Security：	Confidential	Version：	V1.1.7
Scope：	Internal	Status：	Released

FOIRing Flutter SDK API Guide

Nanjing Foinoe Co., Ltd

■ Records

Version	*Date*	*Remarks*
V1.0.0	2023.11.15	First version
V1.0.1	2023.12.04	Add ECG function
V1.0.2	2023.12.18	1)Add OTA prompt and PPG explanation; 2)RemoveSendBle(SendType.step)
V1.0.3	2023.12.20	Integrate Bluetooth protocol SDK into Smartring_Plugin facilitates unified management
V1.0.4	2024.01.23	1)Support infrared and red raw data 2)Get device information part 5 3)Add blood oxygen measurement settings API
V1.0.5	2024.03.21	Add SR23ECG function
V1.0.6	2024.04.19	1. Modify the history callback function parameters. 2. Add the acquisition of stress and temperature fluctuations.
V1.0.7	2024.04.22	1. Increase historical data battery percentage output 2. Add OTA firmware upgrade function through network download
V1.0.8	2024.04.26	1. Add ECG finger detection function 2. Add the ring to the device information 1 callback to see if it supports ECG information
V1.0.9	2024.07.03	1. Increase the temperature baseline output, 2. Modify the ECG sampling rate support range 3. ECG measurement switch disp_sRC field added value 2
V1.1.0	2024.07.23	Modify the loading method of the iOS integration. a library
V1.11	2024.11.11	Fixed the issue of null sportsMode value in historical data
V1.1.2	2024.11.28	The ECG function requires adding image indicators to the files that must be imported for IOS and Android, as well as the issue of not being able to find the release version library for IOS development
V1.1.3	2024.11.29	Filter historical records and real-time measurement of heart rate anomalies
V1.1.4	2025.01.15	Add SR28 interface

V1.1.5	2025.02.28	Increase SR28 blood glucose function
V1.1.6	2025.04.18	Fixed the issue where the device returned abnormal data when connecting to SR28 to measure ECG using the Demo App.
V1.1.7	2025.04.30	Update sleep algorithm library.

■

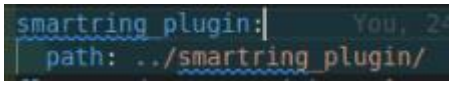
■

■ Table of Contents

1. Integration Instructions

1.1. Project Import

Add smartring_ The plugin project is placed in the same file directory as the main project, and then added to the pubspec.yaml of the main project.



Smartring_plugin engineering department is responsible for implementing native code functions for Flutter, including sleep, resting heart rate, respiratory rate, blood oxygen saturation, heart rate immersion, etc., as well as Bluetooth protocol parsing.

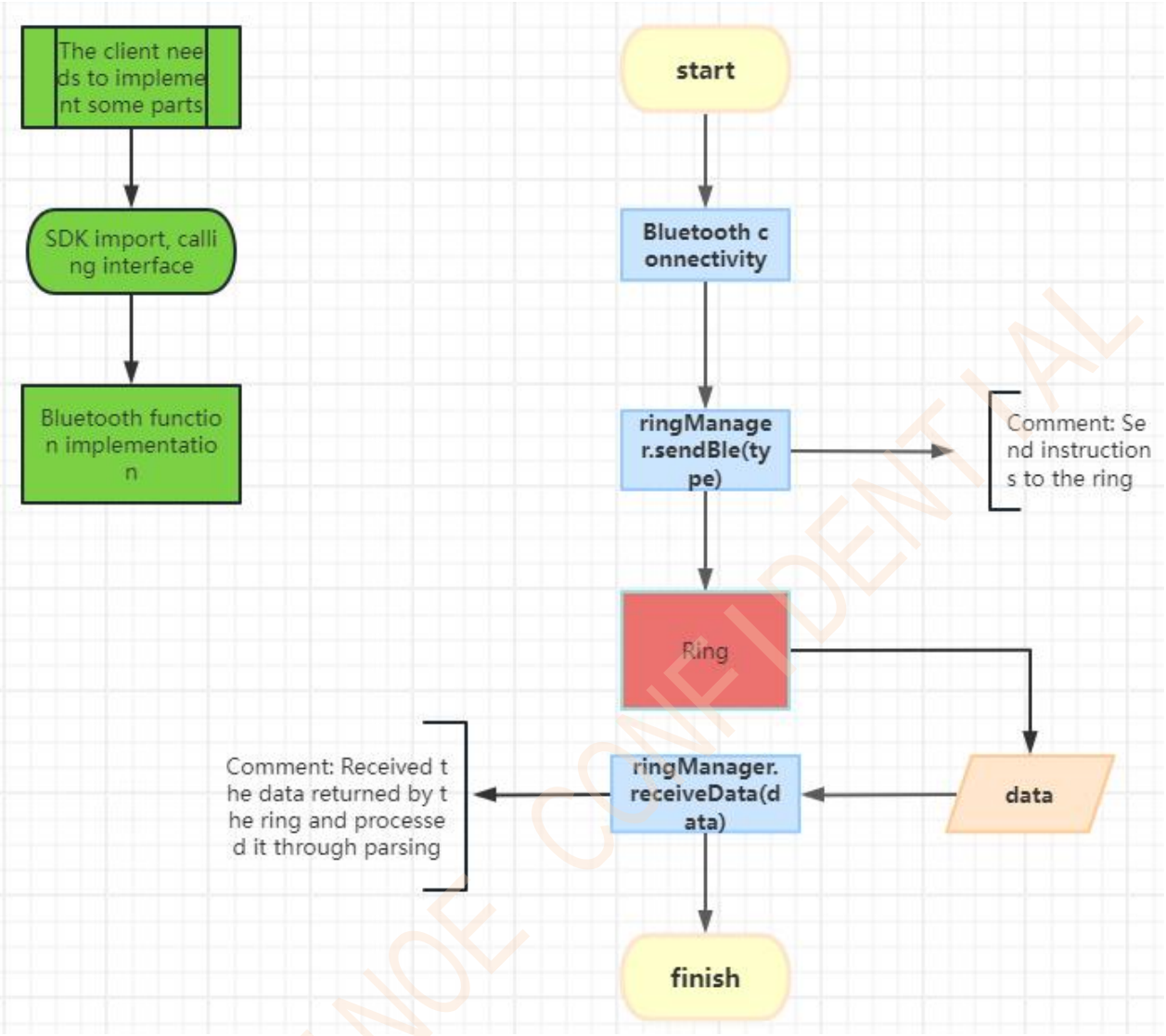
1.2. References

2. API Description

2.1. Function Description

The SDK establishes a connection with the device through BLE. You can easily communicate with devices, provide data services such as blood oxygen and body temperature through the SDK. You can use this SDK to develop Flutter applications based on Dart.

2.2. Flow Chart



2.3. Interface

2.3.1. Initialization

Initialize when need to access:

import ‘package:smartring_plugin/sdk/core.dart’;(Responsible for Bluetooth protocol transmission and processing)

import‘package:smartring_plugin/sdk/common/ble_protocol_constant.dart’;（Command Set for Bluetooth Protocol）

import ‘../Bluetooth/Bluetooth_manager’;(Bluetooth function encapsulation)

RingManager.instance.sendBle(type,data);(Assemble the data and send it to the ring)

RingManager.instance.registerProcess(ReceiveType,(data){});(Register the callback interface to receive the data sent by the ring for processing).

2.3.2. Turn on and off the sport mode

Support SR09 and SR23 projects.

1) Function:

SendBle(SendType.setSportModePatameters,{ “switch” ,” timeInterval” ,” duration” })

2) Parameter:

- SendType.setSportModePatameters: Send Type;
- switch: 1 on, 0 off;
- timeInterval: Record data time interval (10-180 seconds);
- duration: Duration time (5-180 minutes);

3) Callback:

None

4) Sample Code:

```
void onSportMode() {
  if (sportValue["sportMode"].value == 1) {
    startStep = 0;
    endStep = 0;
    sportStart = true;
    sendBle(SendType.step);
    Future.delayed(const Duration(seconds: 1), () {
      sendBle(SendType.setSportModeParameters, {
        "switch": sportValue["sportMode"].value,
        "timeInterval": sportValue["timeInterval"].value,
        "duration": sportValue["duration"].value
      });
    }); // Future.delayed
    futureCancle = Future.delayed(
        Duration(milliseconds: sportValue["duration"].value * MINUTE + 2000),
        () {
      sportStart = false;
      sendBle(SendType.step);
    }); // Future.delayed
  } else {
    sendBle(SendType.setSportModeParameters, {
      "switch": sportValue["sportMode"].value,
      "timeInterval": sportValue["timeInterval"].value,
      "duration": sportValue["duration"].value
    });
    sportStart = false;
    sendBle(SendType.step);
    futureCancle.timeout(const Duration(milliseconds: 0), onTimeout: () {
      print("[futureCancle]");
    });
  }
}
```

2.3.3. Set heart rate measurement time

Support SR09 and SR23 projects.

1) Function:

SendBle(SendType.setHrTime,{ “time” })

2) Parameter:

SendType.setHrTime: Send Type

time: Heart rate measurement time 10-180 seconds;

3) Callback:

None

4) Sample Code

None

2.3.4. Enable or Disable measurement

Support SR09, SR23 and SR28 projects.

1) Function:

SendBle (SendType. openSingleHealth);

Enable heart rate and HRV measurements

SendBle (SendType. closeSingleHealth);

Disableheart rateand HRV measurements.

SendBle (SendType. openHealth);

Enable heart rate and blood oxygen measurement.

SendBle (SendType. closeHealth)

Disable heart rate and blood oxygen measurement.

2) Parameter:

SendType. openSingleHealth: Send Type

SendType. closeSingleHealth: Send Type

SendType. openHealth: Send Type

SendType. closeHealth: Send Type

3) Callback:

Support SR09 and SR23 projects.

RingManager. registerProcess (ReceiveType. Health, (data) {});

Register heart rate, blood oxygen, and HRV data callback.

Parameter:

- ReceiveType. Health: Receive Type
- (data){}:callback function

Return:

- oxValue:Blood oxygen value
- heartValue:Heart rate values, when the measurement is invalid, the return value is -1.
- hrvValue:Hrv values

Support SR28 project.

RingManager.registerProcess(ReceiveType.PPG_MEASUREMENT,(data){});

Parameter:

- ReceiveType. PPG_MEASUREMENT: Receive Type
- (data){}:callback function

Return:

- blood_oxygen: The range of blood oxygen values is 1-100%; 0xFF is invalid data.
- heart_rate: The range of heart rate values is 0-200; 0xFF is invalid data.
- hrv: Heart rate variability (HRV): To output the RMSSD value, it is necessary to take the RMSSD value with HRV reliability value>60. If the HRV reliability value is<60 or HRV is not measured, the RMSSD value output is 0xFFFF, ranging from 0-65535.
- status:
  0: Measurement not started.
  1: During measurement.
  2: Effective measurement data.
- heart_rate_quality: hart rate quality index(QI), Range 1-100, when the value is 40 or above, it indicates accurate heart rate.
- respiratory_rate: Output the respiratory rate value, which must take the respiratory rate reliability value>50 to output. If the respiratory rate reliability value is<50 or no respiratory rate is measured, the output respiratory rate value is 0xFF, ranging from 0-50.
- oxygen_r_value: Blood oxygen R value.
- ibi: Only output IBI values with a QI value of 100. If the QI value is 0 or no interval is detected, it is 0xFFFF, in milliseconds, with a range of 0-65535.
- stress: Pressure value, no stress value measured is 0xFF.
- cardiac coherence: Range 0-1, no detected Cardiac coherence value of 0xFF.

