HEISHA SDK Integration Guide

V 1.14

【Revision Record】

Time	Version	Remarks
2021-09-15	1.0	First version
2021-12-02	1.14	Compatible with SDK version 1.1.4 and below

Overview

Introduction

HEISHA Android SDK encapsulates the underlying protocol of HEISHA's DNEST device and provides an upper layer API for developers to call to obtain device operation information, control the device to complete related functions, set device operation parameters and other functions, shortening the time for developers to interface, and read the protocol documentation. This integration guide records common questions and API calls in the integrating process.

SDK Architecture

Manager

Includes various managers for SDK protocols.

Product

Define the base classes of HEISHA products and their derived classes.

Component

Defines the various component and modules of the product.

Importing SDK

Important Tips

It is recommended to test our integration demo first and read the integration code in our demo carefully, it will help you a lot in the interfacing process. The integration demo code has been uploaded to the GitHub remote repository at:

https://github.com/Ezio-Wen/HeiSha_SDK_Demo.git

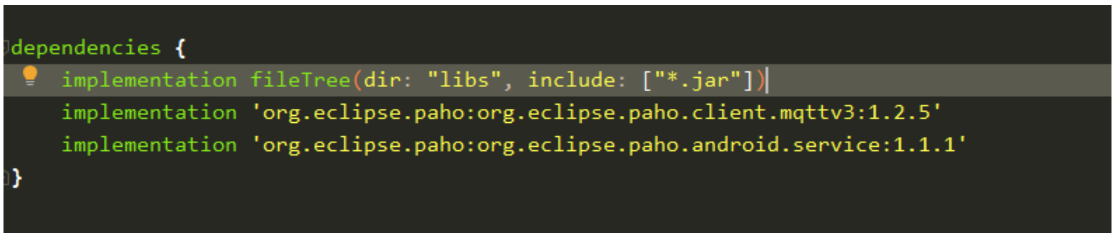
You can also download the demo's integrated installation package to test:

http://118.190.91.165:8080/download/HeiSha/Apps/

Steps to import the HEISHA SDK into Android studio are as follows

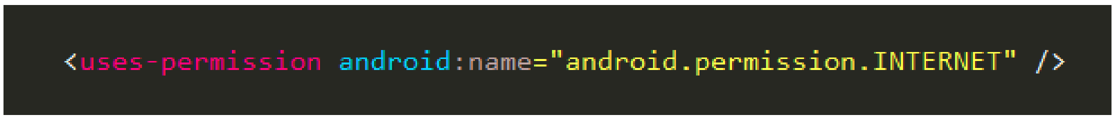
1) To get the SDK, open http://118.190.91.165:8080/download/HeiSha/SDK/ , You can first read the HEISHA SDK Version Description.pdf in the directory of this web page, which documents the version records and updates of the HEISHA SDK for each version. Then click to download the corresponding version of the SDK jar package from the web directory.

2) Copy the HEISHA SDK jar package to the lib directory of the project in Android studio, create a new Android project, then open the build.gradle file in the app directory of the project, add the following dependencies, and then synchronize the project.



```
dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.2.5'
    implementation 'org.eclipse.paho:org.eclipse.paho.android.service:1.1.1'
}
```

3）Open the AndroidManifest.xml of the project and add the network access permission.



