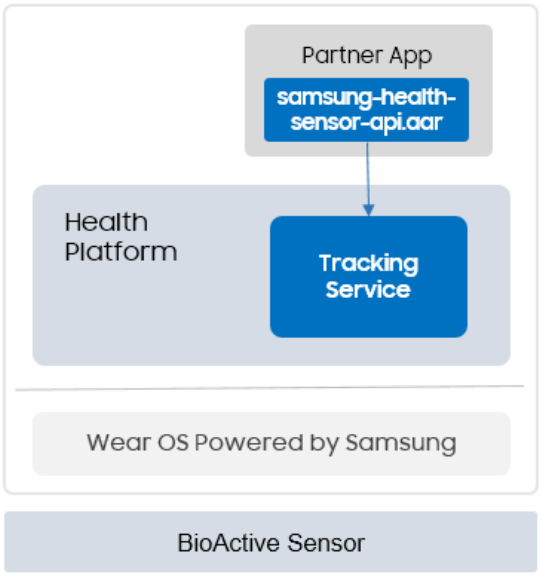
Programming Guide
- Samsung Health Sensor SDK

v1.3.0

The Health Platform is preloaded in the Galaxy Watch, which runs Wear OS powered by Samsung. Its version updates are available through Galaxy Store or Google Play.

The Samsung Health Sensor SDK provides tracking for the watch's raw or processed health sensor data through the Health Platform. Accelerometer, body composition, ECG, heart rate, PPG, skin temperature, SpO2, and sweat loss data can be measured with the SDK. The Galaxy Watch's BioActive sensor runs three powerful health sensors: optical heart rate, electrical heart signal, and bioelectrical impedance analysis sensors. An accessory and a temperature sensor are used to track health sensor data on the Galaxy Watch.

The Samsung Health Sensor SDK enables a Wear OS application to track health sensor data by importing the SDK's library (`samsung-health-sensor-api.aar`) and setting an event listener to receive data events. On-demand data tracking gives a one-time data measurement to the application. Continuous data tracking sends periodical events until the event listener is removed. The measuring frequency and number of retrieved health data in each event are different depending on the tracker type.



Galaxy Watch

The SDK's main features are:

Capabilities

The Samsung Health Sensor SDK provides a list of available tracker types for the watch.

Measuring Watch's Health Sensor Data

A partner application using the SDK can measure the health sensor data of the Galaxy Watch. The following tracker types are supported:

[Continuous Tracking]

The following tracker types can be measured continuously until the tracker type's event listener is unset. The continuous tracker types operate with low Galaxy Watch battery consumption.

- Accelerometer
- Heart rate including IBI (inter-beat interval)
- PPG Green, Red, and IR
- Skin temperature

[On-demand Tracking]

The following tracker types are on-demand tracker types. Use only one on-demand tracker type at a time and only when needed.

- Bioelectrical impedance analysis (BIA)
- Electrocardiogram (ECG)
- PPG Green, Red, and IR
- Skin temperature
- SpO2 (blood oxygen level)

[Other]

Measuring the user's sweat loss amount after running is available with the following tracker type.

- Sweat loss

Developer Mode

The Health Platform supports the Developer mode for application testing and debugging. See Health Platform's Developer Mode for more information.

Glossary

Glossary	Description
BIA	Bioelectrical impedance analysis
BioActive Sensor	A 3-in-1 sensor that uses a single chip embedded in the Galaxy Watch
ECG	Electrocardiogram
Health Platform	Preloaded in the Galaxy Watch running Wear OS powered by Samsung. It provides the Health Tracking Service to measure health sensor data.

Health Tracking Service	Helps to connect a Wear OS application and the Health Platform and to measure the watch's health sensor data.
IBI	Inter-beat interval
PPG	Photoplethysmogram
Samsung Health Sensor SDK	Enables an application to measure health sensor data on the Galaxy Watch running Wear OS powered by Samsung. Raw or processed data is retrieved by setting an event listener for each tracker type. A partner application imports the 'samsung-health-sensor-api.aar' SDK's library and can track the watch's health sensor data.

Health Sensor Data Specifications

> **Note**
>
> Data measured by the Samsung Health Sensor SDK is for fitness and wellness information only, not for the diagnosis or treatment of any medical condition.