4) Sample Code

```
ringManager.registerProcess(ReceiveType.Health, (data) {
  if(data["oxValue"]!=25){
    oxValue.value = data["oxValue"].toString();
  }
  heartValue.value = data["heartValue"].toString();
  hrvValue.value = data["hrvValue"].toString();
});
//PPG_MEASUREMENT
ringManager.registerProcess(ReceiveType.PPG_MEASUREMENT, (data) {
  debugPrint("PPG_MEASUREMENT data=$data ");
});
```

2.3.5. Enable battery data acquisition

Support SR09, SR23 and SR28 projects.

1) Function:

SendBle (SendType. batteryDataAndState)

2) Parameter:

SendType. batteryDataAndState : Send Type

3) Callback:

RingManager. registerProcess (ReceiveType. BatteryDataAndState, (data) {});

Register battery data callback

Paramter:

- ReceiveType. BatteryDataAndState: Receive Type
- (data){}:callback function

Return:

- Status: 0: Discharging, 1: Charging;
- BatteryValue: Battery voltage;
- BatteryPer: Battery level percentage;

4) Sample Code

```
ringManager.registerProcess(ReceiveType.BatteryDataAndState, (Map data) {
  if (data.isNotEmpty) {
    var isWireless = false;
    String bleName = blueToothManager.getDeviceName();
    if (bleName.isNotEmpty) {
      isWireless = !bleName.toUpperCase().contains('W') ? false : true;
    }
    var charging = data["status"] == 1;
    var result = charging ? "charging" : "uncharged";
    var battery_per = 0;
    if (data["batteryPer"] != null) {
      battery_per = data["batteryPer"];
    } else {
      battery_per = nexring_plugin.toBatteryLevel(
          data["batteryValue"], charging, isWireless);
    }
    batteryValue.value = data["batteryValue"];
    batteryState.value = result;
    batteryPer.value = battery_per;
  }
});
```

2.3.6. Get device information - Part1

Support SR09, SR23 and SR28 projects.

1) Function:

SendBle(SendType.deviceInfo1)

2) Parameter:

SendType.deviceInfo1: Send Type

3) Callback:

RingManager. registerProcess (ReceiveType. DeviceInfo1, (data) {});

Register Device Information Data Callback.

Paramter:

- ReceiveType. DeviceInfo1
- (data){}:callback function

Return:

- color: The color of the ring;
- size: The size of the ring;
- bleAddress: Bluetooth MAC address of the ring;
- deviceVer: The device version number of the ring;
- switchOem:

  0: OEM certification switch off

  1: OEM certification switch on

- chargingMode:

  0: Magnetic Charging in Box(With Battery)

  1: Common Wireless Charging

  2: NFC Wireless Charging

  3: Magnetic Charging in Box(Without Battery)

  4: USB Cable Magnetic Charging

- mainChipModel: Chip type;
- productIteration: Product Iteration;
- hasSportsMode: Does it support sports mode or not;
- IsSupportEcg: Does it support ECG function

4) Sample Code

```
ringManager.registerProcess(ReceiveType.DeviceInfo1, (Map data) {
  if (data.isNotEmpty) {
    var color = "";
    if (data["color"] == 0) {
      color = "Deep Black";
    } else if (data["color"] == 1) {
      color = "Silver";
    } else if (data["color"] == 2) {
      color = "Gold";
    } else if (data["color"] == 3) {
      color = "Rose Gold";
    }
    devColor.value = color;
    devSize.value = data["size"];
    devAddress.value = data["bleAddress"];
    devVersion.value = data["deviceVer"];
    switchOem.value = data["switchOem"];
    chargingMode.value = data["chargingMode"];
    mainChipModel.value = data["mainChipModel"];
    productIteration.value = data["productIteration"];
    hasSportsMode.value = data["hasSportsMode"];
    if (switchOem.value && isStartOem) {
      isStartOem = false;
      oemVerify();
    }
  }
});
```

2.3.7. Get device information - Part2

Support SR09, SR23 and SR28 projects.

1) Function:

SendBle(SendType.deviceInfo2)

2) Parameter:

SendType.deviceInfo2: Send Type

3) Callback:

RingManager. registerProcess (ReceiveType. DeviceInfo2, (data) {});

Register Device Information Data Callback

Paramter:

- ReceiveType. DeviceInfo2
- (data){}:callback function

Return:

- sn: Device serial number;
- bindStatus: Device binding status;
- samplingRate: PPG Sampling rate;

4) Sample Code

```
ringManager.registerProcess(ReceiveType.DeviceInfo2, (Map data) {
  if (data.isNotEmpty) {
    sn.value = data["sn"];
    bindStatus.value = data["bindStatus"];
    samplingRate.value = data["samplingRate"];
  }
});
```

2.3.8. Device shutdown

Support SR09, SR23 and SR28 projects.

1) Function:

SendBle(SendType.shutDown)

2) Parameter:

SendType.shutDown: Send Type

3) Callback:

None

4) Sample Code

None

2.3.9. Time synchronization

Support SR09, SR23 and SR28 projects.

1) Function:

SendBle(SendType. timeSyn)

2) Parameter:

SendType. timeSyn: Send Type

3) Callback:

None

4) Sample Code

None

Note:Whenload the application each time, it should send this command to synchronize the time.

2.3.10. Device binding

Support SR09, SR23 and SR28 projects.

1) Function:

SendBle(SendType. deviceBind)

2) Parameter:

SendType. deviceBind: Send Type

3) Callback:

None

4) Sample Code

None

2.3.11. Device unbinding

Support SR09, SR23 and SR28 projects.

1) Function:

SendBle(SendType. deviceUnBind)

2) Parameter:

SendType. deviceUnBind: Send Type

3) Callback:

None

4) Sample Code

None

2.3.12. Device restart

Support SR09, SR23 and SR28 projects.

1) Function:

SendBle(SendType. restart)

2) Parameter:

SendType.restart : Send Type

3) Callback:

None

4) Sample Code

None

2.3.13. Restore factory settings

Support SR09, SR23 and SR28 projects.

1) Function:

SendBle(SendType. restoreFactorySettings)

2) Parameter:

SendType. restoreFactorySettings: Send Type

3) Callback:

None

4) Sample Code

None

2.3.14. Clear historical data

Support SR09, SR23 and SR28 projects.

1) Function:

SendBle(SendType. cleanHistoricalData)

2) Parameter:

SendType. cleanHistoricalData: Send Type

3) Callback:

None

4) Sample Code

None

2.3.15. Get finger temperature

Support SR09, SR23 and SR28 projects.

1) Function:

SendBle(SendType. temperature)

2) Parameter:

SendType. temperature: Send Type

3) Callback:

RingManager. registerProcess (ReceiveType. Temperature, (data) {});

Register finger temperature callback.

Parameter:

- ReceiveType. Temperature
- (data){}:callback function

Return:

- data: Finger temperature value

4) Sample Code

```
ringManager.registerProcess(ReceiveType.Temperature, (data) {
  if (data.isNotEmpty) {
    temperatureRes.value = " Temperature :$data";
  }
});
```

2.3.16. Get the quantity of historical data

Support SR09, SR23 and SR28 projects.

1) Function:

SendBle(SendType. historicalNum)

2) Parameter:

SendType. historicalNum: Send Type

3) Callback:

RingManager. registerProcess (ReceiveType. HistoricalNum, (data) {});

Registration History Data Quantity Callback

Paramter:

- ReceiveType. HistoricalNum
- (data){}:callback function

Return:

- uum: Number of historical data;
- minUUID: Minimum value of uuid;
- maxUUID: The maximum value of uuid;

4) Sample Code

```
ringManager.registerProcess(ReceiveType.HistoricalNum, (Map data) {
  if (data.isNotEmpty) {
    endUUID = data["maxUUID"];
    startUUID = data["minUUID"];
    numUUID = data["num"];
  }
});
```

2.3.17. Get historical data

Support SR09, SR23 and SR28 projects.

1) Function:

SendBle(SendType. historicalData)

2) Parameter:

SendType. historicalData: Send Type

3) Callback:

RingManager. registerProcess (ReceiveType. HistoricalData, (data) {});

Registration History Data Callback

Parameter:

- ReceiveType. HistoricalData
- (data){}:callback function

Return:

- uuid: UUID value, accumulated from 1;
- historyArray:A historical data array that returns all historical data in the form of an array, containing the following data:
  - hrv: HRV value;
  - timeStamp: TimeStamp;
  - HeartRate: Heart rate, when the measurement is invalid, the return value is -1;
  - motionDetectionCount: Motion value;
  - detectionMode:

    0: Heart rate measurement mode

    1: Blood oxygen measurement mode;

  - wearStatus:

    0: Not worn,

    1: Wearing;

  - chargeStatus:

    0: Discharging

    1: Charging;

  - uuid: UUID value, accumulated from 1;
  - temperature: Finger temperature;
- step: Step value;
- ox: Blood oxygen
- rawHr: The raw heart rate data set is used for judgment purposes;
- sportsMode: Is sports mode enabled or not;
- batteryLevel: Percentage of the battery; (SR09 firmware version V2.2.2 and above are effective. This data is not accurate for versions before V2.2.2 and is not used as a reference.)

IMPORTANT:

1. Use the sleep algorithm to calculate sleep data (2.3.18 Get Sleep Data). The Time Stamp, Heart Rate, HRV, Motion Detection Count, Step, OX and other data in the historical data need to be stored in an array. You need to use (wearStatus==1, chargeStatus==0，heartRate）=50，heartRate<=175) as the filtering condition. For specific operations, refer to the dealHistoryData method in the demo, and then substitute the array data into the smartring_plugin.sleepAlgorithm(arr) method to calculate the sleep data. It is recommended to store the historical data in the database first after the end time of the last sleep yesterday, and then clean up the historical data.

2. It is recommended that the starting data of the historical data sent to the sleep algorithm should be the first historical data after the end time of the last sleep yesterday, and the end data should be the last historical data obtained from the ring.

4) Sample Code

```
ringManager.registerProcess(ReceiveType.HistoricalData, (Map data) {
  if (data.isNotEmpty) {
    var uuid = data['uuid'];
    if (uuid != endUUID) {
      historyStart.value="Getting data started";
      double process = (uuid - startUUID) / numUUID;
      progressValue.value = process;
      historyRawDataArray.add(data);
    } else {
      historyStart.value="Finish";
      historyRawDataArray.add(data);
      progressVisit.value = false;
      dealHistoryData(historyRawDataArray);
    }

  }
});
```

Note: Please send the command of historical data quantity before this command.

2.3.18. Sleep data calculation

Support SR09, SR23 and SR28 projects.

