Classified:	*Confidential*	*Version:*	*V1.0.6*
Range:	*Internal*	*Status:*	*Released*

NexRing JS SDK APIGuide

File No:	*LT_SW_2023042701*	*Category:*	*RD*
Prepared:	*Chenzq*	*Date:*	*2023-4-27*
Reviewed:	*Yangxd*	*Date:*	*2023-4-27*
Approved:	*Tangzp*	*Date:*	*2023-4-27*

■ Linktop Communication Technology Co., Ltd.

History

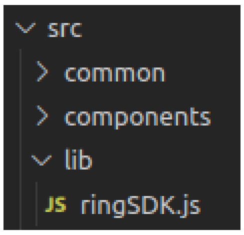
Version	*Release date*	*Description*
V1.0.0	2023.04.27	*First version*
V1.0.1	2023.04.27	*Update sleep algorithm*
V1.0.2	2023.04.27	*Add OEM certification and modify compatibility issues with HRV and power devices*
V1.0.3	2023.07.07	*Add OTA function*
V1.0.4	2023.07.19	*Modify sleep algorithm implementation*
V1.0.5	2023.08.10	*Compatible with historical data acquisition of old and new firmware, with step counting algorithm type added for step counting*
V1.0.6	2023.08.04	*Increase exercise mode, increase calorie count, increase hrv output in manual mode, and increase raw waveform data output in manual mode*

Contents

1.Integration Instructions

1.1Project Import

（1）Import the ringSDK.js files into the LIB directory under the source directory of the development project.

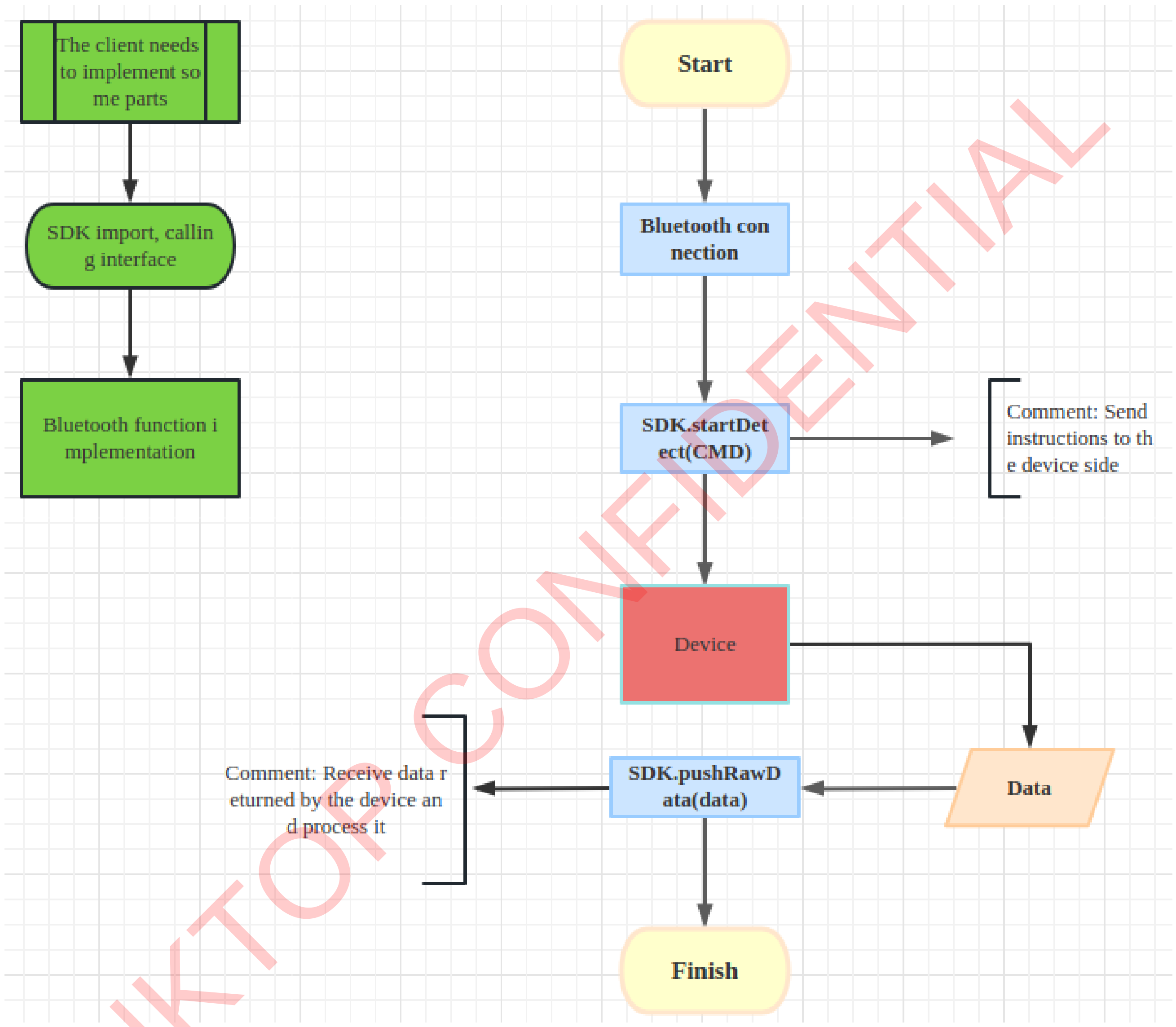


2. APIDescription

*2.1*FunctionDescription

The SDK establishes a connection with the device through BLE. You can easily communicate with the device through the SDK and extract data services such as electrocardiogram, blood oxygen, body temperature, etc. provided by the device. You can use this SDK to develop React Native applications and WeChat mini programs based on JavaScript. This SDK only provides data output such as electrocardiogram, blood oxygen, body temperature, etc. for user development and testing purposes.

2.2Flow Chart


The client needs to implement some parts
SDK import, calling interface
Bluetooth function implementation
Start
Bluetooth connection
SDK.startDetect(CMD)
Comment: Send instructions to the device side
Device
Data
SDK.pushRawData(data)
Comment: Receive data returned by the device and process it
Finish

2.3Interface

1． Initialization

Initialize when need to access:

ImportSDKfrom"../lib/ringSDK"

Bluetooth connection parameters：

export const FILTER_UUID="FEF5";

export const UUID_SERVICE="00001822-0000-1000-8000-00805F9B34FB";

export const UUID_SERVICE_IOS="1822";

export const WRITE_UUID="000066FE-0000-1000-8000-00805F9B34FB";

export const NOTIFY_UUID="000066FE-0000-1000-8000-00805F9B34FB";

FILTER_UUID:UUID used for filtering during Bluetooth scanning

UUID_SERVICE:After successful Bluetooth connection, the Android end service UUID

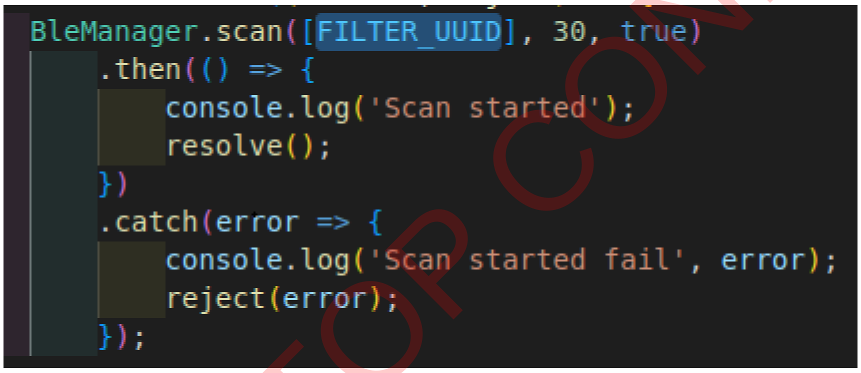
UUID_SERVICE_IOS:IOS end service UUID after successful Bluetooth connection

WRITE_UUID：UUID for writing data

NOTIFY_UUID: UUID for receiving data

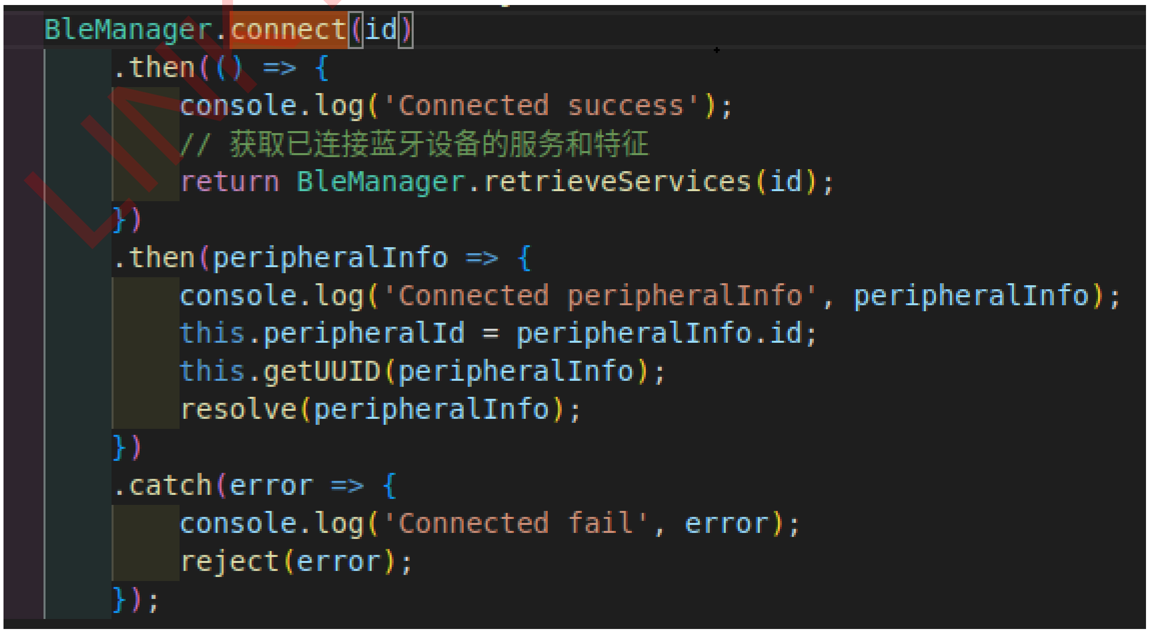
Bluetooth implementation part:

