Development Help Document

BLUETOOTH SDK MOUDEL

Table of Contents

Introduction to SDK

This SDK is designed to assist users in interacting with scanners through Bluetooth communication using Android smartphones. It allows users to send commands to control the scanner and receive data from it.

Development Language: Java

Development Tool: Android Studio

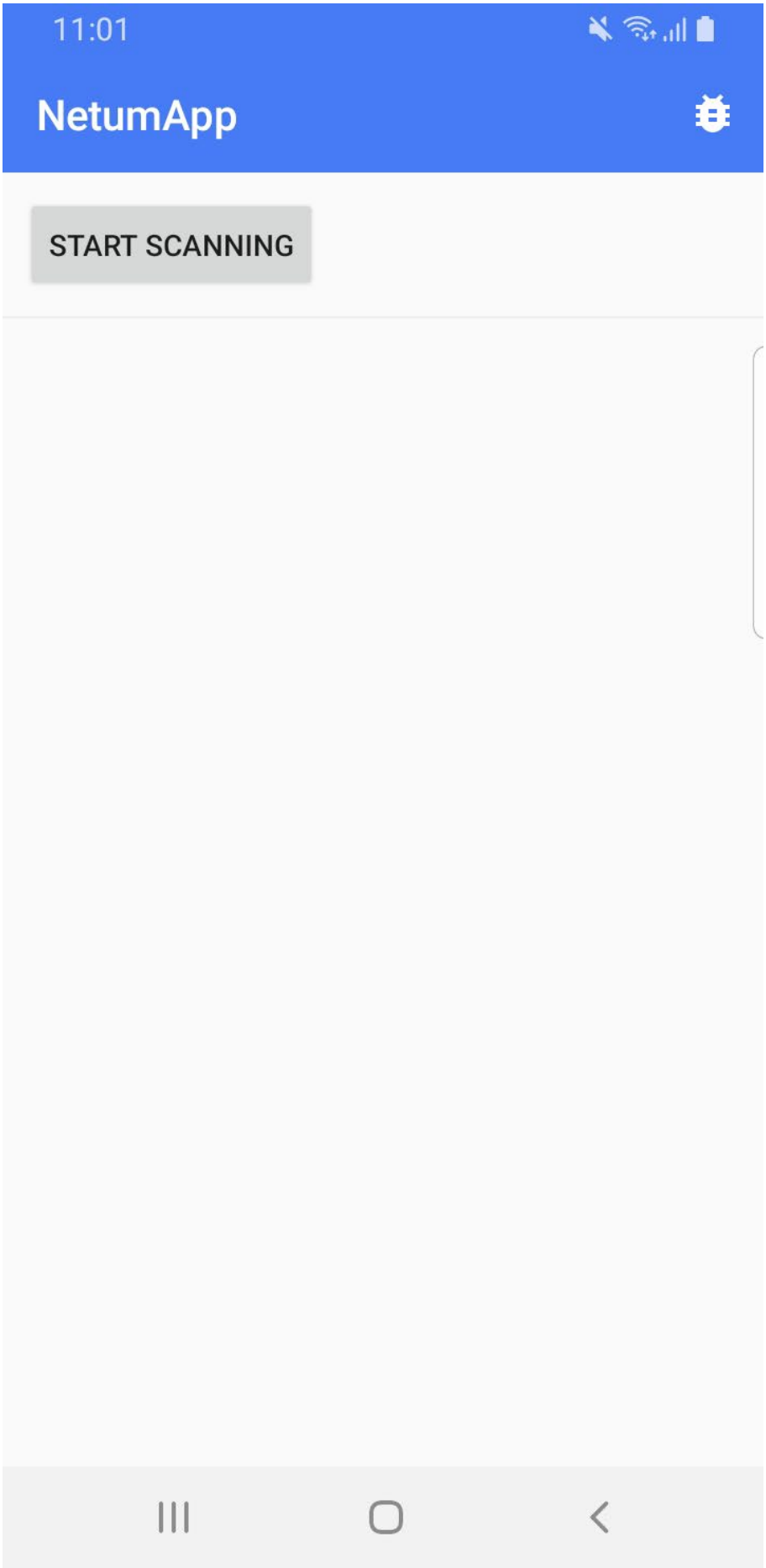
This document summarizes common scenarios for user support and demonstrates program construction using the SDK.

