrackerException.resolve()`

Capability Check

After connecting with Health Platform succeeded, check the watch device's available tracker types.

```
List<HealthTrackerType> availableTrackers =
        healthTracking.getTrackingCapability().getSupportHealthTrackerTypes();
```

Tracking Data

If a required tracker type is available on the watch device, track data with:

```
protected final void trackPpgGreen() {
    // Start PPG Green data tracking.
    HealthTracker ppgGreenTracker = null;
    ppgGreenTracker = healthTracking.getHealthTracker(HealthTrackerType.PPG_GREEN);
    ppgGreenTracker.setEventListener(trackerListener);
}
```

An event for tracking data is retrieved in:

- `HealthTracker.TrackerEventListener.onDataUpdated(List<DataPoint> list)`

Receiving data occurs until unsetting the event listener.

```
private final HealthTracker.TrackerEventListener trackerListener = new
HealthTracker.TrackerEventListener() {
    @Override
    public void onDataReceived(List<DataPoint> list) {

    }

    @Override
    public void onError(HealthTracker.TrackerError error) {
        // Error occurs.
    }

    @Override
    public void onFlushCompleted() {
        // Flushing data is completed.
    }
};
```

If you get `SDK_POLICY_ERROR`, it indicates that your app's SDK policy was not downloaded yet.

> **Note**
>
> An interval time between each DataPoint's timestamp in `TrackerEventListener.onDataReceived()` can be variable. An order of received data points is not changed. Handle the received data orderly with constant timestamp difference. E.g. accelerometer data has 40 milliseconds time interval for each data point. See Data Specifications.

Flushing Data

If a watch's display is off, Health Platform provides batched tracking data. `Flush()` gives collected data instantly.

```
protected final void flushPpgGreen() {
    // Flushing data gives collected data instantly.
    ppgGreenTracker.flush();
}
```

E.g. Tracking accelerometer data gives 300 samples every 12 seconds normally.

- If `Flush()` is called when 250 samples are gathered, a `DataPoint` list in `onDataUpdated()` gives only 250 samples.
- After getting the flush result, `onDataUpdated()` gives 300 samples again.

Stop Tracking

If your app doesn't need to track data any more, unset the registered event listener with:

```
protected final void stopTracking() {
    // Unset the event listener stops tracking data.
    ppgGreenTracker.unsetEventListener();
}
```

Tracking Sweat Loss

> **Note**
>
> The sweat loss feature works after the watch software update in Feb, 2022.
>
> Tracking APIs provide sweat loss tracking for only a running exercise currently.

Samsung Privileged Health SDK provides tracking sweat loss after an exercise. To get a sweat loss data, tracking an exercise is required with Wear OS's Health Services.

The following steps describe how to track sweat loss data.

Getting a HealthTracker Instance for Sweat Loss

After a service connection succeeded, a starting point is getting a `HealthTracker` instance

- `HealthTrackingService.getHealthTracker(HealthTrackerType, TrackerUserProfile, ExerciseType)`

The user profile can set with `TrackerUserProfile.Builder()`.

[SweatLossActivity.java]

```
private TrackerUserProfile profile;

    //...

   // Set the user profile.
   profile = new TrackerUserProfile.Builder()
       .setHeight(height)
       .setWeight(weight)
       .setGender(gender)
       .setAge(age)
       .build();

   // ...
```

Create a `HealthTracker` instance for sweat loss with the user profile above and the `ExerciseType.RUNNING`.

[SweatLossActivity.java]

```
private HealthTracker sweatLossTracker = null;

// If HealthTrackingService.connectService() is called
private final ConnectionListener connectionListener = new ConnectionListener() {
    @Override
    public void onConnectionSuccess() {

        try {
            // Get a HealthTracker instance for sweat loss.
            sweatLossTracker =
                healthTrackingService.getHealthTracker(HealthTrackerType.SWEAT_LOSS,
                profile,
                com.samsung.android.service.health.tracking.data.ExerciseType.RUNNING);
        } catch (final IllegalArgumentException e) {
            // Exception handling
            );
            finish();
        } catch (final UnsupportedOperationException e) {
            // Exception handling
            );
            finish();
        }
```

And set an event listener.

