Android_SDK_V1.0.4

1 Search, connect, disconnect, and reconnect devices

1.1 Initialization Settings

Initialize YCBTClient

```
/***
  * Initialization
  * @param context
  * @param isReconnect whether need to reconnect
  * @param isDebug whether isDebug debugging mode
  */
public static void initClient(Context context, boolean isReconnect, boolean isDebug)
YCBTClient.initClient(this,true, BuildConfig.DEBUG);
```

When initializing, call the Reconnect class

```
public void init(Context context, boolean isReconnect)
```

Parameter description

- context context
- isReconnect sets whether or not to automatically connect back, true connects back, false does not connect back.

sample code (computing)

```
Reconnect.getInstance().init(getApplicationContext(),true);
```

1.2 Search for nearby equipment

```
YCBTClient.startScanBle(bleScanResponse);
```

Bluetooth device information is called back to onScanResponse one by one and can be used to display a list of Bluetooth devices

```
YCBTClient.startScanBle(new BleScanResponse() {
    @Override
    public void onScanResponse(int i, ScanDeviceBean scanDeviceBean) {
            if ( scanDeviceBean != null) {}
    }
}, 6);

  public class ScanDeviceBean implements Comparable<ScanDeviceBean> {
        private String deviceMac;
        private String deviceName;
        private int deviceRssi;
        public BluetoothDevice device;
}
```

Bluetooth device information contains: device mac, device name, rssi, BluetoothDevice object

```
YCBTClient.connectBle(mac, bleConnectResponse);
```

1.3 Connecting equipment

Connecting devices via device mac

```
YCBTClient.connectBle(mac, bleConnectResponse);
```

1.4 Global listening to Bluetooth device connection status

```
//add list
YCBTClient.registerBleStateChange(bleConnectResponse);
//remove list
YCBTClient.unRegisterBleStateChange(bleConnectResponse);
YCBTClient.registerBleStateChange(new BleConnectResponse() {
        @Override
        public void onConnectResponse(int code) {
                EventBusMessageEventeventBusMessageEvent = new EventBusMessageEvent();
                if (code == Constants.BLEState.Disconnect) {
                        // disconn
                        eventBusMessageEvent.belState =EventBusMessageEvent.DISCONNECT;
                } else if (code == Constants.BLEState.ReadWriteOK) {
                        //connection successful
                        eventBusMessageEvent.belState =EventBusMessageEvent.CONNECTED;
                } else if (code == Constants.BLEState.TimeOut) {
                        //Connection timeout
                        eventBusMessageEvent.belState = EventBusMessageEvent.TIMEOUT;
                } else if (code == Constants.BLEState.Disconnecting) {
                        //being disconn
                        eventBusMessageEvent.belState = EventBusMessageEvent.DISCONNECTING;
                }else {
                        //being connected
                        eventBusMessageEvent.belState = EventBusMessageEvent.CONNECTING;
                }
                EventBus.getDefault().post(eventBusMessageEvent);
        }
});
```

1.5 Disconnecting a Bluetooth device

```
YCBTClient.disconnectBle();
```

1.6 Determining Bluetooth connection status

```
if(YCBTClient.connectState() == Constants.BLEState.ReadWriteOK){
    //connected and read/write successful
}

  public static class BLEState {
        public static final int TimeOut =0x01; //Timeout
        public static final int NotOpen = 0x02;
        public static final int Disconnect = 0x03; //Not connected
        public static final int Disconnecting = 0x04; //Disconnecting in progress
        public static final int Connecting = 0x05; //Connected
        public static final int Connected = 0x06; //Connected
        public static final int ServicesDiscovered = 0x07; //DiscoveredServices
        public static final int CharacteristicDiscovered = 0x08; // Characteristic Discovered
        public static final int CharacteristicNotification = 0x09; // Enable Notification
        public staticfinal int ReadWriteOK = 0x0a; //Read/Write Success
}
```

1.7 Setting whether to reconnect

```
/**
  * Set whether reconnect is required
  * @param isReconnect whether reconnect is required
  */
public static void setReconnect(boolean isReconnect)
YCBTClient.setReconnect(true);
```

2 OTA upgrades

2.1 Connecting equipment

Connecting devices via device mac

```
YCBTClient.connectBle(mac, bleConnectResponse);
```

2.2 OTA upgrade

Call the update firmware method, start sending the firmware to the device, the callback can get the current progress, send the firmware to the device is complete, will disconnect the Bluetooth connection, you need to reconnect, and then call the update firmware method, this time, the device will display the interface of the firmware upgrade, the progress bar starts from 0 again, after the completion of the firmware upgrade, the device will automatically restart!
Calling Firmware Upgrade
Jelly Platform Upgrade
Start sending firmware, the interface shows a progress bar, the progress bar will be full automatically disconnect, listen to the connection status, if disconnected need to be connected back to the connection, after connecting to call the firmware upgrade method again, this time the device enters the upgrade state, the progress bar starts from 0 again, the upgrade is complete, the device will automatically reboot!

```
YCBTClient.upgradeFirmware(this, mac, name, filePath, dfuCallBack);
```

Non-Jeremy's Platform
Sending firmware and upgrades can be accomplished with a single call!

2.3 UI upgrade

Pass the path to the upgrade file and get the current progress in the callbacks

```
YCBTClient.watchUiUpgrade(path, new BleDataResponse() {
    @Override
    public void onDataResponse(int code, float ratio, HashMap resultMap) {
        Objectprogress1 = resultMap.get("progress");
        Object data = resultMap.get("data");
        if (progress1 != null) {
            // current progress percentage
                progress = (float) progress1;
        }else if (data != null){
            if((int)data == 0){
            //upgrade successful
            }else{
            //upgrade failed
            }


}
```

3 Synchronize and delete health history data

Obtaining health data from the device means querying the device for data such as heart rate, blood pressure, sleep, and blood oxygen, which are detected either actively or automatically by the device.
All devices have different health data types, so please refer to your device's return value and function support switches to invoke them. Among the data types supported by the device, the first 5 data types (Exercise, Sleep, Heart Rate, Blood Pressure, Combined Data) are supported by most of the devices, and the rest are supported by customized versions. Note:Don't use Heart Rate, Steps, Sleep, Blood Pressure obtained by Combined Data, that is to say, you can only use Blood Oxygen, Breathing Rate, Body Temperature, Body Fat, etc. obtained by this type. In other words, the data obtained by this type can only be used for blood oxygen, respiratory rate, body temperature, body fat, etc. Heart rate, step count, sleep, blood pressure can only be obtained by separate types.
The data in the device will not be deleted voluntarily, so the app should delete it voluntarily after getting it, otherwise it will get the same data when getting it next time, and once the data in the device exceeds the storage size, it will be deleted automatically.
After querying the data, the result will be returned in the response, and each type returns a different result, which needs to be converted to the corresponding object. Specific types have been listed one by one, refer to the content of 3.1.

3.1 Access to data

```
public static void healthHistoryData(int dataType, BleDataResponse bleDataResponse)
```

Parameter Description:
dataType: type
bleDataResponse: data callbacks
Type:

    Sport, Constants.DATATYPE.Health_HistorySport
    Sleep, Constants.DATATYPE.Health_HistorySleep
    Heart Rate, Constants.DATATYPE.Health_HistoryHeart
    Blood Pressure, Constants.DATATYPE.Health_HistoryBlood
    Combined Data,Constants.DATATYPE.Health_HistoryAll
    Lipids, Uric Acid, Constants.DATATYPE.Health_HistoryComprehensiveMeasureData

```
 YCBTClient.healthHistoryData(Constants.DATATYPE.Health_HistorySport,bleDataResponse);
```

Data callback bleDataResponse:

```
@Override
public void onDataResponse(int code, float ratio, HashMap resultMap) {
        if (code == 0 && resultMap != null) {
                // get success
                BloodResponse mBloodResponse = new Gson().fromJson(String.valueOf(resultMap), BloodResponse.class);
                List<BloodResponse.DataBean> mBloodData = mBloodResponse.getData();
        } else {
                //failed to get
        }
}
public class BloodResponse {
    private int code;
    private List<DataBean> data;//different data returned depending on the type
    private int dataType;
}
```

Health data types and corresponding return value types

```
public static class SportDataBean {
    private long sportStartTime;// start timestamp (seconds)
    private long sportEndTime;// end timestamp (seconds)
    private int sportStep;// number of steps (steps)
    private int sportDistance;// distance (meters)
    private int sportCalorie;// calories (kcal)
}
```

```
public static class SleepDataBean {
    private int deepSleepCount; // deepSleepCount
    private int lightSleepCount; // lightSleepCount
    private long startTime; // startSleepTime
    privatelong endTime; // endSleepTime
    private int deepSleepTotal; // deepSleepTotal
    private int lightSleepTotal; // lightSleepTotal
    @SerializedName(value = "rapidEyeMovementTotal", alternate = {"remTimes"})
    public int rapidEyeMovementTotal; // eyeMovementTotal
    public int wakeCount;// number of times awake
    public int wakeDuration;// length of time awake
    private List<SleepData> sleepData = new ArrayList<>(); // sleep data
}

  public static class SleepData {
    @SerializedName(value = "sleepStartTime", alternate = {"stime"})
    private long sleepStartTime; // start time
    @SerializedName(value = "sleepLen", alternate = {"sleepLong"})
    private int sleepLen; // sleep length
    privateint sleepType; // deepSleepLightSleep flag, flag type is SleepType below
}

  public static class SleepType{
    public static final int unknow = -1;// unknown
    public static final int deepSleep= 0xF1;// deepSleep
    public static final int lightSleep = 0xF2;// lightSleep
    public static final int rem = 0xF3;// rapid eye movement
    public static final int awake = 0xF4;// awake
}
public static class DataBean {
    private int heartValue;// heart rate value
    private long heartStartTime;// start timestamp in seconds
}
public static class DataBean {
    private long bloodStartTime;// start timestamp
    private int bloodDBP;// diastolic blood pressure
    private int bloodSBP;// systolic blood pressure
}
public static class AllDataBean {
    public int heartValue;// heart rate value
    public int hrvValue;// HRV
    public int cvrrValue;// CVRR
    public int OOValue;// oxygen value
    publicint stepValue;// number of steps
    public int DBPValue;// diastolic pressure
    public int tempIntValue;// integer part of temperature
    public int tempFloatValue;// decimal part of temperature
    public long startTime;// starttimestamp
    public int SBPValue;// systolic blood pressure
    public int respiratoryRateValue;// respiratory rate value
    public int bodyFatIntValue;// body fat integer part
    public int bodyFatFloatValue;// body fat decimal part
    public int bloodSugarValue;// blood sugar*10 value
}
```

```
public static class ComprehensiveMeasureDataBean {
    //tc
    public int cholesterolInteger;//lipid integer part
    public int cholesterolFloat;//lipid decimal part
    //tg
    publicint triglycerideCholesterolInteger;//not used
    public int triglycerideCholesterolFloat;//not used
    //ldlc
    public int highLipoproteinCholesterolInteger;//not used
    public int highLipoproteinCholesterolFloat;//not used
    //hdlc
    public int lowLipoproteinCholesterolInteger;//not used
    public intlowLipoproteinCholesterolFloat;
    public int bloodFatModel;//lipid measurement model

    public int uricAcid;//uric acid value
    public int uricAcidModel;//uric acid measurement model

    public intbloodKetoneInteger;//not used
    public int bloodKetoneFloat;//not used
    public int bloodKetoneModel;//not used

    public int bloodSugarInteger;//not used
    public intbloodSugarFloat;// not used
    public int bloodSugarModel;// not used

    public long time;// start timestamp
}
```

3.2 Deletion of data

```
YCBTClient.deleteHealthHistoryData(type,bleDataResponse);
```

Parameter Description:

Type:

Sport, Constants.DATATYPE.Health_DeleteSport
Sleep, Constants.DATATYPE.Health_DeleteSleep
Heart Rate, Constants.DATATYPE.Health_DeleteHeart
Blood Pressure, Constants.DATATYPE.Health_DeleteBlood
Combined Data,Constants.DATATYPE.Health_DeleteAll
Lipids, uric acid, Constants.DATATYPE.Health_DeleteComprehensiveMeasureData

bleDataResponse: data callbacks

```
@Override
public void onDataResponse(int code, float ratio, HashMap resultMap) {
        if (code == 0) {
                // deletion successful
        } else {
                // deletion failed
        }
}
```

4 Dials

Dial download includes checking dial information in device, app switching dials, deleting dials, device operating dials, app customizing dials.

4.1 Querying Device Dial Information

```
// Call the Get Dial interface, pass in callback parameters
YCBTClient.watchDialInfo(new BleDataResponse() {
    @Override
    public void onDataResponse(int i, float v, HashMap hashMap) {
          if (i = = 0 && hashMap != null) {
                maxDials = (int) hashMap.get("maxDials");// maximum number of dials
                currDials = (int) hashMap.get("currDials");// current number of dials
                List<DialsBean> dials = (List<DialsBean>) hashMap.get("dials");//data of all dials
                List<DialsBean> customDials = (List<DialsBean>) hashMap.get("customDials");//data of custom dials, some devices don't support this feature
          }
    }
});

  //information
public class DialsBean {
    public int dialplateId;//
    public int blockNumber;//number of block packets currently received
    public boolean isCanDelete;//whether it can be deleted
    public int dialVersion;//dial version
}
```

4.2 Deleting dials

Calling methods are different for Jelly platforms and non-Jelly platforms

```
//Judge the platform
if (YCBTClient.getChipScheme() == Constants.Platform.JieLi) {
        //JieLi platform,delete according to the dial file name
        String fileName = datas.get(position).fileName;
        //Interceptdial name, example:/watch0
        YCBTClient.jlWatchDialDelete(fileName.substring(fileName.lastIndexOf("/")), new BleDataResponse() {
                @Override
                public voidonDataResponse(int i, float v, HashMap hashMap) {
                            if (i == 0) {
                              //delete successfully
                          }else{
                              //delete failed
                          }

        });
} else {
        //non-jersey platforms, deletion based on dial dialplateId
        DialResultBean.Datadata = datas.get(position);
        data.state = 0;
        data.progress = 0;
        data.blockNumber = 0;
        YCBTClient.watchDialDelete(data.dialplateId, newBleDataResponse() {
                @Override
                public void onDataResponse(int i, float v, HashMap hashMap) {
                          if (i == 0) {
                              //deletion succeeded
                          }else{
```

```
                        //deletion failed
                    }
               }
       });
}
```

4.3 Switching dials

The parameters for switching dials and deleting dials are the same

```
//Judge the platform
if (YCBTClient.getChipScheme() == Constants.Platform.JieLi) {
    //JieLi platform,delete according to the dial file name
    String fileName = datas.get(position).fileName;
    //Interceptdial name
    YCBTClient.jlWatchDialSetCurrent(fileName.substring(fileName.lastIndexOf("/")), new BleDataResponse() {
            @Override
            public voidonDataResponse(int code, float v, HashMap hashMap) {
                    if (code == 0) {
                        //Setup successful
                    }else{
                        //Setup failed
                    }

    });
} else {
    //Non-jersey platforms, deletion based on dial dialplateId
    YCBTClient.watchDialSetCurrent(datas.get(position).dialplateId, new BleDataResponse() {
            @Override
            public void onDataResponse(int code, float v, HashMaphashMap) {
                    if (code == 0) {
                        // setup successful
                    }else{
                        // setup failed
                    }
            }
    });
}
```

4.4 Downloading the dial

Before downloading a new dial, you need to delete an old dial if you exceed the maximum number of dials.