Program Demonstration

Program Installation

1.Unzip the SDK development package and locate the file within the folder: app -> release -> app-release.apk

2.Install the APK program on your Android device. After installation, open the software as shown below:


11:01
NetumApp
START SCANNING

Bluetooth BLE

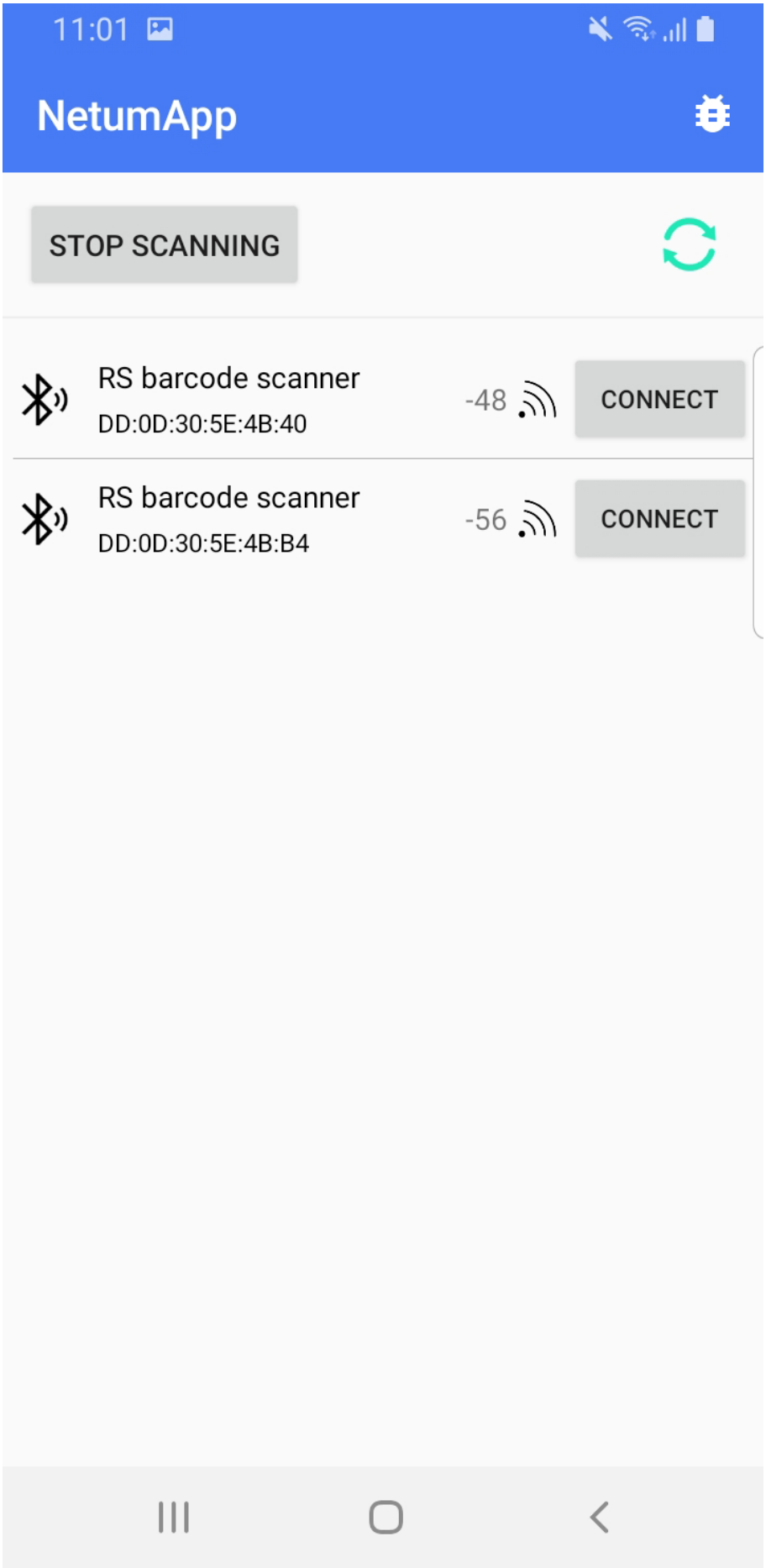
1. Use the scanner to scan the command code below to switch the scanner to Bluetooth mode.