Continuous Tracker Types

A Wear OS application can retrieve the following health sensor data continuously with a periodic event until the event is unset.

Tracker Type	Raw / Processed	Description
`ACCELEROMETER_CONTINUOUS`	Raw	Provides x, y, and z axis values measured with a 25 Hz frequency. Measured data is retrieved with `AccelerometerSet` data points in `TrackerEventListener.onDataReceived()`.
`HEART_RATE_CONTINUOUS`	Processed	Heart rate data including inter-beat interval (IBI) measured with a 1 Hz frequency. **[Watch display - on]** Measured data is retrieved with 1 `HeartRateSet` data point in `TrackerEventListener.onDataReceived()`. **[Watch display - off]** Measured data is retrieved with `HeartRateSet` data points in `TrackerEventListener.onDataReceived()`. ** IBI values for the complete tracking times are stored in the first data point. The other data points contain NULL.*
`PPG_CONTINUOUS`	Raw	Includes PPG green, IR and red data measured with a 25 Hz frequency. Measured data is retrieved with `PpgSet` data points in `TrackerEventListener.onDataReceived()`.
`SKIN_TEMPERATURE_CONTINUOUS`	Processed	Skin temperature and ambient temperature around the Galaxy Watch. This is not the same as body temperature. **[Watch display – on]** Measured data is retrieved with 1 `SkinTemperatureSet` data point in `TrackerEventListener.onDataReceived()`. **[Watch display – off]** Measured data is retrieved with `SkinTemperatureSet` data points in `TrackerEventListener.onDataReceived()`.

		** Tracking skin temperature is available with **Galaxy Watch5 series and later** models.* ** For Galaxy Watch5, the skin temperature is measured after updating watch software to Android 13 (API level 33) or higher.*

On-demand Tracker Types

See Using On-demand Tracker Type for more information on using an on-demand tracker type.

Tracker Type	Raw / Processed	Description
BIA_ON_DEMAND	Processed	Body composition data. Measured data is retrieved with 1 `BiaSet` data point in `TrackerEventListener.onDataReceived()`.
ECG_ON_DEMAND	Raw	ECG data measured with a 500 Hz frequency. Measured data is retrieved `EcgSet` data points in `TrackerEventListener.onDataReceived()`.
PPG_ON_DEMAND	Raw	Includes PPG green, IR, and red data measured with a 100 Hz frequency. Measured data is retrieved 1 `PpgSet` data point in `TrackerEventListener.onDataReceived()`.
SKIN_TEMPERATURE_ON_DEMAND	Processed	Skin temperature and ambient temperature around the Galaxy Watch. This is not the same as body temperature. Measured data is retrieved 1 `SkinTemperatureSet` data point in `TrackerEventListener.onDataReceived()`. ** Tracking skin temperature is available with **Galaxy Watch5 series and later** models.* ** For Galaxy Watch5, the skin temperature is measured after updating watch software to Android 13 (API level 33) or higher.*
SPO2_ON_DEMAND	Processed	Blood oxygen data. Measured data is retrieved 1 `Spo2Set` data point in `TrackerEventListener.onDataReceived()`. ** Tracking SpO2 is available with watch **Health Platform v1.3.0** or above.*

Other

Measuring the user's sweat loss amount after a running is available with the following tracker type.

Tracker Type	Raw / Processed	Description

SWEAT_LOSS	Processed	Sweat loss amount for a running exercise. 1 `SweatLossSet` data point is retrieved.

Using the On-demand Tracker Type

Please follow these guidelines when using an on-demand tracker type:

- Use the tracker in a foreground application, not in the background.
  Do not use more than one on-demand tracker at the same time.
- An on-demand tracker type is not intended for continuous measurement.
  Track on-demand type sensors in 30 seconds.
- During on-demand tracker type data measurement, tracking a continuous tracker type can give invalid values.

Development Environment

Target Device

The Samsung Health Sensor SDK supports the following watch devices that run on Wear OS Powered by Samsung.