1) Function:

Support SR09, SR23

FOIRing_plugin.sleepAlgorithm(sleepArray)

Note: Historical data must be obtained first

Support SR28

FOIRing_plugin.sleepNewAlgorithm(sleepArray,newAlgorithmHistoryData)

2) Parameter:

sleepArray:

- ts: timeStamp
- hr:heart rate
- hrv:hrv
- motion: Motion detection count
- steps: step value
- ox: blood oxygen data

IMPORTANT:

We need to filter data with wearStatus==1, chargeStatue==0, HeartRate>=50, and HeartRate<=175. Please refer to the data retrieval interface (2.3.17 Get Historical Data)

newAlgorithmHistoryData:

- type: 0-sleep, 1-wake-up
- ts:sleep time stamp or wake time stamp.
- bed_rest_duration: time in bed, unit: minute, range 0-64800
- wake_order: Awakening sequence, range 0-25

IMPORTANT:

Data acquisition interface view (2.3.35 Get Vital Signs Historical Data, ACC Sleep Data and Activity Data, (7) RingManager.registerProcess(ReceiveType.SLEEP_HISTORY,(data){}))

3) Return

- 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: Array of awakening time periods;
- napList: Array of sporadic nap time periods;

4) Sample Code

```
void getNewSleepData() {
  if (sleepArray.isEmpty) {
    showToast("Please obtain historical data first");
    return;
  }
  final sleepResult =
      smartring_plugin.sleepNewAlgorithm(sleepArray, newAlgorithmHistoryData)
  sleepTimeArray = [];
  sleepTimePeriodArray = [];
  newSleepAnalysis.value = "";
  for (var i = 0; i < sleepResult.length; i++) {
    var data = sleepResult[i];
    var lightTime = 0;
    var deepTime = 0;
    var remTime = 0;        You, 13小时前 • Uncommitted changes
    var wakeTime = 0;
    var napTime = 0;
    var startTime = data["startTime"];
    var endTime = data["endTime"];
    var stagingList = data["stagingList"];
    for (var i = 0; i < stagingList.length; i++) {
      var staging = stagingList[i];
      switch (staging["type"]) {
        case smartring_plugin.SleepType.WAKE:    "smartring": Unknown word.
          wakeTime = staging["endTime"] - staging["startTime"] + wakeTime;
          break;
        case smartring_plugin.SleepType.NREM1:    "smartring": Unknown word.
          lightTime = staging["endTime"] - staging["startTime"] + lightTime;
          break;
        case smartring_plugin.SleepType.NREM3:    "smartring": Unknown word.
          deepTime = staging["endTime"] - staging["startTime"] + deepTime;
          break;
        case smartring_plugin.SleepType.REM:    "smartring": Unknown word.
          remTime = staging["endTime"] - staging["startTime"] + remTime;
          break;
        case smartring_plugin.SleepType.NAP:    "smartring": Unknown word.
          napTime = staging["endTime"] - staging["startTime"] + napTime;
          break;
      }
    }
    sleepTimeArray.add({
```

```
void getSleepData() {
  if (sleepArray.isEmpty) {
    showToast("Please obtain historical data first");
    return;
  }
  final sleepResult = nexring_plugin.sleepAlgorithm(sleepArray);
  print("getSleepData  sleepResult=$sleepResult");
  sleepTimeArray = [];
  sleepTimePeriodArray = [];
  sleepAnalysis.value="";
  for (var i = 0; i < sleepResult.length; i++) {
    var data = sleepResult[i];
    var lightTime = 0;
    var deepTime = 0;
    var remTime = 0;
    var wakeTime = 0;
    var napTime = 0;
    var startTime = data["startTime"];
    var endTime = data["endTime"];
    var stagingList = data["stagingList"];
    // print("i=$i  stagingList=$stagingList ");
    for (var i = 0; i < stagingList.length; i++) {
      var staging = stagingList[i];
      switch (staging["type"]) {
        case nexring_plugin.SleepType.WAKE:
          wakeTime = staging["endTime"] - staging["startTime"] + wakeTime;
          break;
        case nexring_plugin.SleepType.NREM1:
          lightTime = staging["endTime"] - staging["startTime"] + lightTime;
          break;
        case nexring_plugin.SleepType.NREM3:
          deepTime = staging["endTime"] - staging["startTime"] + deepTime;
          break;
        case nexring_plugin.SleepType.REM:
          remTime = staging["endTime"] - staging["startTime"] + remTime;
          break;
        case nexring_plugin.SleepType.NAP:
          napTime = staging["endTime"] - staging["startTime"] + napTime;
          break;
      }
    }
    sleepTimeArray.add({
      "deepSleep":
          "deepTime= ${(deepTime ~/ HOUR)}h${((deepTime % HOUR) ~/ MINUTE)}m",
      "lightTime":
          "lightTime= ${(lightTime ~/ HOUR)}h${((lightTime % HOUR) ~/ MINUTE)}m",
      "remTime":
```


2.3.19. Calculation of resting heart rate

Support SR09, SR23 and SR28 projects.

1) Function:

FOIRing_plugin.restingHeartRate(data)

Note: Historical data must be obtained first

2) Parameter:

- data:hrArray contain timestamp and heartRate

3) Return

- ts:timeStamp
- data:restingHeartRate

4) Sample Code

```
void getRestingHeartRate() {
  if (hrArray.isEmpty) {
    showToast("Please obtain historical data first");
    return;
  }
  List restingHeartRateArray = nexring_plugin.restingHeartRate(hrArray);
  String result = "";
  for (var i = 0; i < restingHeartRateArray.length; i++) {
    var data = restingHeartRateArray[i];
    result =
        "part$i time:${nexring_plugin.formatDateTime(data["ts"],isFull:false)} restingHeartRate:${data["data"]} $result";
  }
  restingHeartRate.value = result;
}
```

2.3.20. Respiratory rate calculation

Support SR09, SR23 and SR28 projects.

1) Function:

FOIRing_plugin.respiratoryRate(sleepTimePeriodArray,sleepArray)

Note: Sleep data calculation must be performed first

2) Parameter:

- sleepTimePeriodArray: The start and end times of sleep
- sleepArray: contain timeStamp,heartRate,hrv,motion,step, Blood oxygen

3) Return

- startTime:The start of sleep
- endTime:end times of sleep
- respiratoryRate: respiratoryRate value

4) Sample Code

```
void getRespiratoryRate() {
  if (sleepTimePeriodArray.isEmpty) {
    showToast(
        "Please obtain sleep data first, or there is no sleep data available");
    return;
  }
  List respiratoryRateArray =
      nexring_plugin.respiratoryRate(sleepTimePeriodArray, sleepArray);
  String result = "";
  for (var i = 0; i < respiratoryRateArray.length; i++) {
    var data = respiratoryRateArray[i];
    result =
        "part$i startTime:${nexring_plugin.formatDateTime(data["startTime"])} endTime:${nexring_plugin.
  }
  respiratoryRate.value = result;
}
```

2.3.21. Blood oxygen saturation calculation

Support SR09, SR23 and SR28 projects.

1) Function:

FOIRing_plugin.oxygenSaturation(sleepTimePeriodArray,sleepArray)

Note: Sleep data calculation must be performed first

2) Parameter:

- sleepTimePeriodArray: The start and end times of sleep
- sleepArray: contain timeStamp,heartRate,hrv,motion,step, Blood oxygen

3) Return

- startTime: The start of sleep
- endTime: end times of sleep
- oxygen: oxygen value

4) Sample Code

```
void getOxSaturation() {
  if (sleepTimePeriodArray.isEmpty) {
    showToast(
        "Please obtain sleep data first, or there is no sleep data available");
    return;
  }
  var oxSaturationArray =
      nexring_plugin.oxygenSaturation(sleepTimePeriodArray, sleepArray);
  String result = "";
  for (var i = 0; i < oxSaturationArray.length; i++) {
    var data = oxSaturationArray[i];
    result =
        "part$i startTime:${data["startTime"]} endTime:${data["endTime"]} oxygen:${data["oxygen"]} $result";
  }
  oxSaturation.value = result;
}
```

2.3.22. Heart rate immersion calculation

Support SR09, SR23 and SR28 projects.

1) Function:

FOIRing_plugin.heartRateImmersion(sleepTimePeriodArry,sleepArray,hrArray)

Note: Sleep data calculation must be performed first

2) Parameter:

- sleepTimePeriodArray: The start and end times of sleep
- sleepArray: contain timeStamp,heartRate,hrv,motion,step, Blood oxygen
- hrArray: contain timeStamp,heartRate

3) Return

- Time:time
- restingHeartRate: restingHeartRate value

4) Sample Code

```
void getHrImmersion() {
  if (sleepTimePeriodArray.isEmpty) {
    showToast(
        "Please obtain sleep data first, or there is no sleep data available");
    return;
  }
  var hrImmersionArray = nexring_plugin.heartRateImmersion(
      sleepTimePeriodArray, sleepArray, hrArray);
  String result = "";
  for (var i = 0; i < hrImmersionArray.length; i++) {
    var data = hrImmersionArray[i];
    result =
        "part$i time:${data["time"]}  restingHeartRate:${data["restingHeartRate"].toStringAsFixed(1)} $result";
  }
  hrImmersion.value = result;
}
```

2.3.23. PPG raw data switch

Support SR09, SR23 and SR28 projects.

1) Function:

sendBle(SendType.setHealthPara,{“samplingRate”,switch})

Note: a). Whenyou turn on this, please call sendBle (SendType. openHealth) first;

b). The common version Ring suppport IR raw data only.

2) Parameter:

- SendType.setHealthPara: Send Type
- SamplingRate: Sampling rate, this value is in device information part2;
- Switch: 1 on, 0 off

3) Callback:

RingManager. registerProcess (ReceiveType. IRsource, (data) {})

Register raw data callback

Parameter:

- ReceiveType. IRsource
- (data){}:callback function

Return:

- data:PPGraw data (IR data)

  Note: This callback will receive 8 raw data at once, which are signed data, ranging from -32768 to 32767.

4) Sample Code


```
ringManager.registerProcess(ReceiveType.IRresouce, (data) {
  irWaveList.addAll(data);
  update.value = !update.value;
  if (irWaveList.length > 600) {
    irWaveList.removeRange(0, 8);
  }
});
```

2.3.24. PPG raw data switch – Enhanced

Support SR09, SR23 and SR28 projects.

1) Function:

sendBle(SendType.setHealthPara,{“samplingRate”,switch})

Note: a). Whenyou turn on this, please call sendBle (SendType. openHealth) or SendBle (SendType. openSingleHealth)first.

b). The ring firmware version need 2.1.5 or later.

2) Parameter:

- SendType.setHealthPara: Send Type
- SamplingRate: Sampling rate, this value is in device information part2;
- Switch: 1 on, 0 off

3) Callback:

RingManager.registerProcess(ReceiveType.GreenOrIr,(data){})

Register raw data callback

Parameter：

- ReceiveType.GreenOrIr
- (data){}:callback function

Return：

- data[“irOrGreen”]:PPG raw data (Infrared or Green)

  When call SendBle (SendType. openSingleHealth), it is Green Light raw data;

  When call SendBle (SendType. Health), it is Infrared Light raw data;

- data[“redOrGreen”]: PPG raw data (Red)

  When call SendBle (SendType. openSingleHealth),it will be empty.

  When callSendBle (SendType. Health), it is RedLight raw data;

Note: You will receive 8 raw data (Infrared,Red or Green) at a time, which are signed numbers and range from -32768 to 32767.

4) Sample Code

```
ringManager.registerProcess(ReceiveType.GreenOrIr, (data) {
  irWaveList.addAll(data["irOrGreen"]);
  redWaveList.addAll(data["redOrGreen"]);
  update.value = !update.value;
  if (irWaveList.length > 600) {
    irWaveList.removeRange(0, 8);
  }
  if (redWaveList.length > 600) {
    redWaveList.removeRange(0, 8);
  }
});
```

2.3.25. Blood oxygen measurement settings

Support SR09 project.

1) Function:

sendBle(SendType.oxSetting,{ “switch”, timeInterval”})

The ring firmware version need 2.1.6 or later.

2) Parameter:

- timeInterval：Blood oxygen measurement interval, range of 5-360 minutes (measured between 0:00am and 6:00am)
- switch: Blood oxygen measurement switch, 1 on, 0 off

  Note: SR09 is enabled by default,it will measure SPO2 at 2:00am and 4:00am

3) Callback:

None

4) Sample Code

None

2.3.26. Get device information – Part5

Support SR09 and SR23 projects.