%#IFSNO$4

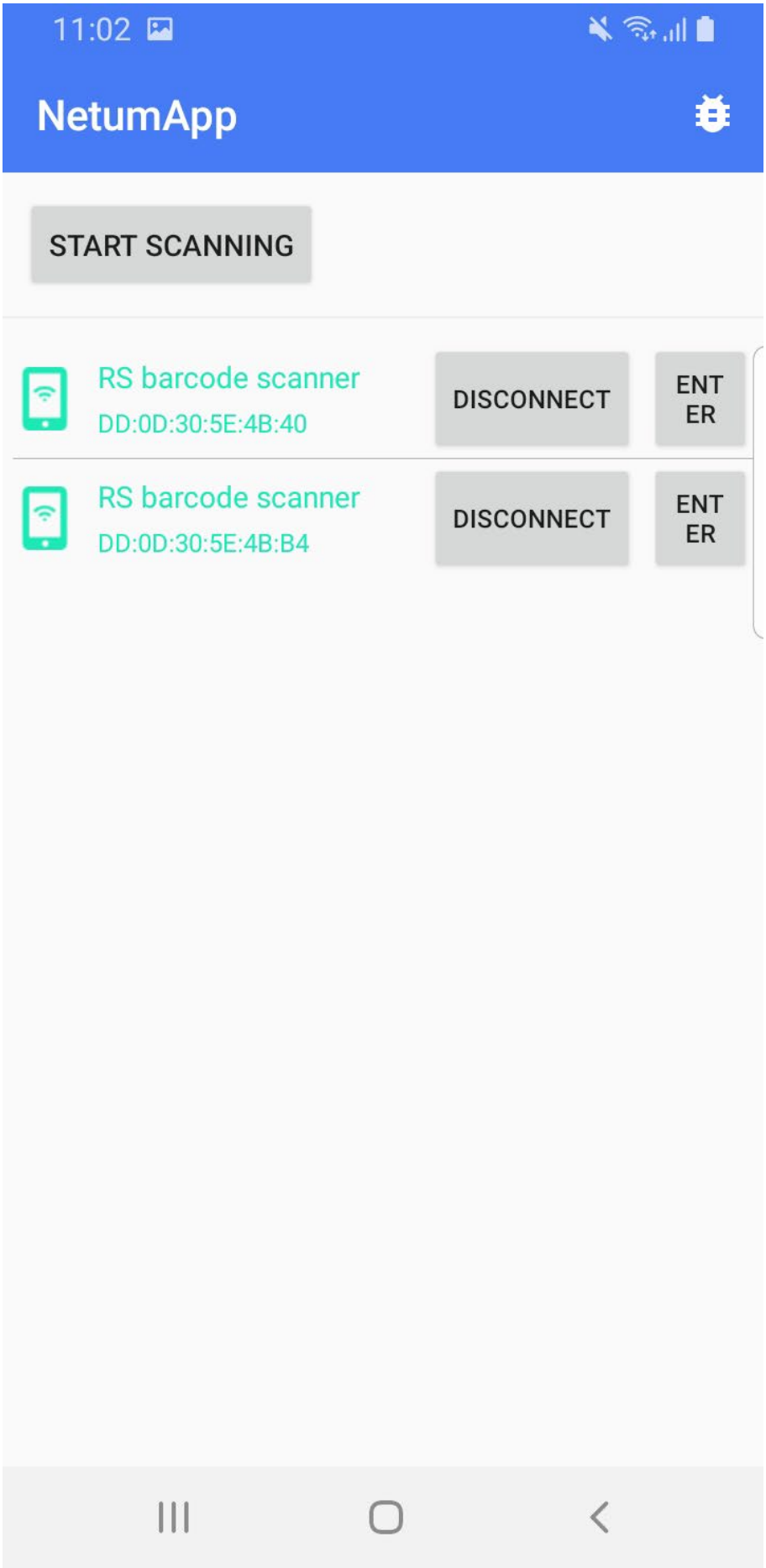
2. Use the scanner to scan the command code below to change the scanner's Bluetooth mode to Bluetooth BLE.