- Galaxy Watch4 series and later models

SDK Content

Folder Structure	Content Description
Document	API Reference Programming Guide
Library	Library of the Samsung Health Sensor SDK. Import it in your application project. - `samsung-health-sensor-api.aar`
Sample code	Measure blood oxygen Measure blood oxygen and heart rate Transfer measured heart rate to a connected phone Measure skin temperature

Limitations

- The emulator is not supported.
- Data measured by the Samsung Health Sensor SDK is for fitness and wellness information only, not for the diagnosis or treatment of any medical condition.

Developer Support

If you experience trouble using the Samsung Health Sensor SDK, please submit a query at Developer Support with a bug report log and a detailed issue description.

You can get the bug report by following these steps:

1) Reproduce the trouble case on a watch.
2) Open the Galaxy Wearable application on the phone and select More (Ξ) > Settings > About Galaxy Wearable.
3) Touch [Galaxy Wearable] 10 times.
4) Select [Run Watch Dump] and wait.
5) Find and share My Files > Internal Storage > Download > Log > GearLog > bundled-bugreport-xx.zip.

> **Note**
>
> For Wear OS's Health Services APIs issues, use "create a new issue" at the Android Developer site.

Connect Galaxy Watch with Android Studio

To debug your application, configure a Galaxy Watch by following these steps.

> **Note**
>
> If you have a Galaxy Watch cradle, you only need to complete the “Turn on Developer mode and adjust its settings” section, place a watch on a watch cradle and connect a 5 pin USB cable to your PC.

Connect your watch to Wi-Fi

1. Go to **Settings > Connection > Wi-Fi** and make sure that Wi-Fi is enabled.
2. From the list of available Wi-Fi networks, choose and connect to the same one your PC is using.

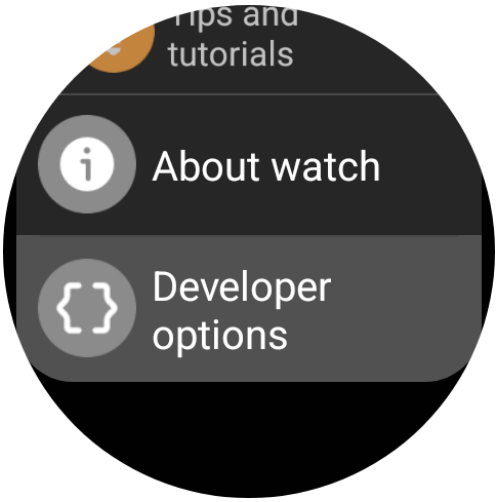
Turn on Developer mode and adjust its settings

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

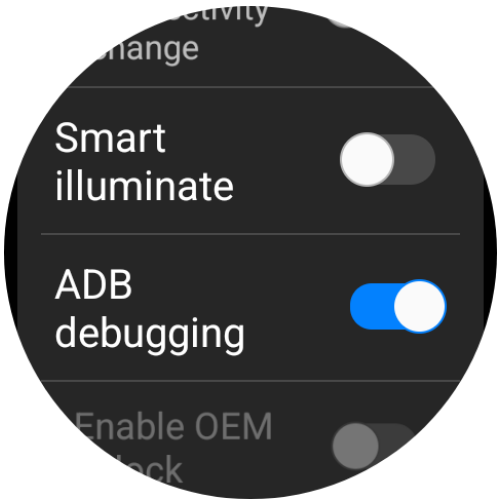
1. On your watch, go to **Settings > About watch > Software** and tap **Software version** 5 times.

2. Upon successful activation of the **Developer mode**, a toast message is displayed as on the image below.

3. Afterwards, **Developer options** is visible under **Settings**.



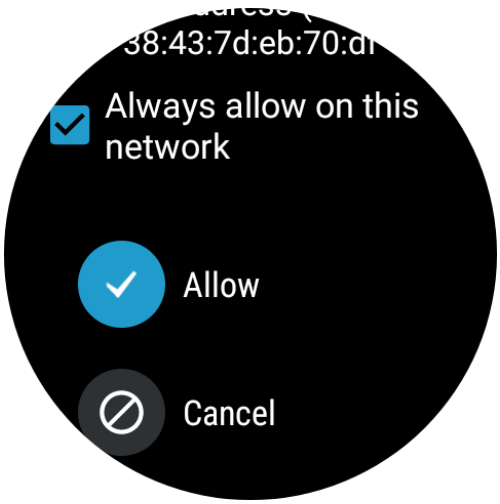
4. Tap **Developer options** and enable ADB debugging:



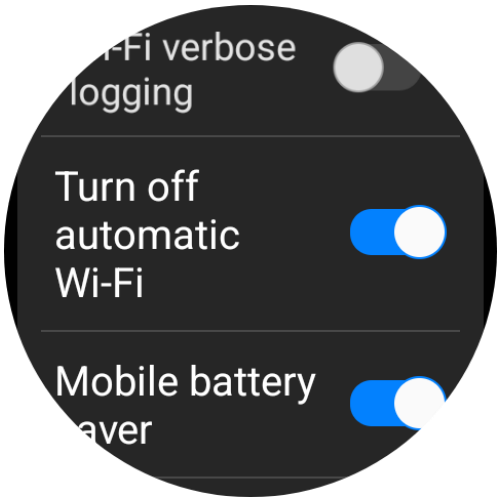
   - In **Developer options**, find **Wireless debugging**

- Turn on **Wireless debugging**

- When prompted to allow debugging over this Wi-Fi network – tap **Always allow on this network** and tap **Allow**



- Go back to **Developer options** and select **Turn off automatic Wi-Fi**



Note

There may be differences in the settings depending on your One UI version.

Connect your Galaxy Watch to Android Studio

1. Go to **Settings > Developer options > Wireless debugging** and choose **Pair new device**.

2. Take note of the **Wi-Fi pairing code**, **IP address & Port**.

3. In Android Studio, go to **Terminal** and type:

   `adb pair <IP address>:<port> <Wi-Fi pairing code>`

4. When prompted, tap **Always allow from this computer** to allow debugging.

5. After successfully pairing the watch, type:

   `adb connect <IP address of your watch>:<port>`

   Upon successful connection, you see the following message in Android Studio's Terminal:

   `connected to <IP address of your watch>`

The watch is now connected to the Android Studio's debugger.

Health Platform's Developer Mode

The Samsung Health Sensor SDK features are implemented in the watch Health Platform. They work properly after registering partner application information in a Samsung health system. The health team registers partner application information including an application package name and signature (sha256) after approving a partner request. A registered application signature in the Samsung health system should be from the application's release key. See more information on partner requests on the Developer site.

The Health Platform on a Galaxy Watch allows a Wear OS application using the Samsung Health Sensor SDK to track health sensor data only if the running application's package name and signature are both matched with the registered information. Otherwise, the SDK gives `SDK_POLICY ERROR`.

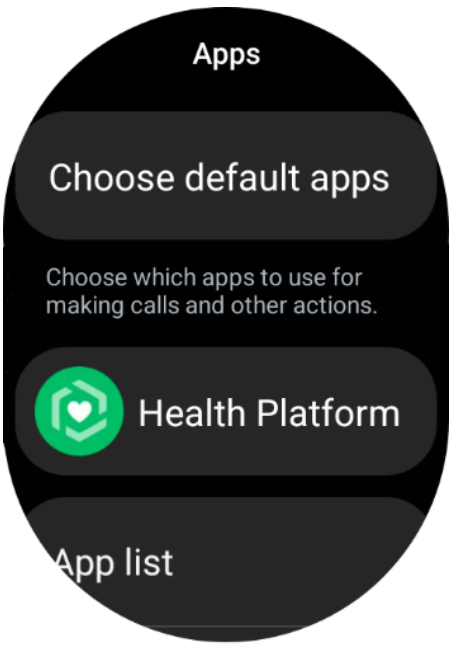
If you would like to test your application with a different application key, use the Health Platform's Developer mode.

> **Note**
>
> The Health Platform's Developer mode is ONLY for testing or debugging your application. It is NOT for application users. Please do not provide a developer mode guide for your application users.