[SweatLossActivity.java]

```
        // ...

        // Set an event listener for getting sweat loss data
        sweatLossTracker.setEventListener(trackerEventListener);

    }
    // ...
};
```

Starting an Exercise with Health Services

Sweat loss is retrieved after an exercise. Getting exercise data is available with Health Services.

Create an `ExerciseClient` and set an event listener for an exercise data update.

[WHSManager.java]

```
private ExerciseClient exerciseClient;

private void initExerciseClient() {
    if (exerciseClient == null) {
        exerciseClient = HealthServices.getClient(mContext).getExerciseClient();
        exerciseClient.setUpdateListener(exerciseUpdateListener);
    }
}
```

And start a running exercise with setting data types for `STEPS_PER_MINUTE` and `DISTANCE`.

`STEPS_PER_MINUTE` will be used in `HealthTracker.setExerciseData()` and `DISTANCE` is helpful to check an error case when a sweat loss's status is not zero.

[WHSManager.java]

```
protected void startExercise() {
    isExceriseStarted = true;
    Set<DataType> dataType = new HashSet<>();
    dataType.add(DataType.STEPS_PER_MINUTE);
    dataType.add(DataType.DISTANCE);

    ExerciseConfig.Builder exerciseConfigBuilder =
            ExerciseConfig.builder()
                    .setExerciseType(ExerciseType.RUNNING)
                    .setAggregateDataTypes(dataType)
                    .setDataTypes(dataType)
                    .setShouldEnableGps(useGps);

}
```

Updated data will be retrieved in `ExerciseUpdateListener`.

[WHSManager.java]

```
private final ExerciseUpdateListener exerciseUpdateListener =
        new ExerciseUpdateListener() {

            @Override
            public void onExerciseUpdate(ExerciseUpdate update) {

                ExerciseState state = update.getState();
                if (state == ExerciseState.AUTO_ENDED || state ==
ExerciseState.TERMINATED) {
                    // Set the exercise state to STOP and return.
                }

                update.getLatestMetrics().forEach(this::setMetricView);

            }

            // Override the listener's other APIs.

        };
```

Setting the Exercise State to START

After setting a running exercise data measurement with Health Services, set the exercise state to START.

[SweatLossActivity.java]

```
@Override
public void startExercise() {
    try {
        // Set the exercise state with START.
        sweatLossTracker.setExerciseState(ExerciseState.START);
    } catch (final IllegalStateException e) {
        // Exception handling
        });
    }

}
```

Setting Exercise Data

Measured exercise data with Galaxy watch is retrieved in `ExerciseUpdateListener.onExerciseUpdate()` of Health Services.

[WHSManager.java]

```
private final ExerciseUpdateListener exerciseUpdateListener =
        new ExerciseUpdateListener() {

            @Override
            public void onExerciseUpdate(ExerciseUpdate update) {

                ExerciseState state = update.getState();
                if (state == ExerciseState.AUTO_ENDED || state ==
ExerciseState.TERMINATED) {
                    // Set the exercise state to STOP and return.
                }

                update.getLatestMetrics().forEach(this::setMetricView);

            }

            // Override the listener's other APIs.
```

Take retrieved exercise data's `STEPS_PER_MINUTE` to set exercise data with Samsung Privileged Health SDK's `HealthTracker.setExerciseData()`.

Checking the exercise data's duration and distance is helpful. If the exercise's duration is less than 5 minutes or the distance is less than 2 kilometers, Samsung Privileged Health SDK will give an error in a sweat loss' STATUS.

For more errors of `SweatLossSet.STATUS`, see Samsung Privileged Health SDK's Tracking API Reference.

```
        private void setMetricView(DataType dataType, List<DataPoint> dataPoints) {

            if (dataType.getName().equals(STEPS_PER_MINUTE)) {
                int listSize = dataPoints.size();

                float[] spmdata = new float[listSize];
                long[] timeStamp = new long[listSize];
                for (int i = 0; i < listSize; ++i){
                    spmdata[i] = (float) Iterables.
                            get(dataPoints, i).getValue().asLong();
                    timeStamp[i] = getUtcTimeFromSystemElapsedTime(
                            Iterables.get(dataPoints, i)
                                    .getEndDurationFromBoot()
                                    .toMillis());
            }
            if (isExceriseStarted)
            {
                callback.spmCallback(spmdata, timeStamp);
            }
        }
        if(dataType.getName().equals(DISTANCE)) {
            int listSize = dataPoints.size();

            double[] distancedata = new double[listSize];
            for (int i = 0; i < listSize; ++i){
                distancedata[i] = Iterables.get(dataPoints, i).getValue().asDouble();
            }
            if (isExceriseStarted)
            {
                callback.distanceCallback(distancedata);
            }
        }
}
```

And set exercise data with Samsung Privileged Health SDK's `HealthTracker.setExerciseData()`.