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

Calling API

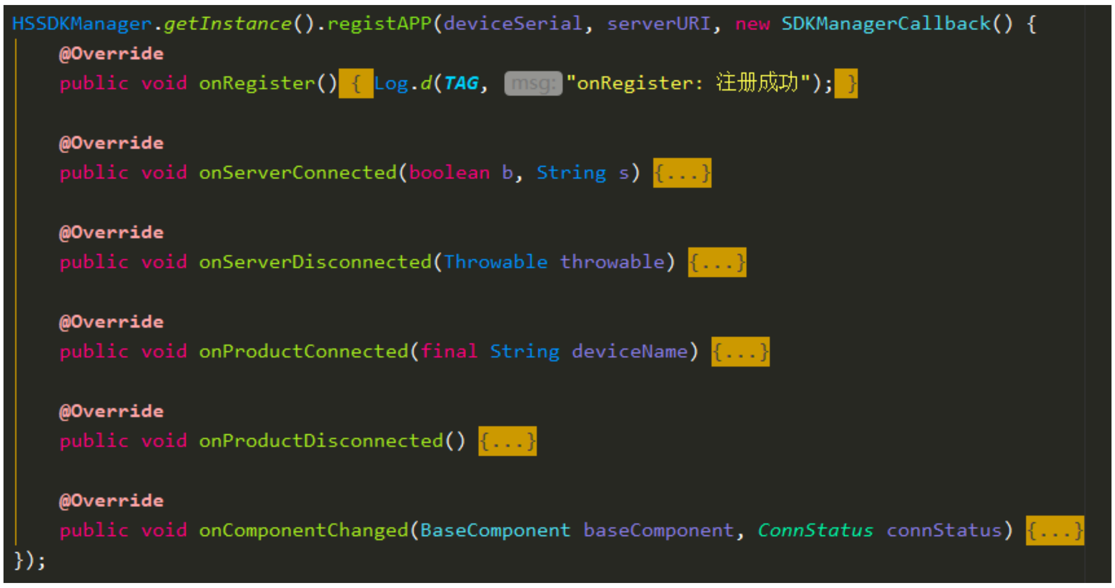
Tips:

The API documentation in JavaDoc form has been uploaded to the server and is available at：

http://118.190.91.165:8080/download/HeiSha/API/HeiSha_SDK/

Connecting devices and acquiring device components

1) Method of calling the register app, passing in the serial number of the connected HEISHA device, the URL of the connected MQTT server and an instance of the SDK Manager callback interface.



```
HSSDKManager.getInstance().registAPP(deviceSerial, serverURI, new SDKManagerCallback() {
    @Override
    public void onRegister() { Log.d(TAG, msg: "onRegister: 注册成功"); }

    @Override
    public void onServerConnected(boolean b, String s) {...}

    @Override
    public void onServerDisconnected(Throwable throwable) {...}

    @Override
    public void onProductConnected(final String deviceName) {...}

    @Override
    public void onProductDisconnected() {...}

    @Override
    public void onComponentChanged(BaseComponent baseComponent, ConnStatus connStatus) {...}
});
```

After successful registration, the SDK will automatically connect to the server and wait for the device to be logged on line, in the above several callback functions,

onRegister() Callback after successful registration；

onServerConnected() Callback when the SDK successfully connects to the server;

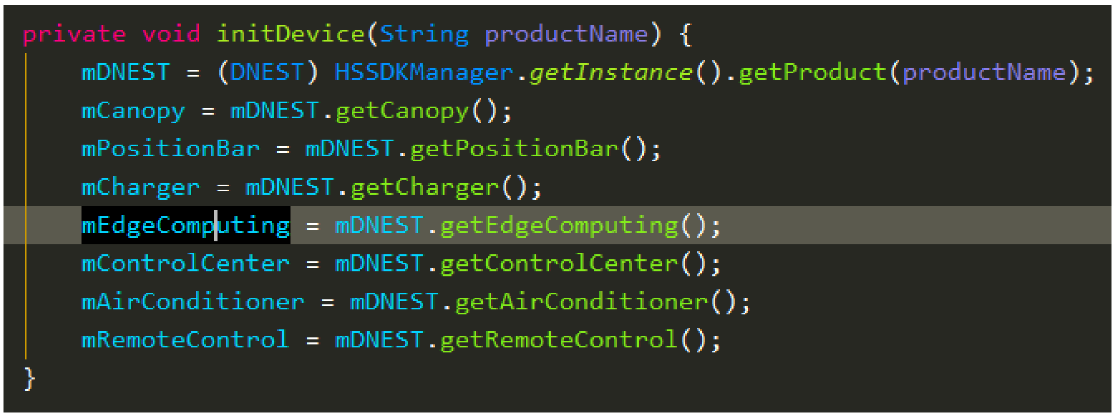
onServerDisconnected() Callback when the SDK is disconnected from the server;

onProductConnected() Callback when the device is logged in and online, where we generally initialize the device instance and component instance and register the event callback listeners for each component;

onProductDisconnected() Callback when the SDK is disconnected from the device;