Bluetooth scanning

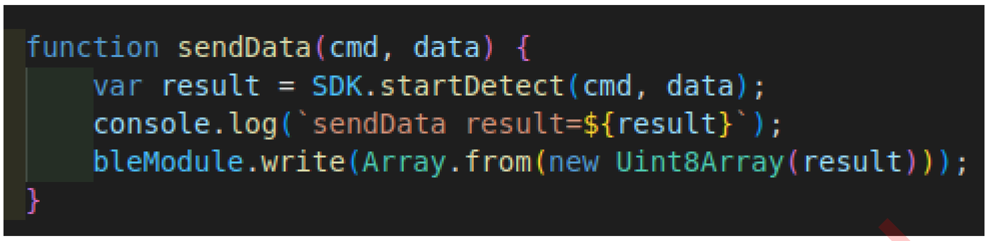


```
BleManager.scan([FILTER_UUID], 30, true)
    .then(() => {
        console.log('Scan started');
        resolve();
    })
    .catch(error => {
        console.log('Scan started fail', error);
        reject(error);
    });
```

Bluetooth connection

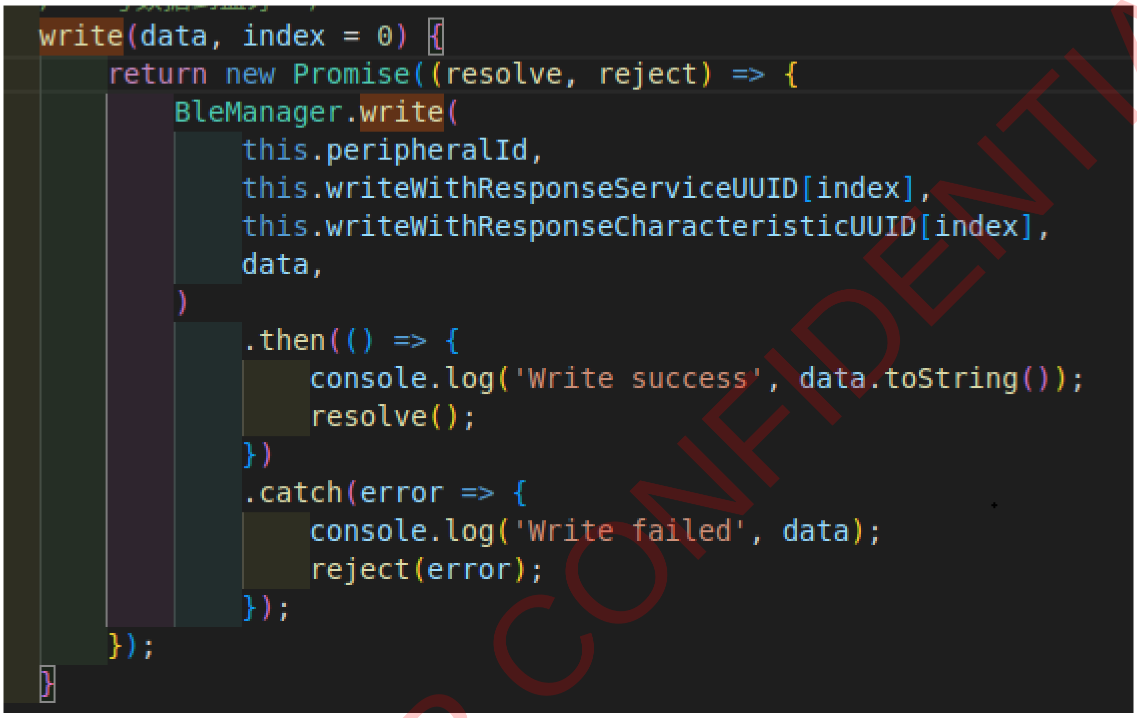


```
BleManager.connect(id)
    .then(() => {
        console.log('Connected success');
        // 获取已连接蓝牙设备的服务和特征
        return BleManager.retrieveServices(id);
    })
    .then(peripheralInfo => {
        console.log('Connected peripheralInfo', peripheralInfo);
        this.peripheralId = peripheralInfo.id;
        this.getUUID(peripheralInfo);
        resolve(peripheralInfo);
    })
    .catch(error => {
        console.log('Connected fail', error);
        reject(error);
    });
```



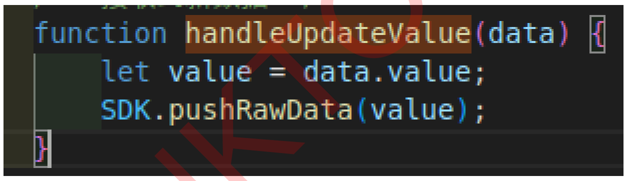
```
function sendData(cmd, data) {
    var result = SDK.startDetect(cmd, data);
    console.log(`sendData result=${result}`);
    bleModule.write(Array.from(new Uint8Array(result)));
}
```

Bluetooth sending data:



```
write(data, index = 0) {
    return new Promise((resolve, reject) => {
        BleManager.write(
            this.peripheralId,
            this.writeWithResponseServiceUUID[index],
            this.writeWithResponseCharacteristicUUID[index],
            data,
        )
            .then(() => {
                console.log('Write success', data.toString());
                resolve();
            })
            .catch(error => {
                console.log('Write failed', data);
                reject(error);
            });
    });
}
```

Bluetooth receiving data:



```
function handleUpdateValue(data) {
    let value = data.value;
    SDK.pushRawData(value);
}
```

2． Start Interface

startDetect(type);

Call startDetect to start the test interface after Bluetooth connection, parameter type =>

const step = "step";

const timeSyn = "timeSyn";

const openSingleHealth = "openSingleHealth";

const closeSingleHealth = "closeSingleHealth";

const openHealth = "openHealth";

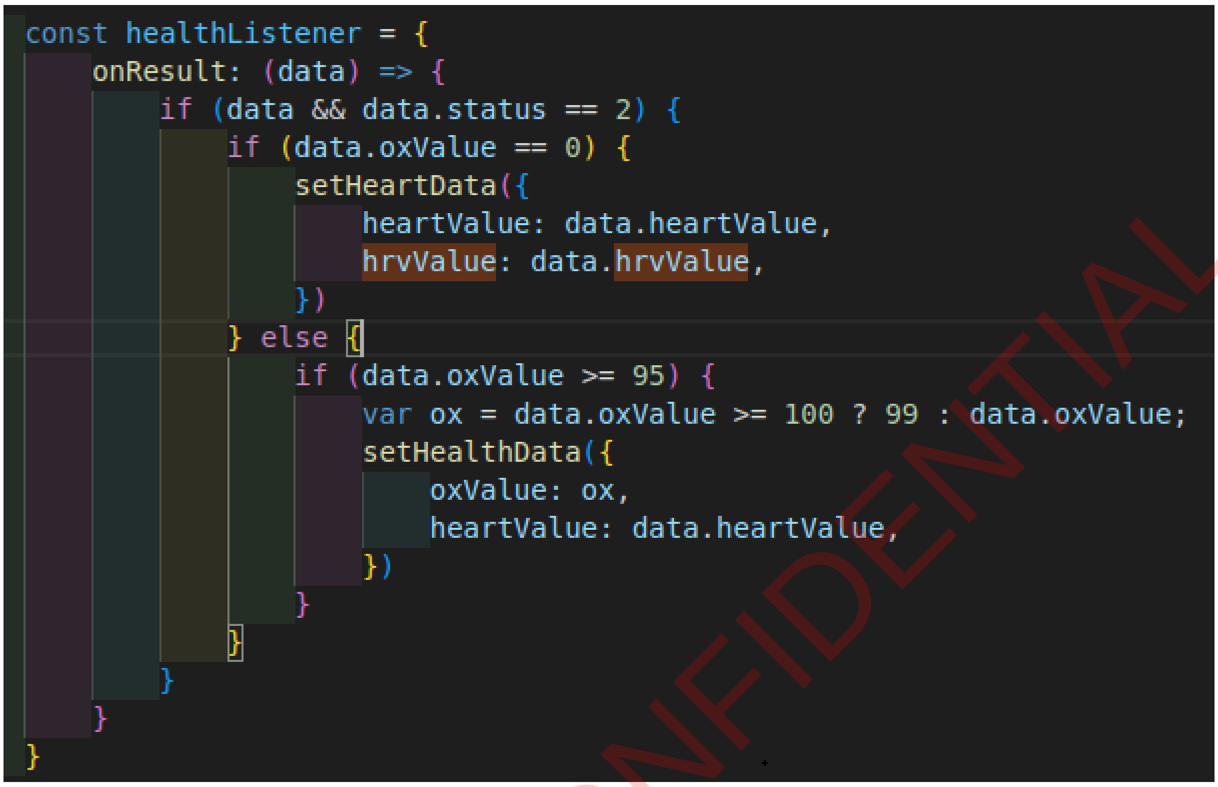
```
const closeHealth = "closeHealth";
const temperature = "temperature";
const shutDown = "shutDown";
const restart = "restart";
const restoreFactorySettings = "restoreFactorySettings";
const historicalNum = "historicalNum";
const historicalData = "historicalData";
const cleanHistoricalData = "cleanHistoricalData";
const deviceInfo1 = "deviceInfo1";
const deviceInfo2 = "deviceInfo2";
const batteryDataAndState = "batteryDataAndState"
const deviceBind = "deviceBind"
const deviceUnBind = "deviceUnBind"
```

Example: startDetect (step) enables step counting measurement， It is recommended to call startDetect (timeSyn) synchronization time every time the ring is connected

2． Callback Interface

1) <u>SDK.registerHealthListener(healthListener);</u>

Registration of heart rate and blood oxygen callback function



```
const healthListener = {
    onResult: (data) => {
        if (data && data.status == 2) {
            if (data.oxValue == 0) {
                setHeartData({
                    heartValue: data.heartValue,
                    hrvValue: data.hrvValue,
                })
            } else {
                if (data.oxValue >= 95) {
                    var ox = data.oxValue >= 100 ? 99 : data.oxValue;
                    setHealthData({
                        oxValue: ox,
                        heartValue: data.heartValue,
                    })
                }
            }
        }
    }
}
```