1) Function:

sendBle(SendType.deviceInfo5)

The ring firmware version need 2.1.6 or later.

2) Parameter：

- SendType.deviceInfo1：Send Type

3) Callback:

RingManager.registerProcess(ReceiveType.DeviceInfo5,(data){})

Registration device information 5 data callback

Parameter：

- ReceiveType. DeviceInfo5
- (data){}:callback function

Return：

- data[“hrMeasurementTime”]:Heart rate measurement time, in seconds
- data[“oxMeasurementInterval”]: Blood oxygen measurement interval, in minutes
- data[“oxMeasurementSwitch”]: Blood oxygen measurement switch, 0 off, 1 on

4) Sample Code：

```
ringManager.registerProcess(ReceiveType.DeviceInfo5, (Map data) {
  if (data.isNotEmpty) {
    hrMeasurementTime.value = data["hrMeasurementTime"];
    oxMeasurementInterval.value = data["oxMeasurementInterval"];
    oxMeasurementSettingSwitch.value = data["oxMeasurementSettingSwitch"];
  }
});
```

2.3.27. OTA

Support SR09, SR23 and SR28 projects.

OTA function calls are encapsulated and then ota_ In the file of data.dart.You can choose between local firmware upgrade or network download firmware upgrade. For network upgrade, first select the firmware you want to download. Optional firmware options include FOIRing03, sr23, sr26, sr09 (using wireless charging), sr09n (using NFC charging)

Note:

Please select the correct firmware for upgrading, you check the charging mode from the device information:

- The charging mode is Common Wireless Charging, please select firmware named as SR09W.
- The charging mode is NFC Wireless Charging, please select firmware named as SR09N.

If the wrong OTA firmware upgrade is selected, it will result in the device not functioning properly.

2.3.28. Check OEM certification

Support SR09, SR23 and SR28 projects.

1) Function:

bleData.sendBle(SendType.deviceInfo1)

2) Parameter:

SendType.deviceInfo1: Send Type

3) Callback:

RingManager.registerProcess(ReceiveType.OEMResult,(data){})

Registration OEM authentication result callback.

Parameter:

- ReceiveType.OEMResult
- (data){}:callback function

Return:

- data:

  0: Verification fail

  1: Verification successful

4) Sample Code

```
void oemVerify() {
  ringManager.startOEMVerify((cmd, [data]) {
    print(" startOEMVerify cmd=$cmd data=$data");
    var sendData = data;
    if (SendType.startOEMVerifyR2 == cmd) {
      sendData = nexring_plugin.aes128_decrypt(data["sn"], data["txt"]);
    }
    sendBle(cmd, sendData);
  });
}
```

```
ringManager.registerProcess(ReceiveType.OEMResult, (data) {
  oemResult.value = data ? "Verification successful" : "Verification fail";
});
```

Note:

The OEM authentication will be processed in the SDK, and the upper layer application does not need to handle it. This interface is only used for application layer queries.

Before conducting OEM authentication, it is necessary to obtain whether the switch in the device information part1 data is turned on.If the OEM authentication switch is turned on without OEM authentication, most of the Ring command sending functions will not be able to be used.

2.3.29. ECG

Support SR23 and SR28 projects.

1) Function:

bleData.sendBle(SendType.setEcg,{ "samplingRate" ," switch" ," dispSrc" })

2) Parameter:

- SendType.setEcg: Send Type

- samplingRate: The default sampling rate is 512Hz, using parameters .For example: ECG_PPG_SAMPLE_RATE.ECG_PPG_SMAPLE_RATE_512
  When the clock frequency is 0, the sampling rate is 500Hz, and when the clock frequency is 1, the sampling rate is 512Hz
- switch:
  0: Turn offECG detection;
  1: Turn on ECG detection
- clockFrequency:
  0: The clock frequency is 32000 Hz
  1: The clock frequency is 32768 Hz
- dispSrc:
  0: Raw data; support SR23 and SR28 projects
  1: Output data through algorithm library calculation; support SR23 project
  2: Raw data and output data calculated by ring algorithm library same time support SR23 project

3) Callback

a) **When dispSrc=0** support SR23 and SR28 projects

RingManager.registerProcess(ReceiveType.EcgRaw,(data){})

Register ECG raw data callback

Parameter:

- ReceiveType.EcgRaw
- (data){}:callback function

Return:

- data[“ecgList”]:ECG raw data array, array length 5, value range -131072~131071
- data[“dataCount”]:The number of data output by ECG

Note:

If you use the ECG algorithm of the server, please send the ECG raw data output by the ring to the server. The ECG algorithm of the server requires at least **40 seconds** of **valid**ECG raw datawith a sampling rate of **500Hz** to be uploaded, otherwise the ECG algorithm will not be able to output ECG measurement results.

b) **When dispSrc=1, the ECG data is calculated by the ring algorithm library and output as a callback function** support SR23 project

RingManager.registerProcess(ReceiveType.EcgAlgorithm,(data){})

Register waveform data callback after device algorithm

Parameter:

- ReceiveType.EcgAlgorithm
- (data){}:callback function

Return:

- data[“ecgList”]:ECG raw data array, array length 7, value range-32768~32767
- data[“dataCount”]:The number of data output by ECG

RingManager.registerProcess(ReceiveType.EcgAlgorithmResult,(data){})

Callback of result data after device algorithm registration

Parameter:

- ReceiveType.EcgAlgorithmResult
- (data){}:callback function

Return:

- data[“heartRate”]:heart rate

  0: Invalid;

  1: Below 40;

  254: Above 200

- data[“resultOfArrhythmia”]:Arrhythmia detection

  0: Arrhythmic examination not completed with no results.

  1: The arrhythmia examination has been completed, and no abnormal events have been found

  2: Arrhythmic examination completed, no good enough electrocardiogram collected to make any decision

  3: Arrhythmic examination completed and bradycardia detected.

  4: Arrhythmia detection completed, atrial fibrillation detected.

  5: Arrhythmic examination completed, detected tachycardia

  6: Arrhythmic examination completed with abnormalities. However, bradycardia, atrial fibrillation, and tachycardia cannot be confirmed

  When data ["resultOfArrhythmia"]=2, one of the following four reasons will be output:

  - data ["low_amplitude"]:

    1: The amplitude of the electrocardiogram is very low. Please ensure that the contact surface is clean and try again.

    0: None

  - data[“significant_noise”]:

    1: There is obvious noise in the electrocardiogram signal. Please ensure that the device is not held too tightly and try again.

    0: None

  - data[“unstable_signal”]:

    1: The electrocardiogram signal is unstable. Please stay still and try again.

    0: None

  - data[“not_enough_data”]:

1: Not enough data, please make sure you stay stationary and try again.

0: None

- data[“rmssd”]:rmssd data
- data[“sdnn”]:sdnn data
- data[“pressureIndex”]: Pressure index: effective value 0-100, 0xFF indicates invalid
- data[“bmr”]: Basic metabolic rate, unit: calories per minute
- data[“active_cal”]: Active energy consumption, unit: calories per minute
- data[“signalQuality”]:

  0: The electrocardiogram signal does not exist. The user is not connected to the ECG device.

  1: Changing the status indicates that the user is connected to the electrocardiogram device, but the electrocardiogram signal quality is poor and the heartbeat cannot be determined. This may be due to excessive noise on the electrocardiogram signal. No electrocardiogram algorithm can run at this level of signal quality.

  2: This state indicates that the heartbeat can be determined in the electrocardiogram signal, but a large amount of noise has been detected in the signal. User verification/identification cannot be performed at this level of signal quality.

  3: This state indicates that the heartbeat can be determined in the electrocardiogram signal, and the signal is clear enough without noise interference, suitable for the operation of all electrocardiogram algorithms.

- data[“present”]:

  1: Users connect to electrocardiogram devices;

  0: User not connected to electrocardiogram device

- data[“alive”]:

  1: Detected user heartbeat;

  0: No user heartbeat detected

c) **When dispSrc=2**, support SR23 project

It can get the raw data by

```
RingManager.registerProcess(ReceiveType.EcgRaw,(data){})
```

And

It can get the results from ring algorithm by

```
RingManager.registerProcess(ReceiveType.EcgAlgorithm,(data){})
```

If you need to enable the software ECG algorithm, please initiate ECG raw data detection, and then pass the ECG raw data into channel. send (data ["ecgList"]) through RingManager. registerProcess (ReceiveType. EcgRaw, (data) {})

The following is the callback of the raw data after being processed by the software ECG algorithm:

a) channel.setMessageHandler((message){})

Return:

- message[“type”]:type

  "Wave": waveform data

- message ["data"]: ECG waveform data, type of shaping, range: -32768~32767

"HR": Heart rate

"Mood Index": Mood Index

- 1-20 Calm down
- 21-40 Relax
- 41-60 balance
- 61-80 incentives
- 81-100 excitement/anxiety/excitement

"RR": peak to peak value

“HRV”: Heart rate variability

“RESPIRATORY RATE”:Respiratory Rate

b) Lead on/off detect(Finger Dectections) support SR23 and SR28 projects.

RingManager.registerProcess(ReceiveType.EcgFingerDetect,(data){})

Register ECG finger detection callback function

Parameter:

- ReceiveType.EcgFingerDetect
- (data){}:callback function

Return:

- data[“fingerDetect”]:

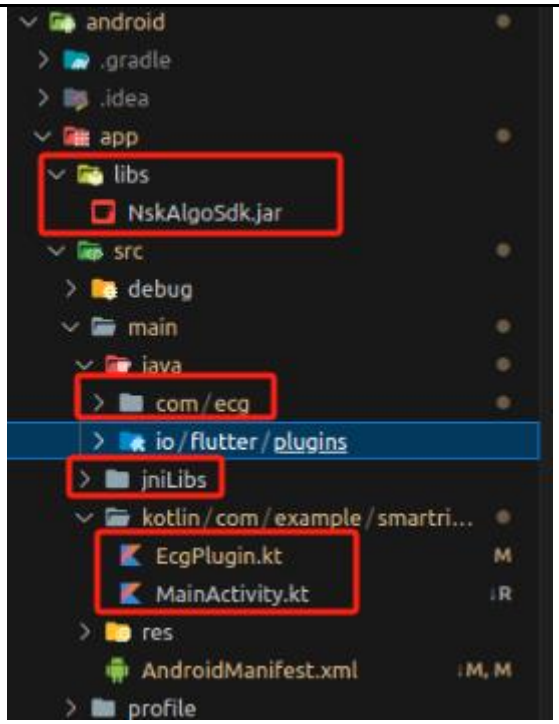
  0: No finger contact detected with the ring;

  1: Detected finger contact with ring

4) Project Compilation Configuration

a) Android

You need to create a libs folder in Android/app, Then add NskAlgoSdk.jar from the android/app/libs directory of the smartstring_flutter project to the libs folder, place the com/egg folder in the Android/app/src/main/Java directory, create a jniLibs folder in the Android/app/src/main/Java directory, add the libNskAlgo.so library to the jniLibs folder, and add the EcgPlugin.kt file to the same folder as MainActivity.kt, Add EcgPlugin (this, FlutterEngine. dartExecutor. binaryMessenger) to the MainActivity.kt file, and add a link to the jniLibs directory and the NskAlgoSdk. jar package in the build. gradle file

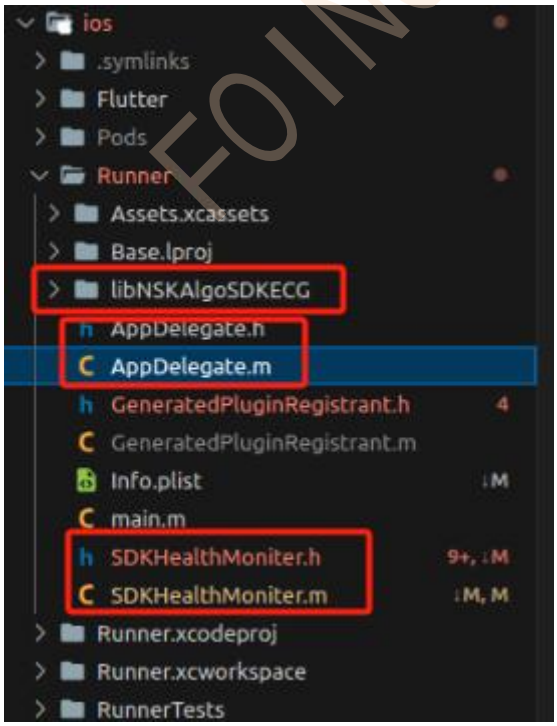


b) iOS

Add the SDKHealthMonitor. h, SDKHealthMonitor. m, and libNSKAlgoSDKECG folders from the ios/Runner directory of the smartstring_flutter project to the ios directory, add the sdkHc03 variable in AppDelegate. h, and initialize SDKHealthMoniter in AppDelegate.