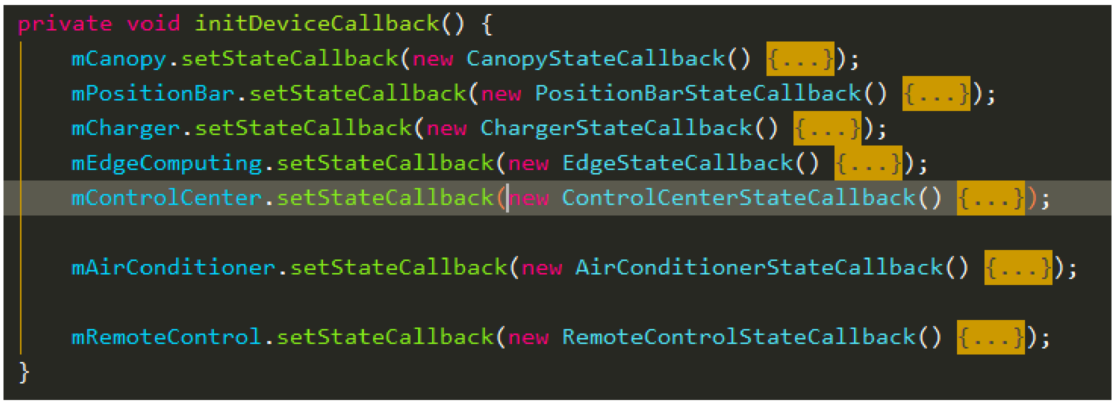
onComponentChanged() will call back when the connection status of the main master module of the device changes.

After the device is online, instantiate the device and the components of the device,



```java
private void initDevice(String productName) {
    mDNEST = (DNEST) HSSDKManager.getInstance().getProduct(productName);
    mCanopy = mDNEST.getCanopy();
    mPositionBar = mDNEST.getPositionBar();
    mCharger = mDNEST.getCharger();
    mEdgeComputing = mDNEST.getEdgeComputing();
    mControlCenter = mDNEST.getControlCenter();
    mAirConditioner = mDNEST.getAirConditioner();
    mRemoteControl = mDNEST.getRemoteControl();
}
```

Register event callback listeners for each component



```
private void initDeviceCallback() {
    mCanopy.setStateCallback(new CanopyStateCallback() {...});
    mPositionBar.setStateCallback(new PositionBarStateCallback() {...});
    mCharger.setStateCallback(new ChargerStateCallback() {...});
    mEdgeComputing.setStateCallback(new EdgeStateCallback() {...});
    mControlCenter.setStateCallback(new ControlCenterStateCallback() {...});

    mAirConditioner.setStateCallback(new AirConditionerStateCallback() {...});

    mRemoteControl.setStateCallback(new RemoteControlStateCallback() {...});
}
```

Except for the ControlCenter module, all other components have only two listening events, namely

onUpdate(), which is called when a property is reported in the component module, and onOperateResult(), which is called when an operation is performed on the component's function and the result is received.

The ControlCenter module's onGetConfigVersionInfo(), called when the version of the device configurable parameters is obtained.

onGetConfig(), Called back when equipment-specific configurable parameters are obtained;

onSetConfig(); After setting the parameters, the device will provide feedback on the result of the setting and the reason for the error.

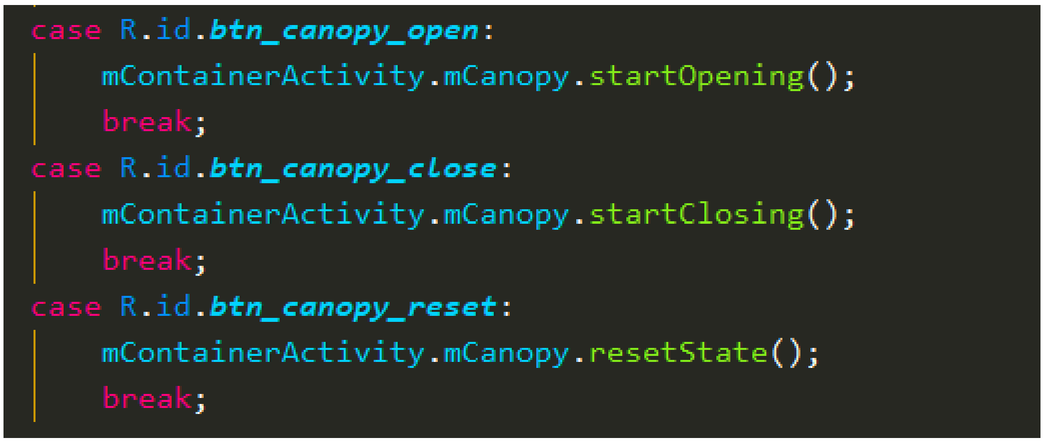
OnThingPost(); Called back when various events are reported by the device, the event levels are:

THING_LEVEL_INFORMATION、THING_LEVEL_WARNING and THING_LEVEL_ERROR.

Operate the functions provided by each component

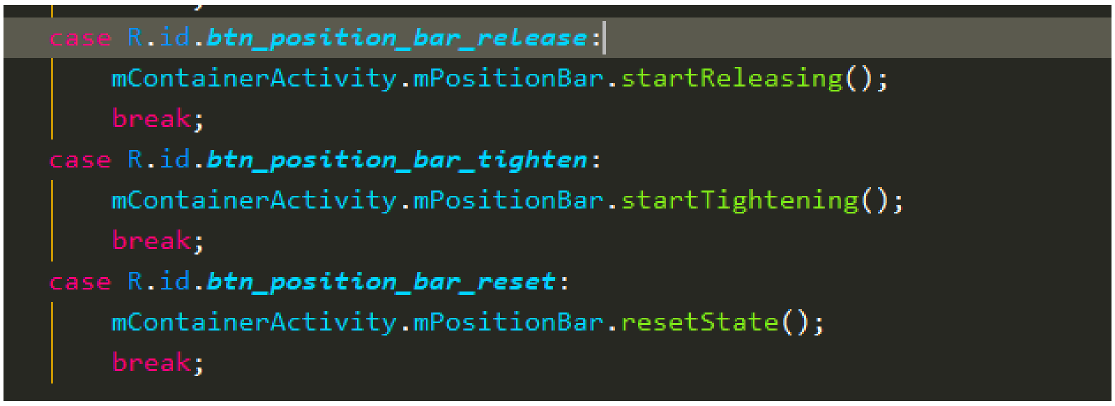
module

1) Open、close and reset Canopy:



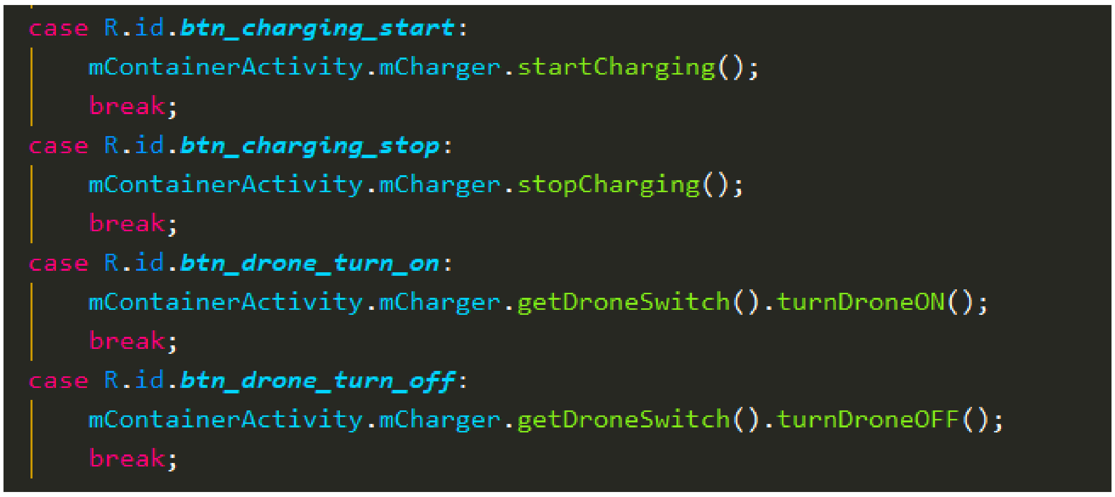
```
case R.id.btn_canopy_open:
    mContainerActivity.mCanopy.startOpening();
    break;
case R.id.btn_canopy_close:
    mContainerActivity.mCanopy.startClosing();
    break;
case R.id.btn_canopy_reset:
    mContainerActivity.mCanopy.resetState();
    break;
```