Callbackfunction

onResult:(data)

data.status =>0: Measurement not started; 1: During measurement; 2: Measurement data is valid

data.heartValue=>Heart rate data

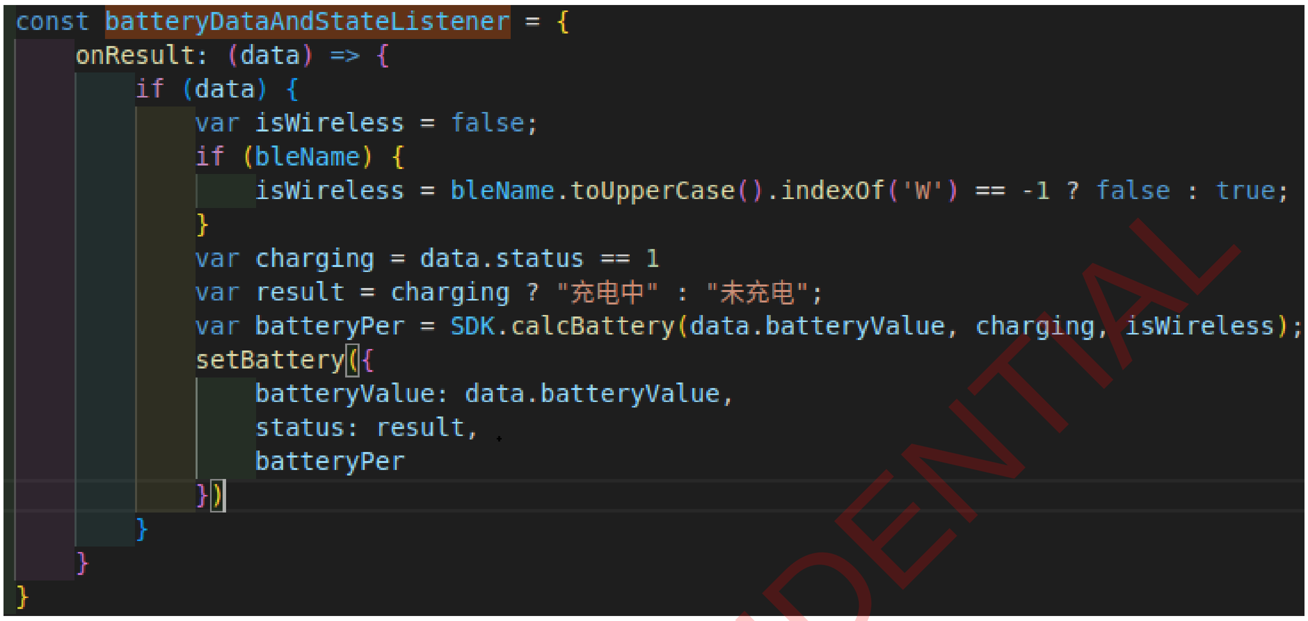
data.oxValue=>Blood oxygen data

data.hrvValue=>hrv data

.

2) SDK.registerHealthListener(batteryDataAndStateListener);

Registration of power callback function



```
const batteryDataAndStateListener = {
    onResult: (data) => {
        if (data) {
            var isWireless = false;
            if (bleName) {
                isWireless = bleName.toUpperCase().indexOf('W') == -1 ? false : true;
            }
            var charging = data.status == 1
            var result = charging ? "充电中" : "未充电";
            var batteryPer = SDK.calcBattery(data.batteryValue, charging, isWireless);
            setBattery({
                batteryValue: data.batteryValue,
                status: result,
                batteryPer
            })
        }
    }
}
```

onResult:(data)

isWireless:Determine whether it is wireless charging or wired charging by determining whether the Bluetooth name contains W

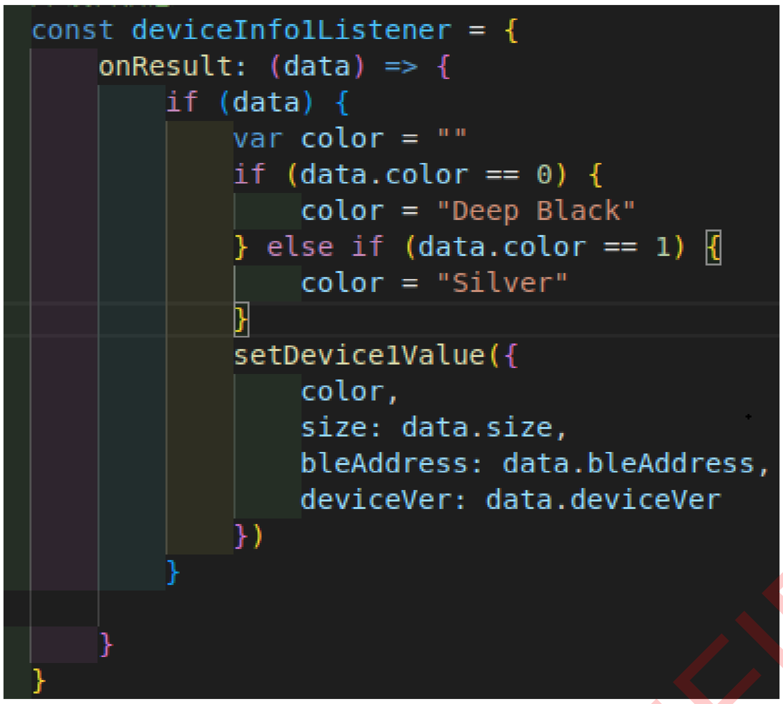
data.status=>0: Not charged; 1: Charging

data.batteryValue=>battery voltage

batteryPer=>Battery percentage (SDK. calcBattery calculates battery percentage)

3) SDK.registerDeviceInfo1Listener(deviceInfo1Listener)

Device Information 1 Callback Function Registration



```
const deviceInfo1Listener = {
    onResult: (data) => {
        if (data) {
            var color = ""
            if (data.color == 0) {
                color = "Deep Black"
            } else if (data.color == 1) {
                color = "Silver"
            }
            setDevice1Value({
                color,
                size: data.size,
                bleAddress: data.bleAddress,
                deviceVer: data.deviceVer
            })
        }

    }
}
```

onResult:(data)

data.color=>Ring color 0: Deep Black; 1:Silver

data.size=>Ring size

data.bleAddress=>Bluetooth MAC address of the ring

data.deviceVer=>The device version number of the ring

data. switchOem =>0: OEM certification switch off 1: OEM certification switch on

data.chargingMode=>0: Magnetic charging with battery charging compartment 1: Wireless charging 2: NFC wireless charging 3: Magnetic charging without charging stand 4: USB cable magnetic charging

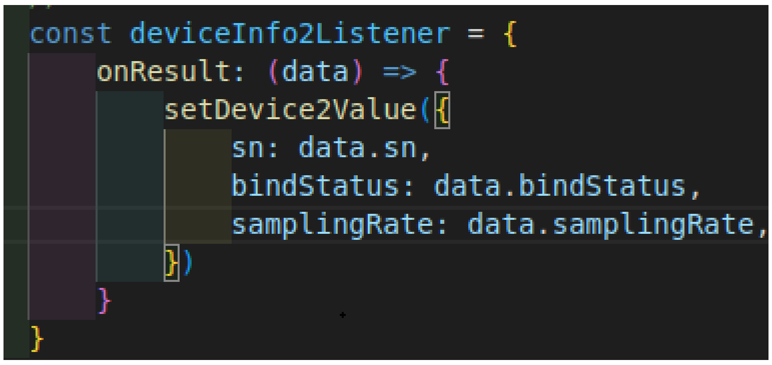
data.mainChipModel=> Main chip model

data.productIteration=> Product iteration

data.hasSportsMode=> Does it support sports mode

4) SDK.registerDeviceInfo2Listener(deviceInfo2Listener);

Device Information 2 Callback Function Registration



```
const deviceInfo2Listener = {
    onResult: (data) => {
        setDevice2Value({
            sn: data.sn,
            bindStatus: data.bindStatus,
            samplingRate: data.samplingRate,
        })
    }
}
```

onResult:(data)