蓝牙 BLE 模式

3. Open the app and navigate to the Bluetooth BLE device search and pairing interface.

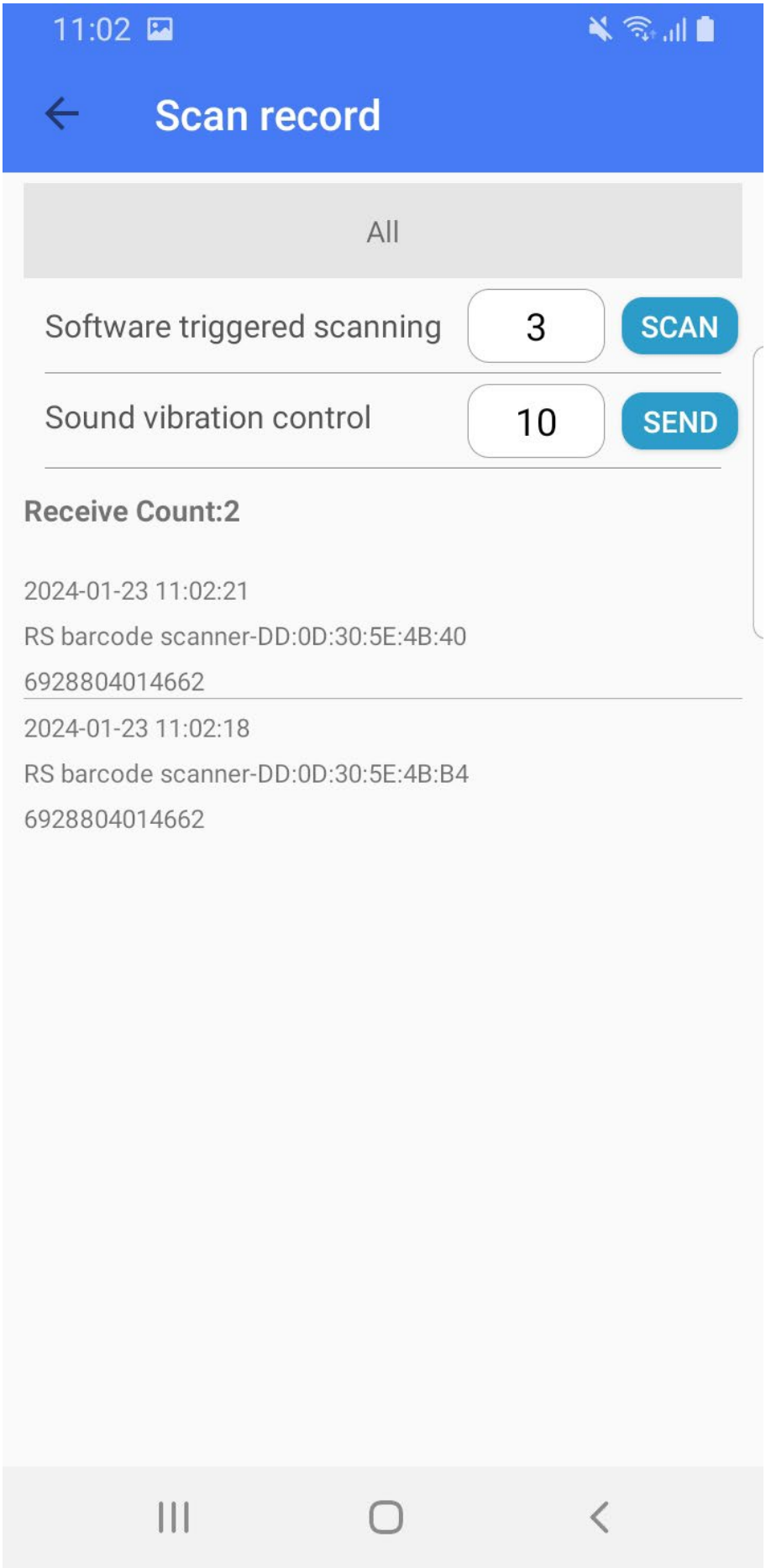

11:01
NetumApp
STOP SCANNING
RS barcode scanner
DD:0D:30:5E:4B:40
-48
CONNECT
RS barcode scanner
DD:0D:30:5E:4B:B4
-56
CONNECT

4. Select "RS barcode scanner" to connect to the scanner. Upon successful connection, the main interface displays as follows:


11:02
NetumApp
START SCANNING
RS barcode scanner
DD:0D:30:5E:4B:40
DISCONNECT
ENT
ER
RS barcode scanner
DD:0D:30:5E:4B:B4
DISCONNECT
ENT
ER

- The main interface is used to display Bluetooth devices detected and connected scanners.

5. Click on the menu option button located at the top-right corner of the main interface to access the testing and receiving interface.


11:02
Scan record
All
Software triggered scanning
3
SCAN
Sound vibration control
10
SEND
Receive Count:2
2024-01-23 11:02:21
RS barcode scanner-DD:0D:30:5E:4B:40
6928804014662
2024-01-23 11:02:18
RS barcode scanner-DD:0D:30:5E:4B:B4
6928804014662

a) Device Selection: You can choose the device you wish to control.

b) Software-triggered scanning command with a set range of 1-7 seconds.

c) Sound and vibration control with a set range of 0-26.

SDK sound(n=0x30~0x4A)	"$BUZZ#Bn"	see right beep/led table
e.g. SDK sound 0	"$BUZZ#B0"	1 high short beep
e.g. SDK sound 26	"$BUZZ#BJ"	High-high-low-low beep

Beep / LED Action	Value
1 high short beep	0
2 high short beeps	1
3 high short beeps	2
4 high short beeps	3
5 high short beeps	4
1 low short beep	5
2 low short beeps	6
3 low short beeps	7
4 low short beeps	8
5 low short beeps	9
1 high long beep	10
2 high long beeps	11
3 high long beeps	12
4 high long beeps	13
5 high long beeps	14
1 low long beep	15
2 low long beeps	16
3 low long beeps	17
4 low long beeps	18
5 low long beeps	19
Fast warble beep	20
Slow warble beep	21
High-low beep	22
Low-high beep	23
High-low-high beep	24
Low-high-low beep	25
High-high-low-low beep	26

d) Custom sound and vibration control with three parameters:

Time: Continuous action time of sound or vibration, range: 10-2540 ms.