2) Unlock、lock and reset Charge Bars:



```
case R.id.btn_position_bar_release:
    mContainerActivity.mPositionBar.startReleasing();
    break;
case R.id.btn_position_bar_tighten:
    mContainerActivity.mPositionBar.startTightening();
    break;
case R.id.btn_position_bar_reset:
    mContainerActivity.mPositionBar.resetState();
    break;
```

3) Start charging, stop charging, turn on and off the drone:



```
case R.id.btn_charging_start:
    mContainerActivity.mCharger.startCharging();
    break;
case R.id.btn_charging_stop:
    mContainerActivity.mCharger.stopCharging();
    break;
case R.id.btn_drone_turn_on:
    mContainerActivity.mCharger.getDroneSwitch().turnDroneON();
    break;
case R.id.btn_drone_turn_off:
    mContainerActivity.mCharger.getDroneSwitch().turnDroneOFF();
    break;
```

4) Power on/off edge computing modules such as Android and NVIDIA:

```
case R.id.btn_android_turn_on:
    mContainerActivity.mEdgeComputing.androidTurnOn();
    break;
case R.id.btn_android_turn_off:
    mContainerActivity.mEdgeComputing.androidTurnOff();
    break;
case R.id.btn_nvidia_turn_on:
    mContainerActivity.mEdgeComputing.NVIDIATurnOn();
    break;
case R.id.btn_nvidia_turn_off:
    mContainerActivity.mEdgeComputing.NVIDIATurnOff();
    break;
```

5) Switching on/off the drone remote control and plugging the USB cable of the remote control.

```
case R.id.btn_rc_turn_on:
    mContainerActivity.mRemoteControl.RCTurnOn();
    break;

case R.id.btn_rc_turn_off:
    mContainerActivity.mRemoteControl.RCTurnOff();
    break;

case R.id.btn_rc_usb_plug_in:
    mContainerActivity.mRemoteControl.RCUSBPlugIn();
    break;

case R.id.btn_rc_usb_pull_out:
    mContainerActivity.mRemoteControl.RCUSBPullOut();
    break;
```

6) After the operation is completed, you can get the result of the operation in the callback function ControlCenter.onOperateResult().

7) Get the current value of the configurable parameters, just pass in the specific parameter index, and after the completion of the acquisition, you can get the value of the acquired parameters in the callback function ControlCenter.onGetConfig():

```
mContainerActivity.mControlCenter.getConfigParameter(ConfigParameter.SERVICE_PARAM_POST_RATE_CANOPY);
```

8) Set the configurable parameter value, pass in the specific parameter index and the parameter value to be set, and after the setting is completed, the setting result can be obtained in the callback function ControlCenter.onSetConfig():

```
private void setParam(ConfigParameter parameter, int value) {
    if (mContainerActivity.isServerConnected && mContainerActivity.isDeviceConnected) {
        mContainerActivity.mControlCenter.setConfigParameter(parameter, value);
    }
}
```

FQA

1) Q：How to call API to lock and unlock the charge bars?

A: Just use the obtained PositionBar component instance to call the lock and unlock methods.

```
mPositionBar.startTightening();

mPositionBar.startReleasing();
```

2) Q： How to parse the error status of the charge bars limiter sensors obtained by calling PositionBar.getBarLimitSwitchFaultStateSet()?

A： Calling getBarLimitSwitchFaultStateSet() method will return a byte type value, the 8bit of this value from low to high code sensor S1 to S8 fault state respectively, 0 means normal, 1 means fault.

3) Q： How to parse the error status of the driving motor of the charge bars obtained by calling mPositionBar.getMotorFaultStateSet()?

A： Calling getMotorFaultStateSet() method will return a byte type value. The low 4bit of this value from low to high indicates the motor Motor1 to Motor4 fault state respectively, 0 means normal, 1 means fault.