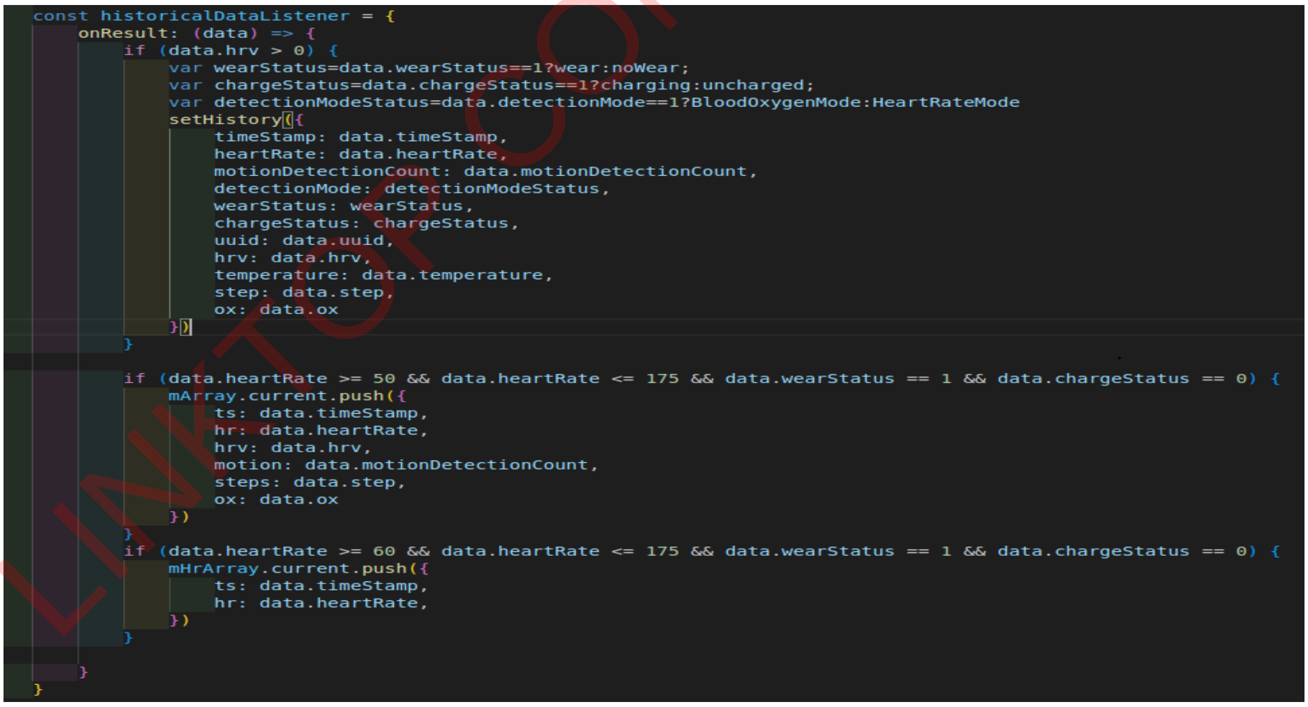
data.sn=> Device SN number

data.bindStatus=> Binding State

data,samplingRate=> Blood oxygen IR waveform data sampling rate

5) <u>SDK.registerHistoricalDataListener(historicalDataListener)</u>

Registration of Historical Data Callback Function



```
const historicalDataListener = {
    onResult: (data) => {
        if (data.hrv > 0) {
            var wearStatus=data.wearStatus==1?wear:noWear;
            var chargeStatus=data.chargeStatus==1?charging:uncharged;
            var detectionModeStatus=data.detectionMode==1?BloodOxygenMode:HeartRateMode
            setHistory({
                timeStamp: data.timeStamp,
                heartRate: data.heartRate,
                motionDetectionCount: data.motionDetectionCount,
                detectionMode: detectionModeStatus,
                wearStatus: wearStatus,
                chargeStatus: chargeStatus,
                uuid: data.uuid,
                hrv: data.hrv,
                temperature: data.temperature,
                step: data.step,
                ox: data.ox
            })
        }

        if (data.heartRate >= 50 && data.heartRate <= 175 && data.wearStatus == 1 && data.chargeStatus == 0) {
            mArray.current.push({
                ts: data.timeStamp,
                hr: data.heartRate,
                hrv: data.hrv,
                motion: data.motionDetectionCount,
                steps: data.step,
                ox: data.ox
            })
        }
        if (data.heartRate >= 60 && data.heartRate <= 175 && data.wearStatus == 1 && data.chargeStatus == 0) {
            mHrArray.current.push({
                ts: data.timeStamp,
                hr: data.heartRate,
            })
        }

    }
}
```

onResult:(data)

data.hrv=>hrv data

data.timeStamp=>time stamp

data.heartRate=>heart rate

data.motionDetectionCount=>Motion detection count data.detectionMode=>Measurement mode 0: Heart rate measurement mode; 1: Blood oxygen measurement mode

data.wearStatus=>Wearing status 0: Not worn; 1: Wearing

data.chargeStatus=>Charging state 0: not charged; 1: Charging

data.uuid=>Uuid value, accumulated from 1

data.temperature=>finger temperature

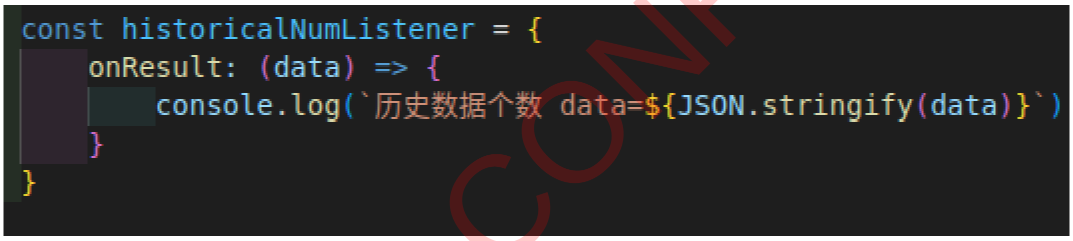
data.step=>step number

data.ox=>Blood oxygen

It is recommended to read the historical records, save the data in the database, and then clear the historical records

6) <u>SDK.registerHistoricalNumListener(historicalNumListener);</u>

Registration of callback function for the number of historical data



```
const historicalNumListener = {
    onResult: (data) => {
        console.log(`历史数据个数 data=${JSON.stringify(data)}`)
    }
}
```

onResult:(data)

data.num=>Number of historical data

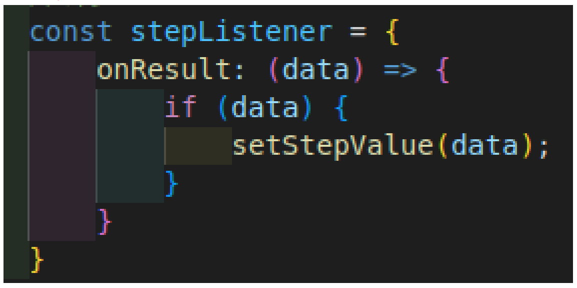
data.minUUID=>Minimum uuid

data.maxUUID=>Maximum uuid

Please note that the number of historical data must be requested before requesting historical data

7) <u>SDK.registerStepListener(stepListener);</u>

Step callback function registration



```
const stepListener = {
    onResult: (data) => {
        if (data) {
            setStepValue(data);
        }
    }
}
```

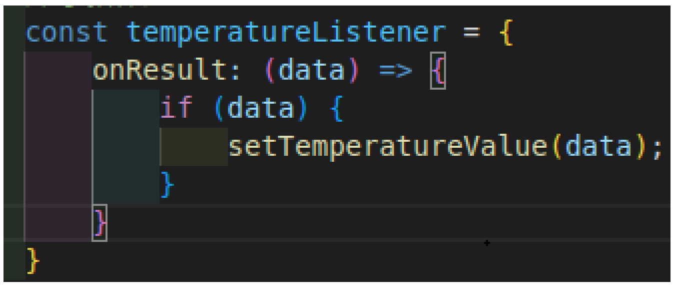
onResult:(data)

data.stepCount=> step number

data.StepAlgorithm=> The step count algorithm used returns LIS2DS12: The original step count data of the device is used. If no data is returned, the APP's step count algorithm is used

8) SDK.registerTemperatureListener(temperatureListener)

Finger temperature callback function registration



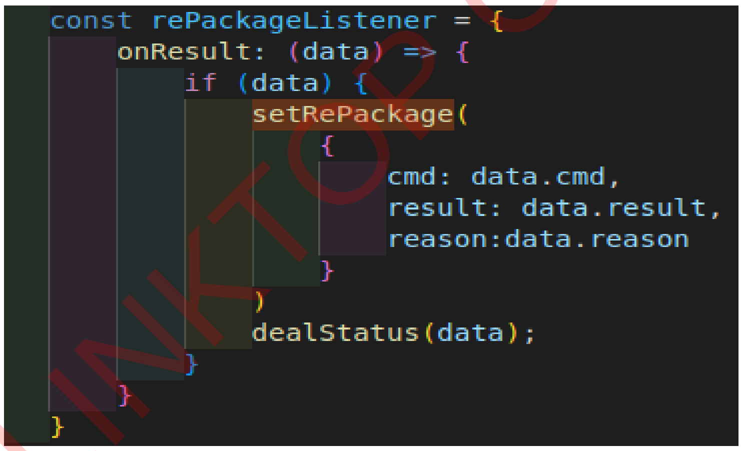
```
const temperatureListener = {
    onResult: (data) => {
        if (data) {
            setTemperatureValue(data);
        }
    }
}
```

onResult:(data)

data=>finger temperature

9) SDK.registerRePackageListener(rePackageListener);

Package callback function registration



```
const rePackageListener = {
    onResult: (data) => {
        if (data) {
            setRePackage(
                {
                    cmd: data.cmd,
                    result: data.result,
                    reason:data.reason
                }
            )
            dealStatus(data);
        }
    }
}
```

onResult:(data)

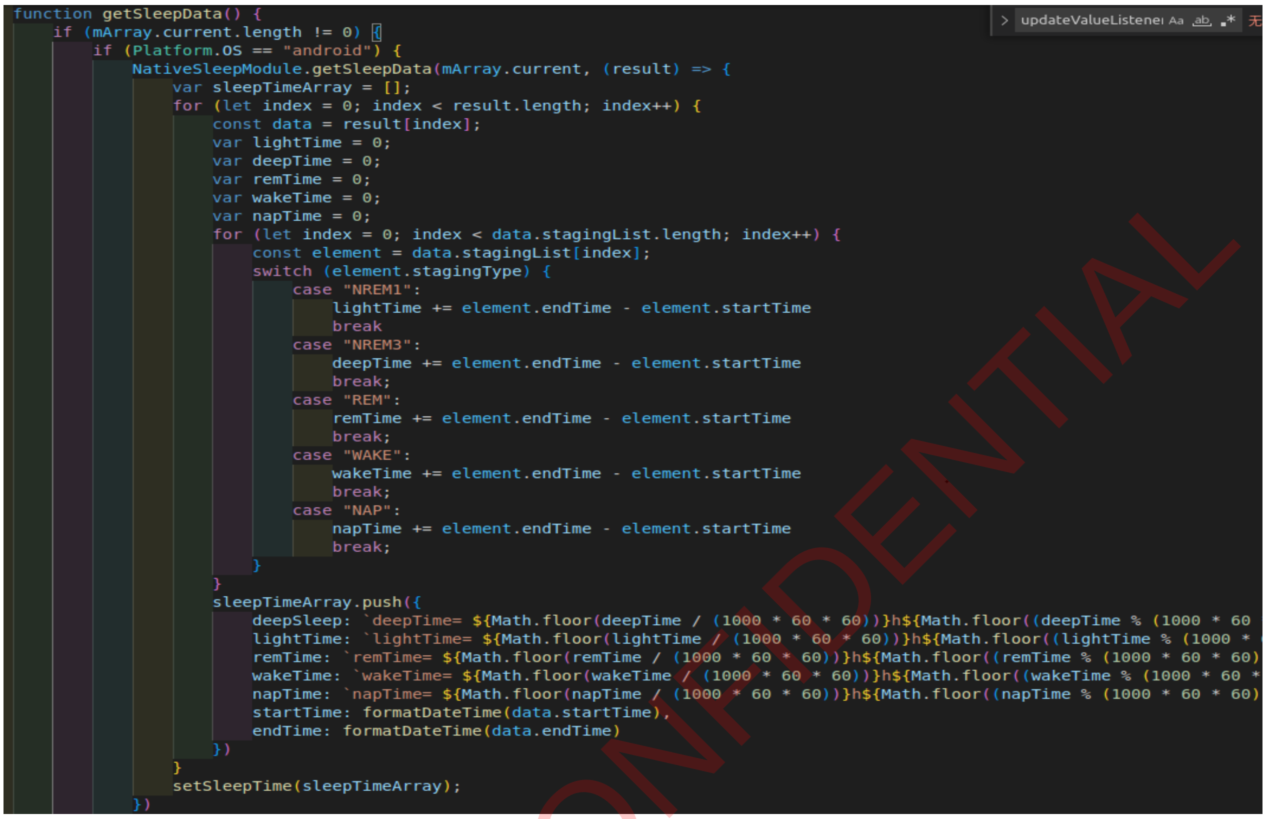
data.cmd=>Instructions issued

data.result=>Return the result of issuing the command

data.reason=> Return to the reason for the failure to issue the command

10) NativeSleepModule.getSleepData(Array,function)