Turn on Developer Mode

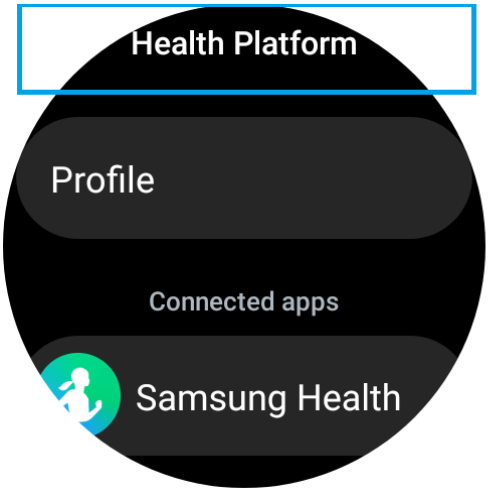
Developer mode can be enabled by following these steps:

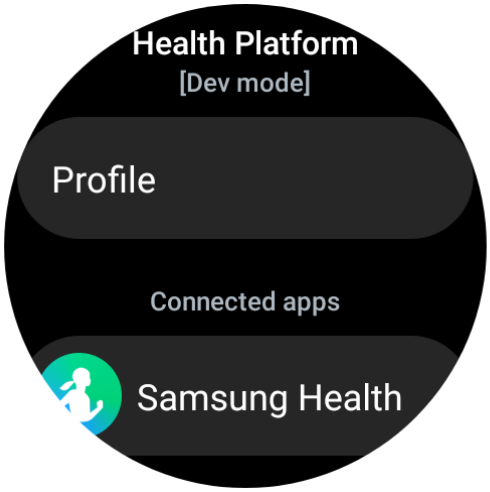
1. Pull up the display and select the **Settings** icon.
2. Select **Apps** and swipe down. Select **Health Platform**.



3. Tap the **Health Platform** title part quickly about 10 times.

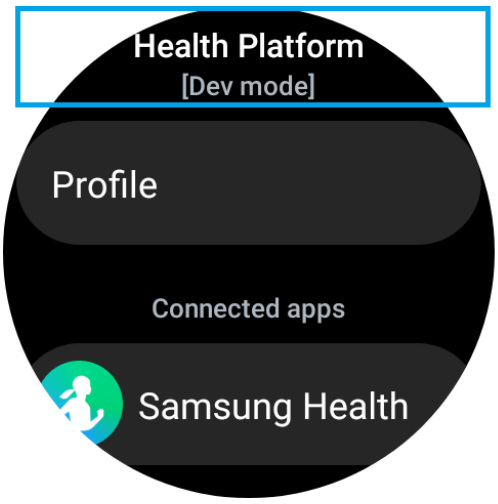
4. **[Dev mode]** displays after the Health Platform title.
   This means the Developer mode is enabled.

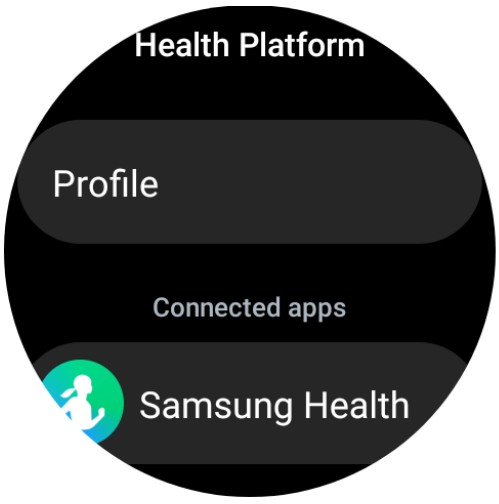




Turn off Developer Mode

Tapping the **Health Platform** title disables the Developer mode.





Tracking Service

This is a short tutorial on how to use the Samsung Health Sensor SDK. Follow each step and run your application with the Tracker Service feature.

Importing Samsung Health Sensor Library

Import the library into your application project's `libs` folder:

`samsung-health-sensor-api.aar`

app/build.gradle

Set Java version as 1.8 for a compile option, and add a dependency for the Samsung Health Sensor SDK library in `app/build.gradle`.

[app/build.gradle]

```
android {

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}

dependencies {

    implementation fileTree(dir: 'libs', include: '*.aar')
    // ...
}
```

Gradle Version

Use Gradle version 7.2 or lower.