4.4.1 Jelly Platform Download

```
/**
  * dial download , dial version number whole file CRC
  *
  * @param filePath is dial file path, must exist
  * @param isNoNeedCheck : whether to skip file checking false : dial file need file checking true : custom background file does not need file checking, but need conversion tool for algorithm conversion
  * @param dataResponse code 0x00: End download 0x01: Start download data dialState Ox00: Received successfully 0x01: Parameter error 0x02: Number of dials exceeds maximum storage
  */
public static void jlWatchDialDownload( String filePath, boolean isNoNeedCheck, BleDataResponse dataResponse)
```

```
YCBTClient.jlWatchDialDownload(path, false, new BleDataResponse() {
    @Override
    public void onDataResponse(int i, float v, HashMap hashMap) {
        if (code == 0) { //Ox00: Received successfully 0x01: Parameter error 0x02: Number of dials exceeds maximum storage
            // Determine if this is a return progress
            if((int) hashMap.get("dataType") == Constants.DATATYPE.WatchDialProgress){
                int progress = (int) ((float) hashMap.get("progress"));//current download progress
            }else{
                //download complete
            }
        }else{
            //download failed
        }
    }
});
```

4.4.2 Non-jersey platform downloads

```
/**
  * Dial download, dial version number, whole file CRC
  *
  * @param type 0x00: end download 0x01: start download
  * @param dialDatas Binary data for dials
  * @param dialId ID of dials, IDs are the same for an update, different for a new file
  * @param dialBlock received Number of blocks.No breakpoint information, then 0
  * @param dialVersion dial version number
  * @param dataResponse code 0x00: end download 0x01: start download data dialState Ox00: receive success 0x01: parameter error 0x02: the number of dials exceeds the maximum number of stored
  */
  public static void watchDialDownload(int type, byte[] dialDatas, int dialId, int dialBlock, int dialVersion, BleDataResponse dataResponse)
FileInputStream inputStream = new FileInputStream(new File(path));
byte[] buffer = new byte[1024];
int len = 0;
ByteArrayOutputStream bos = newByteArrayOutputStream();
while ((len = inputStream.read(buffer)) != -1) {
        bos.write(buffer, 0, len);
}
bos.flush();
YCBTClient.watchDialDownload(0x01, bos.toByteArray(), id, datas.get(position).blockNumber, Integer.parseInt(position).Integer.parseInt(datas.get(position).dialVersion), new BleDataResponse() {
        @Override
        public void onDataResponse(int i, float v, HashMaphashMap) {
            if (code == 0) { //Ox00: Received successfully 0x01: Parameter error 0x02: Number of dials exceeds maximum storage
                // Determine if this is a return progress
                if((int) hashMap.get("dataType") == Constants.DATATYPE.WatchDialProgress){
                    int progress = (int) ((float) hashMap.get("progress"));//current download progress
                }else{
                    //download complete
                }
            }else{
                //download failed
            }
        }
});
```

4.5 Customizing Dials

Get dial reference 4.1 Query the device dial information and get both the system dial and customized dials
clarification
Custom Dial is based on the custom dial source file provided by the vendor, and modifies the image and text color to generate a new dial file, which is downloaded to the device. If you don't modify the image, you can pass in null and the SDK will keep the original background image and thumbnail image. Once the new dial file is generated, call the dial download method directly to download it to the device. The document gives a dial BMP query information method, may be in the app development interface, or generate thumbnails when you want to use the relevant parameters.

```
/***
  * getScreenParameters
  * @param dataResponse dataResponse hashmap.get("data");
  * screenType 0:round 1:square
  * screenWidth screen width in pixels
  * screenHeight screen height
  *screenCorner screenCorner
  */
public static void getScreenParameters(BleDataResponse dataResponse)
YCBTClient.getScreenParameters(new BleDataResponse() {
    @Override
    public void onDataResponse(int i, float v, HashMap hashMap) {
          if (hashMap != null) {
              // screenCorner
              int screenCorner = (int) hashMap.get("screenCorner");
          }
    }
}
```

Generate customized dial data

```
/*
  * Modify the custom dial file
  * @param dst Destination file address byte[]
  * @param size bin file byte[]
  * @param bg_src Background image byte[]
  * @param thumb_src Thumbnail image byte[]
  * @param x Position of the date display x coordinate short
  * @param y The y coordinate of the position where the date is displayed short
  * @param r The date font color value
  * @param g The date font color value
  * @param b The date font color value
  *
  * @return true: modify successful false: modify failed
  *
  */
public boolean modifyBinFile(byte[] dst, byte[] src, byte[] bg_src, byte[] thumb_src, int x, int y, byte r, byte g, byte b)
```

4.5.1 Jelly Platform

```
/**
  * Jelly Platform bitmap image conversion
  *
  * @param inPath Input file path <p>example:in.png ,in.jpg</p>
  * @param outPath Output file path <p>example:out.res, out</p>
  * @param dataResponse code 0x00: success 0x01: failure
```

```
 */
public static void jlSaveCustomizeDialBg(String inPath, String outPath, BleDataResponse dataResponse)
/**
 * jlInstallCustomizeDial
 * @param bgFilePath file path
 * @param dataResponse callback
 */
public static void jlInstallCustomizeDial(String bgFilePath, BleDataResponse dataResponse)
/**
 * Set the position and color of jieli dial text
 * @param position One of 9 positions
 * 0x01:up
 * 0x02:down
 * 0x03:left
 * 0x04:right
 * 0x05:top left
 * 0x06:top right
 * 0x07:bottom left
 * 0x08:bottom right
 * 0x09:center
 * @param color Color rgb565 format
 *@param dataResponse
 */
public static void jieliSetDialText(int position,int color,BleDataResponse dataResponse)
```

4.5.2 Non-jelly platforms

Getting, switching, and deleting are the same as any other dial.

```
/*
 * Get custom dial background image size
 *
 */
public ImageBean getBmpSize(byte[] binArray)
/*
 * background image bmp888 to bmp565
 *
 */
public byte[] toBmp565(byte[] binArray, int size, boolean isFlip)
/*
 * Modify the custom dial file
 * @param dst Destination file address byte[]
 * @param size bin file byte[]
 * @param bg_src Background image byte[]
 * @param thumb_src Thumbnail image byte[]
 * @param x Position of the date display x coordinate short
 * @param y The y coordinate of the position where the date is displayed short
 * @param r The date font color value
 * @param g The date font color value
 * @param b The date font color value
 *
 * @return true: modify successful false: modify failed
 *
```

```
  */
public boolean modifyBinFile(byte[] dst, byte[] src, byte[] bg_src, byte[] thumb_src, int x, int y, byte r, byte g, byte b)
```

4.6 Listening Dial Switching

```
YCBTClient.deviceToApp(new BleDeviceToAppDataResponse() {
        @Override
        public void onDataResponse(int code, HashMap hashMap) {
        //listen for device operations, dial switching or deletion
        if (code == 0 && hashMap ! = null && hashMap.get("dataType") != null && (int) hashMap.get("dataType") == Constants.DATATYPE.DeviceSwitchDial) {
            if (YCBTClient.getChipScheme() == Constant..JieLi) {
                //JieLi Platform
                String deviceName = hashMap.get("datas").toString();
                for (int i = 0; i < datas.size(); i++) {
                    DialResultBean.Data data = datas.get(i);
                    String fileName = data.fileName;
                    String substring = fileName.substring(fileName.lastIndexOf("/"));
                    // Determine and modify the current state of the dial
                    if (substring.equalsIgnoreCase(deviceName)) {
                        // current dial in use
                        data.state = 4;
                        oldDialPosition = i;
                    } else {
                        if (data.state == 4) {
                            data.state = 3;
                        }
                    }
                }
            } else {
                // non-Jellyfish platform
                byte[]ids = (byte[]) hashMap.get("datas");
                int id = (ids[0] & 0xff) + ((ids[1] & 0xff) << 8) + ((ids[2] & 0xff) << 16) + ((ids[3] &0xff) << 24);
                for (int i = 0; i < datas.size(); i++) {
                  // Determine and modify the current state of the dial
                    DialResultBean.Data data = datas.get(i);
                    if (data.state == 4) {
                        data.state= 3;
                    }
                    if (data.dialplateId == id) {
                        // current dial in use
                        data.state = 4;
                        oldDialPosition = i;
                    }
                }
            }
        }
        }
}
```

4.7 Setting up custom dials

Customized dials for Jelly and non-Jelly settings are encapsulated.

```
 /**
     * Set custom dial
     * @param context context
     * @param imgPath image path
     * @param thumb      nailPath thumbnail path
* @param customDialId di      al id
bin file path
     * @param pointX Text x-axis Jelly dial no x-axis y-axis parameter, can pass 0
     * @param pointY Text y-axis
     * @param position Text position //Up:1 Down:2 Left:3 Right:4 Up:5 Up:6 Down:7 Down:8 Middle:9
     * @param parseColor Text colorrgb565 format
     * @param isCanDelete whether can delete
     * @param dialProgressListener callback,status:0normal,1installation of the number of up to the limit,2failure,progress:progress
     */
   public static void setDialCustomize(Context context,String imgPath, String thumbnailPath,
                                            int customDialId, String saveFileName, int pointX, int pointY, int position,
                                            int parseColor , boolean isCanDelete, DialUtils.DialProgressListener dialProgressListener)
```

5 Take a picture

There are two ways to start taking a picture, one is to start the device to enter the photo mode, and the other is to start the app to enter the photo mode. Entering Photo Mode
There are also two ways to perform the photo-taking action, one by tapping the app and the other by tapping the device. Taking a real picture
This is done on the cell phone, and if the device is tapped to take the picture, you need to reply to the device when the picture is taken whether the picture was taken successfully or not.

5.1 App Enabling and Disabling Photo Mode

```
/**
  * APP control bracelet take photo interface
  *
  * @param type 0x00: exit take photo mode 0x01: enter take photo mode
  * @param dataResponse 0x00 Success 0x01 Failure
  */
public static void appControlTakePhoto(int type, BleDataResponse dataResponse)
YCBTClient.appControlTakePhoto(1, dataResponse);
```

5.2 Turning the device on and off photo mode

```
 YCBTClient.deviceToApp(new BleDeviceToAppDataResponse() {
            @Override
            public void onDataResponse(int i, HashMap hashMap) {
                if (hashMap != null) {
                    if (i == 0) {//
                        int dataType = (int) hashMap.get("dataType");
                        int data = -1;
                        if (hashMap.get("data") != null)
                            data = (int) hashMap.get("data");
                        switch (dataType) {
                          case Constants.DATATYPE.DeviceTakePhoto://Camera take photo control
                            if (PermissionUtil.openCameraPermission(context) && PermissionUtil.openSDCardPermission(context)) {
                              //0x00:exit photo mode 0x01:enter photo mode 0x02:take photo
                                if (data == 1) {
                                                }else {
                                                        // send event notification to exit photo mode
                                                }
                                        }
                                        break;
                                }
                        }
                }
        }
 });
```

When the status is received as photo, the photo is still operated on the cell and the device is only responsible for reporting the current status.

5.3 Cross-operation

Whenever the interaction logic of taking a picture crosses over, such as when a cell phone is activated and the device is exited, or when a device is activated and the cell phone is exited. Be aware of the status
To change the number of the USB flash drive, adjust the corresponding USB port.

6 Contacts

The Address Book function only sends user names and numbers to the device for storage, and the maximum number of addresses that can be stored on the device is 30. During the entire process of transferring contacts, you only need to execute synchronization once to turn it on and off, but you need to execute it repeatedly to send the contact data, because only one record can be sent per execution.