Sleep data algorithm（android）



```
function getSleepData() {
    if (mArray.current.length != 0) {
        if (Platform.OS == "android") {
            NativeSleepModule.getSleepData(mArray.current, (result) => {
                var sleepTimeArray = [];
                for (let index = 0; index < result.length; index++) {
                    const data = result[index];
                    var lightTime = 0;
                    var deepTime = 0;
                    var remTime = 0;
                    var wakeTime = 0;
                    var napTime = 0;
                    for (let index = 0; index < data.stagingList.length; index++) {
                        const element = data.stagingList[index];
                        switch (element.stagingType) {
                            case "NREM1":
                                lightTime += element.endTime - element.startTime
                                break
                            case "NREM3":
                                deepTime += element.endTime - element.startTime
                                break;
                            case "REM":
                                remTime += element.endTime - element.startTime
                                break;
                            case "WAKE":
                                wakeTime += element.endTime - element.startTime
                                break;
                            case "NAP":
                                napTime += element.endTime - element.startTime
                                break;
                        }
                    }
                    sleepTimeArray.push({
                        deepSleep: `deepTime= ${Math.floor(deepTime / (1000 * 60 * 60))}h${Math.floor((deepTime % (1000 * 60
                        lightTime: `lightTime= ${Math.floor(lightTime / (1000 * 60 * 60))}h${Math.floor((lightTime % (1000 *
                        remTime: `remTime= ${Math.floor(remTime / (1000 * 60 * 60))}h${Math.floor((remTime % (1000 * 60 * 60)
                        wakeTime: `wakeTime= ${Math.floor(wakeTime / (1000 * 60 * 60))}h${Math.floor((wakeTime % (1000 * 60 *
                        napTime: `napTime= ${Math.floor(napTime / (1000 * 60 * 60))}h${Math.floor((napTime % (1000 * 60 * 60)
                        startTime: formatDateTime(data.startTime),
                        endTime: formatDateTime(data.endTime)
                    })
                }
                setSleepTime(sleepTimeArray);
            })
```

Parameters:

Array: Historical data array

Return value:

DeepSleep: deep sleep value

LightTime: Light sleep value

RemTime: Rapid eye movement sleep value

WakeTime:Awakening time value

NapTime:Sporadic naps

SleepTimePeriod. startTime: sleep start time

SleepTimePeriod. endTime: Sleep end time

DeepList: Array of deep sleep time periods

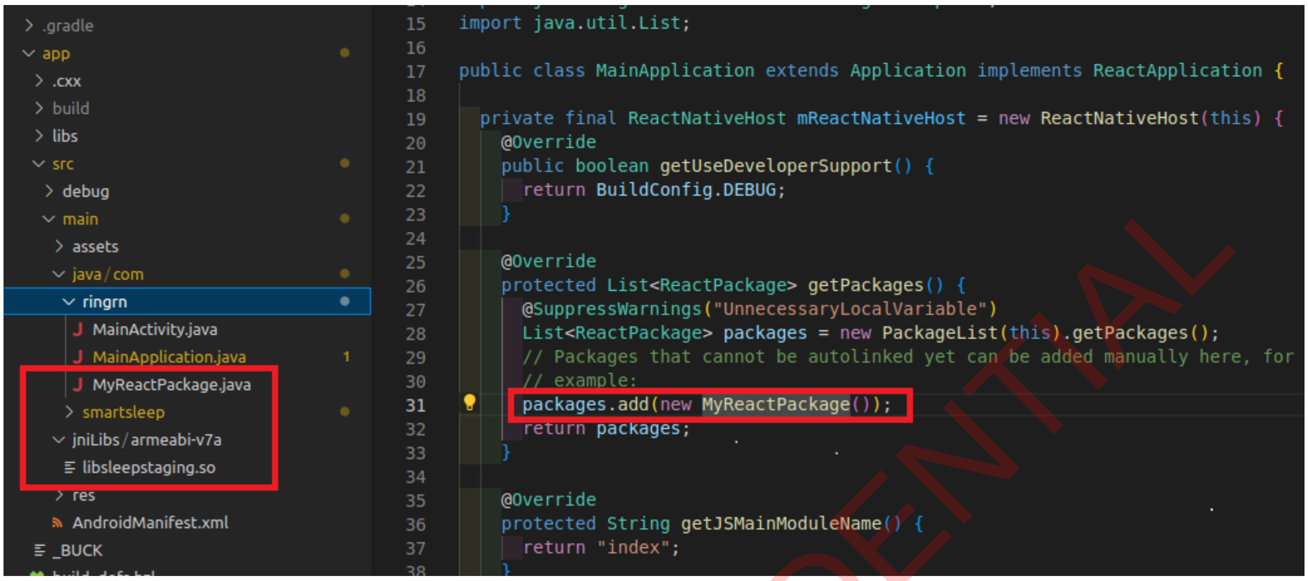
LightList: array of mild sleep time periods

RemList: Array of fast eye movement sleep time periods

WakeList:Awakening time period array

NapList: Sporadic naps period array

Code to be added to the Android directory

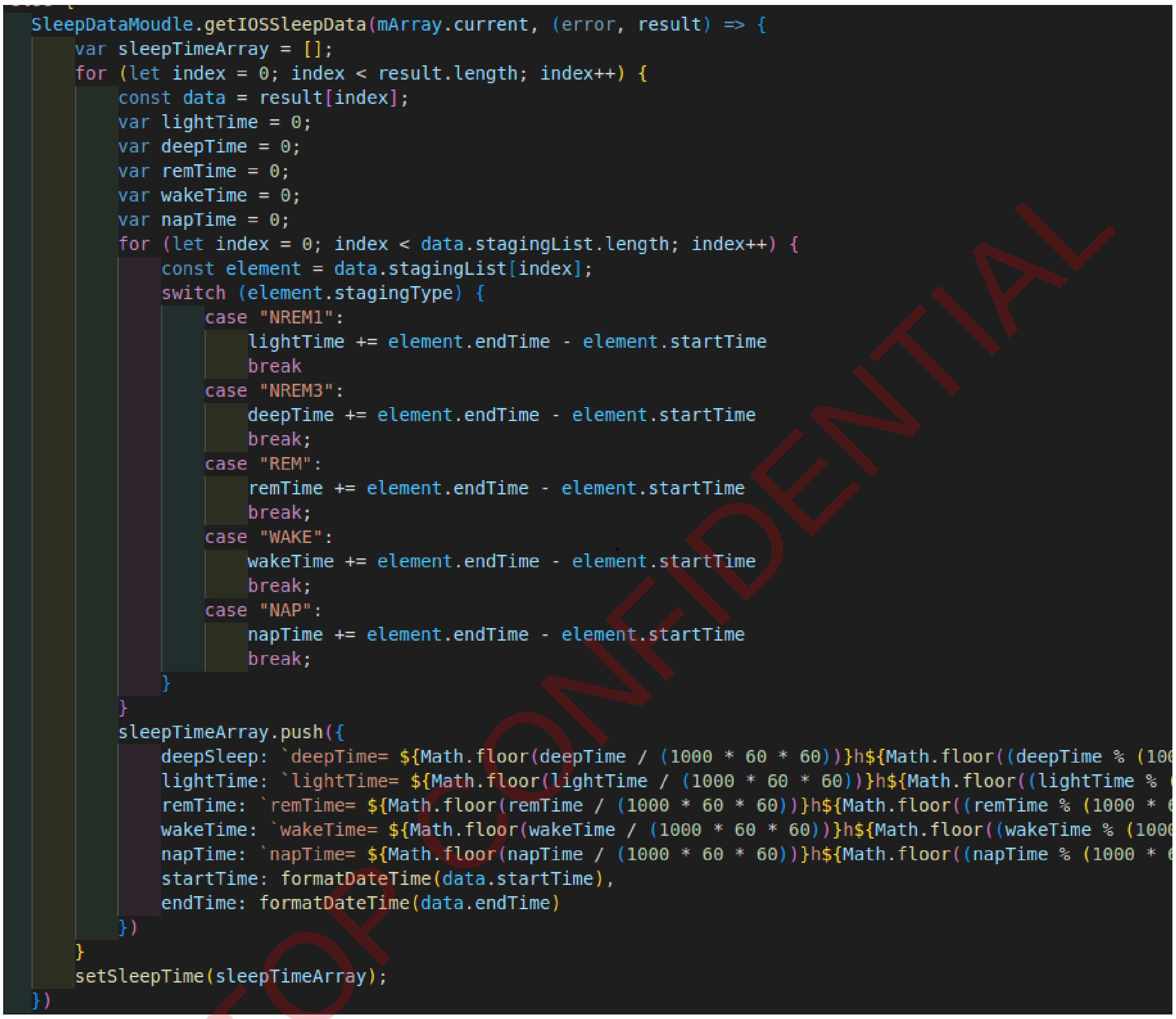


```
> .gradle
v app
  > .cxx
  > build
  > libs
  v src
    > debug
    v main
      > assets
      v java / com
        v ringrn
          J MainActivity.java
          J MainApplication.java        1
          J MyReactPackage.java
          > smartsleep
      v jniLibs / armeabi-v7a
          libsleepstaging.so
      > res
      AndroidManifest.xml
  _BUCK
```

```java
import java.util.List;

public class MainApplication extends Application implements ReactApplication {

  private final ReactNativeHost mReactNativeHost = new ReactNativeHost(this) {
    @Override
    public boolean getUseDeveloperSupport() {
      return BuildConfig.DEBUG;
    }

    @Override
    protected List<ReactPackage> getPackages() {
      @SuppressWarnings("UnnecessaryLocalVariable")
      List<ReactPackage> packages = new PackageList(this).getPackages();
      // Packages that cannot be autolinked yet can be added manually here, for
      // example:
      packages.add(new MyReactPackage());
      return packages;
    }

    @Override
    protected String getJSMainModuleName() {
      return "index";
```