Type: Control type, 0 = control both sound and vibration, 1 = control sound, 2 = control vibration.

Frequency: Sound frequency, range: 100-5200 Hz.

Continue beep	"$BUZZ#BKttnff"	tt=02~FF, beep 10ms~2.54s; n=0~2, 0. beep+vibration 1. only beep, 2. only vibration. ff=00~FF(100~5200HZ), freqence=100+ff*20.

e) Reception list for receiving scanner scan data.

6. Click on the connected scanner in the main interface to enter the scanner's individual control interface (under construction).

Bluetooth SPP

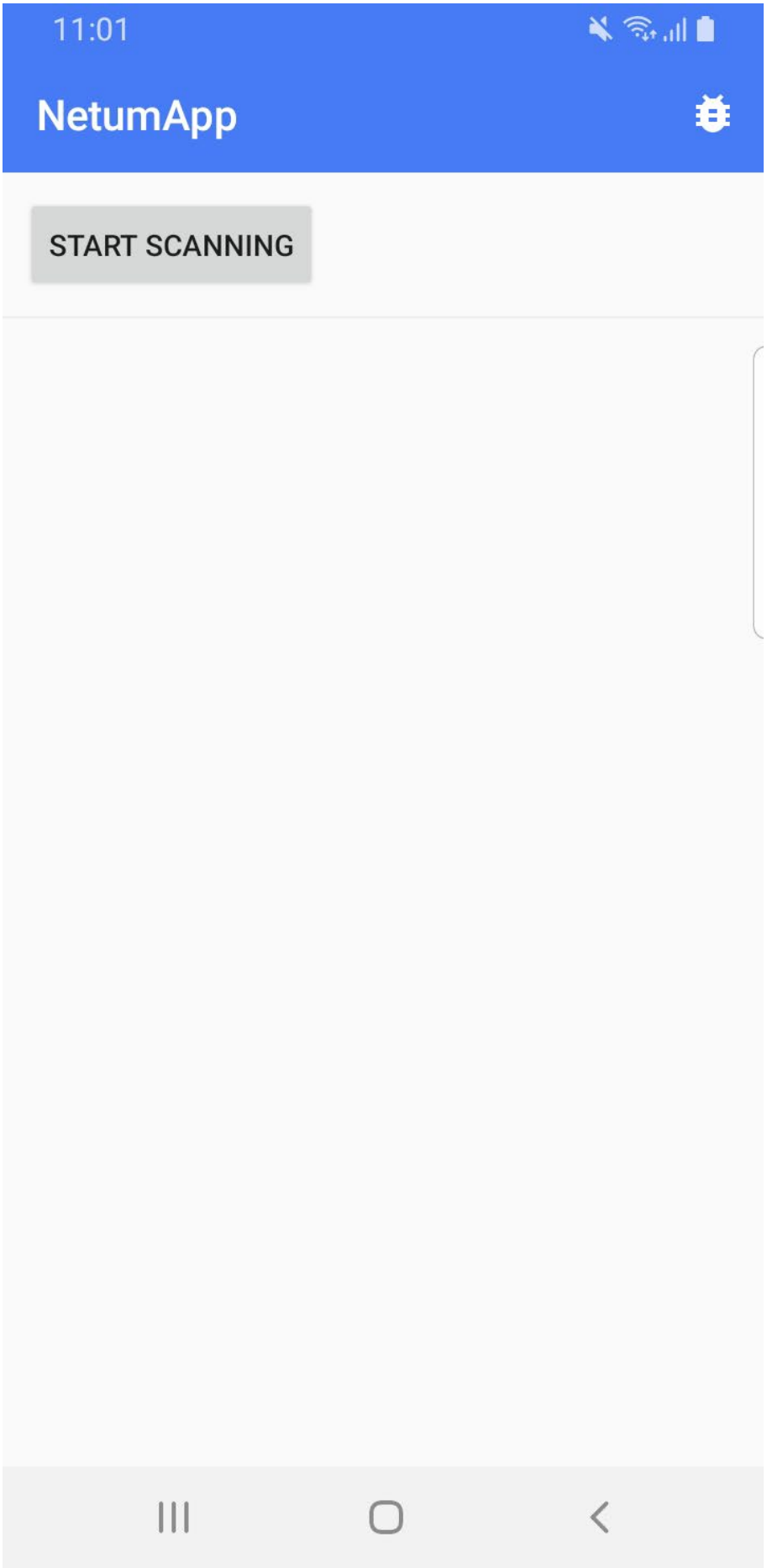
1. Use the scanner to scan the command code below to switch the scanner to Bluetooth mode.