6.1 Access to Address Book

```
/**
 * Get Address Book
 */
public static void getDeviceContacts(Context context, BleDataResponse bleDataResponse)
 YCBTClient.getDeviceContacts(ContactsActivity.this, new BleDataResponse() {
        @Override
        public void onDataResponse(int i, float v, HashMaphashMap) {
                if (hashMap ! = null && hashMap.get("data") != null) {
                        List<ContactsBean> beans = (ArrayList) hashMap.get("data");
                }
        }
});

public class ContactsBean implements Serializable {
        public shortid;// address book id
        public String name;// name
        public String number;// number
}
```

6.1 Entering the Synchronized Address Book

The device will not enter the actual synchronized data until it is set to start synchronization. This needs to be performed only once.

```
/**
 * Start/end push contacts
 *
 * @param type operator 0x00:end, 0x02:start
 * @param dataResponse 0x00 sync success 0x01 sync failure
 */
public static void appPushContactsSwitch(int type, BleDataResponse dataResponse)
```

6.2 Sending Address Book Data

6.2.1 Non-jersey platforms

```
/**
 * Push a single address book
 *
 * @param phoneNumber cell phone number
 * @param name name (UTF-8 string) can not be more than the maximum of 8 Chinese characters
 * @param dataResponse 0x00 synchronization success 0x01 synchronization failure
 */
public static voidappPushContacts(String phoneNumber, String name, BleDataResponse dataResponse)

 /**
 * Push a set of contacts, the internal implementation is still a loop to read the data, and push a single contacts
 *
 * @param contactsBeans contactsentity class collection
 * @param dataResponse 0x00 synchronization success 0x01 synchronization failure
 */
public static void appNewPushContacts(Context context, List<ContactsBean> contactsBeans, BleDataResponse dataResponse)
YCBTClient.appPushContactsSwitch(2, dataResponse) // start
for (ContactsBean bean : contactsBeans) {
    YCBTClient.appPushContacts(bean.number,bean.name, dataResponse);//push a single
}
YCBTClient.appPushContactsSwitch(0, dataResponse);//end
```

6.2.2 Jelly Platform

Sending messages from the Geritol platform's address book uses the platform's own API, which is proprietary.

```
List<DeviceContacts> contacts = new ArrayList<>();
for (ContactsBean bean : contactsBeans) {
    DeviceContacts deviceContacts = newDeviceContacts(bean.id, bean.name, bean.number);
    if (bean.name != null) {
        bean.name = ByteUtil.getData(bean.name, 20);
    }
    contacts.add(deviceContacts);
 }
   UpdateContactsTask task = new UpdateContactsTask(WatchManager.getInstance(), context, contacts);
task.setListener(new SimpleTaskListener() {
    @Override
    public void onBegin() {
    }

    @Override
    public void onError(int code, String msg) {
        if (dataResponse != null) {
            HashMap<String, String> map = new HashMap<String, String>();
            map.put("msg", msg);
            dataResponse.onDataResponse(-1, 0, map);
```

```
                }
        }

        @Override
        public void onFinish() {
                if (dataResponse != null) {
                        dataResponse.onDataResponse(0, 0, null);
                }
        }
});
task.start()
```

7 ECG test

ECG detection includes start and stop ECG, get the result of ECG, and refer to the Demo's case study on plotting ECG waveforms.
Examples of relevant methods are given in the file. ECG detection is turned on and off by the app, and the recommended measurement time is 60 ~ 90%.
seconds. The test process will acquire the measured data. Similarly, the device itself can initiate ECG measurements, and the app can acquire the relevant information.

7.1 Setting the wearing position

When the device is worn in a position that does not match the setup position, the resulting waveform is reversed.

```
/***
    * Setting left and right hand wear
    * @param leftOrRight 0x00: left hand 0x01: right hand
    * @param dataResponse 0x00 sync successful 0x01 sync failed
        */
public static void settingHandWear(int leftOrRight, BleDataResponse dataResponse)
YCBTClient.setHandWear(Constants.HandWear.Left, new BleDataResponse() {
        @Override
        public void onDataResponse(int code, float ratio,HashMap resultMap) {
                if (code == 0) {
                        Log.i("EcgMeasurDialog", "Your selection is:Left Handed"                ;
}
        }
});
```

7.2 Starting and ending ECG measurements

After initiating ECG measurement, it is necessary to place the finger of another hand in relation to the electrode position to make contact and measure the on/off value.
After starting, the device will report the measured data.

```
/****
  * Start ECG real-time testing
  * @param dataResponse
  * @param realDataResponse
  */
public static void appEcgTestStart(BleDataResponse dataResponse,BleRealDataResponse realDataResponse)

/***
  * End ECG real-time test
  * @param dataResponse
  */
public static void appEcgTestEnd(BleDataResponse dataResponse)
```

```java
YCBTClient.appEcgTestStart(dataResponse, new BleRealDataResponse() {
        @Override
        public void onRealDataResponse(int i, HashMap hashMap) {
                if ( hashMap != null) {
                        int dataType = (int) hashMap.get("dataType");
                        if (i == Constants.DATATYPE.Real_UploadECG) {
                                List<Integer> data = (List<Integer&gt;) hashMap.get("data");
                                person(data);
                        } else if (i == Constants.DATATYPE.Real_UploadECGHrv) {
                                if (!hrv_is_from_device && hashMap.get( "data") ! = null && ((float) hashMap.get("data")) != 0) {
                                        mHRV = (int) ((float) hashMap.get("data"));
                                }
                        } else if (i == Constants.DATATYPE.Real_UploadECGRR) {
                                float param = (float) hashMap.get("data");
                        } else if (i == Constants.DATATYPE.Real_UploadBlood) {
                                mHeart = (int) hashMap.get("heartValue");//heart rate
                                mDBP = (int) hashMap.get("bloodDBP");//high pressure
                                mSBP = (int) hashMap.get("bloodSBP");//low pressure
                                if (hashMap.get("hrv") ! = null && ((int) hashMap.get("hrv")) != 0) {
                                        hrv_is_from_device = true;
                                        mHRV = (int) hashMap.get("hrv");
                                }
                        } else if (i == Constants.DATATYPE.AppECGPPGStatus) {
                            //Some customized devices will return PPG data
                        }
                }
        }
});

  YCBTClient.appEcgTestEnd(new BleDataResponse() {
        @Override
        public void onDataResponse(int code, float ratio, HashMap resultMap) {
                if (code == 0) {
                        if (mEcgMeasureList.size() > 2800) {
                                // heart rate analysis
                        }
                }
        }
} });

  private void person(List<Integer> datas) {
        mEcgMeasureList.addAll(datas);
        int index =0;
        for (Integer data : datas) {
                if (index % 3 == 0) {
                        int value = data / 40;
                        drawLists.add(value > 500 ?500 : value);
                }
                index++;
        }
}
```

7.2 Obtaining ECG results

```
  AITools.getInstance().getAIDiagnosisResult(new BleAIDiagnosisResponse() {
        @Override
        public void onAIDiagnosisResponse(AIDataBeanaiDataBean) {
                if (aiDataBean != null) {
                        short heart = aiDataBean.heart;//heart rate
                        // diagnosis type 1 normal 4 ,5 ,9 abnormal
                        mDiagnoseType = aiDataBean.qrstype;// type 1 normal heart beat 5 ventricular premature heart beat 9 atrial premature heart beat 14 noise
                        // whether atrial fibrillation
                        isAfib =aiDataBean.is_atrial_fibrillation;// whether at
                        // store data
                        saveData();
                }
        }
} });

  public class AIDataBean implements Serializable {
        public short heart;// heart rateheart<= 50 suspected bradycardia heart>= 120 suspected tachycardia hrv >= 125 suspected sinus arrhythmia
        public int qrstype;//type 1 normal 2 ventricular flutter 3 junctional tachycardia 4 junctional escape beats 5 ventricular tachycardia 6 ventricular escape beats 7 left bundle branch block 8 right bundle branch block 9 atrial tachycardia 10Atrial escape
beats 14 Abnormal ECG (abnormal waveform)
        public boolean is_atrial_fibrillation;//is_atrial_fibrillation (AF)
}
```

7.3 Synchronizing ecg data and PPG data

```
// Synchronize ECG
public static void collectEcgList(BleDataResponse dataResponse)
// Synchronize PPG data
public static void collectPpgList(BleDataResponse dataResponse)
YCBTClient.collectEcgList(new BleDataResponse() {
    @Override
    public void onDataResponse(int code, float v, HashMap resultMap) {
            if (code == 0) {
                    if (resultMap == null) {
                            return;
                    }
                    EcgSyncListResponse ecgSyncListResponse = new Gson().fromJson(String.valueOf(resultMap), EcgSyncListResponse.class);
                    if (ecgSyncListResponse == null || ecgSyncListResponse.data == null) {
                            return;
                    }
                    List<EcgSyncListResponse.DataBean> dataBeans =ecgSyncListResponse.data;
                    for (EcgSyncListResponse.DataBean dataBean : dataBeans) {
                            List<Integer> lists = SharedPreferencesUtil.readEcgListMsg(
                                            new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date(dataBean.collectStartTime)),
                                            EcgActicvity.this);
                            if (lists != null && lists.size() > 0) {
                                    // Delete the data in the watch after getting it
                                    deleteEcgInfo(dataBean.collectSendTime);
                            } else {
                                    datas.add(dataBean);
                            }
                    }
            }
    }
});

  publicclass EcgSyncListResponse {
        public int collectType;
        public int code;
        public List<DataBean> data;

        public class DataBean {
                public intcollectDigits;
                public int collectType;
                public int collectSN; // number of bracelet data bars
                public int collectTotalLen;
                public long collectSendTime;
                public longcollectStartTime;
                public int collectBlockNum;
        }
}
/***
  * Synchronize history raw data
  * @param type 0x00: Electrocardiogram data (ECG) 0x01: PPG data 0x02: 3-axis Acceleration data 0x03: 6-axis sensor data (3-axis Acceleration + 3-axis Gyroscope)
  * 0x04: 9-axis sensor data (3-axis Acceleration + 3-axis Gyroscope + 3-axis Magnetometer) 0x05: 3-axis Magnetometer data0x06: Inflatable blood pressure data 0x07: PPI data
  * @param dataResponse
  */
public static void collectHistoryListData(int type, BleDataResponse dataResponse)

  /***
  * Delete history raw data
  * @param type 0x00: Electrocardiogram data (ECG) 0x01: PPG data 0x02: 3-axis acceleration data 0x03: 6-axis sensor data (3-axis acceleration + 3-axis gyroscope)
  * 0x04: 9-axis sensor data (3-axis acceleration + 3-axis gyroscope + 3-axis magnetometer) 0x05: 3-axis magnetometer data 0x06: inflated blood pressure data
  * @param timestamp timestamp, if it is (0xFFFFFFFF) means delete all
  * @param dataResponse
  */
public static void deleteHistoryListData(int type, long timestamp, BleDataResponse dataResponse)
```

7.4 Deleting device history data

```
/***
  * Delete historical raw data
  * @param type 0x00: Electrocardiogram data (ECG) 0x01: PPG data 0x02: 3-axis acceleration data 0x03: 6-axis sensor data (3-axis acceleration + 3-axis gyroscope)
  * 0x04: 9-axis sensor data (3-axis acceleration + 3-axis gyroscope + 3-axis magnetometer) 0x05:3-axis magnetometer data0x06: Inflatable blood pressure data
  * @param timestamp timestamp, if it is (0xFFFFFFFF) means delete all
  * @param dataResponse
  */
public static void deleteHistoryListData(int type, long timestamp, BleDataResponse dataResponse)
YCBTClient.deleteHistoryListData(0, sendTime, new BleDataResponse() {
        @Override
        public void onDataResponse(int code, float ratio, HashMapresultMap) {
                if (code == 0) {
                    // Delete Success
                } else {
                    // Delete Fail
                }
        }
})
```

7.5 Obtaining ECG and PPG data for device startup measurements

1. If it is an ECG measurement initiated by the device, only the measured ECG or PPG data can be obtained, no other data is available.
2. For operations related to acquiring data, please refer to the section on Historical Acquisition Data.

8 Movement

```
public class SportType {
        public static final int RESERVED = 0;//
        public static final int RUN = 1;//run
        public static final int SWIMMING = 2;//swim
        publicstatic final int RIDE = 3;
        public static final int FITNESS = 4;// Fitness
        public static final int SCRAP = 5;//
        public static final int SKIPPING_ROPE = 6;//JUMP ROPE
        public static final int BASKETBALL = 7;// Basketball
        public static final int WALKING = 8;// Walking
        public static final int BADMINTON = 9;// Badminton
        public static final intFOOTBALL = 10;//Football
        public static final int MOUNTAINEERING = 11;//Mountaineering
        public static final int PING_PONG = 12;//Table tennis
        public static final int FREE_MODE = 13;/ //FREE_MODE
        public static final int RUN_INDOORS = 14;//Indoor running (or treadmill)
        public static final int RUN_OUTSIDE = 15;//Outdoor running
        public static final int WALK_OUTDOOR = 16;///Outdoor Walking
        public static final int WALK_INDOOR = 17;//Indoor Walking
        public static final int WALK_AND_RUN = 18;//Walking and Running Mode
        public static final int INDOOR_CYCLING = 19;//IndoorRIDING (or MOTORCYCLE)
        public static final int STEPPER = 20;//STEPPER
        public static final int ROWING_MACHINE = 21;//ROWING MACHINE
        public static final int REAL_TIME_MODE = 22;//Real TimeMonitor Mode
        public static final int SIT_UPS = 23;// Sit
        public static final int LEAPING_MOTION = 24;// Jumping Movement
        public static final int WEIGHT_TRAINING = 25;// Weight Training
        public static final int YOGA = 26;// Yoga
        public static final int ONFOOT = 0x1B;// Hiking
        public static final int VOLLEYBALL = 0x1C;// Volleyball
        public static final intKAYAK = 0x1D;// Kayaking
        public static final int ROLLER_SKATING = 0x1E;// Roller skating
        public static final int TENNIS = 0x1F;// Tennis
        public static final int GOLF = 0x20;//Golf
        public static final int ELLIPTICAL_MACHINE = 0x21;// Elliptical
        public static final int DANCE = 0x22;// Dance
        public static final int ROCK_CLIMBING = 0x23;// Climbing
        public static final int AEROBICS = 0x24;// Exercise
        public static final int OTHERSPORTS = 0x25;// Other Sports
}
```