Note that. a static library needs to be added in Target ->build phases ->Link Binary With Libraries in xcode.

Then Target Build Setting Other Link Flags add all_ Load (. a library in libNSKAlgoSDECG).



c) Flutter

Add var channel=const BasicMessage Channel ("ecgMessage Channel", StandardMessage Codec()) to the code;

Receive data from Native:

channel.setMessageHandler((message){})

Send data to Native:

channel.send()

2.3.30. Common Callback of Issue Instructions

Support SR23 and SR28 projects

When the APP sends a command to Ring, Ring returns the execution status of the command through this callback. This callback should be registered during initialization so that it can take effect when the first command is sent. Note that the returned result only indicates whether the command will be accepted by Ring, and does not indicate that a specific result will be returned. Ring only confirms receipt of the command. For example, when sending the command SendBle (SendType. deviceInfo1), it will return 0 through this callback, and the actual result is in RingManager RegisterProcess (ReceiveType. DeviceInfo1, (data) {}) returns; If the return is a situation other than 0, such as 1 or 2, it means that Ring did not execute the command, in RingManager RegisterProcess (ReceiveType. DeviceInfo1, (data) {}) will not have a result in this callback.

RingManager.registerProcess(ReceiveType.RePackage,(data){})

- Cmd: Issuing command instructions;
- Result: The result of issuing the command;
- Reason: Failure reason;

```
ringManager.registerProcess(ReceiveType.RePackage, (data) {
  print(
      "cmd=${data['cmd']}  result=${data['result']}  reason=${data['reason']} ");
  repackage.value =
      "cmd:${data['cmd']} result:${data['result']} reason:${data['reason']} ";
});
```

2.3.31. Temperature fluctuations

Support SR09 and SR23 projects.

Due to the involvement of business logic and database factors, the implementation of temperature fluctuation data is only implemented in the demo. The database part can refer to the demo database or directly use this database based on the actual database used. Before calling this function, historical records should be obtained to ensure sleep data is available.

The following is the implementation code in the demo:

```
Future<void> getTemperature() async {
  final temperatureFluctuateData =
      await healthModel.getSleepTemperatureFluctuateData(
          DateTime.now().subtract(const Duration(days: 30)));

  if (temperatureFluctuateData != null) {
    //   var ftcW = temperatureFluctuateData["ftcW"];
    // debugPrint('temperatureFluctuateData["ftcW"]=${ftcW}');
    ftcW.value = (temperatureFluctuateData["ftcW"] * 100).round() / 100;
    tempArr.value = temperatureFluctuateData["temperatureArray"];
    ftcBase.value = temperatureFluctuateData["ftcBase"];
  }
}
```

1) **Function:**

   getSleepTemperatureFluctuateData(DateTime.now())

   Temperature fluctuation function

2) **Parameter:**

   DateTime: The date you need to fill in to obtain temperature fluctuations. In the demo, DateTime. now() is used to obtain the temperature fluctuation data for that day.

3) **Return value:**

   - FtcW: is the temperature fluctuation value(When the temperature fluctuation value is outside the range of -0.8-0.8 ° C, it is judged as fever)
   - TemperatureArray: The temperature difference (temperature value - temperature baseline value) that fluctuates during sleep is used to draw a waveform graph.
   - FtcBase: is the baseline value of temperature fluctuations

2.3.32. GetStressData

Support SR09 and SR23 projects.

Due to the involvement of business logic and database factors, the implementation of stress data is only implemented in the demo. The database part can refer to the demo database or directly use this database based on the actual database used. Before calling this function, historical records should be obtained to ensure that there is sleep data. At least 5 days of stress value (5 days exceeding 3 hours of sleep data) are required for the first 14 days to have stress data.

The following is the implementation code in the demo:

```
Future<void> getPressure() async {
  final pressureBaseLineData =
      await healthModel.getPressureBaseLine(DateTime.now());
  if (0 < pressureBaseLineData["downCount"] &&
      pressureBaseLineData["downCount"] < 5) {
    stressDays.value = "还有${pressureBaseLineData["downCount"]}天提供压力数据";
  } else {
    stressDays.value = "";
  }
  var baseLine = pressureBaseLineData["baseLine"];
  if (baseLine > 0) {
    await healthModel.storePressureZone();
    pressureArray.value = {};
    motionArray.value = [];
    healthModel.getPressureDataByDate(DateTime.now()).then((array) {
      for (var element in array) {
        pressureArray.addAll(element.allZoneList);
        motionArray.addAll(element.allMotionList);

        pressureBaseLine.value = element.pressureBaseLine;
      }
    });
  }
}
```

1. **Base Line**

   1) **Function:**

   getPressureBaseLine(DateTime.now())

   Get stress baseline

   2) **Parameter**

   DateTime: You need to fill in the date to obtain the pressure baseline, and in the demo, DateTime.now() is used to obtain the pressure baseline data for that day.

   3) **Return**

   - downCount:How many days are we still missing to obtain stress data
   - baseLine:Pressure baseline value.

2. **Pressure data**

   1) **Function:**

   getPressureDataByDate (DateTime.now())

   Pressure data function

   2) **Parameter**

   DateTime: The date you need to fill in to obtain stress data. In the demo, DateTime. now() is used to obtain the stress data for the current day.

   3) **Return**

   - allZoneList:Pressure data is used to draw wave pressure graphs
   - allMontionList: Motion data is used to draw motion waveform graphs

2.3.33. Clear historical data of new algorithms

Support SR28 project.

1) **Function:**

bleData.sendBle(SendType.cleanNewHistoryData)

2) Parameter:

SendType._cleanNewHistoryData: Send Type

3) Callback:

None

4) Sample Code

None

2.3.34. User Information

Support SR28 project.

1) Function:

bleData.sendBle(SendType.userInfo,{ "height", "weight", "age", "function", "sex" })

2) Parameter:

- SendType.userInfo: Send Type
- function:

  0:get

  1:set
- sex:

  0:male

  1:female
- age:age, range 1-115
- height: Height, unit mm, range 1200-3000
- weight, unit kg, range 30-200

3) Callback:

RingManager.registerProcess(ReceiveType.USER_INFO,(data){})

Registration user information result callback.

Parameter:

- ReceiveType.USER_INFO
- (data){}:callback function

Return:

- sex:

  0:male

  1:female
- age:age, range 1-115

- height: Height, unit mm, range 1200-3000
- weight, unit kg, range 30-200

4) **Sample Code**

```
//user info callback interface registration function
ringManager.registerProcess(ReceiveType.USER_INFO, (data) {
  userInfoValue.assignAll(data);
  debugPrint("userInfoValue=$userInfoValue");
});
```

2.3.35. Get New Historical Records

Support SR28 project.

Before requesting new algorithm historical data, please first request the total entries of the new algorithm historical data (2.5.36)

1) **Function:**

bleData.sendBle(SendType.setNewAlgorithmHistory)

2) **Parameter:**

- SendType._setNewAlgorithmHistory: Send Type

3) **Callback:**

(1) RingManager.registerProcess(ReceiveType.NEW_ALGORITHM_HISTORY,(data){})

Register the callback function for vital signs historical data.

Parameter:

- ReceiveType.NEW_ALGORITHM_HISTORY
- (data){}:callback function

Return:

- timeStamp：time stamp.
- uuid: Record unique UUID, starting from 1 and accumulating, invalid value is 0xFFFFFF.
- ibi: interbeatinterval (IBI)value,invalid value is 0xFFFF, unit: ms
- stress: Stress value, actual value range 0.0-1.0, invalid value is 0xFF.

  - Stress value is 0.0-0.1, indicating high stress, actual output value 0.0 is invalid value.
  - Stress value is 0.1-0.5, indicating medium stress.
  - Stress value is 0.5-1.0, indicating no stres.
- cardiac_coherence: Cardiac coherence value,the value range is 0.00-1.00, invalid value is 0xFF.

  - Cardiac coherence value is 0-0.03, indicating high pressure, and the actual output value of 0 is an invalid value.
  - Cardiac coherence value is 0.03-0.1, indicating medium pressure.
  - Cardiac coherence value is 0.1-1.00, indicating no pressure.
- hrv: Output the RMSSD value of HRV, invalid value is 0xFFFF.
- respiratory_rate: Respiratory rate value, invalid value is 0xFF.

- heart_rate: Heart rate value, the heart rate value measured, range: 0-200bpm, invalid value is 0xFF.

Note: The new historical data is recorded and stored in the device according to the set measurement interval time, and a maximum of 7x24 hours of measurement records are stored. Measurement records exceeding 168 hours directly overwrite the oldest measurement records, unless the device is restored to factory settings, or the App sends a clear command to clear all measurement records.

(2) RingManager.registerProcess(ReceiveType. EXCLUDED_SWIMMING_ACTIVITY_HISTORY, (data){})

Activity detection data callback function for activities other than swimming

Parameter:

- ReceiveType.EXCLUDED_SWIMMING_ACTIVITY_HISTORY
- (data){}:callback function

Return:

- timeStamp：time stamp
- uuid: Record unique UUID, starting from 1 and accumulating, invalid value is 0xFFFFFF
- distance:distance within 5 minutes. unit: Km.
- step:steps in 5 minutes.
- total_energy:total calories burned in 5 minutes, including basal metabolic rate, unit: Kcal.
- total_active_energy:total active calories burned in 5 minutes, excluding basal metabolic rate, unit: Kcal.
- vo2:oxygen uptake, range 0-255, unit: ml/kg/min
- vo2Max:VO2 Max, range 0-255, unit: ml/kg/min
- active_type:

  0: No activity or very mild activity

  1: Rhythmic and non rhythmic activities that cannot be classified under other categories

  2: Walking

  3: Running

  4: Swimming, when the activity type is detected as swimming, switch to storing swimming activity data

  5: Other rhythmic activities
- exercise_type:

  0: No continuous exercise

  1: Continuous exercise

(3) RingManager.registerProcess(ReceiveType. EXERCISE_ACTIVITY_HISTORY,(data){})

The stored activity data callback during workout.