%#IFSNO$4

2. Use the scanner to scan the command code below to change the scanner's Bluetooth mode to Bluetooth SPP.

AT+MODE=1

7. Open the app and navigate to the Bluetooth SPP device search and pairing interface.


11:01
NetumApp
START SCANNING

8. While the app is in search mode, the scanner scans the specified format connection setup code. Follow the steps below to actively connect the phone:

   a) Scan the clear pairing record setup code.

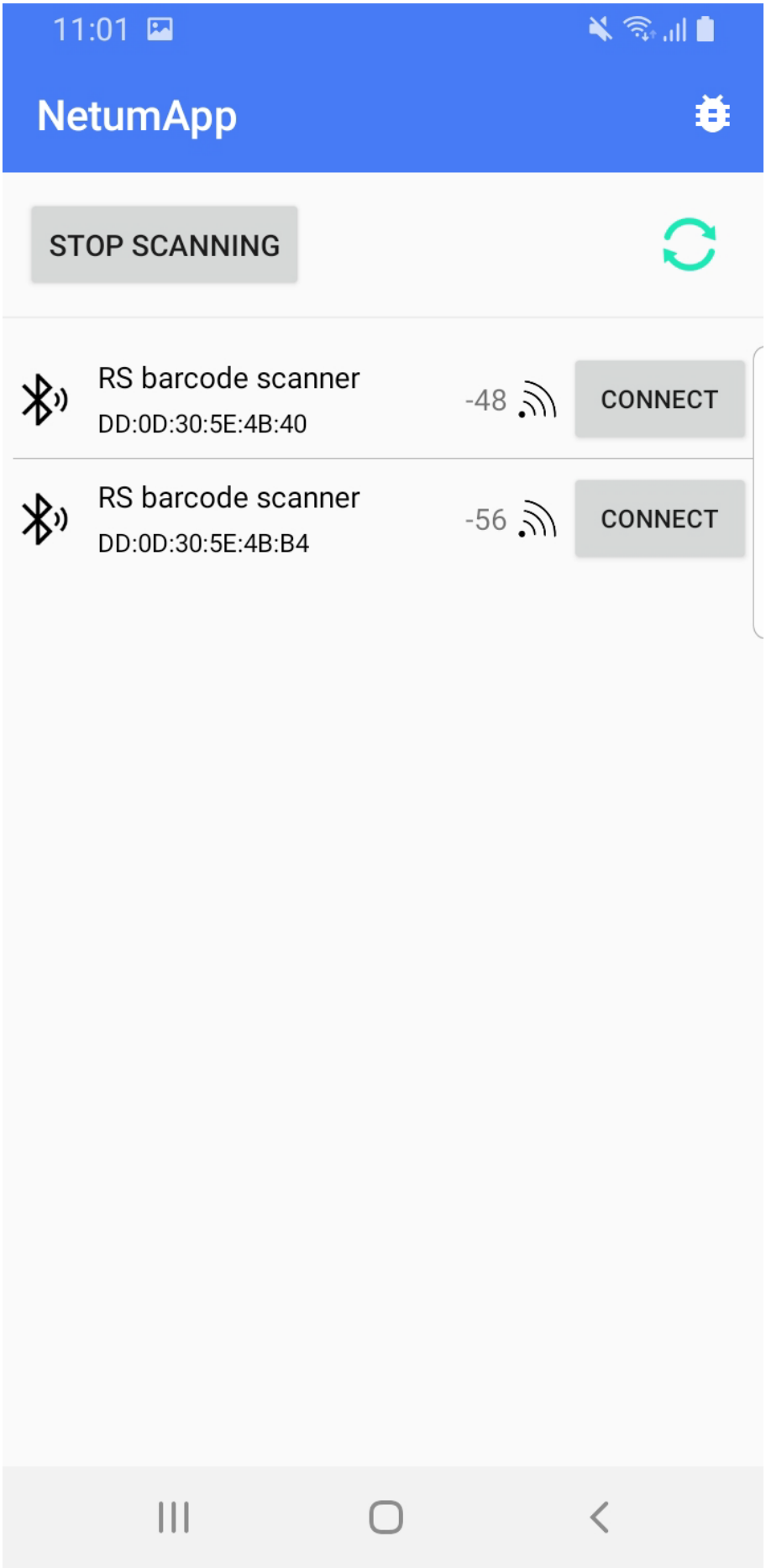
   Un-pair the scanner

   b) Scan the Bluetooth address setup code.

   Format 1: %%88BD45335E2C$, where 88BD45335E2C is the phone's Bluetooth address.