8.1 Startup and shutdown of the movement

```
/***
  * Sport Mode
  * @param sportSwitch 0x00: Stop 0x01: Start 0x02: Pause 0x03: Continue
  * @param sportType Type Selection 0x00: Reserved 0x01: Running 0x02: Swimming 0x03: Cycling 0x0C: Table Tennis
  * 0x04: Fitness 0x08: Walking 0x05: Reserved 0x09: Badminton 0x06: Rope Skipping 0x0A: Soccer 0x07: Basketball 0x0B: Mountaineering
  * 0x0C: Table Tennis 0x0D: Free Mode 0x0E: Indoor Running 0x0F: Outdoor Running 0x10: Outdoor Walking 0x11: Indoor Walking
  * 0x12: Walking Running Mode 0x13: Indoor Cycling 0x14: Stepping Machine 0x15: Rowing Machine0x16:Real Time Monitor Mode 0x17:Sit-ups 0x18:Jumping Exercise 0x19:Weight Training 0x1A:Yoga
  * @param dataResponse 0x00 Synchronization Success 0x01 Synchronization Failure
  */
public static void appRunMode(int sportSwitch, int sportType, BleDataResponse dataResponse)
YCBTClient.appRunMode(Constants.SportState.Start,Constants.SportType.RUN,dataResponse);
/***
  * Register to listen for real-time data
  */
public static void appRegisterRealDataCallBack(BleRealDataResponse realDataResponse)
YCBTClient.appRegisterRealDataCallBack(new BleRealDataResponse() {

        @Override
        public void onRealDataResponse(int dataType, HashMap dataMap) {
                if (dataMap == null) {
                        return;
                }
                if (YCBTClient.isSupportFunction(Constants.FunctionConstant.ISHASREALEXERCISEDATA)) {
                        if (dataType ==Constants.DATATYPE.Real_UploadOGA) {
                                if (mSportType != SportType.RIDE){
                                        mRunInfo.distance = (int) dataMap.get("sportsRealDistance");// total distance
                                        mRunInfo.calorie = (int) dataMap.get("sportsRealCalories");// calories
                                }
                                mRunInfo.runTime = (int) dataMap.get("recordTime");// duration of the sport
                                mRunInfo.heart = (int) dataMap.get("heartRate");// heart rate
                                sportStep = (int) dataMap.get("sportsRealSteps");// total steps
                        }
                } else {
                        if (dataType == Constants.DATATYPE.Real_UploadSport) {
                                if (isFirst) { // Initial distance InitialCalories Initial Steps
                                        startDistance = (int) dataMap.get("sportDistance");
                                        startCalorie = (int) dataMap.get("sportCalorie");
                                        startSportStep = (int) dataMap.get("sportStep");
                                        isFirst = false;
                                }
                                sportStep = (int) dataMap.get("sportStep");//steps
                                sportDistance = (int) dataMap.get("sportDistance");//distance
                                sportCalorie = (int) dataMap.get("sportCalorie");//calorie
                                mRunInfo.distance = (sportDistance - startDistance);//total distance
                                mRunInfo.calorie = (sportCalorie - startCalorie);//total calories
                                sportStep = (sportStep - startSportStep);//total steps
                        } else if (dataType == Constants.DATATYPE.Real_UploadHeart) {
                                heartValue = (int) dataMap.get("heartValue");// heart rate
                                mRunIn        fo.heart = heartValue;
}
}
} }
});

  public class RunInfo implements Serializable {
        public inttype = -1; // sport type: 0: reserved, 1: running, 2: swimming, 3: cycling, 4: fitness, 5: reserved 6: rope skipping 7: basketball 8: walking 9: badminton 10: soccer 11: hiking 12: table tennis
        public long beginDate; // start time
        public float distance; // total distance bracelet
        public float distances; // total distance bracelet mileage distance / 1000
        public int calorie;// calories
        public String minkm; // pace
        public int heart; // heart rate
        public int runTime; // exercise duration
        public float kmh; // speed
        public String mapCoordinatesList; // mapStartEndCoordinates Track
        public Boolean isUpload; // whether to upload
}
YCBTClient.deviceToApp(new BleDeviceToAppDataResponse() {
    @Override
    public void onDataResponse(int i, HashMap hashMap) {
            if (hashMap != null) {
                    if (i == 0) {
                            int dataType = (int) hashMap.get("dataType");
                            int data = -1;
                            if (hashMap.get("data") != null)
                                    data = (int) hashMap.get("data");
                            switch (dataType) {
                                    case Constants.DATATYPE.DeviceSportModeControl:
                                            if (hashMap.get("datas") != null) {
                                                    // Determine if it is a command to turn on motion, open the motion interface
                                                    if (((byte[]) hashMap.get("datas")).length >= 2 && ((byte[]) hashMap.get("datas"))[0] == 1) {
                                                            Intentintent = new Intent(context, SportRunningActivity.class);
                                                            intent.putExtra("Title", ((byte[]) hashMap.get("datas"))[1] & 0xff);
                                                            startActivity(intent);
                                                    } else {
                                                            //update campaign data
                                                            EventBus.getDefault().post(new EventBusExitExerciseEvent((byte[]) hashMap.get("datas")));
                                                    }

                                            break;
                            }
                    }
            }
    }
});
```

```
/**
  * Read real time data of bracelet movement
  *
  * @param type 0x00: off 0x01:on
  * @param dataResponse
  */
public static void appRealSportFromDevice(int type, final BleDataResponse dataResponse)
YCBTClient.appRealSportFromDevice(type, new BleDataResponse() {
        @Override
        public void onDataResponse(int i, float v, HashMap hashMap) {
        }
});
if (type == 1) {
        YCBTClient.appRegisterRealDataCallBack(new BleRealDataResponse() {
                @Override
                public void onRealDataResponse(int i, HashMap hashMap) {
                        if (i == Constants.DATATYPE.Real_UploadSport) {
                                if (hashMap != null && hashMap.size() > 0) {
                                        sportStep = (int) hashMap.get("sportStep");// steps
                                        sportDistance = (int) hashMap.get("sportDistance");//Distance
                                        sportCalorie = (int) hashMap.get("sportCalorie");// calories
                                }
                        }

        });
}
```

8.2 Motion history data

Most of the devices do not record the data related to the movement from the start of operation and require the app to handle it by itself.
For some customized devices, motion data is saved.

9 Settings

9.1 Shutdown, reset, reboot

```
/**
  * Shutdown, reboot, enter transport mode control
  * In transport mode, it will be very power-saving and need to be charged to turn on.
  * @param type 0x01: shutdown 0x02: enter transport mode 0x03: reboot 0x04:device active disconn
  * @param dataResponse 0x00: setup successful 0x01: setup failed
  */
public static void appShutDown(int type, BleDataResponse dataResponse)
YCBTClient.appShutDown(1,dataResponse);
```

9.2 Restoring factory settings

```
/***
  * Restore factory settings
  * @param dataResponse 0x00: setup successful 0x01: setup failed
  */
public static void settingRestoreFactory(BleDataResponse dataResponse)
YCBTClient.setRestoreFactory(dataResponse);
```

9.3 Emptying the data queue

The sdk instructions are stored in a queue, first-in-first-out, and you can call this method to clear all instructions

```
/***
  * Empty the data que
```

```
 */
public static void resetQueue()
YCBTClient.resetQueue();
```

9.4 Setting the language

```
/***
  * Language setting
  * @param langType 0x00:English 0x01: Chinese 0x02: Russian 0x03: German 0x04: French
  * 0x05: Japanese 0x06: Spanish 0x07: Italian 0x08: Portuguese
  * 0x09: Korean 0x0A: Polish 0x0B: Malay 0x0C: Traditional Chinese0xFF:Other
  * @param dataResponse 0x00: Setting Success 0x01: Setting Failure
  */
public static void settingLanguage(int langType, BleDataResponse dataResponse)
YCBTClient.setRestoreFactory(1,dataResponse);
```

9.5 Setting units

```
/***
  * Unit setting (set to 0 for unwanted settings)
  * @param distanceUnit 0x00:km 0x01:mile distance
  * @param weightUnit 0x00:kg 0x01:lb 0x02:st body weight
  * @param temperatureUnit 0x00: °C 0x01: °F temperature
  * @param timeFormat 0x00:24 hours 0x01:12 hours timeFormat
  * @param dataResponse 0x00:Success 0x01:Failure
  */
public static void settingUnit(int distanceUnit, int weightUnit, int temperatureUnit, int timeFormat, BleDataResponse dataResponse)
```

9.6 Targeting

Failure is returned if the parameter is incorrectly assigned, or if the device does not support the target setting.

```
/***
  * Goal Settings
  * @param goalType 0x00:Steps 0x01:Calories 0x02:Distance 0x03:Sleep 0x04:ExerciseTime 0x05:ValidSteps
  * @param target Goal value (0x00 if type is Sleep)
  * @param sleepHour Sleep goal:Hour (0x00 if type isnon-sleep (type 0x00)
  * @param sleepMinute Sleep goal:minutes (type 0x00)
  * @param dataResponse
  */
public static void settingGoal(int goalType, int target, sleepHour, int sleepMinute, BleDataResponse dataResponse)
YCBTClient.setGoal(0x00, 1000, 0x00, 0x00,dataResponse);
```

9.7 User information settings

```
/***
  * UserInfo setting
  * @param height height cm 50 ~ 255
  * @param weight weight(kg)
  * @param sex gender 0 male 1 female
  * @param age age 1~150
  * @param dataResponse
  */
public static void settingUserInfo(int height, int weight, int sex, int age, BleDataResponse dataResponse)
YCBTClient.setUserInfo(170,50,1,30,dataResponse);
```

9.8 Setting up a sedentary reminder

```
/***
  * Set sedentary reminder
  * @param time1StartHour time period 1 start hour
  * @param time1StartMin time period 1 start minute
  * @param time1EndHour time period 1 end hour
  * @param time1EndMin time period 1 end minute
  * @paramtime2StartHour time2StartHour
  * @param time2StartMin time2StartMin
  * @param time2EndHour time2EndHour
  * @param time2EndMin time2EndMin
  * @param intervalTime interval time in minutes
  * @param repeat Week repeat
  * bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
  * total switch Sunday Saturday Friday Thursday Wednesday Tuesday Monday
  * 0 off 1 on
  * @param dataResponse
  */
public static void settingLongsite(inttime1StartHour, int time1StartMin, int time                                        1EndHour, int time1EndMin,
int time2StartHour, int time2StartMin, int tim
                                                        int intervalTime, int repeat, BleDataResponse dataResponse)
YCBTClient.setLongsite(mSedentaryAmStartHour, mSedentaryAmStartMin, mSedentaryAmEndHour, mSedentaryAmEndMin, mSedentaryPmStartHour , mSedentaryPmStartMin, mSedentaryPmEndHour, mSedentaryPmEndMin, mSedentaryIntervalTime,
mWeek,dataResponse);
```

9.9 Notification Reminder Switch Settings

```
/***
  * Notification Reminder Switch Settings
  * @param on Notification Reminder Master Switch 0x00: Off 0x01: On
  * @param sub1 Reminder Item Sub-Switch Bit7:Incoming Calls Bit6:Messenger Bit5:Email Bit4:WeChat Bit3:QQ Bit2:Sina Weibo Bit1:facebook Bit0:twitter:Off1:on
  * @param sub2 reminder sub-switch Bit7:Messenger Bit6:WhatsAPP Bit5:Linked in Bit4:Instagram Bit3:Skype Bit2:Line Bit1:Snapchat Bit0:Telegram: off 1:on
  * @param sub3 Reminder item switch Bit7:Reserved Bit6:Reserved Bit5:Reserved Bit4:Reserved Bit3:Reserved Bit2:Reserved Bit1:Viber Bit0:Other 0: Off 1: On
  * @param dataResponse
  */
public static void settingNotify (int on, int sub1, int sub2, int sub3, BleDataResponse dataResponse)
YCBTClient.setNotify(on, mMessageOne, mMessageTwo, mMessageThree,dataResponse);
```

9.10 Loss Prevention Alerts

Loss prevention means that the bracelet vibrates when the signal connecting the device to the cell phone becomes weak or is disconnected. The last three values in the anti-loss type have the same effect.

```
/***
  * AntiLose alert setting
  * @param type 0x00: No AntiLose 0x01: Near AntiLose 0x02: Medium AntiLose (default) 0x03: Far AntiLose
  * @param dataResponse 0x00: Success 0x01: Failure
```

```
 */
public static void settingAntiLose (int type, BleDataResponse dataResponse)
YCBTClient.setAntiLose(3, dataResponse);
```

9.11 Health monitoring

Health monitoring means that the device will measure the corresponding data at a fixed time and save it.
For health monitoring, heart rate and temperature are common functions that can be used in most scenarios. Others, such as oxygen monitoring, may only be used by individual devices.

9.11.1 Heart rate monitoring

```
/****
 * Heart rate monitoring mode setting
 * @param mode mode 0x00: manual mode 0x01: auto mode
 * @param intervalTime Heart rate monitoring interval in auto mode (1-60 minutes)
 * @param dataResponse
 */
public static void settingHeartMonitor(int mode, int intervalTime, BleDataResponse dataResponse)
YCBTClient.setHeartMonitor(0x01, 60, dataResponse);
```

9.11.2 Temperature monitoring

```
/**
 * Temperature Monitoring
 *
 * @param on_off Temperature Monitor Switch
 * @param interval Interval Minutes
 * @param dataResponse
 */
public static void settingTemperatureMonitor(boolean on_off, int interval, BleDataResponse dataResponse)
YCBTClient.setTemperatureMonitor(true, 60, dataResponse);
```

9.11.3 Ambient light detection

```
/**
 * Ambient Light Detection Setting
 *
 * @param on_off Ambient Light Detection Switch
 * @param interval Ambient Light Monitoring Interval (minutes)
 * @param dataResponse
 */
public static void settingAmbientLight(boolean on_off, int interval, BleDataResponse dataResponse)
YCBTClient.setAmbientLight(true, 60, dataResponse);
```

9.11.4 Oxygen monitoring

```
/**
 * Blood Oxygen Mode Setting
 *
 * @param on_off Blood Oxygen Mode Switch
 * @param interval interval minutes
 * @param dataResponse
 */
public static void settingBloodOxygenModeMonitor(boolean on_off, int interval, BleDataResponse dataResponse)
YCBTClient.setBloodOxygenModeMonitor(true, 60, dataResponse);
```

9.11.5 Environmental temperature and humidity monitoring

```
/**
 * AmbientTemperatureAndHumidityDetectionModeSetting
 *
 * @param on_off AmbientTemperatureAndHumidityDetectionModeSwitch
 * @param interval interval minutes
 * @param dataResponse
 */
public static void settingAmbientTemperatureAndHumidity(boolean on_off, int interval, BleDataResponse dataResponse)
YCBTClient.setAmbientTemperatureAndHumidity(true, 60, dataResponse);
```

9.12 Health value warning

9.12.1 Alerts on Cardiovascular Rate

```
/***
 * Heart rate alarm setting
 * @param on Alarm switch 0x00: off
 * @param highHeart Highest heart rate alarm threshold 100 - 240
 * @param lowHeart Lowest heart rate alarm threshold 30 - 60
 * @param dataResponse
 */
public static void settingHeartAlarm(int on, int highHeart, int lowHeart, BleDataResponse dataResponse)
YCBTClient.setHeartAlarm(1, 100, 30, dataResponse);
```

9.12.2 Temperature alarms

```
/**
 * Temperature Alarm
 *
 * @param on_off Temperature Alarm Switch
 * @param temp Temperature Alarm Upper Limit (-127 - 127)
 * @param dataResponse
 */
public static void settingTemperatureAlarm(boolean on_off , int temp, BleDataResponse dataResponse)
YCBTClient.setTemperatureAlarm(true, 38, dataResponse);
```