[SweatLossActivity.java]

```
@Override
public void spmCallback(float[] spmData, long[] timeStamp) {
    try {
        sweatLossTracker.setExerciseData(DataType.STEP_PER_MINUTE, spmData,
timeStamp);
    } catch (final IllegalArgumentException e) {
        // Exception handling
        );
    } catch (final IllegalStateException e) {
        // Exception handling
    }

}
```

Setting the Exercise State with STOP

If the exercise is finished, set an exercise state to STOP.

[SweatLossActivity.java]

```
@Override
public void endExercise(boolean exception, String state) {
    if(!state.equals("auto ended")) {
        try {
            sweatLossTracker.setExerciseState(ExerciseState.STOP);
        } catch (final IllegalStateException e) {
            // Exception handling
        }
    }

    // ...

}
```

Checking the Tracker Listener

If the running exercise is finished well, sweat loss data will be retrieved in `HealthTracker.TrackerEventListener`. Check a sweat loss's value and its status.

[SweatLossActivity.java]

```
private final HealthTracker.TrackerEventListener trackerEventListener = new
HealthTracker.TrackerEventListener() {
    @Override
    public void onDataReceived(@NonNull List<DataPoint> list) {
        if (list.size() != 0) {
            float sweatLoss = list.get(0).getValue(ValueKey.SweatLossSet.SWEAT_LOSS);
            int status = list.get(0).getValue(ValueKey.SweatLossSet.STATUS);
                if(status == 0)
                    // No error
                else
                    // Check status and handle an error.
            });
        } else {
            // No data
        }
    }

    @Override
    public void onFlushCompleted() {
        // ...
    }
```

```
    @Override
    public void onError(HealthTracker.TrackerError trackerError) {
        if (trackerError == HealthTracker.TrackerError.PERMISSION_ERROR) {
            // Permission handling
        }
        if (trackerError == HealthTracker.TrackerError.SDK_POLICY_ERROR) {
            // Check a device's network or submit a query in Samsung dev site.
        }
        // ...
    }
};
```

Unsetting the Tracker Event Listener

If you don't need to get sweat loss, unset the registered event listener.

Proguard Rule

Setting the proguard rule in your app in `app/build.gradle` is helpful to debug your app. You can set the following proguard rule for debugging stack traces and comment it for an app release.

```
-keep class com.samsung.android.service.health.tracking.** {
    *;
}
```

User Guide for Measurement

BIA

Measure BIA with Galaxy Watch with following steps.

> **Note**
>
> Measurement results may not be accurate if the user is under 20 years old.

1. Make sure your watch is on tightly. And raise your arms so your armpits are open.

2. Place your middle finger on 2 o’clock key and ring finger on 4 o’clock key on the watch.

3. Touch your watch only. Don’t let your hand on the watch’s keys touch another hand on the watch.

4. Keep touching keys until the measurement is completed

ECG

Measuring ECG data is available with:

1. Make sure your watch is on tightly.

2. Rest your figure lightly on the Watch’s 2 o’clock key.

Heart Rate and PPG (Green / IR / Red)

Make sure your watch is on tightly to measure heart rate and PPG (Green / IR / Red).

SpO2

If you have a trouble to measure SpO2, please check the watch’s location on your wrist and a pose. And please be careful not to move.

Sample Code Description

`TrackingSampleApp` is a sample code app project. You can find codes for how to track each data type like heart rate or PPG green.

Build the app project and run the sample app.

'Sensors' menu enable to track a given sensor data.

As selecting 'START', 'STOP', or 'FLUSH' button, you can monitor retrieved sensor data below.