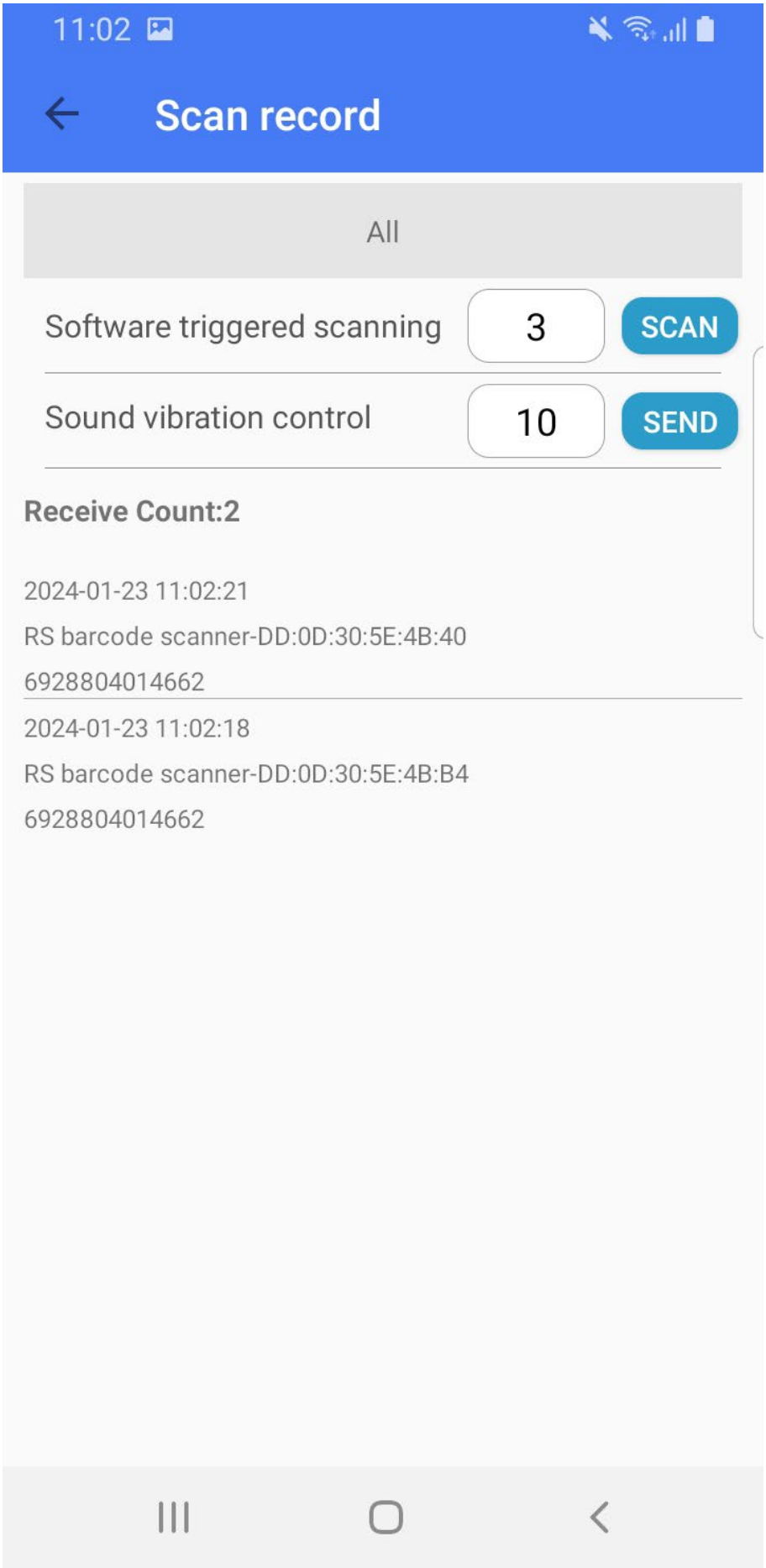
   Format 2: AT+SPPCONN=88BD45335E2C, where 88BD45335E2C is the phone's Bluetooth address.

9. Select "RS barcode scanner" to connect to the scanner. Upon successful connection, the main interface displays as follows:


11:01
NetumApp
STOP SCANNING
RS barcode scanner
DD:0D:30:5E:4B:40
-48
CONNECT
RS barcode scanner
DD:0D:30:5E:4B:B4
-56
CONNECT

- The main interface is used to display Bluetooth devices detected and connected scanners.

10. Click on the menu option button located at the top-right corner of the main interface to access the testing and receiving interface.