9.12.3 Early Warning of Shore Pressure

```
/**
 * Blood Pressure Alarm Settings
 *
 * @param on_off Alarm Switch 0x00:off 0x01:on
 * @param maxSBP Maximum Systolic Blood Pressure Alarm Threshold 140 - 250 mmHg
 * @param maxDBP Maximum Diastolic Blood Pressure Alarm Threshold 90 - 160 mmHg
 * @param minSBPMinimum Systolic Blood Pressure Alarm Threshold 50 - 90 mmHg
 * @param minDBP Minimum Diastolic Blood Pressure Alarm Threshold 40 - 60 mmHg
 * @param dataResponse
 */
public static void settingBloodAlarm(int on_off, int maxSBP, int maxDBP, int minSBP, int minDBP, BleDataResponse dataResponse)
YCBTClient.setBloodAlarm(1, 140,90,50,40,dataResponse);
```

9.12.4 Hypoxia warning

```
/**
 * Blood Oxygen Alarm Setting
 *
 * @param on_off 0x00: off 0x01:on
 * @param minBloodOxygen Minimum Blood Oxygen Alarm Threshold 80- 95
 * @param dataResponse
 */
public static void settingBloodOxygenAlarm(int on_off, int minBloodOxygen, BleDataResponse dataResponse)
YCBTClient.setBloodOxygenAlarm(1, 80,dataResponse);
```

9.12.5 Early warning of respiratory rate

```
/**
 * Respiratory Rate Alarm
 *
 * @param onOff type 0x00: off 0x01: on
 * @param high Highest Respiratory Rate Alarm Threshold
 * @param low Lowest Respiratory Rate Alarm Threshold
 * @param dataResponse code 0x00: Setting Successful 0x01: Setting Failed
 */
public static void settingRespiratoryRateAlarm(int onOff, int high, int low, BleDataResponse dataResponse)
YCBTClient.setRespiratoryRateAlarm(1, 30, 6,dataResponse);
```

9.13 Do Not Disturb Settings

When the watch is in Do Not Disturb mode, all the reminder functions do not work.

```
/**
 * Do Not Disturb mode setting
 *
 * @param on 0x00:off 0x01:on
 * @param startTimeHour startTimeHour: hour
 * @param startTimeMin startTimeMin: minutes
 * @param endTimeHour endTimeHour: hour
 * @param endTimeMin endTimeMin: minutes
 * @param dataResponse code 0x00: set successful 0x01: set failed
 */
public static void settingNotDisturb(int on,
                                     int startTimeHour, int startTimeMin,
                                     int endTimeHour int startTimeHour, int startTimeMin,    int endTimeHour, int endTimeMin, BleDataResponse dataResponse)
YCBTClient.setNotDisturb(1, 6, 30, 16, 30, dataResponse);
```

9.14 Wrist up screen switch

```
/***
 * Wrist Raise Raise Screen Switch Setting
 * @param on 0x00:off 0x01: on
 * @param dataResponse code 0x00: Setting Successful 0x01: Setting Failure
```

```
  */
public static void settingRaiseScreen(int on, BleDataResponse dataResponse)
YCBTClient.setRaiseScreen(1,dataResponse);
```

9.15 Screen Settings

9.15.1 Screen brightness

```
/***
  * Display Brightness setting
  * @param level 0x00:Low 0x01: Medium 0x02: High 0x03: Auto 0x04: Lower 0x05: Higher
  * @param dataResponse code 0x00: Setting Success 0x01: Setting Failure
  */
public static voidsettingDisplayBrightness(int level, BleDataResponse dataResponse)

  public class DisplayBrightness{
        public static final int Low = 0;
        public staticfinal int Middle = 1;
        public static final int High = 2;
        public static final int Auto = 3;
        public static final int Lower = 4;
        public static final int Higher = 5;
}
YCBTClient.setDisplayBrightness(Constants.DisplayBrightness.Middle,dataResponse);
```

9.15.2 Dormant screen time

Note that the time interval is not a specific value, but PauseTime.

```
/**
  * Pause time setting
  *
  * @param level 0x00: 5s 0x01: 10s 0x02: 15s 0x03: 30s
  * @param dataResponse
  */
public static void settingScreenTime(int level,BleDataResponse dataResponse)

public class PauseTime{
        public static final int Time5s = 0;
        public static final int Time10s = 1;
        public static final intTime15s = 2;
        public static final int Time30s = 3;
}
YCBTClient.setScreenTime(Constants.PauseTime.Time5s,dataResponse);
```

9.16 Skin color settings

The skin color setting affects the device's ability to detect health data and ECG tests, etc. In general, the darker the skin, the larger the value for users with more hair.

```
/***
  * Skin color setting
  * @param skinColor 0x00:white 0x01: yellow between white
  * 0x02: yellow 0x03: brown 0x04: brown 0x05: black 0x07: other
  * @param dataResponse
  */
```

```
public static void settingSkin(intskinColor, BleDataResponse dataResponse)

public class SkinColor {
        /**
          * white
          */
        public static final int SKIN_WHITE = 0;
        /**
          * white between yellow
          */
        public static final intSKIN_WHITE_BETWEEN_YELLOW = 1;
        /**
          * YELLOW
          */
        public static final int SKIN_YELLOW = 2;
        /**
          * BROWN
          */
        public static final int SKIN_BROWN = 3;
        /**
          * BROWN
          */
        publicstatic final int SKIN_BROWNNESS = 4;
        /**
          * BLACK
          */
        public static final int SKIN_BLACK = 5;
}
YCBTClient.setSkin(Constants.SkinColor.SKIN_BROWNNESS,dataResponse);
```

9.17 Blood Pressure Range Settings

If the measured photoelectric pressure deviates significantly from the actual pressure, the equipment can be calibrated by setting the pressure level.
Positive.
Note: If the device has a pressure calibration function, it is not necessary to use this function and the pressure calibration can be adjusted directly.

```
/***
  * Blood pressure range setting
  * @param level 0x00:low 0x01: normal 0x02: slightly high 0x03: moderately high 0x04: severely high
  * @param dataResponse
  */
public static void settingBloodRange(int level, BleDataResponse dataResponse)
YCBTClient.setBloodRange(1,dataResponse);
```

9.18 Device Name Settings

The setting name is not allowed to exceed 12 bytes, and the use of special characters is not recommended.
This method is used in the case of industrial production and is not required for the development of general applications.

```
/***
  * Device name setting
  *
  * @param dataResponse
  */
public static void settingDeviceName(String name, BleDataResponse dataResponse)
```

```
YCBTClient.setDeviceName(replace,dataResponse);
```

9.19 Theme Settings

Get Topics

```
/***
  * Get device main interface style configuration
  * @param dataResponse BleDataResponse
  */
public static void getThemeInfo(BleDataResponse dataResponse)
  YCBTClient.getThemeInfo(new BleDataResponse() {
        @Override
        public void onDataResponse(int code, float ratio, HashMap resultMap) {
        }
});
```

Setting up a theme

```
/***
  * Device main interface style configuration
  * @param themeType style(0-(N-1)), N represents the number of main interfaces supported by the device, as queried by the get command
  * @param dataResponse
  */
public static void settingMainTheme(int themeType, BleDataResponse dataResponse)
YCBTClient.setMainTheme(0,dataResponse);
```

9.20 Reminder Settings

9.20.1 Setting the sleep reminder time

After successful setup, the device will vibrate to display the sleep reminder screen when the current time enters the reminder time.

```
/***
  * setSleepRemind time
  * @param startHour hour
  * @param startMin minutes
  * @param repeat weekly repeat
  * bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
  * totalSwitch Sunday Saturday Friday Thursday Wednesday Tuesday Monday
  * @paramdataResponse
  */
public static void settingSleepRemind(int startHour, int startMin, int repeat, BleDataResponse dataResponse)
YCBTClient.settingSleepRemind(1, 1, 0xff,dataResponse);
```

9.20.2 Disconnect or motion reminder settings for reaching standards

When the device reaches the set alert condition, a vibration will be generated.

```
/**
  * Bracelet Status Alert Setting
  *
  * @param on_off Bracelet Status Alert Switch
  * @param type Alert Type 0x00: Bluetooth Disconnected Alert 0x01: Exercise Attainment Alert
  * @param dataResponse
  */
public static void settingBraceletStatusAlert(boolean on_off, int type, BleDataResponse dataResponse)
```

```
YCBTClient.settingBraceletStatusAlert(1,1,dataResponse);
```

9.20.3 Upload reminders

Use to indicate that an upload reminder is generated when the data stored on the device reaches a specified ratio.

```
/**
  * Upload Reminder Setting
  *
  * @param on_off Upload Reminder Switch
  * @param threshold Storage Threshold
  * @param dataResponse
  */
public static void settingUploadReminder(boolean on_off, int threshold, BleDataResponse dataResponse)
YCBTClient.settingSleepRemind(true, 80,dataResponse);
```

9.21 Data Acquisition Configuration

Not all devices support the types listed in the file.
Note the units and values of the last two parameters in the method
The file lists two methods, the second of which is for certain customized devices, and the parts of the two methods are
The parameters are not the same.

```
/***
  * Data collection configuration
  * @param on 0x01: on 0x00: off
  * @param type 0x00: PPG 0x01: Acceleration data 0x02: ECG 0x03: Temperature and humidity 0x04: Ambient light 0x05: Temperature
  * @param collectLong The length of each collection (in seconds) (0 when closed)
  * @param collectInterval collectInterval(unit:minutes) (close fill 0)
  * @param dataResponse
  */
public static void settingDataCollect(int on, int type, int collectLong, int collectInterval, BleDataResponse dataResponse)
/**
  * Data acquisition configuration in different working modes
  *
  * @param mode mode 0x00: set to normal working mode acquisition configuration 0x01: set to care working mode acquisition configuration 0x02: set to power saving working mode acquisition configuration 0x03: set to motion mode acquisition configuration
  * @param type data type 0x00: PPG, 0x01.Six Axis, 0x02: ECG, 0x03: Temperature and Humidity, 0x04: Ambient Light, 0x05: Body Temperature, 0x06: Heart Rate
  * @param secondTime The length of each acquisition (unit: second) (0 when off)
  * @param interval The interval of acquisition (unit: minute) (0 when off)
  * @dataResponse
  */
public static void settingConfigInDifWorkingModes(int mode, int type, int secondTime, int interval, BleDataResponse dataResponse )
YCBTClient.setDataCollect(1, 0x00, 90, 60, dataResponse);

  YCBTClient.setConfigInDifWorkingModes(0,1, 0x00, 90, 60, dataResponse);
```

9.22 Working Mode Switching

When the device enters a different operating mode, the monitoring time or sampling frequency of the device may change.

```
/**
  * Working Mode Switching Setting
  *
  * @param level 0x00: Set to Normal Working Mode 0x01: Set to Care Working Mode 0x02: Set to Power Saving Working Mode 0x03: Set to Custom Working Mode
  * @param dataResponse
```

```
 */
public static void settingWorkingMode(int level, BleDataResponse dataResponse)
YCBTClient.setWorkingMode(3,dataResponse);
```

9.23 Accident monitoring switch (reserved)

No device is supported for the time being.

```
/**
  * Accident monitoring mode setting
  *
  * @param on_off Accident monitoring mode switch
  * @param dataResponse
  */
public static void settingAccidentMode(boolean on_off, BleDataResponse dataResponse)
YCBTClient.setAccidentMode(true,dataResponse);
```

9.24 Pedometer Time Setting

Set the pedometer frequency of the device, the fixed values are 10, 5, 1, the unit is minutes.

```
/**
  * settingStepCountingStateTime
  *
  * @param interval CountingStateTime minutes
  * @param dataResponse
  */
public static void settingStepCountingStateTime(int interval, BleDataResponse dataResponse)
YCBTClient.setStepCountingStateTime(10,dataResponse);
```

9.25 Bluetooth Interval Setting

Note that the unit and value of the time interval in the parameter, the value must be an integer multiple of 0.625ms.

```
/**
  * setBluetoothBroadcastInterval
  *
  * @param time BroadcastInterval Time Unit ms 20 ~ 10240ms
  * @param dataResponse
  */
public static void settingBluetoothBroadcastInterval(int time, BleDataResponse dataResponse)
YCBTClient.setBluetoothBroadcastInterval(2000,dataResponse);
```

9.26 Bluetooth Transmit Power Settings

The transmit power should preferably not be negative and needs to be greater than or equal to 0 DBM.

```
/**
  * setBluetoothTransmittingPower
  *
  * @param power Transmitting Power >= 0 dbm
  * @param dataResponse
  */
public static void settingBluetoothTransmittingPower(int power, BleDataResponse dataResponse)
YCBTClient.setBluetoothTransmittingPower(0,dataResponse);
```

9.27 Exercise Heart Rate Interval Setting

Setting the心率 range for different types of exercise.

```
/**
  * ExerciseHeartRateZone setting
  *
  * @param type 0:RecuperateHeartRate, 1:RecreationalWarmup, 2:CardioStrength, 3:FatLossSculpting, 4:ExerciseLimit, 5:Empty
  * @param minHeart MinHeart rate in this mode
  * @param maxHeart MaxHeart rate in this mode
  * @param dataResponse
  */
public static void settingExerciseHeartRateZone(int type, int minHeart, int maxHeart, BleDataResponse dataResponse)
YCBTClient.setExerciseHeartRateZone(0,60,100,dataResponse);
```

9.28 Logo switch for the insurance sector

Only control the display of the device's safety function interface.

```
/**
  * Insurance interface display switch control
  *
  * @param on_off 0x00: turn off insurance interface display, 0x01: turn on insurance interface display
  * @param dataResponse
  */
public static void settingInsuranceSwitch(int on_off, BleDataResponse dataResponse)
YCBTClient.setInsuranceSwitch(1,dataResponse);
```

9.29 Setting of the time of oscillation of Mardiator

You can modify the trailing vibration time, note that the unit is milliseconds.

```
/**
  * Vibration time setting
  *
  * @param type type 0x00:alarm
  * @param time vibration time in milliseconds
  * @param dataResponse
  */
public static void settingVibrationTime(int type, int time, BleDataResponse dataResponse)
YCBTClient.setVibrationTime(0,1000,dataResponse);
```

9.30 Events

The basic processing and functions of events are relatively similar to those of alarms. The biggest difference between events and alarms is that you can add note names, and events are only supported by some customized devices.