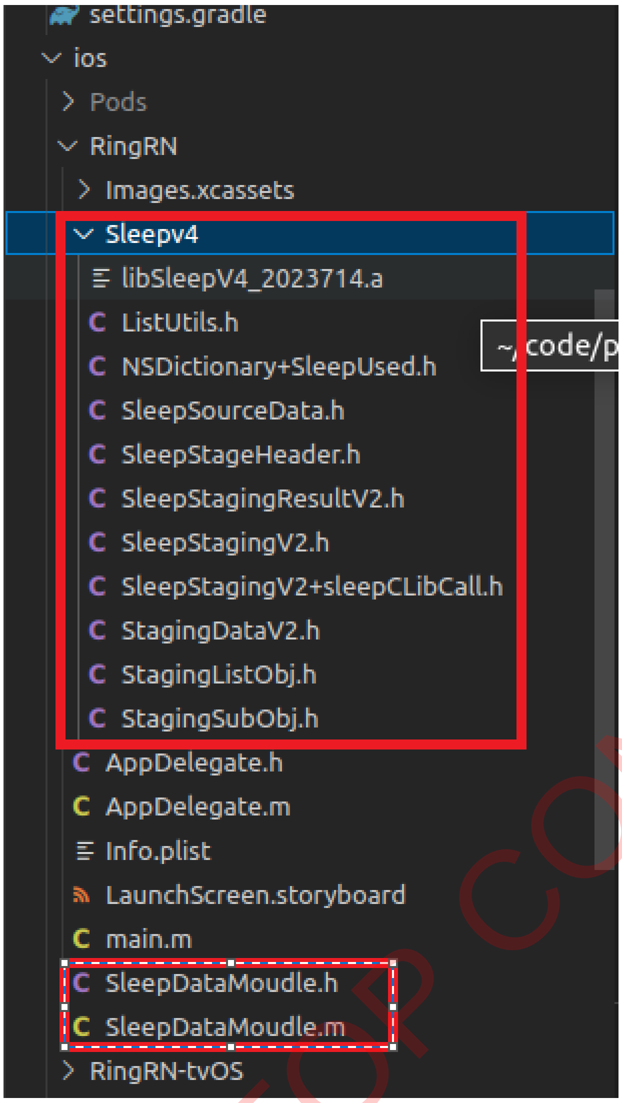
SleepDataModule.getIOSSleepData(Array,function)

Sleep data algorithm（ios）



```
SleepDataMoudle.getIOSSleepData(mArray.current, (error, result) => {
    var sleepTimeArray = [];
    for (let index = 0; index < result.length; index++) {
        const data = result[index];
        var lightTime = 0;
        var deepTime = 0;
        var remTime = 0;
        var wakeTime = 0;
        var napTime = 0;
        for (let index = 0; index < data.stagingList.length; index++) {
            const element = data.stagingList[index];
            switch (element.stagingType) {
                case "NREM1":
                    lightTime += element.endTime - element.startTime
                    break
                case "NREM3":
                    deepTime += element.endTime - element.startTime
                    break;
                case "REM":
                    remTime += element.endTime - element.startTime
                    break;
                case "WAKE":
                    wakeTime += element.endTime - element.startTime
                    break;
                case "NAP":
                    napTime += element.endTime - element.startTime
                    break;
            }
        }
        sleepTimeArray.push({
            deepSleep: `deepTime= ${Math.floor(deepTime / (1000 * 60 * 60))}h${Math.floor((deepTime % (100
            lightTime: `lightTime= ${Math.floor(lightTime / (1000 * 60 * 60))}h${Math.floor((lightTime %
            remTime: `remTime= ${Math.floor(remTime / (1000 * 60 * 60))}h${Math.floor((remTime % (1000 *
            wakeTime: `wakeTime= ${Math.floor(wakeTime / (1000 * 60 * 60))}h${Math.floor((wakeTime % (1000
            napTime: `napTime= ${Math.floor(napTime / (1000 * 60 * 60))}h${Math.floor((napTime % (1000 *
            startTime: formatDateTime(data.startTime),
            endTime: formatDateTime(data.endTime)
        })
    }
    setSleepTime(sleepTimeArray);
})
```

Code to be added to the iOS directory

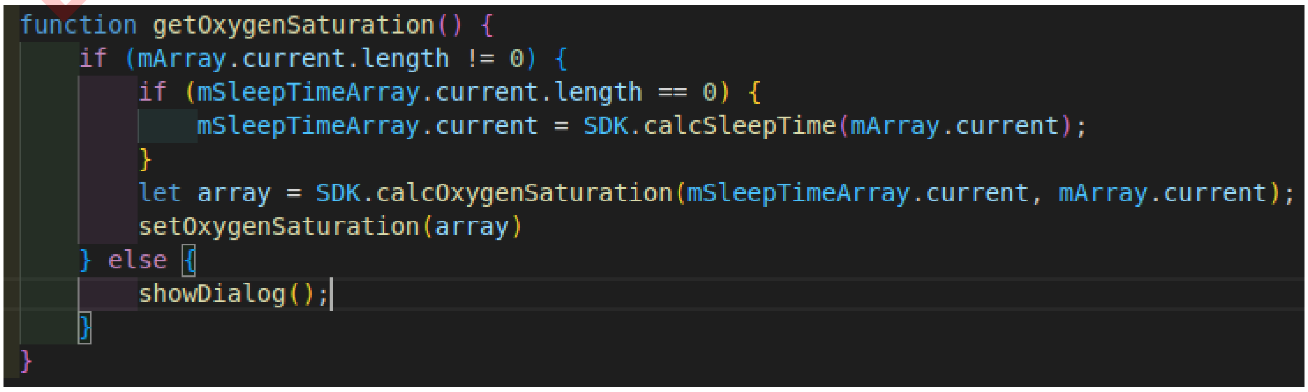


```
settings.gradle
ios
  Pods
  RingRN
    Images.xcassets
    Sleepv4
      libSleepV4_2023714.a
      ListUtils.h
      NSDictionary+SleepUsed.h
      SleepSourceData.h
      SleepStageHeader.h
      SleepStagingResultV2.h
      SleepStagingV2.h
      SleepStagingV2+sleepCLibCall.h
      StagingDataV2.h
      StagingListObj.h
      StagingSubObj.h
    AppDelegate.h
    AppDelegate.m
    Info.plist
    LaunchScreen.storyboard
    main.m
    SleepDataMoudle.h
    SleepDataMoudle.m
  RingRN-tvOS
```

Note that. a Static library needs to be added in xcode's build phases=>Link Binary With Libraries

11) SDK.calcOxygenSaturation(SleepTimeArray, Array)

Oxygen saturation algorithm



```
function getOxygenSaturation() {
    if (mArray.current.length != 0) {
        if (mSleepTimeArray.current.length == 0) {
            mSleepTimeArray.current = SDK.calcSleepTime(mArray.current);
        }
        let array = SDK.calcOxygenSaturation(mSleepTimeArray.current, mArray.current);
        setOxygenSaturation(array)
    } else {
        showDialog();
    }
}
```

Parameters:

SleepTimeArray: sleep data

Array: Historical data

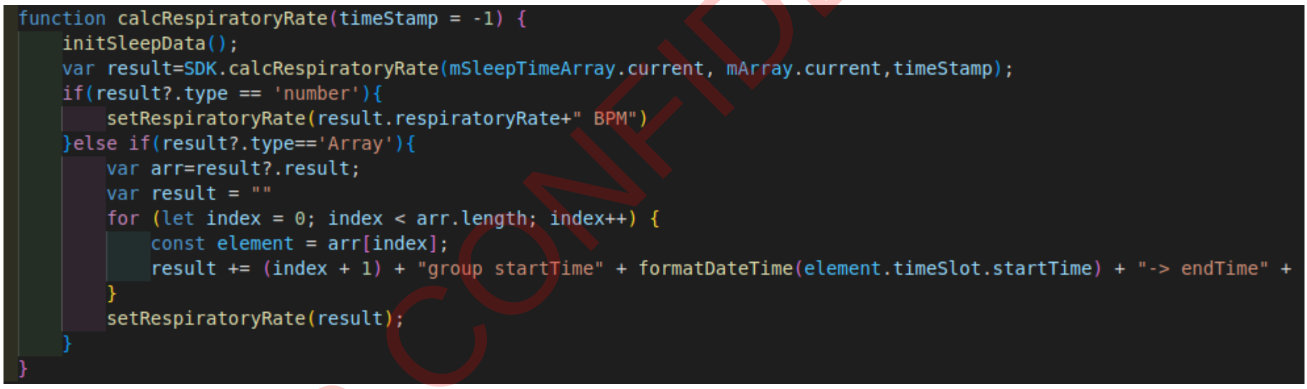
Return value:

StartTime: Sleep start time

EndTime: Sleep end time

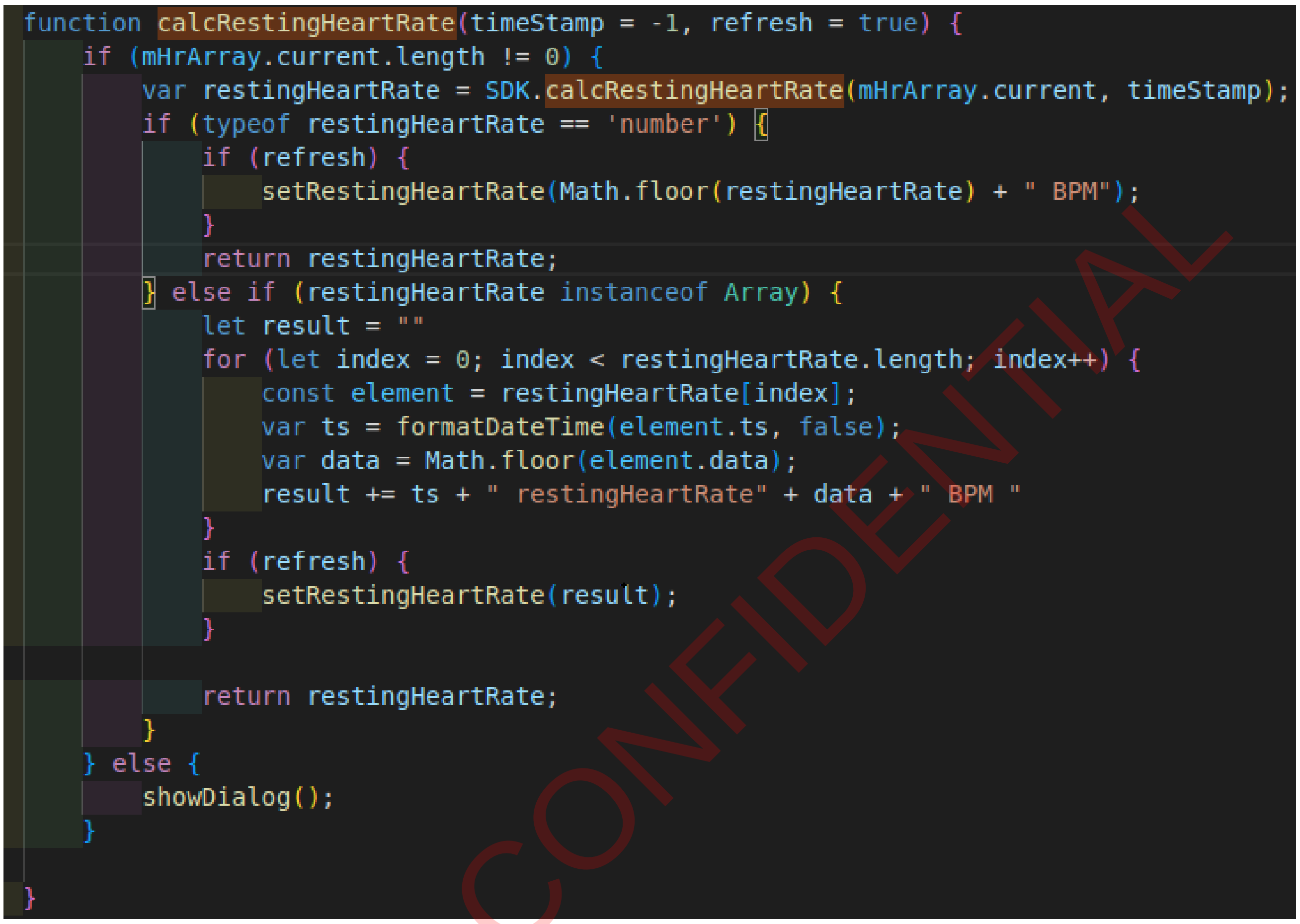
Oxygen: oxygen saturation value

12） **GetRespiratory**



```
function calcRespiratoryRate(timeStamp = -1) {
    initSleepData();
    var result=SDK.calcRespiratoryRate(mSleepTimeArray.current, mArray.current,timeStamp);
    if(result?.type == 'number'){
        setRespiratoryRate(result.respiratoryRate+" BPM")
    }else if(result?.type=='Array'){
        var arr=result?.result;
        var result = ""
        for (let index = 0; index < arr.length; index++) {
            const element = arr[index];
            result += (index + 1) + "group startTime" + formatDateTime(element.timeSlot.startTime) + "-> endTime" +
        }
        setRespiratoryRate(result);
    }
}
```

13) SDK.calcRestingHeartRate(HrArray, timeStamp);



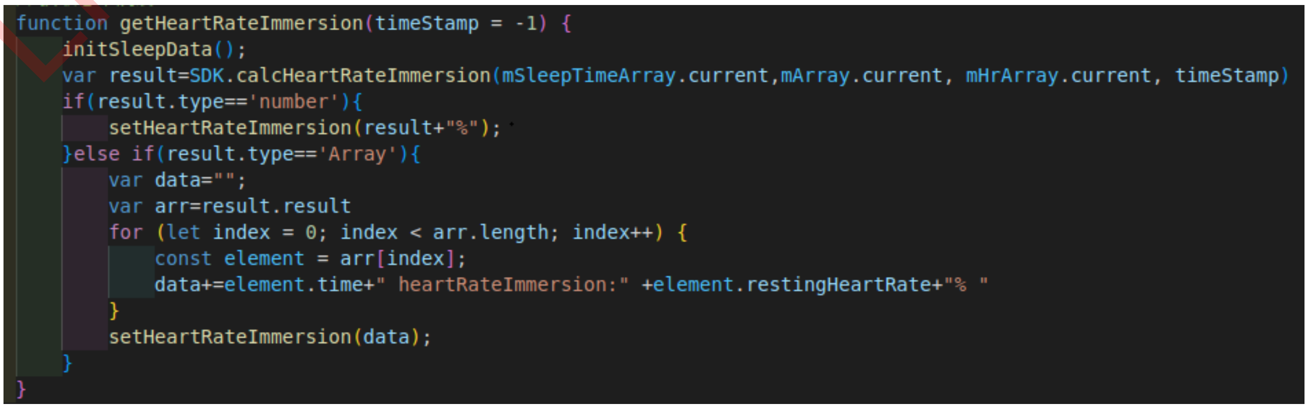
```
function calcRestingHeartRate(timeStamp = -1, refresh = true) {
    if (mHrArray.current.length != 0) {
        var restingHeartRate = SDK.calcRestingHeartRate(mHrArray.current, timeStamp);
        if (typeof restingHeartRate == 'number') {
            if (refresh) {
                setRestingHeartRate(Math.floor(restingHeartRate) + " BPM");
            }
            return restingHeartRate;
        } else if (restingHeartRate instanceof Array) {
            let result = ""
            for (let index = 0; index < restingHeartRate.length; index++) {
                const element = restingHeartRate[index];
                var ts = formatDateTime(element.ts, false);
                var data = Math.floor(element.data);
                result += ts + " restingHeartRate" + data + " BPM "
            }
            if (refresh) {
                setRestingHeartRate(result);
            }

            return restingHeartRate;
        }
    } else {
        showDialog();
    }

}
```

Parameters:

HrArray: Array of historical heart rate and timestamp

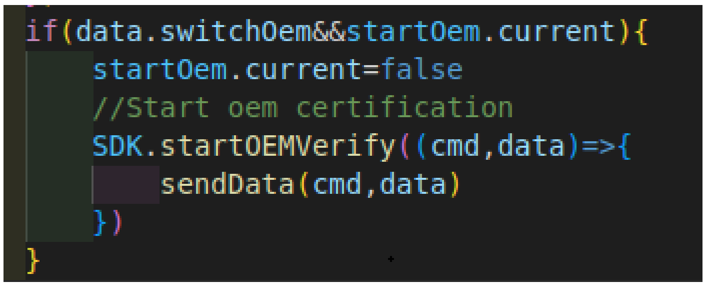
Timestamp: timestamp (calculated when the value is -1, the average heart rate of all sleep time periods. When the value is within a certain sleep time period, the resting heart rate of the current sleep period is calculated)

14) GetHeart Rate Variability



```
function getHeartRateImmersion(timeStamp = -1) {
    initSleepData();
    var result=SDK.calcHeartRateImmersion(mSleepTimeArray.current,mArray.current, mHrArray.current, timeStamp)
    if(result.type=='number'){
        setHeartRateImmersion(result+"%");
    }else if(result.type=='Array'){
        var data="";
        var arr=result.result
        for (let index = 0; index < arr.length; index++) {
            const element = arr[index];
            data+=element.time+" heartRateImmersion:" +element.restingHeartRate+"% "
        }
        setHeartRateImmersion(data);
    }
}
```

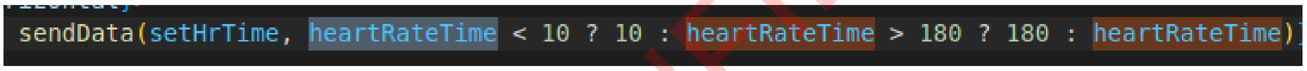
15) OEM Certification



```
if(data.switchOem&&startOem.current){
        startOem.current=false
        //Start oem certification
        SDK.startOEMVerify((cmd,data)=>{
            sendData(cmd,data)
        })
}
```

First, call the interface to obtain device information 1 and then obtain the OEM switch status of device information 1. When the OEM switch status is 1 enabled, call startOEMVerify to start OEM certification. If the ring has OEM certification enabled, OEM certification must be performed first before other functions can operate normally.