Connecting with the Health Platform

Connect to the Health Platform with:

- `HealthTrackingService.connectService()`

[SubActivity.java]

```
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 SubActivity 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 tracker event listener with:

- `HealthTracker.TrackerEventListener`

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

- `HealthTrackerException.resolve()`

[SubActivity.java]

```
    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(SubActivity.this);
            }
        }

    };
```

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

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

Capability Check

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

[SubActivity.java]

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

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

        // Check Capability.
        List<HealthTrackerType> availableTrackers =
            healthTracking.getTrackingCapability().getSupportHealthTrackerTypes();
        if (!availableTrackers
             .contains(HealthTrackerType.SPO2_ON_DEMAND)) {
            // SpO2 is not supported on a watch.
        }
        // Tracking Data.
    }
// ...
```

Permission Request

The Samsung Health Sensor SDK requires permissions. Add permissions in your application's manifest.

[AndroidManifest.xml]

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

Check the granted permissions for BODY_SENSORS or ACTIVITY_RECOGNITION with:

- `ContextCompat.checkSelfPermission(@NonNull Context context, @NonNull String permission)`

[PermissionActivity.java]

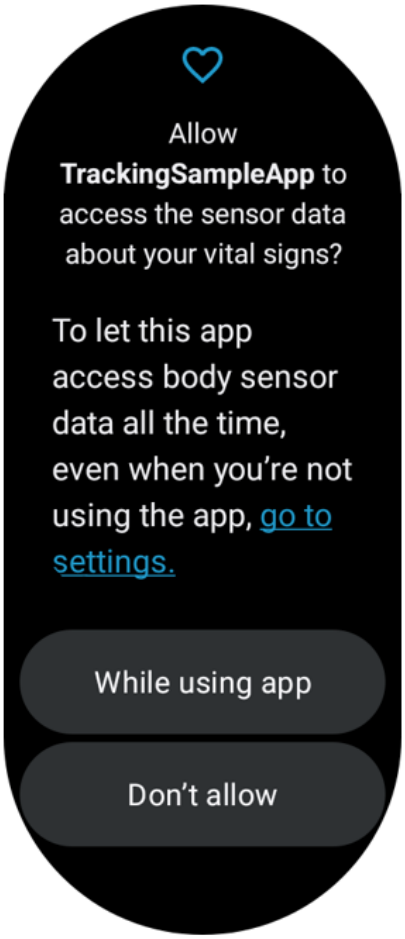
```
public static boolean checkPermission(@Nullable Context context,
                                      @NonNull String[] permissions) {
    for (String permission : permissions) {
        if (context == null || ActivityCompat.checkSelfPermission(context,
            permission) == PackageManager.PERMISSION_DENIED) {
            Log.i(TAG, "checkPermission : PERMISSION_DENIED : " + "permission");
            return false;
        } else {
            Log.i(TAG, "checkPermission : PERMISSION_GRANTED : " + "permission");
        }
    }
    return true;
}
```

If there is no permission, request the permission:

[PermissionActivity.java]

```
import androidx.core.app.ActivityCompat;

public class PermissionActivity extends FragmentActivity {
    private ArrayList<String> mPermissions;
    private static final String PERMISSION_KEY = "permissions";
    private static final int PERMISSION_REQ_TAG = 1;



    public void requestPermission(ArrayList<String> mPermissions) {
        // ...

        String[] permissions = new String[mPermissions.size()];
        permissions = mPermissions.toArray(permissions);
        ActivityCompat.requestPermissions(this, permissions, PERMISSION_REQ_TAG);

        // ...
    }

    @Override
    public void onRequestPermissionsResult(int requestCode,
                                           @NonNull String[] permissions,
                                           @NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        if (requestCode == PERMISSION_REQ_TAG) {
        // ...
        for (int p : grantResults) {
            if (p == PackageManager.PERMISSION_DENIED) {
                Log.i(TAG, "onRequestPermissionsResult : permission denied");
                finish();
                return;
            }
        }
        // ...
    }
}
```

The permission request displays a pop-up to get the user’s consent.