9.30.1 Event Alert Settings

The event name should be lengthy, and the time between events uses an enumerated value rather than a specific value, which is in minutes.
After successful setup, the corresponding event id will be returned, 1 ~ 10

```
/**
  * Event reminder settings
  *
  * @param code 0x01: add event reminder 0x02: delete event reminder 0x03: modify event reminder Maximum 10 events
  * @param index time id 1-10
  * @param on_off 0x00: off 0x01: on
  * @param type 0x00: alarm 0x01:Custom event
  * @param hour Response hour
  * @param min Response minute
  * @param weekRepeat Repeat
  * bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
  * Reserved Sunday Saturday Friday Thursday Tuesday Monday
  * @param interval Interval 0x00.Single 0x01:10min 0x02:20min 0x03:30min
  * @param name Name of the event Maximum 4 Chinese
  * @param dataResponse
  */
public static void settingEventReminder(int code, int index. int on_off, int type, int hour, int min, int weekRepeat, int interval, String name, BleDataResponse dataResponse)
//add event
YCBTClient.setEventReminder(1,1,1,1,1,1,0,0, "name",dataResponse);

  //delete event, by specifying the event number you can directly delete
YCBTClient.setEventReminder(2 ,1,1,1,1,1,0,0, "name",dataResponse);
```

9.30.2 Event Alert Switch Control

Sets whether or not device events take effect. If false, all events have no effect.

```
/**
  * Event Reminder Switch Control
  *
  * @param on_off 0x00: off 0x01:on
  * @param dataResponse
  */
public static void settingEventReminderSwitch(int on_off, BleDataResponse dataResponse)
YCBTClient.setEventReminderSwitch(1,dataResponse);
```

9.30.3 Getting Event Alerts

```
/***
  * Get current bracelet event reminder info
  * @param dataResponse
  */
public static void getEventReminderInfo(BleDataResponse dataResponse)
YCBTClient.getEventReminderInfo(dataResponse);
```

9.31 Time

```
/**
  * Synchronization time
  *
  * @param dataResponse
  */
public static void settingTime(BleDataResponse dataResponse)
YCBTClient.setTime(dataResponse);
```

9.32 Setting up the phone system

```
/***
  * Phone OS setting
  * @param os 0x00: Android 0x01: IOS
```

```
 * @param devVersion version number
 * @param dataResponse
 */
public static void settingAppSystem(int os,String devVersion, BleDataResponse dataResponse)
YCBTClient.setAppSystem(0, Build.VERSION.RELEASE,dataResponse);
```

9.33 Setting the device mac

```
/**
 * APP setDeviceMAC
 *
 * @param mac Device Mac address in format 12:31:0A:AB:AF:9F or 12310AABAF9F
 * @param dataResponse
 */
public static void settingDeviceMac(String mac, BleDataResponse dataResponse)
```

10 Obtaining Device Data

10.1 Obtaining Basic Information

```
/***
 * Get basic device information
 * @param dataResponse BleDataResponse Device ID, Firmware version number, Battery status, Battery level, Binding status
 */
public static void getDeviceInfo(BleDataResponse dataResponse)
YCBTClient.getDeviceInfo(new BleDataResponse() {
        @Override
        public void onDataResponse(int code, float v, HashMap resultMap) {
            if (code == 0) {
                if ( resultMap != null) {
                    String backVal = resultMap.toString();
                    Gson gson = new Gson();

                    //Product type, 0x00:Watch/Bracelet 0x01:Ring
                    String type = (String) YCBTClient.readDeviceInfo(Constants.FunctionConstant.DEVICETYPE);

                    BandBaseInfo bandBaseInfo = gson.fromJson(backVal, BandBaseInfo.class);
        }
});

public classBandBaseInfo {
        private int dataType;// data
        private int code;// return code
        private BandBaseInfoModel data;

        public static class BandBaseInfoModel{
                privateString deviceBatteryValue;//battery level
                private String deviceVersion;//device firmware version
        }

  }
```

10.2 Getting Alarm Clock Information

```
/**
  * Query Alarm Clock
  *
  * @param dataResponse
  */
public static void settingGetAllAlarm(BleDataResponse dataResponse)
YCBTClient.setGetAllAlarm(new BleDataResponse() {
        @Override
        public void onDataResponse(int code, float ratio, HashMap resultMap) {
                if (code == 0) {
                        List data = (ArrayList) resultMap.get("data");
                        if (mMeAlarmClock != null) {
                                mMeAlarmClock.clear();
                        }
                        if (data.size() != 0) {
                                for (int i = 0; i < data.size(); i++) {
                                        HashMap dataMap = (HashMap) data.get(i);
                                        int alarmHour = (int) dataMap.get("alarmHour"); // hour
                                        int alarmMin= (int) dataMap.get("alarmMin"); // minute
                                        int alarmDelayTime = (int) dataMap.get("alarmDelayTime"); // snooze time
                                        int alarmType = (int) dataMap.get("alarmType"); // type
                                        int alarmRepeat = (int) dataMap.get("alarmRepeat"); // alarmRepeat
                                        String mAlarmRepeat = clockRepeatToValueArray(alarmRepeat);
                                        char[] ar = mAlarmRepeat.toCharArray(); // char array
                                        mMeAlarmClock.add(new MeAlarmClock(alarmHour, alarmMin, WeekUtil.getLable(ar, context),mAlarmRepeat, ar[ar        .length - 1] + ""));
}
}
}
} }
});
```

10.2.1 Adding an alarm clock

```
/***
  * Add Alarm Clock
  * @param type 0x00:Wake up 0x01:Sleep 0x02:Exercise 0x03:Medication 0x04:Appointment 0x05:Party 0x06:Meeting 0x07:Customize
  * @param startHour Start time hour
  * @param startMin Start time minute
  * @paramweekRepeat Repeat & switch
  * bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
  * total switch Sunday Saturday Friday Thursday Wednesday Tuesday Monday
  * 0 off 1 on
  * @param delayTime Snooze time (in:minutes) 0-59
  * @param dataResponse Data returnResult
  */

  public static void settingAddAlarm(int type, int startHour, int startMin, int weekRepeat, int delayTime, BleDataResponse dataResponse)
```

10.2.2 Modifying the alarm clock

```
/***
  * Modify Alarm Clock
  * @param startHour Previous Alarm Hour
  * @param startMin Previous Alarm Minute
  * @param newType 0x00:Wake Up 0x01:Sleep 0x02:Exercise 0x03:Take Medication 0x04:Appointment 0x05:Party 0x06:Meeting 0x07:Customize
  * @paramnewStartHour Start time hour
  * @param newStartMin Start time minute
  * @param newWeekRepeat Repeat & on/off
  * bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
  * totalSwitch Sunday Saturday Friday Thursday Wednesday Tuesday Monday
  * 0Off1On
  * @@param newDelayTime snooze time (unit:minute) 0-59
  * @param dataResponse 0x00: setSuccess 0x01: setFail
  */
public static void settingModfiyAlarm(int startHour, int startMin,
                                      int newType,
                                      int newStartHour, int newStartMin,
                                      int newWeekRepeat, int newDelayTime, BleDataResponse dataResponse)
```

10.2.3 Deleting alarms

```
/***
  * Delete Alarm
  * @param startHour Alarm Start Hour
  * @param startMin Alarm Start Minute
  * @param dataResponse
  */
public static void settingDeleteAlarm(int startHour, int startMin, BleDataResponse dataResponse)
```

10.3 Obtaining Dividend Information

```
/***
  * Get current bracelet chip scheme
  * @param dataResponse 0x00: NRF52832 0x01: RTK8762C 0x02: RTK8762D
  */
public static void getChipScheme(BleDataResponse dataResponse)
YCBTClient.getChipScheme(new BleDataResponse() {
    @Override
    public void onDataResponse(int code, float ratio, HashMap resultMap) {
        if (code == 0 & amp;& resultMap != null) {
                chipScheme = (int) resultMap.get("chipScheme");
                if(chipScheme == Platform.Nordic){

                }
          }

});

  public class Platform {
        public static final intNordic = 0;//nordic Platform NRF52832
        public static final int Realtek = 1;//Renex Platform RTK8762C
        public static final int Realtek2 = 2;//Renex Platform RTK8762D
    public staticfinal int JieLi = 3;//JieLi platform jl701n
}
```

10.4 Equipment type

```
/***
  * Get device name or model information
  * @param dataResponse BleDataResponse Device name or model information
  */
public static void getDeviceName(BleDataResponse dataResponse)
  YCBTClient.getDeviceName(new BleDataResponse() {
        @Override
        public void onDataResponse(int i, float v, HashMap hashMap) {
                if (i == 0 && hashMap != null) {
                        String deviceName = (String) hashMap.get("data");
                }
        }
});
```

11 app sends notification to device

Message push, notification devices

```
/**
  * Message Push Notification Bracelet
  *
  * @param type 0x00: Incoming Call 0x01:SMS 0x02:Email 0x03:Other 0x04:QQ 0x05:WeChat 0x06:Sina Weibo 0x07:Twitter 0x08:Facebook 0x09:Messenger
  * 0x0A:Whatsapp0x0B:LinkedIn 0x0C:Instagram 0x0D:Skype 0x0E:Line 0x0F:Snapchat 0x10 Reject call
  * @param title Title of the message
  * @param content Content of the message The content of the message is more or less determined by the size of the screen and the font size
  * @paramdataResponse 0x00: Success 0x01: Fail
  */
public static void appSengMessageToDevice(final int type, final String title, final String content, final BleDataResponse dataResponse)
YCBTClient.appSengMessageToDevice(0x01, "title", "content",dataResponse);
```

12 Control equipment

12.1 Finding equipment

After the method is adjusted, the watch will vibrate.

```
/**
  * Find device
  *
  * @param mode 0x01: start finding device 0x00: end finding device
  * @param remindNum number of device reminders
  * @param remindInterval remindInterval seconds
  * @param dataResponse 0x00: success 0x01: failure
  */
  public static void appFindDevice(int mode, int remindNum, int remindInterval, BleDataResponse dataResponse)
```

```
// On, 5 reminders with 2 seconds interval
YCBTClient.appFindDevice(0x01, 5, 2, bleDataResponse);
```

12.2 Tamponade Calibration

Gauge calibration refers to the calibration of the photoelectric gauge. After performing gauge calibration, it is not necessary to use the gauge level setting, i.e., only one of the two is used, and gauge calibration has a high priority.

```
/****
  * Blood Pressure Cal
  * @param sbp systolic dbp diastolic
  * @param dataResponse 0x00: Calibration Successful 0x01: Calibration Failed - Parameter Error 0x02: Calibration Failed - Device is not testing the Blood Pressure or the Blood Pressure is not yet out of value
  */
public static void appBloodCalibration(int sbp, int dbp, BleDataResponse dataResponse)
YCBTClient.appBloodCalibration(SBP, DBP,bleDataResponse);
```

12.3 Temperature Calibration

Temperature calibration is used to calibrate temperature sensors for more accurate temperature measurements in the case of a device's growth, and is not normally required for the development of applications.

```
/**
  * Temperature cal
  *
  * @param tempInt Temperature integer part (-127 - 127)
  * @param tempFloat Temperature fractional part (0-99)
  * @param dataResponse
  */
public static void appTemperatureCorrect(int tempInt, int tempFloat, BleDataResponse dataResponse)
YCBTClient.appTemperatureCorrect(40, 5,bleDataResponse);
```

12.4 Modify the color of the two-dimensional body temperature code

The color of the two-dimensional temperature code can be modified for some individually customized devices.

```
/**
  * Temperature two-dimensional code control
  *
  * @param code temperature color 0x00: green 0x01: red 0x02: orange
  * @param dataResponse
  */
public static void appTemperatureCode(int code, BleDataResponse dataResponse)
YCBTClient.appTemperatureCorrect(0,bleDataResponse);
```

12.5 Weather data

Whether or not the watch supports tomorrow's weather setting can be judged based on the function attributes or return values.
Temperatures in the sky are all regulated temperatures.
Only the first six values of the sky type are available, and the values listed below are only available for some specially customized devices.
The remaining optional parameters are available only for customized devices, and null is always available for all other devices.

12.5.1 Today's weather

```
/***
  * Today's Weather Information DataResponse
  * @param lowTemp bottomTemp
  * @param highTemp topTemp
```

```
 * @param curTemp currentTemperature
 * @param type Weather Type 1(sunny), 2(cloudy), 3(wind), 4(rain), 5(Snow), 6(foggy), 0(unknown)
 * @param dataResponse
 */
public static void appTodayWeather(String lowTemp, String highTemp, String curTemp, int type, BleDataResponse dataResponse)
YCBTClient.appTodayWeather(min, max, temp, cont,bleDataResponse);
```

12.5.2 Tomorrow's weather

```
/***
 * Tomorrow's Weather Information Data Transfer
 * @param lowTemp Bottom Temperature
 * @param highTemp Highest Temperature
 * @param curTemp Current Temperature
 * @param type Weather Type 1(sunny), 2(cloudy), 3(wind), 4(rain), 5(Snow), 6(foggy), 0(unknown)
 * @param dataResponse
 */
public static void appTomorrowWeather(String lowTemp, String highTemp, String curTemp, int type, BleDataResponse dataResponse)
YCBTClient.appTomorrowWeather(tomorrow_min, tomorrow_max, "20", tomorrow_cont,bleDataResponse);
```

12.6 Acquisition of real-time data

Obtaining real-time data from devices is relatively special, turning on and receiving are in two ways, and to differentiate, the return data type and control type are not exactly the same, the demo will list the process and steps of obtaining the data, and for the content not listed, it may appear in other usage scenarios, and if it doesn't appear in the whole file, then it may not be supported or not needed.