Parameter:

- ReceiveType.EXERCISE_ACTIVITY_HISTORY
- (data){}:callback function

Return:

- timeStamp：time stamp
- step: steps within the storage interval time (e.g. 10 seconds)
- distance: distance within the storage interval time (e.g. 10 seconds), in Kcal
- speed: the speed during workout, with an output unit of 0.1m/s and a range of 0-200
- step_frequency:cadence, unit: SPM
- total_energy: total calories burnedwithin a storage interval time (e.g. 10 seconds), including basal metabolic rate, in Kcal
- total_active_energy: total activity calories burned within a storage interval time (e.g. 10 seconds), excluding basal metabolic rate, unit: Kcal
- current_energy_consumed: current energy consumption within a storage interval time (e.g. 10 seconds), unit: MET
- exercise_type: workout types, 0x00 Other, 0x01 Running, 0x02 Walking, 0x03 Swimming Pool, 0x04 Open Water Swimming, 0x05 Indoor Cycling, 0x06 Outdoor Cycling, 0x07 Yoga, 0x08 Mindfulness
- heart_rate: Heart rate value, range 0-200, unit: bpm

(4) RingManager.registerProcess(ReceiveType. EXERCISE_VITAL_SIGNS_HISTORY,(data){})

The stored vital signdata callback during workout.

Parameter:

- ReceiveType.EXERCISE_VITAL_SIGNS_HISTORY
- (data){}:callback function

Return:

- timeStamp：time stamp
- heart_rate: Heart rate output within the storage interval time, invalid value is 0xFF, range: 0-200, unit: bpm.
- hrv: Output the RMSSD value of HRV within the storage interval time, invalid value is 0xFFFF
- respiratory_rate: Respiratory rate valuewithin the storage interval time, invalid value is 0xFF, range: 0-50
- ibi: interbeat interval value within the storage interval time, invalid value is 0xFFFF, unit: ms, range: 0-65535
- stress: stress value within the storage interval time (e.g. 10 seconds), invalid value is 0xFF
- cardiac_coherence: cardiac coherence value within a storage interval (e.g. 10 seconds), wherethe cardiac coherence value of 0xFF is not measured.
- vo2: Oxygen uptake within the storage interval time (e.g. 10 s), range 0-255, unit: ml/kg/min, no VO2 value detected is 0xFF.
- vo2Max: VO2 Max output within the storage interval time (e.g. 10 seconds), range 0-255, unit: ml/kg/min, no VO2 Max value detected is 0xFF.

- temperature: finger temperature value, invalid value is 0xFF, measured when the activity type is mindfulness, not measured for other activity types.
- exercise_type: workout types, same workout type as selected in the App.
  0x00 Other, 0x01 Running, 0x02 Walking, 0x03 Swimming Pool, 0x04 Open Water Swimming, 0x05 Indoor Cycling, 0x06 Outdoor Cycling, 0x07 Yoga, 0x08 Mindfulness
- active_type: Output by the ring algorithm.
  0: No activity or very mild activity
  1: Rhythmic and non rhythmic activities that cannot be classified under other categories
  2: I'm walking
  3: Running
  4: Swimming, when the activity type is detected as swimming, switch to storing swimming activity data
  5: Other rhythmic activities
- exercise_status: Exercise status, output by the ring algorithm.
  0-No continuous exercise, 1-Continuous exercise

(5) RingManager.registerProcess(ReceiveType. SWIMMING_EXERCISE_HISTORY,(data){})

The stored activity data callback during swimming workout.

Parameter:

- ReceiveType.SWIMMING_EXERCISE_HISTORY
- (data){}:callback function

Return:

- timeStamp：time stamp
- total_stroke_count: Total number of strokes, cumulative swimming strokes for all laps
- total_stroke_time: Total swimming time, accumulated swimming time since the last reset, excluding turning, stopping, and pausing. Unit in seconds, range 0-65535
- total_distance: Total swimming distance, cumulative swimming distance since the last reset, unit: meters, range 0-20000
- swimming_pace: Swimming pace, swimming pace for all laps, in minutes per kilometer
- swimming_laps: Swimming laps, swimming laps since last reset
- average_swimming_efficiency: Average swimming efficiency, the average swimming efficiency of all laps (SWOLF). SWOLF is the sum of the time per lap (in seconds) and the number of strokes per lap

(6) RingManager.registerProcess(ReceiveType. SINGLE_LAP_SWIMMING_HISTORY,(data){})

The stored lab activity data callback during swimming workout.

Parameter:

- ReceiveType.SINGLE_LAP_SWIMMING_HISTORY
- (data){}:callback function

Return:

- timeStamp：time stamp
- stroke_count: Number of strokes per lap, swimming strokes per lap
- swimming_time: Single lap swimming time, swimming time per lap, unit s, range 0-65535
- stroke_rate: Single circle stroke rate, stroke rate per circle, unit: times/min, range 0-180
- swimming_posture: Single lap swimming posture, swimming posture for each lap, 0-unknown type, 1-crawl swimming style, 2-breaststroke, 3-backstroke, 4-butterfly
- swimming_pace: Single lap swimming pace, swimming pace per lap, in minutes per kilometer
- swimming_efficiency: Swimming efficiency(SWOLF), sum of time per lap (in seconds) and number of strokes per lap.
- circle_detection: Circle detection status, swimming circle detection indicator
- swimming_distance: Single lap swimming distance, unit: meters

(7) RingManager.registerProcess(ReceiveType. SLEEP_HISTORY,(data){})

ACC Sleep detection data callback function.

Parameter:

- ReceiveType.SLEEP_HISTORY
- (data){}:callback function

Return:

- timeStamp：stored timestamp of sleep events.
- uuid: Record unique UUID, starting from 1 and accumulating, invalid data: 0xFFFFFF
- timestamp_type: 0-sleep, 1-wake-up
- sleep_timeStamp:sleep time stamp or wake time stamp.
- bed_time: time in bed, unit: minute, range 0-64800
- wake_index: Awakening sequence, range 0-25

We need to input the data of (2.3.17 Old Historical Data) and timestamp_type, sleep_timeStamp, bed_time, and wake_index into the new sleep algorithm (not yet provided) to calculate the sleep data

(8) RingManager.registerProcess(ReceiveType. DAILY_ACTIVITY_HISTORY,(data){})

Daily activity data callback function.

Parameter:

- ReceiveType.DAILY_ACTIVITY_HISTORY
- (data){}:callback function

Return:

- year: year
- month: month
- day:day
- total_walk_steps: total walking steps: range 0-65535
- total_run_steps: total running steps: range 0-65535

- total_other_steps: total other steps: range 0-65535
- total_distance: total distance, unit: Km
- total_energy: total calories burned, including basal metabolic rate, unit: Kcal
- total_active_energy: total activity calories burned, excluding basal metabolic rate, unit: Kcal
- current_energy_consumed: Current energy consumption, unit: MET

(9) RingManager.registerProcess(ReceiveType. TEMPERATURE_HISTORY,(data){})

Sleep skin temperature history data callback function.

Parameter:

- ReceiveType.TEMPERATURE_HISTORY
- (data){}:callback function

Return:

- timeStamp：time stamp
- temperature1: temperature
- temperature2: temperature
- temperature3: temperature
- temperature4: temperature
- temperature5: temperature

  Note: The temperature history data is stored every 5 minutes, with a timestamp of temperature 5. Temperature 4 time=Temperature 5 time -60 seconds, Temperature 3 time=Temperature 5-120 seconds, Temperature 2 time=Temperature 5-180 seconds, and Temperature 1 time=Temperature 5-240 seconds The temperature history data is only measured and stored after the ring detects the start of sleep and stops measuring after sleep ends. Not stored at other times

(10) RingManager.registerProcess(ReceiveType. STEP_TEMPERATURE_ACTIVITY_INTENSITY_HISTORY,(data){})

Step count/body temperature/activity intensity historical data callback interface function

Parameter:

- ReceiveType.STEP_TEMPERATURE_ACTIVITY_INTENSITY_HISTORY
- (data){}:callback function

Return:

- timeStamp：time stamp
- uuid: Record unique UUID, starting from 1 and accumulating, invalid data: 0xFFFFFF
- step: Step count, the sum of the walking and running steps output by the new algorithm, unsigned shaping, value range 0-65535
- temperature: finger temperature value, 0x00 indicates that the measured temperature is below 20 ° C. 0xFF represents invalid data.
- activity_intensity: Activity intensity, range: 1-4

-1. Sitting still for extended periods of time

-2. Low activity intensity

-3. Moderate activity intensity

-4. High activity intensity

- acc_sd: Using the maximum acceleration of 4G, the X, Y, and Z accelerometer values are Around 36863-36862, representing 2G to -2G, monitor the ACC target for 42 seconds at each measurement interval (2 minutes)Accurate deviation

4) Sample

Code

```
//NEW_ALGORITHM_HISTORY
ringManager.registerProcess(ReceiveType.NEW_ALGORITHM_HISTORY, (data) {
  debugPrint(
      "NEW_ALGORITHM_HISTORY data=$data historyCount=$historyCount  numUUID=${numUUID.value}  ");
  historyCount += 1;
  newHistoryProcess.value =
      "${data["uuid"] - startUUID.value + 1}/${numUUID.value}";
  newHistoryData.add(jsonEncode(data));
  if (historyCount == numUUID.value) {
    newHistoryReceiveFinish();
  }
});
```

```
//EXCLUDED_SWIMMING_ACTIVITY_HISTORY
ringManager.registerProcess(ReceiveType.EXCLUDED_SWIMMING_ACTIVITY_HISTORY,
    (data) {
  debugPrint("EXCLUDED_SWIMMING_ACTIVITY_HISTORY data=$data ");
  historyCount += 1;
  excludedSwimmingActivityHistoryData.add(jsonEncode(data));
  if (historyCount == numUUID.value) {
    newHistoryReceiveFinish();
  }
});

//EXERCISE_ACTIVITY_HISTORY
ringManager.registerProcess(ReceiveType.EXERCISE_ACTIVITY_HISTORY, (data) {
  debugPrint("EXERCISE_ACTIVITY_HISTORY data=$data ");
  historyCount += 1;
  exerciseActivityHistoryData.add(jsonEncode(data));
  if (historyCount == numUUID.value) {
    newHistoryReceiveFinish();
  }
});

//SWIMMING_EXERCISE_HISTORY
ringManager.registerProcess(ReceiveType.SWIMMING_EXERCISE_HISTORY, (data) {
  debugPrint("SWIMMING_EXERCISE_HISTORY data=$data ");
  historyCount += 1;
  swimmingExerciseHistoryData.add(jsonEncode(data));
  if (historyCount == numUUID.value) {
    newHistoryReceiveFinish();
  }
});

//SINGLE_LAP_SWIMMING_HISTORY
ringManager.registerProcess(ReceiveType.SINGLE_LAP_SWIMMING_HISTORY,
    (data) {
  debugPrint("SINGLE_LAP_SWIMMING_HISTORY data=$data ");
  historyCount += 1;
  singleLapSwimmingHistoryData.add(jsonEncode(data));
  if (historyCount == numUUID.value) {
    newHistoryReceiveFinish();
  }
});
```

```
//STEP_TEMPERATURE_ACTIVITY_INTENSITY_HISTORY
ringManager.registerProcess(
    ReceiveType.STEP_TEMPERATURE_ACTIVITY_INTENSITY_HISTORY, (data) {
  debugPrint("STEP_TEMPERATURE_ACTIVITY_INTENSITY_HISTORY data=$data ");
  historyCount += 1;
  stepTemperatureActivityIntensityHistoryData.add(jsonEncode(data));
  if (historyCount == numUUID.value) {
    newHistoryReceiveFinish();
  }
});

//SLEEP_HISTORY
ringManager.registerProcess(ReceiveType.SLEEP_HISTORY, (data) {
  debugPrint("SLEEP_HISTORY data=$data ");
  historyCount += 1;
  sleepHistoryData.add(jsonEncode(data));
  newAlgorithmHistoryData.add({
    "ts": data["sleep_timeStamp"],
    "type": data["timeStamp_type"],
    "bed_rest_duration": data["bed_time"],
    "awake_order": data["wake_index"]
  });
  if (historyCount == numUUID.value) {
    newHistoryReceiveFinish();
  }
});

//DAILY_ACTIVITY_HISTORY
ringManager.registerProcess(ReceiveType.DAILY_ACTIVITY_HISTORY, (data) {
  debugPrint("DAILY_ACTIVITY_HISTORY data=$data ");
  historyCount += 1;
  dailyActivityHistoryData.add(jsonEncode(data));
  if (historyCount == numUUID.value) {
    newHistoryReceiveFinish();
  }
});
//EXERCISE_VITAL_SIGNS_HISTORY
ringManager.registerProcess(ReceiveType.EXERCISE_VITAL_SIGNS_HISTORY,
    (data) {
  debugPrint("EXERCISE_VITAL_SIGNS_HISTORY data=$data ");
  historyCount += 1;
  exerciseVitalSignsHistoryData.add(jsonEncode(data));
  if (historyCount == numUUID.value) {
    newHistoryReceiveFinish();
  }
});

//TEMPERATURE_HISTORY
ringManager.registerProcess(ReceiveType.TEMPERATURE_HISTORY, (data) {
  debugPrint("TEMPERATURE_HISTORY data=$data ");
  historyCount += 1;
  temperatureHistoryData.add(jsonEncode(data));
  if (historyCount == numUUID.value) {
    newHistoryReceiveFinish();
  }
});
```

2.3.36. Total entries for obtaining new historical data

Support SR28 project.