Tracking Data

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

```
protected final void trackPpgContinuous() {
    // Start PPG Continuous data tracking.
    HealthTracker ppgContinuousTracker = null;
    ppgContinuousTracker =
        healthTracking.getHealthTracker(HealthTrackerType.PPG_CONTINUOUS);
    ppgContinuousTracker.setEventListener(trackerListener);
}
```

An event for tracking data is retrieved in:

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

Receiving data occurs until the event listener is unset.

```
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 application's SDK policy was not downloaded yet.

> **Note**
>
> Although frequency of data is fixed (for example, for the accelerometer, it's 25 Hz), interval time between each `DataPoint`'s timestamp in `TrackerEventListener.onDataReceived()` may vary. The order of received data points is not changed, and handles the received data in order, regarding timestamp differences. See Data Specifications.

Flushing Data

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

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

For example, tracking continuous skin temperature normally gives 60 samples every 1 hour.

- If `Flush()` is called when 25 samples have been gathered, a `DataPoint` list in `onDataUpdated()` gives only 25 samples.
- After getting the flush result, `onDataUpdated()` again provides another 60 samples after a 1 hour period.

Stop Tracking

If your application doesn't need to track data anymore, unset the registered event listener with:

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

Tracking Sweat Loss

> **Note**
>
> Tracking sweat loss is available only for a running exercise.

The Samsung Health Sensor SDK provides tracking sweat loss after an exercise. Implementation of the Sweat loss tracker differs a little from other tracker types, as it requires one more step – feeding the tracker with additional data (the 'steps per minute' value) provided from another source such as Wear OS's Health Services library.

In addition, the tracker object requires one more element – `TrackerUserProfile` (which includes weight, height, age, and gender).

The following steps describe how to track sweat loss data.

Getting a HealthTracker Instance for Sweat Loss

After successfully connecting to the `HealthTrackingService`, a starting point is to get a `HealthTracker` instance.

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

The `TrackerUserProfiles` should be set with `TrackerUserProfile.Builder()`.

```
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 (from the example above) and the `ExerciseType.RUNNING`.

```
private HealthTracker sweatLossTracker = null;

// After 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.
        } catch (final UnsupportedOperationException e) {
            // Exception handling.
        }
    }

    // Override other functions of the listener

};
```

Set the event listener. We discuss the listener later.

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

Starting an Exercise with Health Services

During the exercise, you should keep feeding sweat loss tracker with `STEPS_PER_MINUTE`.

Such data is available through Health Services.

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

```
private ExerciseClient exerciseClient;

private void initExerciseClient() {
    if (exerciseClient == null) {
        exerciseClient =
            HealthServicesClientProvider.getClient(context).getExerciseClient();
        exerciseClient.setUpdateCallback(exerciseUpdateCallback);
    }
}
```

Configure a running exercise so that it can track `STEPS_PER_MINUTE` and `DISTANCE`.

`STEPS_PER_MINUTE` is later sent to health tracker using `HealthTracker.setExerciseData()` and `DISTANCE` is helpful to check an error case when a sweat loss's status is not zero.

```
final ExerciseConfig.Builder exerciseConfigBuilder =
        ExerciseConfig.builder(ExerciseType.RUNNING)
                .setDataTypes(new HashSet<>(Arrays.asList(DataType.DISTANCE,
STEPS_PER_MINUTE)))
                .setIsAutoPauseAndResumeEnabled(false)
                .setIsGpsEnabled(false);
```

The data from Health Services comes in `ExerciseUpdateCallback`.

```
private final ExerciseUpdateCallback exerciseUpdateCallback =
    new ExerciseUpdateCallback() {

        @Override
        public void onExerciseUpdateReceived(ExerciseUpdate update) {
            // ‘Steps per minute’ values are represented as Sample Data Points
            extractStepsPerMinute(update.getLatestMetrics().getSampleDataPoints());
            // Use update.getLatestMetrics().getIntervalDataPoints() to get access
            // to such data as ‘distance’
        }

        // Override other functions of the callback

    };
```

Setting the Exercise State to START

Start the exercise using ExerciseClient.startExerciseAsync().