12.6.1 Enabling access to real-time data

```
/**
 * Read the real-time data of the bracelet's movement
 *
 * @param type 0x00: off 0x01:on
 * @param mode 0x00:Steps, Miles, Calories 0x01:Heart Rate 0x07:Comprehensive Data
 * (0x02:Blood Oxygen 0x03:Blood Pressure 0x04:HRV 0x05:Respiratory Rate 0x06:Motion Mode Real-time Data TemporarilyNot used)
 * @param dataResponse
 */
public static void appRealDataFromDevice(int type, int mode, final BleDataResponse dataResponse)
YCBTClient.appRealDataFromDevice(1, 0x00, dataResponse);
```

12.6.2 Receiving real-time data from the device

```
/***
 * Register to listen for real-time data
 */
public static void appRegisterRealDataCallBack(BleRealDataResponse realDataResponse)
YCBTClient.appRegisterRealDataCallBack(new BleRealDataResponse() {
        @Override
        public void onRealDataResponse(int dataType, HashMap hashMap) {
                if(dataType == Constants.DATATYPE.Real_UploadECG){
                        List<Integer> data = (List<Integer>) hashMap.get("data");
                }
        }
});
```

12.7 Waveform Upload

This method is used internally in the SDK and may be required for special scenarios of some customized devices.
When the function is turned on, the device will report the waveform, and the SDK will send it as a notification after receiving it internally, and you need to listen to the notification. You can refer to the previous section for receiving data.

```
/****
  * Waveform upload control
  * @param enable 0x00: stop uploading 0x01: start uploading (upload packet without serial number) 0x02: start uploading (upload packet with 8-bit serial number)
  * @param type 0x00: Optical Waveform 0x01: Cardiac Waveform 0x02: Multi-axis Sensor Waveform 0x03: Ambient Light
  * @paramdataResponse 0x00: request successful 0x01: type not supported
  */
public static void appControlWave(int enable, int type, BleDataResponse dataResponse)
YCBTClient.appControlWave(1, 0x00, dataResponse);
```

12.8 Health Data Measurement

App startup measurements are accomplished in two parts: starting the test and receiving the measurement data.

12.8.1 Turning on and off measurements

After the device starts measurement, the values during measurement are reported actively. For parsing the received data, see the code demonstration in 12.6.2. After the measurement starts, the user exits the measurement screen or the measurement ends, the device reports the status, and the SDK reports the status to the user.
The state is sent out.

```
/**
  * APP start real-time measurement
  *
  * @param onOff switch 0: off 1: single measurement 2: multiple measurements (reserved)
  * @param type type 0x00:heart rate 0x01:blood pressure 0x02:blood oxygen 0x03:respiratory rate 0x04:body temperature 0x05:blood glucose 0x06:uric acid 0x07:blood ketones
  * @paramdataResponse 0x00: setup success 0x01: setup failure (parameter error)
  */
public static void appStartMeasurement(int onOff, int type, BleDataResponse dataResponse)
YCBTClient.appStartMeasurement(1, 5,dataResponse);
/***
  * Register to listen for real time data
  */
public static void appRegisterRealDataCallBack(BleRealDataResponse realDataResponse)

 YCBTClient.appRegisterRealDataCallBack(new BleRealDataResponse() {
       @Override
       public void onRealDataResponse(int dataType, HashMap dataMap) {
           if (hashMap != null){
               if (dataType == Constants.DATATYPE.Real_UploadHeart) {
                   Integer heartValue = (Integer)dataMap.get("heartValue");// heart rate value
               }
               if (dataType ==Constants.DATATYPE.Real_UploadBlood) {
                   // blood pressure
                   String bloodSBP = hashMap.get("bloodSBP").toString();//systolic blood pressure
                   String bloodDBP = hashMap.get("bloodDBP").toString();//diastolic blood pressure
               }
               if (dataType == Constants.DATATYPE.Real_UploadBloodOxygen){
                    Integer bloodOxygenValue = (Integer) hashMap.get("bloodOxygenValue"); // Blood Oxygen
               }
               if (i == Constants.DATATYPE.Real_UploadComprehensive) {
                    // Axillary Temperature Measurement
                    int tempInteger = (int) hashMap.get("tempInteger");
```

```
                int tempFloat = (int) hashMap.get("tempFloat");
                double temp = Double.parseDouble(tempInteger + "."+ tempFloat); // body temperature
            }

        }
    }
});

 /**
  * Get single temperature data, after turning on the measurement, call this method in a loop to get the temperature
  */
private void readRealTemp() {
    YCBTClient.getRealTemp(new BleDataResponse() {
        @Override
        public void onDataResponse(int i, float v, HashMap hashMap) {
            if (i == 0) {
                String temp = (String) hashMap.get("tempValue");
            }
        }
    });
}

YCBTClient.deviceToApp(new BleDeviceToAppDataResponse() {
    @Override
    public void onDataResponse(int i, HashMap hashMap) {
      if (i == 0 && hashMap != null) {
        switch ((int) hashMap.get("dataType")) {
             case Constants.DATATYPE.DeviceMeasurementResult:
             if (hashMap.get("datas") != null) {
                byte[] data = (byte[]) hashMap.get("datas");
                //data[0] 0x00:HeartRate, 0x01:BloodPressure, 0x02:BloodOxygen, 0x03:BreathRate, 0x04:BodyTemperature
                //data[1] 0x00:User exited the measurement, 0x01:MeasurementSuccess,0x02:Measurement Failed
               if (data[1] == 1) {
                 //measurement ended
               }else if (data[1] == 2) {
                 //measurement failed
               }else{
                 //measurement canceled
               }
             }
              break;
        }
      }
    }
}
```

12.9 Health parameters, early warning information

When a warning message is sent, a vibration will occur when the device reaches a specified condition.

```
/***
  * Health parameters, warning information sent
  * @param warnState warningState 0x00: no warning 0x01: warning in effect
  * @param healthState healthState 0x00:unknown 0x01:excellent 0x02:good 0x03:average 0x04:poor 0x05:sick
```

```
 * @paramhealthIndex HealthIndex 0~120
 * @param friendWarn friendWarn 0x00:no warning 0x01:warning in effect Medium
 * @param dataResponse
 */
public static void appHealthArg(int warnState, int healthState, int healthIndex, int friendWarn, BleDataResponse dataResponse)
YCBTClient.appHealthArg(1, 1, 60,1,dataResponse);
```

12.10 Friends and Family News

This method is only supported by certain customized devices.

```
/**
 * Family and friends messages
 *
 * @param index The subscript of the emoticon 0-4
 * @param hour Hour
 * @param min Minute
 * @param name Family and friends nicknames
 * @param dataResponse 0x00: The bracelet shows vibration success 0x01: The bracelet is not worn 0x02: Other errors
 */
public static void appEmoticonIndex(int index, int hour, int min, String name, BleDataResponse dataResponse)
YCBTClient.appEmoticonIndex(4, 9, 40, "I'm Dabo1",dataResponse);;
```

12.11 Data write-back

This section can be used only for functions specific to certain customized devices and can be ignored for other devices.

12.11.1 Health value write-back

```
/**
 * Health value write back to bracelet
 *
 * @param healthValue healthValue
 * @param healthState healthState
 * @param dataResponse
 */
public static void appHealthWriteBack(int healthValue, String healthState, BleDataResponse dataResponse)
YCBTClient.appHealthWriteBack(1, 1, dataResponse);
```

12.11.2 Sleep Data Write Back

```
/**
 * APP sleep data write back to bracelet
 *
 * @param deepSleepTimeHour deepSleep(hour)
 * @param deepSleepTimeMin deepSleep(minute)
 * @param lightSleepTimeHour lightSleep(hour)
 * @paramlightSleepTimeMin light sleep (minutes)
 * @param totalSleepTimeHour total sleep (hours)
 * @param totalSleepTimeMin total sleep (minutes)
 * @param dataResponse 0x00: Successful reception of the bracelet 0x01: Setting failure - parameter error
 */
public static void appSleepWriteBack(int deepSleepTimeHour, int deepSleepTimeMin, int lightSleepTimeHour, int lightSleepTimeMin, totalSleepTimeHour, int totalSleepTimeMin, BleDataResponse dataResponse)
YCBTClient.appSleepWriteBack(3, 40, 2, 20, 6, 0,dataResponse);
```

12.11.3 Write-back of My Information

```
/**
  * APP user's personal information written back to the bracelet
  *
  * @param type 0x00: user insurance class information, followed by content as a string, UTF8 encoding 0x01: user membership status information, followed by content as a string, UTF8 encoding
  * @param content message content
  * @param dataResponse 0x00.User insurance class information, followed by content as a string, UTF8 encoding 0x01: User member status information, followed by content as a string, UTF8 encoding
  */
public static void appUserInfoWriteBack(int type, String content, BleDataResponse dataResponse)
YCBTClient.appUserInfoWriteBack(0, "content", dataResponse);
```

12.11.4 Upgrade Progress Writeback

```
/**
  * Writeback of upgrade progress
  *
  * @param on_off 0x00: turn off reminder 0x01: turn on reminder
  * @param percent Percentage
  * @param dataResponse 0x00: bracelet received success 0x01: setup failure-parameter error
  */
public static void appUpgradeReminder(int on_off, int percent, BleDataResponse dataResponse)
YCBTClient.appUpgradeReminder(1, 50, dataResponse);
```

12.11.5 Motion Data Write Back

```
/**
  * Exercise data writeback
  *
  * @param step Effective Steps
  * @param type Exercise Type 0x00:Recuperative Rest 0x01:Casual Warmup 0x02:Cardio Intensive 0x03:Fat Loss Plastic 0x04:Exercise Limit 0x05:Empty State
  * @param dataResponse 0x00: Bracelet Sync Successful 0x01:synchronization failed
  */
public static void appEffectiveStep(int step, int type, BleDataResponse dataResponse)
YCBTClient.appEffectiveStep(10, 0, dataResponse);
```

12.11.6 Calculating heart rate synchronization

```
/**
  * APP Calculate Heart Rate Synchronization
  *
  * @param heart Effective Heart Rate
  * @param dataResponse 0x00: Bracelet Synchronization Successful 0x01: Bracelet Synchronization Failed
  */
public static void appEffectiveHeart(int heart, BleDataResponse dataResponse)
YCBTClient.appEffectiveHeart(78, dataResponse);
```

12.11.7 Measurement data write-back

Note that all values are one value except for the value of diastolic blood pressure, which is written with systolic blood pressure in the front and diastolic blood pressure in the back, and that whatever the values are, they must be passed as an array.
If the value passed has a small number, the whole part comes first, followed by the small part.

```
/**
  * APP measurement data written back to the bracelet
  *
  * @param type 0x00: heart rate 0x01: blood pressure 0x02: blood oxygen 0x04:    respiratory rate 0x05: HRV
* @param value1 Measurement of the type of the return value of only 1byte, this byte to fill in the measured value (such as heart rate, respiratory rate, etc.), to return the two-byte (such as blood pressure, this byte to fill in the systolic pressure) *
@param value2 Measurement of the type of the return value of only 1byte, this byte to assume that 0 (such as blood pressure, this byte to fill in the
  * @param value2 Measurement type return value only 1byte, this byte default is considered 0 (e.g. heart rate, respiratory rate, etc.), return two bytes (e.g. blood pressure, this byte is filled with diastolic blood pressure)
  * @param dataResponse 2bytes 0x00: Heart Rate 0x01: Blood Pressure 0x02: Oxygen 0x04: Respiratory rate0x05: HRV 0x00 Success 0x01 Failure
  */
public static void appWritebackData(int type, int value1, int value2, BleDataResponse dataResponse)
YCBTClient.appWritebackData(1,135,94, dataResponse);
```

12.12 Sensor Data Storage Switch Control

Controls whether or not the device records relevant data, valid only for a particular device.

```
/**
  * Sensor Data Storage Switch Control
  *
  * @param type 0x00: PPG 0x01: Acceleration Data 0x02: ECG 0x03: Temperature and Humidity 0x04: Ambient Light 0x05: Body Temperature
  * @param on_off 0x00: off 0x01: on
  * @param dataResponse
  */
public static void appSensorSwitchControl(int type, int on_off, BleDataResponse dataResponse)
YCBTClient.appSensorSwitchControl(0,1, dataResponse);
```

12.13 Sending cell phone numbers

```
/**
  * Send the current phone model
  *
  * @param model Phone model
  * @param dataResponse 0x00: Push Successful 0x01: Push Failed - Parameter Error
  */
public static void appMobileModel(String model, BleDataResponse dataResponse)
YCBTClient.appMobileModel("model", dataResponse);
```

12.14 Messaging

Send message has a fixed value, only when index is 6, content passes the value, in other cases, it passes null.

```
/**
  * APP Information Push
  *
  * @param type Type 0x00 There is a new weekly report generated, please check it on the APP.
  * 0x01 There is a new monthly report generated, please go to APP to che
  * 0x02 Received a message from friends and family, please go to APP
  * 0x03 It's been a long time since you took a measurement, take a measurement.
  * 0x04 You have successfully booked a consultation.
  * 0x05 The consultation you have booked will start in one hour.
  * 0x06 custom content
  * @param message Message content Message content is available when type is 0x06
  * @param dataResponse 0x00: bracelet synchronization success 0x01: bracelet synchronization failure
  */
public static void appPushMessage(int type, String message, BleDataResponse dataResponse)
YCBTClient.appPushMessage(0,null, dataResponse);
```

12.15 Calibration of ambient temperature and humidity (reserved)

Calibration of the relevant sensors for more accurate measurements

```
/**
  * Temperature and Humidity Cal
  *
  * @param tempInt Temperature integer part (-127 - 127)
  * @param tempFloat Temperature fractional part (0-9)
  * @param humidInt Humidity integer part (0-99)
  * @param humidFloat Humidity fractional part (0-9)
  * @paramdataResponse 0x00 success 0x01 failure
  */
public static void appPushTempAndHumidCalibration(int tempInt, int tempFloat, int humidInt, int humidFloat, BleDataResponse dataResponse)
YCBTClient.appPushTempAndHumidCalibration(37,8,30,0, dataResponse);
```

12.16 Methylene glycol calibration

Glucose calibration is used to calibrate the data of glucose measurements, and when this method is performed, the glucose test results will be more accurate.

```
/****
  * Blood Sugar Cal
  * @param bloodSugarInteger Blood Sugar Integer part
  * @param bloodSugarFloat Blood Sugar Fractional part
  * @param type 0:Fasting 1:After breakfast 2:Before lunch 3:After lunch 4:Before dinner 5:After dinner
  * @param dataResponse0x00: Calibration Successful 0x01: Calibration Failed - Parameter Error 0x02: Calibration Failed - Device is not testing blood pressure or blood pressure is not yet out of value
  */
public static void appBloodSugarCalibration(int bloodSugarInteger, int bloodSugarFloat, int type, BleDataResponse dataResponse)
YCBTClient.appBloodSugarCalibration(4,6,0, dataResponse);
```