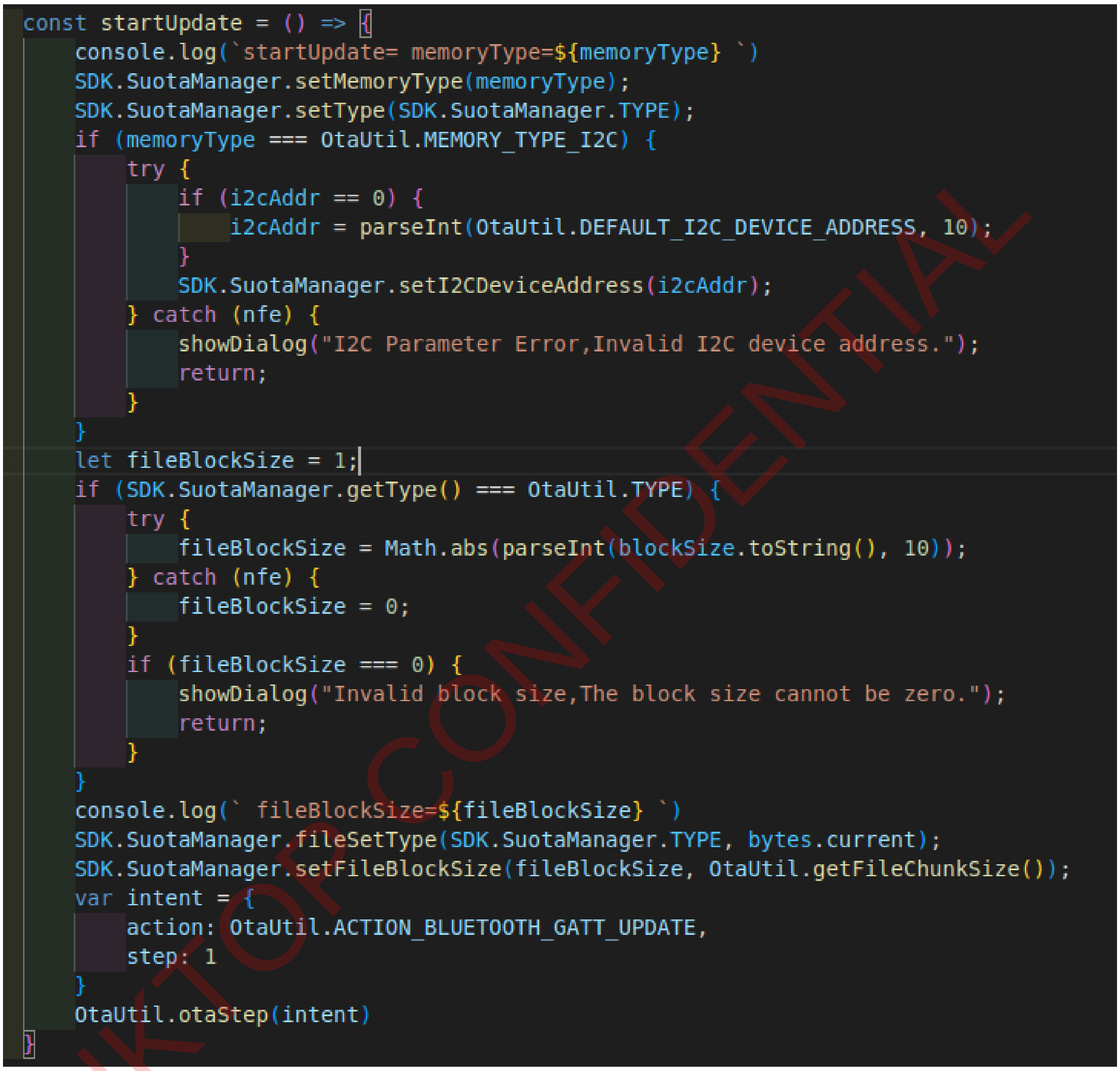
16) Heart Rate measurement time setting



```
sendData(setHrTime, heartRateTime < 10 ? 10 : heartRateTime > 180 ? 180 : heartRateTime)
```

The time range for heart rate measurement is between 10 and 180 seconds

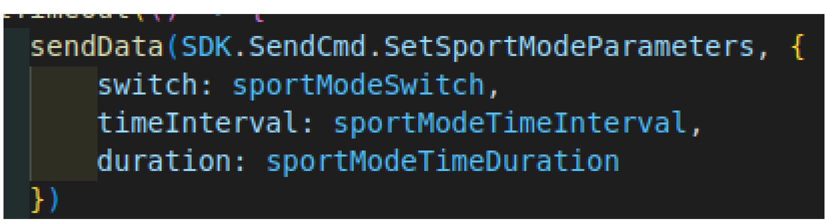
17) OTA



```
const startUpdate = () => {
    console.log(`startUpdate= memoryType=${memoryType} `)
    SDK.SuotaManager.setMemoryType(memoryType);
    SDK.SuotaManager.setType(SDK.SuotaManager.TYPE);
    if (memoryType === OtaUtil.MEMORY_TYPE_I2C) {
        try {
            if (i2cAddr == 0) {
                i2cAddr = parseInt(OtaUtil.DEFAULT_I2C_DEVICE_ADDRESS, 10);
            }
            SDK.SuotaManager.setI2CDeviceAddress(i2cAddr);
        } catch (nfe) {
            showDialog("I2C Parameter Error,Invalid I2C device address.");
            return;
        }
    }
    let fileBlockSize = 1;
    if (SDK.SuotaManager.getType() === OtaUtil.TYPE) {
        try {
            fileBlockSize = Math.abs(parseInt(blockSize.toString(), 10));
        } catch (nfe) {
            fileBlockSize = 0;
        }
        if (fileBlockSize === 0) {
            showDialog("Invalid block size,The block size cannot be zero.");
            return;
        }
    }
    console.log(` fileBlockSize=${fileBlockSize} `)
    SDK.SuotaManager.fileSetType(SDK.SuotaManager.TYPE, bytes.current);
    SDK.SuotaManager.setFileBlockSize(fileBlockSize, OtaUtil.getFileChunkSize());
    var intent = {
        action: OtaUtil.ACTION_BLUETOOTH_GATT_UPDATE,
        step: 1
    }
    OtaUtil.otaStep(intent)
}
```

The ring project defaults to selecting SPI for OTA upgrade, and the default parameter configuration does not need to be changed

18) Sports mode



```
sendData(SDK.SendCmd.SetSportModeParameters, {
    switch: sportModeSwitch,
    timeInterval: sportModeTimeInterval,
    duration: sportModeTimeDuration
})
```

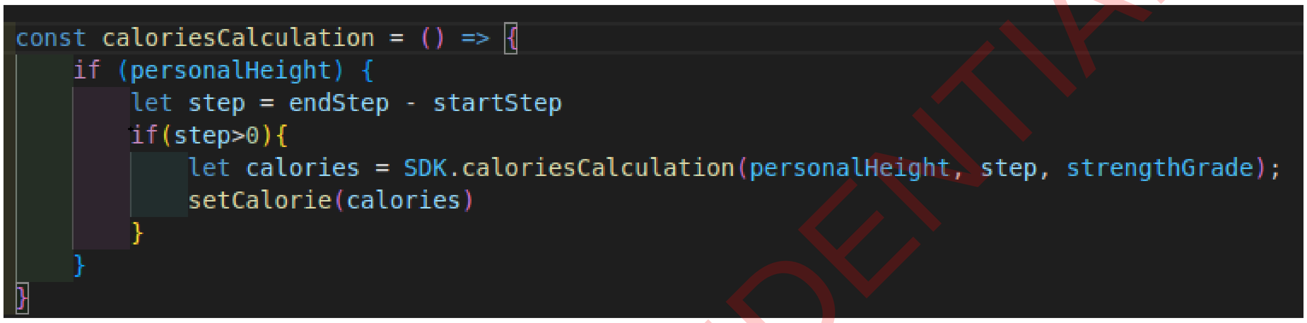
switch: 1: Turn on sports mode, 0: Turn off sports mode

TimeInterval: Data collection time interval (heart rate, steps, etc.) in seconds (10-180 seconds)

duration: Duration of sports mode in minutes (5-180min)

The data of sports mode can be obtained by calling the acquisition history interface

19) Calorie



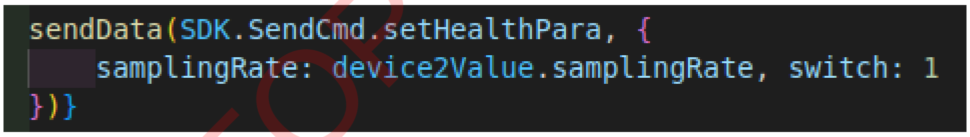
```
const caloriesCalculation = () => {
    if (personalHeight) {
        let step = endStep - startStep
        if(step>0){
            let calories = SDK.caloriesCalculation(personalHeight, step, strengthGrade);
            setCalorie(calories)
        }
    }
}
```

personalHeight: The height of the ring wearer, in cm

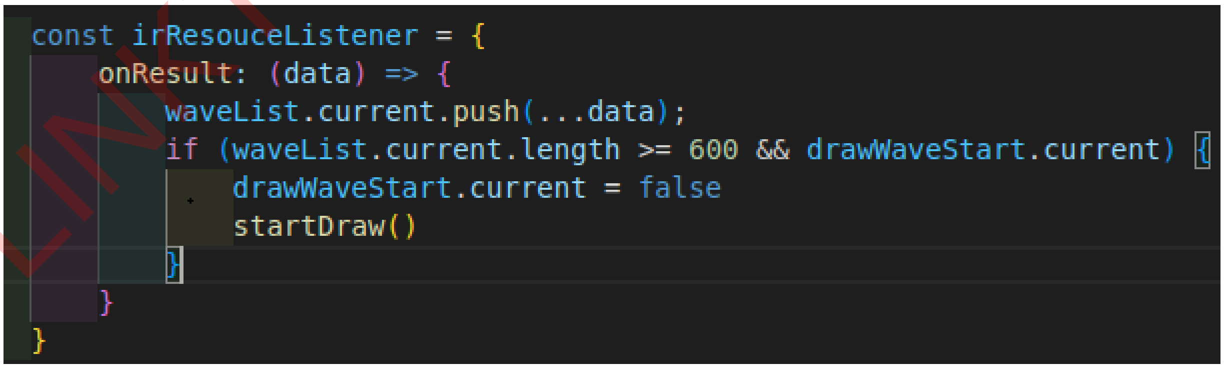
step: The number of steps the ring wearer takes to move

strengthGrade: The intensity of exercise is divided into three levels: 0.1 for high intensity exercise, 0.08 for moderate intensity exercise, and 0.05 for low intensity exercise

20) Infrared source data



```
sendData(SDK.SendCmd.setHealthPara, {
    samplingRate: device2Value.samplingRate, switch: 1
})}
```



```
const irResouceListener = {
    onResult: (data) => {
        waveList.current.push(...data);
        if (waveList.current.length >= 600 && drawWaveStart.current) {
            drawWaveStart.current = false
            startDraw()
        }
    }
}
```

samplingRate: The sampling rate is obtained through the device information 2 interface

switch: 1: open, 0: close

irResouceListener: waveform data reception

After opening the infrared source data, opening the blood oxygen measurement can obtain the source data of blood oxygen IR