After configuring and starting the exercise in Health Services, it is time to set the exercise state to START on the sweat loss tracker.

```
try {
    // Set the exercise state to START.
    sweatLossTracker.setExerciseState(ExerciseState.START);
} catch (final IllegalStateException e) {
    // Exception handling.
}
```

Extracting Exercise Data and Sending them to the Tracker

The example below shows how to extract 'steps per minute' from onExerciseUpdateReceived(ExerciseUpdate exerciseUpdate) (they reside in exerciseUpdate.getLatestMetrics().getSampleDataPoints()).

```
private void extractStepsPerMinute(List<SampleDataPoint<?>> dataPoints) {
    final int listSize = dataPoints.size();

    final float[] spmData = new float[listSize];
    final long[] timeStamp = new long[listSize];

    for (SampleDataPoint<?> element : dataPoints) {
        if (element.getDataType().equals(STEPS_PER_MINUTE)) {

            spmData[dataPoints.indexOf(element)] = (Long)
                Objects.requireNonNull(element).getValue();
            timeStamp[dataPoints.indexOf(element)] =
getUtcTimeFromSystemElapsedTime(Objects.requireNonNull(element).getTimeDurationFromBoot().toM
illis());
        }
    }
    if (listSize > 0) {
        sweatLossTracker.setExerciseData(STEPS_PER_MINUTE, stepsPerMinuteValues,
stepsPerMinuteTimeStamps);
    }
}
```

We send the exercise data - STEPS_PER_MINUTE to the Samsung Health Sensor SDK's sweat loss tracker using HealthTracker.setExerciseData().

Monitoring the distance and duration of the exercise is helpful. If the exercise's duration is less than 5 minutes or the distance is less than 2 kilometers, the Samsung Health Sensor SDK gives an error in a sweat loss' STATUS.

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

Setting the Exercise State with STOP

If you decide to finish the exercise, set the exercise state to `STOP` on the tracker.

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

You should also stop the exercise in the context of Health Services using `ExerciseClient.endExerciseAsync()`.

Checking the Tracker Listener

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

```
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

After getting the final sweat loss tracker, unset the registered tracker's event listener.

```
sweatLossTracker.unsetEventListener();
```

For more information about sweat loss tracking, refer to the Sweat Loss Monitor blog.

User Guide for Measurement

BIA

Measure BIA with Galaxy Watch with the 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. Move your watch higher on your wrist.

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

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

4. Raise your arms so your armpits are open.

5. Keep touching the keys until the measurement is completed.

ECG

Measuring ECG data is available by following these steps:

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 trouble measuring SpO2, please check the watch’s location on your wrist as well as your pose. And please be careful not to move.

Sample Code Description

You can find sample applications using the Samsung Health Sensor SDK on the Samsung Developer site.

Measure Blood Oxygen

The application enables measuring SpO2 on the user's wrist. It's prepared for developers to show how to obtain SpO2 data using Samsung Health Sensor.

See Measure Blood Oxygen Level on Galaxy Watch for a detailed description.

Heart Rate Tracker with the Off-body Sensor

The application measures heart rate while the watch is worn. It shows developers how they can utilize Samsung Health Sensor library with the off-body sensor. Continuous heart rate measurement can be performed as long as the watch is worn.

See Heart Rate Tracking with the Off-body Sensor for more details.

Measure Blood Oxygen and Heart Rate

The application enables measuring SpO2 and heart rate on the user's wrist. It's prepared for developers to show how to obtain data from several sensors at the same time using Samsung Health Sensor.

See Measure Blood Oxygen Level and Heart Rate on Galaxy Watch for a detailed description.

ECG Monitor

The application measures ECG over time and guides how to use on-demand trackers. It's prepared for developers to show how to overcome most common challenges with ECG measurement and how to check the validity of received data.

See ECG Monitor for more details.

Measure Skin Temperature

The main idea is to utilize the Samsung Health Sensor SDK in order to perform wrist skin and ambient temperature measurement. As a result of this code lab, the user can launch the Wear OS health application on the Samsung Galaxy Watches.

See Measure skin temperature on Galaxy Watch with Samsung Health Sensor SDK for a detailed description.

Sweat Loss Monitor

Sweat Loss Monitor is a watch application that tracks sweat loss after an exercise using various health sensor data based on a rich set of sensors that are supported by the Galaxy Watch.

See **Sweat Loss Monitor** for implementation details and description.