12.17 Sending measured values

Note that all values are one value except for the value of diastolic blood pressure, which is written with systolic blood pressure in the front and diastolic blood pressure in the back, and that whatever the values are, they must be passed as an array.
If there are small numbers in the passed value, the whole part comes first, and the small number part comes last. Note: For this command, the range of the decimal part is 0-9.

```
/****
  * Placement of measurements
  * @param type type 0x00:Heart Rate 0x01:Blood Pressure 0x02:Blood Oxygen 0x03:Respiratory Rate 0x04:HRV 0x05:Blood Sugar 0x06:Temperature
  * @param timestamp timestamp
  * @param valueInteger Integer portion of the value Blood Pressure:Systolic Blood Pressure SBP
  *@param valueFloat Decimal part of value Default is 0 Blood Pressure:Diastolic Blood Pressure DBP Temperature:Fractional part Blood Glucose:Fractional part
  * @param dataResponse code 0x00:Success in placing command 0x01:Failure in placing command
  * byte[] data = hashMap.get("datas") data[0] means the model and type correspond to data[1] 0 means send down successfully 1 means send down failed
  */
public static void appSendMeasureNumber(int type, long timestamp, int valueInteger, int valueFloat,BleDataResponse dataResponse)

  /****
  * Send Measurement Data
  * @param type type 0x00:HeartRate 0x01:BloodPressure 0x02:BloodOxygen 0x03:RespiratoryRate 0x04:HRV 0x05:BloodGlucose 0x06:Temperature
  * @param timestamp timestamp
  * @paramvalueInteger Integer portion of value
  * @param valueFloat Decimal portion of value Default is 0 BP:Diastolic DBP Temperature:Decimal portion Glucose:Decimal portion
  * @param maxInteger Integer portion of maximum value
  * @param maxFloat Decimal portion of maximum value Default is 0 BP:DiTemperature:Decimal part Glucose:Decimal part
  * @param minInteger Integer part of the minimum value
  * @param minFloat Decimal part of the minimum value Default is 0 Blood pressure:Diastolic DBP Temperature:Decimal part Glucose:Decimal part
  * @param dataResponse code 0x00:Successful command 0x01:Command Failure
  * byte[] data = hashMap.get("datas") data[0] denotes the model and type corresponds to data[1] 0 denotes send down success 1 denotes send down failure
  */
public static void appSendMeasureNumber(int type, long timestamp, int valueInteger, int valueFloat, int maxInteger, int maxFloat, int minInteger, int minFloat, BleDataResponse dataResponse)
//heart rate 80
YCBTClient.appSendMeasureNumber(0,1670938122,80,0, dataResponse);

  //blood pressure 135/94
  YCBTClient.appSendMeasureNumber(1,1670938122,135, 94, dataResponse);    //glucose Current value 4.6, Maximum value 5.3, Minimum value 3.7 YCBTClient.94, dataResponse);

  // Meringue Current value 4.6, Maximum value 5.3, Minimum value 3.7
YCBTClient.appSendMeasureNumber(5,1670938122,4,6,5,3,3,7, dataResponse)
```

12.18 Lipid calibration

Lipid calibration is used to calibrate lipid measurements. When this method is performed, the lipid results will be more accurate.

```
/**
  * Lipid calibration, lipid cal
  *
  * @param model Calibration model (reserved)
  * @param totalCholesterolInteger Total cholesterol integer portion 2-10 Unit: mmol/L
  * @param totalCholesterolFloat Total cholesterol fractional portion 0-99 (Decimal pointretain two decimal places) Unit: mmol/L
  * @param dataResponse 0x00: Calibration Successful 0x01: Calibration Failed - Parameter Error
  */
public static void appLipidCalibration(int model, int totalCholesterolInteger, int totalCholesterolFloat int totalCholesterolFloat, BleDataResponse dataResponse)
int value1 = (int)(5.55f * 100);
YCBTClient.appLipidCalibration(0, value1 / 100, value1 % 100, bleDataResponse);
```

12.19 Uric acid calibration

Uric acid calibration is used to calibrate the data from uric acid measurements. When this method is performed, the results of uric acid testing will be more accurate.

```
/**
  * Uric Acid Calibration, Uric Acid Cal
  *
  * @param model Calibration mode (reserved)
  * @param range Range 179-1190 in µmol/L
  * @param dataResponse 0x00: Calibration Successful 0x01: Calibration Failed - Parameter Error
  */
public static void appUricAcidCalibration(int model, int range, BleDataResponse dataResponse)
  YCBTClient.appUricAcidCalibration(0, (int) umolParseFloat, bleDataResponse);
```

13 Receiving Device Response

When a user operates a device or a device detects some information and the device reports the operation, the SDK listens to the response from the device and processes it according to the type of operation, and sends the operation status of the device in the form of a notification. The SDK sends the status of the operation of the device in the form of a notification, which must be listened to and parsed by the application program according to the type of notification. In addition, some of the contents are listed elsewhere and will not be shown here. If the corresponding data is not received, the device may not support this function.

```
/***
  * Listening for device to send data to App
  */
public static void deviceToApp(BleDeviceToAppDataResponse bleRealTypeResponse)
YCBTClient.deviceToApp(new BleDeviceToAppDataResponse() {
   @Override
```

```
public void onDataResponse(int i, HashMap hashMap) {
    if (hashMap != null) {
        if (i == 0) {//
            int dataType = (int) hashMap.get("dataType");
            int data = -1;
            if (hashMap.get("data") != null)
                data = (int) hashMap.get("data");
            switch (dataType) {
                case Constants.DATATYPE.AppECGPPGStatus://device real-time status upload
                    int EcgStatus = (int) hashMap.get("EcgStatus");
                    int PPGStatus = (int) hashMap.get("PPGStatus");
                    if (PPGStatus == 0) {//0 :wear 1: no wear 2: error
                    }
                    break;
                case Constants.DATATYPE.DeviceFindMobile://find mobile
                    if (data == 0) {//0x00: end
                        handler.removeCallbacks(runnable);
                        SoundPoolUtil.getInstance(MainActivity.this).stop();
                    } else if (data == 1) {//0x01: start
                        SoundPoolUtil.getInstance(MainActivity.this).play(100);
                        handler.removeCallbacks(runnable);
                        handler.postDelayed(runnable, 15000);
                    }
                    break;
                case Constants.DATATYPE.DeviceLostReminder:// anti-lost reminder
                    if (data == 0) {//0: end 1: start
                    }
                    break;
                case Constants.DATATYPE.DeviceAnswerAndClosePhone://answer/reject call
                    if (data == 0) {//0: answer 1: reject
                        answerCall();
                    } else {
                        rejectCall();
                    }
                    break;
                case Constants.DATATYPE.DeviceTakePhoto:// camera take photo control
                    if (PermissionUtil.openCameraPermission(context) && PermissionUtil.openSDCardPermission(context)) {
                        if (data == 1) { //0x00:exit photo mode 0x01:enter photo mode 0x02:take photo
                            startActivity(new Intent(MainActivity.this, CameraActivity.class));
                        } else {
                            EventBus.getDefault().post(new EventBusTakePhotoEvent(data));
                        }
                    }
                    break;
                case Constants.DATATYPE.DeviceStartMusic://Music Control
                    HealthApplication.getInstance().musicCon(data);
                    break;
                case Constants.DATATYPE.DeviceSos://Enable one-button call for help control command
                    break;
                case Constants.DATATYPE.DeviceDrinkingPatterns://Enable drinking pattern control command
                    break;
                case Constants.DATATYPE.DeviceSportModeControl:// bracelet sport mode control
                    if (hashMap.get("datas") != null) {
                        if (((byte[]) hashMap.get("datas")).length >= 2 && ((byte[]) hashMap.get("datas"))[0] == 1) {
```

```
                                        Intent intent = new Intent(context, SportRunningActivity.class);
                                        intent.putExtra("Title", ((byte[]) hashMap.get("datas"))[1] & 0xff);
                                        startActivity(intent);
                                   }else {
                                        EventBus.getDefault().post(new EventBusExitExerciseEvent((byte[]) hashMap.get("datas")));
                                   }
                              }
                              break;
                    }
               }
          }
     }
});

  public class DATATYPE{
        public static final int DeviceFindMobile = 0x0400;//find phone
        public static final int DeviceLostReminder = 0x0401;//loss prevention reminder
        public static final intDeviceAnswerAndClosePhone = 0x0402;//answer/reject calls
        public static final int DeviceTakePhoto = 0x0403;//camera photo control
        public static final int DeviceStartMusic =0x0404;//Music control
        public static final int DeviceSos = 0x0405;//One key to call for help control command
        public static final int DeviceDrinkingPatterns = 0x0406;//Drinking pattern control command
        publicstatic final int DeviceConnectOrDisconnect = 0x0407;//Bracelet Bluetooth Connect/OrDisconnect
        public static final int DeviceSportMode = 0x0408;//Bracelet Sport Mode control command
        public static finalint DeviceSyncContacts = 0x0409;//request to synchronize phone contacts
        public static final int DeviceRest = 0x040A;//notify the phone that the current bracelet is restored to factory settings
        public static final int DeviceEndECG =0x040B;//notify APP that the bracelet has ended the real-time ECG test
        public static final int DeviceSportModeControl = 0x040C;//the bracelet sports mode control
        public static final int DeviceSwitchDial = 0x040D;//Handband notifies APP to switch dial
        public static final int DeviceMeasurementResult = 0x040E;//Handband returns APP start single measurement result
        public static final int DeviceAlarmData = 0x040F;//HandbandReport warning data
        public static final int DeviceInflatedBloodMeasureResult = 0x0410;//Bracelet notify APP accurate blood pressure measurement results
        public static final int DeviceUpgradeResult = 0x0411;//Watch return device upgrade result
        public static final int DevicePPIData = 0x0412;//PPI data
        public static final int DeviceMeasurStatusAndResults = 0x0413;//Watch return invasive measurement status andresults
}
```

14 Customization-related

14.1 Obtaining CGM Device Information

Applicable: H01, this method can get the boot time and serial number of the device, other types are not supported.

```
/**
  * Get CGM start time and transmitter serial number
  * Example: YCBTClient.customizeCGM(new BleDataResponse())
  *
  * @param dataResponse code: 0x00:Success,0x01:Failu re
String) hashMap.get("serial")
  * startTime: start time timestamp, e.g.: (String) hashMap.get("startTime")
  */
public static void getCustomizeCGM(BleDataResponse dataResponse)
YCBTClient.getCustomizeCGM(dataResponse);
```

14.2 Querying and Deleting Device History Data

Application: H01

14.2.1 Querying device data

This method obtains different historical data through different types.

```
/**
  * Customize project data synchronization: 1:Register real time data listener, YCBTClient.appRegisterRealDataCallBack(); 2:Start data synchronization, YCBTClient.startCustomizeDataSync()
  *
  *
  * @param type type:0x01.CGM Blood Sugar,0x02:Physical Therapy
  * @param dataResponse code:0x00:Success,0x01:Failure
  * data:[{"offset":0, "cgm":2, "ele":0},{"offset":0, "cgm":3, "ele":0},.{"offset":0, "cgm":3, "ele":0}]
  */
public static void startCustomizeDataSync(int type, BleDataResponse dataResponse)
 YCBTClient.startCustomizeDataSync(1, dataResponse);
```

14.2.2 Deleting device data

This method allows you to delete the corresponding data by the specified type.

```
/**
  * Delete customized item data
  *
  * @param type type: 0x01:CGM Blood glucose, 0x02:Physical therapy
  * @param key delete key 0x00:Delete same day data,0x01:Delete historical data,0x02:Delete all data,0x03:Delete raw data
  * @param dataResponse code:0x00:Success,0x01:Failure-Deletion Failure,0x02:No Recorded Data,0x03: Failure-Parameter Error
  * state: Whether success,e.g.:(int) hashMap.get("state")
  * type: Type,e.g.:(String) hashMap.get("type")
  */
public static void deleteCustomizeData(int type, int key, BleDataResponse dataResponse)
YCBTClient.deleteCustomizeData(0x01, 0x02,dataResponse);
```

15 Business cards

15.1 Issuance of business cards

Distribute business cards to equipment

```
/**
    * Place business card
    * @param type type 0x00:WeChat 0x01:QQ 0x02:Facebook 0x03:Twitter 0x04:Whatsapp 0x05:Instgram 0xFO:SN code 0xF1:Static code 0xF2:Dynamic code
    * @param sqrString QR code ofUTF-8 byte code End with null byte
    * @param dataResponse 0x00:Receive Successful 0x01:Receive Failed - Parameter Error
    */
public static void appSendCardNumber(int type, String sqrString, BleDataResponse dataResponse)
YCBTClient.appSendCardNumber(0xF0, "5678",dataResponse);
```

15.2 Getting Business Card Information

Get business card information, 0x00:WeChat, 0x01:QQ , 0x02:Facebook, 0x03:Twitter, 0x04:Whatsapp, 0x05:Instgram, 0xFO:SN code, 0xF1:Static code, 0xF2:Dynamic code

```
/**
  * GetCardInfo
  * @param type type 0x00:WeChat 0x01:QQ 0x02:Facebook 0x03:Twitter 0x04:Whatsapp 0x05:Instgram 0xFO:SN code 0xF1:Static code 0xF2:Dynamic code
  * @paramdataResponse
  */
public static void getCardInfo(int type, BleDataResponse dataResponse)
YCBTClient.getCardInfo(0xF0,dataResponse);
```

15.3 Listening to dynamic codes

```
 YCBTClient.deviceToApp(new BleDeviceToAppDataResponse() {
            @Override
            public void onDataResponse(int i, HashMap hashMap) {
                if (i == 0&& hashMap != null) { //
                    Log.i(TAG,new Gson().toJson(dataMap))
                }
            }
 })
```

16 Intellectual Interest

16.1 Smart on and off

```
/**
  * Wit On and Off
  * Example: YCBTClient.setWitOnOff(1,1,new BleDataResponse())
  *
  * @param onOff switch 0x00 off,0x01 on
  * @param type 0x01:short video,0x02:music,0x03:reading,0x04:Photo/Video,0x05:SOS,0x06:Slideshow
  * @param dataResponse code: 0x00:Success,0x01:Failure
  */
public static void setWitOnOff(int onOff, int type, BleDataResponse dataResponse)
YCBTClient.setWitOnOff(isOpen ? 1 : 0, protocolIndex, bleDataResponse);
```