1) Function:

bleData.sendBle(SendType.setNewAlgorithmHistoryNum)

2) Parameter:

- SendType.setNewAlgorithmHistoryNum: Send Type

3) Callback:

RingManager.registerProcess(ReceiveType.NEW_ALGORITHM_HISTORY_NUM,(data){})

Parameter:

- ReceiveType.NEW_ALGORITHM_HISTORY_NUM
- (data){}:callback function

Return:

- num：Number of historical data
- minUUID：Minimum UUID value
- maxUUID：Maximum UUID value

4) Sample Code

```
//NEW_ALGORITHM_HISTORY_NUM
ringManager.registerProcess(ReceiveType.NEW_ALGORITHM_HISTORY_NUM, (data) {
  if (data["num"] == 0) {}
  endUUID.value = data["maxUUID"];
  startUUID.value = data["minUUID"];
  numUUID.value = data["num"];
  historyCount = 0;
  newAlgorithmHistory.value = [];
  newHistoryShow.value = false;
  newHistoryData = [];
  temperatureHistoryData = [];
  excludedSwimmingActivityHistoryData = [];
  dailyActivityHistoryData = [];
  exerciseActivityHistoryData = [];
  exerciseVitalSignsHistoryData = [];
  swimmingExerciseHistoryData = [];
  singleLapSwimmingHistoryData = [];
  stepTemperatureActivityIntensityHistoryData = [];
  sleepHistoryData = [];
  newAlgorithmHistoryData = [];
});
```

2.3.37. Get Real-time Activity Data

Support SR28 projects

1) Function:

bleData.sendBle(SendType.setActiveData)

2) Parameter:

- SendType.setActiveData: Send Type

3) Callback:

RingManager.registerProcess(ReceiveType.ACTIVE_DATA,(data){})

Parameter:

- ReceiveType.ACTIVE_DATA
- (data){}:callback function

Return:

- year：year
- month: month
- day: day
- total_walk_steps: Total walking steps: range 0-65535
- total_run_steps: Total running steps: range 0-65535
- total_other_steps: Total other steps: range 0-65535
- total_distance: Total distance, unit: Km
- total_energy: Total Calories Burned, including basal metabolic rate, in Kcal
- total_active_energy: Total Activity Calories Burned, excluding basal metabolic rate, unit: Kcal
- current_energy_consumed: Current energy consumption, unit: MET

4) Sample Code

```
//ACTIVE_DATA
ringManager.registerProcess(ReceiveType.ACTIVE_DATA, (data) {
  debugPrint("ACTIVE_DATA data=$data ");
  activeData.value = jsonEncode(data);
});
```

2.3.38. Workout Function

Support SR28 projects

1) Function:

bleData.sendBle(SendType.setExercise,{ “function” , “type” , “poolSize” , “exerciseTime” })

2) Parameter:

- SendType.setExercise: Send Type
- function: start/closeworkout, pause/resumeworkout, 0x00- disable workout, 0x01- enable workout, 0x02- pause, 0x03- resume. When it is a close, pause/resume command, other parameters can be set to 0.
- type: workout types,
  0x00 Other, 0x01 Running, 0x02 Walking, 0x03 Swimming Pool, 0x04 Open Water Swimming, 0x05 Indoor Cycling, 0x06 Outdoor Cycling, 0x07 Yoga, 0x08Mindfulness

- poolSize: Pool size: When the activity type is pool swimming, the pool size needs to be set. 0x00 is unknown, 0x01 is 25m, and 0x02 is 50m (parameters are tentative). For other activity types, the pool size should be set to 0xFF
- exerciseTime: Training time, unit: minutes, range: 1-65535. When the training time is set, the workout will automatically end when the training time is up. If set to 0, it means that workout needs to be manually turned off.

  Note:If the workout type is mindfulness, it is recommended to set the training time. Please do not set the training time to 0.

3) Callback:

None

4) Sample Code

None

2.3.39. Set Up Real-time Workout Data Reporting

Support SR28 projects

1) Function:

bleData.sendBle(SendType.setReportingExercise,{ "on_off" })

2) Parameter:

- SendType.setReportingExercise: Send Type
- on_off:

  0: Stop reporting: When the ring receives the stop reporting command, it immediately stops reporting activity data during the exercise period.

  1: Start reporting: The ring reports activity data during the exercise period according to the set interval for storing exercise data, and also stores activity data according to the set interval for storing exercise data

3) Callback:

RingManager.registerProcess(ReceiveType.GET_REPORTING_EXERCISE,(data){})

Parameter:

- ReceiveType.GET_REPORTING_EXERCISE
- (data){}:callback function

Return:

- step: The sum of walking and running steps from the start of the workout to the current time
- distance: The distance from the start of the workoutto the current time, unit: Kcal
- speed: The current running or walking speed, the output value unit is 0.1m/s, range: 0-200
- step_frequency: The current exercise cadence, unit: SPM
- total_energy: The total calorie burned from the start of the exercise to the current time,

including the basal metabolic rate, unit: Kcal

- total_active_energy: The total active calorie burned from the start of the exercise to the current time, excluding the basal metabolic rate, unit: Kcal
- current_energy_consumed: The output current energy consumption, unit: MET
- exercise_type: workout type, 0x00 Other, 0x01 Running, 0x02 Walking, 0x03 Pool Swimming, 0x04 Open water swimming, 0x05 indoor cycling, 0x06 outdoor cycling, 0x07 yoga, 0x08 mindfulness
- heart_rate: Current workout heart rate value, range 0-200, unit: bpm

4) Sample Code

```
//GET REPORTING EXERCISE        You, 1小时前 • Uncommitted changes
ringManager.registerProcess(ReceiveType.GET_REPORTING_EXERCISE, (data) {
  debugPrint("GET_REPORTING_EXERCISE data=$data ");
  reporting_exercise.value = jsonEncode(data);
});
```

2.3.40. Setting Measurement Time and Measurement Interval Time

Support SR28 project.

1) Function:

bleData.sendBle(SendType.setMeasurementTiming,{"function", "type", "time1", "time1Interval", "time2", "time2Interval", "time3Interval" })

2) Parameter:

- SendType.setMeasurementTiming: Send Type
- function: Set/Query, 1-Set, 0-Query
- type: Heart rate/blood oxygen, 0-heart rate/HRV/respiratory rate/temperature measurement, 1-blood oxygen measurement
- time 1: Measurement Time 1 is the measurement time during the non sleep period, which can be set through the app in seconds with a range of 10-120 seconds Heart rate/HRV/respiratory rate/temperature measurement time, default is 15 seconds Blood oxygen measurement time, default is 30 seconds
- time1Interval: The measurement interval time 1 is the measurement interval time during the non sleep period, which can be set through the app in seconds. The setting range is 0-65535 seconds. If 0, it means no measurement will be taken Heart rate/HRV/respiratory rate/temperature measurement interval, default is 300 seconds Blood oxygen measurement interval, default to 0, indicating no measurement
- time 2: Measurement Time 2 is the measurement time of the sleep period, which can be set through the app with a measurement interval in seconds and a range of 10-120 seconds Heart rate/HRV/respiratory rate/temperature measurement time, default is 30 seconds Blood oxygen measurement time, default is 30 seconds
- time2Interval: The measurement interval time 2 is the measurement interval time of the sleep period, which can be set through the app in seconds. The setting range is 0-65535 seconds. If it is 0, it means no

measurement will be taken Heart rate/HRV/respiratory rate/temperature measurement interval, default is 300 seconds The default interval for blood oxygen measurement is 7200 seconds The sleep start time is the reported sleep start time, and the sleep end time is the reported sleep end time

- time3Inerval: The measurement interval time for step count/body temperature/activity intensity can be set through the app Interval time, in minutes, set range: 0-255 minutes. If 0, no measurement will be taken

3) Callback:

RingManager.registerProcess(ReceiveType.SET_MEASUREMENT_TIMING,(data){})

Parameter:

- ReceiveType.SET_MEASUREMENT_TIMING
- (data){}:callback function

Return:

- type: Heart rate/blood oxygen, 0-heart rate/HRV/respiratory rate/temperature measurement, 1-blood oxygen measurement
- time 1: Measurement Time 1 is the measurement time during the non sleep period, which can be set through the app in seconds with a range of 10-120 seconds Heart rate/HRV/respiratory rate/temperature measurement time, default is 15 seconds Blood oxygen measurement time, default is 30 seconds
- time1Interval: The measurement interval time 1 is the measurement interval time during the non sleep period, which can be set through the app in seconds. The setting range is 0-65535 seconds. If 0, it means no measurement will be taken Heart rate/HRV/respiratory rate/temperature measurement interval, default is 300 seconds Blood oxygen measurement interval, default to 0, indicating no measurement
- time 2: Measurement Time 2 is the measurement time of the sleep period, which can be set through the app with a measurement interval in seconds and a range of 10-120 seconds Heart rate/HRV/respiratory rate/temperature measurement time, default is 30 seconds Blood oxygen measurement time, default is 30 seconds
- time2Interval: The measurement interval time 2 is the measurement interval time of the sleep period, which can be set through the app in seconds. The setting range is 0-65535 seconds. If it is 0, it means no measurement will be taken Heart rate/HRV/respiratory rate/temperature measurement interval, default is 300 seconds The default interval for blood oxygen measurement is 7200 seconds The sleep start time is the reported sleep start time, and the sleep end time is the reported sleep end time
- time3Inerval: The measurement interval time for step count/body temperature/activity intensity can be set through the app Interval time, in minutes, set range: 0-255 minutes. If 0, no measurement will be taken

4) Sample Code


```
//SET_MEASUREMENT_TIMING
ringManager.registerProcess(ReceiveType.SET_MEASUREMENT_TIMING, (data) {
  debugPrint("SET_MEASUREMENT_TIMING data=$data ");
  measureTimingValue.assignAll(data);
});
```

2.3.41. Add ECG cloud computing

Support SR28 projects

1) Function:

cloudCalEcg({key,sn,mac,secret,isSpe,data})

2) Parameter:

- key: Provided by our company
- sn: Equipment SN number
- mac: Device MAC address
- secret: Provided by our company
- isSpe:

  true: Call the SPE interface,

  false: Call the arrhythmia interface
- data: The integer voltage value unit of ECG is uV

  Retrieve the data[“ecgList”] data from the RingManager.registerProcess(ReceiveType.EcgRaw,(data){}) of the 2.3.29 ECG, convert it to uV using the convertToVoltage method, and then combine the voltage data and lead data (isTouch is true) to form a ` data, 1 `. json ('\n') format

3) Callback:

SPE interface:

Return data type:

```
{
  "processedSignal": {
    "processedDataSamples": {
      "processedValues": [
        0.1
      ],
      "leadsOn": [
        true
      ]
    },
    "validRPeakLocs": [
      0
    ],
    "samplingFrequency": 0
  },
  "sigQualityInfo": {
    "sigQualityRatings": [
      {
        "startSample": 0,
        "endSample": 0,
        "durationSamples": 0,
        "startTimestampMs": 0,
        "endTimestampMs": 0,
        "durationTime": 0,
```

```
      "startTimestampUTC": "2019-08-24T14:15:22Z",
      "endTimestampUTC": "2019-08-24T14:15:22Z",
      "sigQualityRating": "UNAVAILABLE"
    }
  ],
  "burdens": {
    "LOW": 30,
    "HIGH": 70
  }
},
"processedFile": "string"
}
```


Arrhythmia interface:

Return data type:


```
{
  "arrhythmiaClassifications": [
    {
      "arrClassification": "UNCLASSIFIED",
      "durationSamples": 0,
      "durationTime": 0,
      "startSample": 0,
      "endSample": 0,
      "startTimestampMs": 0,
      "endTimestampMs": 0,
      "startTimestampUTC": "2019-08-24T14:15:22Z",
      "endTimestampUTC": "2019-08-24T14:15:22Z",
      "heartRateMetrics": {
        "min": 0.1,
        "max": 0.1,
        "avg": 0.1
      },
      "beatCount": 0
    }
  ],
  "beatClusters": [
    {
      "beatLevel": {
        "beatClassification": "Q",
        "beatClusterIndex": 0,
        "rpeakLocationSampleIndices": [
          0
        ],
        "rpeakLocationTimestampsUTC": [
          "2019-08-24T14:15:22Z"
        ],
        "rpeakLocationTimestampsMs": [
          0
        ]
      },
      "beatClusterLevel": {
        "template": [
          0.1
        ],
        "numberOfBeats": 0,
        "numberOfClusters": 0
      }
    }
  ],
  "heartRates": [
    {
      "heartRate": 0,
```



```
      "sampleIndex": 0,
      "timestampMs": 0,
      "timestampUTC": "2019-08-24T14:15:22Z",
      "displayStatus": "DO_NOT_DISPLAY"
    }
  ],
  "rhythmBurdens": {
    "UNCLASSIFIED": 0.1,
    "INCONCLUSIVE": 0.05,
    "NORMAL": 0.5,
    "ATRIAL_FIBRILLATION": 0.1,
    "BRADYCARDIA": 0.05,
    "TACHYCARDIA": 0.1,
    "PAUSE": 0.01,
    "UNREADABLE": 0.01,
    "UNKNOWN": 0.01
  },
  "beatBurdens": {
    "Q": 0.4,
    "N": 0.2,
    "V": 0.1,
    "S": 0.05,
    "F": 0.05,
    "UNKNOWN": 0.2
  },
  "ectopicBeatBurdens": {
    "ISOLATED_PVC": {
      "total": 10,
      "burden": 0.5
    }
  },
  "arrhythmiaAnalysisConfig": {
    "bradycardiaThreshold": 0,
    "tachycardiaThreshold": 0,
    "absolutePauseThreshold": 0.1,
    "relativePauseThreshold": 0
  },
  "heartRateTrend": {
    "min": 0,
    "max": 0,
    "average": 0.1
  }
}
```


4) Sample Code

```
/**
 * Registration functions for callback interfaces of ecgRawData
 */
ringManager.registerProcess(ReceiveType.EcgRaw, (data) async {
  if (isTouch) {
    if (!dataCollectionStart) {
      dataCollectionStart = true; // 初始化开始时间
      ecgDataBuffer = []; // 清空缓冲区
      enoughDate = false; // 标记数据是否足够绘制图表
    }
    if (ecgDataBuffer.length > 20000 && !enoughDate) {
      enoughDate = true;
      stopEcg();
    } else {
      ecgDataBuffer.addAll(data['ecgList'].map<int>((int item) => ringManager.convertToVoltage(item)) );
    }
    await channel.send(data['ecgList']);
  } else {
    dataCollectionStart = false;
    enoughDate = false;
    ecgDataBuffer = [];
  }
});
```

```
void getCloudEcg(bool requestSpe) async {
  String data = formatEcgData(ecgDataBuffer);
  await httpManager.cloudCalEcg(
      key: "2901f7613ac7403e9c5fbc0248b6d94f",
      sn: sn.value.toString(),
      mac: devAddress.value.toLowerCase(),
      secret: "ad58bea443d04368beb847510a37a99d",
      isSpe: requestSpe,
      data: data).then((result){
        debugPrint("getCloudEcg result=$result");
        if(result!=null){
          ecgResponse.value = jsonEncode(result);
        }
      });

}
```


2.3.42. Blood Glucose

Support SR28 project.

1) Function:

(1) registerDevice({sn,age,gender,height,weight,familyHistory,highCholesterol})

(2) sendBle(SendType.setPpg,{ "on_off" })

(3) uploadPpgRecord({fasting,within2HrsMeal,startTime,endTime,ppgData})

(4) getPpgResult()

2) Parameter:

registerDevice

- sn: Equipment SN number
- age:age

- gender: Gender, M represents male, F represents female
- height: Height, in millimeters
- weight: Weight, in kilograms
- familyHistory: Family history of diabetes, 0: none, 1: yes
- highCholesterol: History of cholesterol, 0: None, 1: Yes

sendBle

- on_off: 0: Close, 1: Open

uploadPpgRecord

- fasting: Is it a quick measurement
- within2HrsMeal: Is it within 2 hours of meals
- startTime: Measurement start time
- endTime: Measurement end time
- ppgData:[{ppg:[],timestamp:‘hh:mm:ss’},…], A set of 100 data points for ppg，Timestamp cannot have duplicates

3) Callback:

registerDevice Function return value

- state: 0: Success, 1: Server returned an error, 2: SN is invalid
- lease: The effective lease time point, measured in Unix seconds; If there is no valid SN, the value is negative
- ltk: The URL parameter of the uploadPpgRecord method for network requests

RingManager.registerProcess(ReceiveType.PPG_SET,(data){})

- on_off: 0: Close, 1: Open
- led: 0: Green light, 1: Red light, 2: Infrared
- current: Current, 0-255
- autoAdjBrightness: Auto dimming, 0: off, 1: on
- sps: Sampling frequency: 0:25sps, 1:50sps, 2:100sps

RingManager.registerProcess(ReceiveType.PPG_DATA,(data){})

- ppgList: PPG data, a set of 4 data

uploadPpgRecord Function return value

- state: 0: Success, 1: Invalid lease
- mid: The URL parameter of the network request for the getPpgResult method

getPpgResult Function return value

- state: 0: Successful, 1: SN not registered
- data: The format is as follows

{'data': {'measurement_data': {'completed_progress': True,
'end_time': '2024-09-11 03:13:36',
'fasting': True,
'lower_bound': 4.7,
'measurement_id': '80bcce7e-a89c-415c-ad15-475fdaf97b17',
'model_no': 'SR23',
'start_time': '2024-09-11 03:08:36',
'upper_bound': 6.3,
'user_id': 1000043,
'within_2hrs_meal': False},
'risk_data': {'current_level': 0,
'end_time': '2024-09-11 03:13:36',
'insights': '',
'locale_insights': {'en': '', 'id': '', 'zh': ''},
'locale_tips': {'en': [], 'id': [], 'zh': []},
'measurement_id': '80bcce7e-a89c-415c-ad15-475fdaf97b17',
'risk_code': 'RISK000',
'start_time': '2024-09-11 03:08:36',
'tips': []}},
'request_metadata': {'attempts': 1, 'sent_at': '2024-09-11
11:13:59.524107'},
'state': 0}

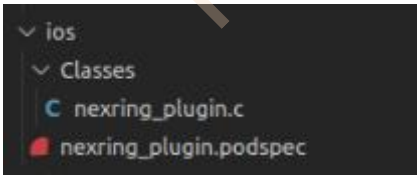
4） **Sample Code**


```
Future<void> registerDevice() async {
  if (sn.value.isEmpty) {
    sendBle(SendType.deviceInfo2);
    await Future.delayed(const Duration(seconds: 1));
  }
  await bloodGlucoseController
      ?.registerDevice(
          sn: sn.value,
          age: userHealthInfo["age"].value,
          gender: userHealthInfo["sex"].value,
          height: userHealthInfo["height"].value,
          weight: userHealthInfo["weight"].value,
          familyHistory: userHealthInfo["family_history"].value,
          highCholesterol: userHealthInfo["high_cholesterol"].value)
      .then((value) => registerResult.value = value);
}
ringManager.registerProcess(ReceiveType.PPG_SET, (data) {
  debugPrint("PPG_SET data=$data ");
});

ringManager.registerProcess(ReceiveType.PPG_DATA, (data) {
  if (startTime.isEmpty) {
    startTime = formatNowTime();
    currentTime = DateTime.now();
  }
  if (ppgDataList.length == 100) {
    if (ppgData.length > 240) {
      endTime = formatNowTime();
      //ppgData数组至少要240个数据才能进行计算
      sendBle(SendType.setPpg, {"on_off": 0});
      ppgSwitch.value = false;
    } else {
      currentTime = currentTime.add(const Duration(seconds: 1));
      ppgData.add({
        "ppg": List.from(ppgDataList),
        "timestamp":
            formatDateTime(currentTime)
      });
      ppgDataList.clear();
    }
  }
  ppgDataList.addAll(data["ppgList"]);
});
```

```
Future<void> uploadPpgData() async {      You, 11分钟前 • Uncommitted ch
  await bloodGlucoseController
      ?.uploadPpgRecord(
          fasting: true,
          within2HrsMeal: false,
          startTime: startTime,
          endTime: endTime,
          ppgData: ppgData)
      .then((value) {
    uploadResult.value = value;
  });
}

Future<void> getBloodGlucoseData() async {
  //服务端计算血糖数据需要时间，延迟5秒后再获取计算结果
  Future.delayed(const Duration(seconds: 5), () async {
    await bloodGlucoseController?.getPpgResult().then((data) {
      if (data != null) {
        if (data['state'] == 0) {
          if (data['data'] != null) {
            ppgMeasureResult.value =
                "lower_bound: ${data['data']["measurement_data"]["lower_
          }else{
            ppgMeasureResult.value = "no data";
          }
        } else if (data['state'] == 1) {
          ppgMeasureResult.value = "sn not registered";
        }
      }
    });
  }); // Future.delayed
}
```


3. iOS development considerations

All static libraries required for development are placed in smartring_plugin Under the Classes directory of the iOS directory of the project.



Note that the. a static library needs to be stored in the. a library folder, Classes, in the target ->build Setting ->library search path in xcode by directly dragging the smartstring_pluging project into the library search path.

Then that. a static library needs to be added in the Target ->Build Phases ->Link Binary With Libraries section of xcode.

Then Target->build Setting->Other Link Flags add -all_ Load

The release version needs to change the 'No' of the 'Enable Testability' release in the 'build Settings' to' Yes'