11:02
Scan record
All
Software triggered scanning
3
SCAN
Sound vibration control
10
SEND
Receive Count:2
2024-01-23 11:02:21
RS barcode scanner-DD:0D:30:5E:4B:40
6928804014662
2024-01-23 11:02:18
RS barcode scanner-DD:0D:30:5E:4B:B4
6928804014662

a) Device Selection: You can choose the device you wish to control.

b) Software-triggered scanning command with a set range of 1-7 seconds.

c) Sound and vibration control with a set range of 0-26.

SDK sound(n=0x30~0x4A)	"$BUZZ#Bn"	see right beep/led table
e.g. SDK sound 0	"$BUZZ#B0"	1 high short beep
e.g. SDK sound 26	"$BUZZ#BJ"	High-high-low-low beep

Beep / LED Action	Value
1 high short beep	0
2 high short beeps	1
3 high short beeps	2
4 high short beeps	3
5 high short beeps	4
1 low short beep	5
2 low short beeps	6
3 low short beeps	7
4 low short beeps	8
5 low short beeps	9
1 high long beep	10
2 high long beeps	11
3 high long beeps	12
4 high long beeps	13
5 high long beeps	14
1 low long beep	15
2 low long beeps	16
3 low long beeps	17
4 low long beeps	18
5 low long beeps	19
Fast warble beep	20
Slow warble beep	21
High-low beep	22
Low-high beep	23
High-low-high beep	24
Low-high-low beep	25
High-high-low-low beep	26

d) Custom sound and vibration control with three parameters:

Time: Continuous action time of sound or vibration, range: 10-2540 ms.

Type: Control type, 0 = control both sound and vibration, 1 = control sound, 2 = control vibration.

Frequency: Sound frequency, range: 100-5200 Hz.

Continue beep	"$BUZZ#BKttnff"	tt=02~FF, beep 10ms~2.54s; n=0~2, 0. beep+vibration 1. only beep, 2. only vibration. ff=00~FF(100~5200HZ), freqence=100+ff*20.

e) Reception list for receiving scanner scan data.

Click on the connected scanner in the main interface to enter the scanner's individual control interface (under construction).

Quick Use

System Permissions

1. Add the following permission to the Android application manifest file (AndroidManifest.xml):

```
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission
android:name="android.permission.USES_POLICY_FORCE_LOCK" />
<uses-permission android:name="android.permission.DISABLE_KEYGUARD" />
<!-- Permission required for floating window -->
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"
/>
```

Initialization

- Initialize only once, call before using methods in the library, not necessarily in the Application.

```
SPPManager.getInstance().init(getApplication());
```

- Global Configuration

```
SPPManager.getInstance()
        .enableLog(true)
        .setReConnectCount(1, 5000)
        .setConnectOverTime(20000)
        .setOperateTimeout(5000);
```

- Configure Logs

By default, the runtime logs in the library are enabled. If preferred, they can be disabled.

```
SPPManager enableLog(boolean enable)
```

- Configure Reconnection

Set the number of reconnect attempts and the reconnect interval in milliseconds, defaulting to 0 attempts for no reconnection.

```
Manager setReConnectCount(int count)
```

- Configure Split Write

Set the data length for split write, defaulting to 20 bytes per package.

```
Manager setSplitWriteNum(int num)
```

- Configure Connection Timeout

Set the connection timeout in milliseconds, defaulting to 10 seconds.

```
Manager setConnectOverTime(long time)
```

- Configure Operation Timeout

Set the timeout for readRssi, setMtu, write, read, notify, indicate operations in milliseconds, defaulting to 5 seconds.

BLE Mode Scan and Connect Devices

- Configure Scan Rules

```
BleScanRuleConfig scanRuleConfig = new BleScanRuleConfig.Builder()
        .setDeviceName(true, names)   // Only scan devices with specified
broadcast names, optional.
        .setDeviceMac(mac)                // Only scan devices with specified
MAC addresses, optional.
        .setAutoConnect(isAutoConnect)      // AutoConnect parameter during
connection, optional, default is false.
        .setScanTimeOut(10000)             // Scan timeout, optional, default
is 10 seconds.
        .setFilter(true)                  // Set device filtering, optional,
default is true.
        .build();
Manager.getInstance().initScanRule(scanRuleConfig);
```

- Scan

```
BLEManager.getInstance().scan(new BleScanCallback() {
    @Override
    public void onScanStarted(boolean success) {
         // Start scanning (main thread)
        mDeviceAdapter.clearScanDevice();
        mDeviceAdapter.notifyDataSetChanged();
        img_loading.startAnimation(operatingAnim);
        img_loading.setVisibility(View.VISIBLE);
        btn_scan.setText(getString(R.string.stop_scan));
    }

    @Override
    public void onLeScan(BleDevice bleDevice) {
        super.onLeScan(bleDevice);
    }

    @Override
    public void onScanning(BleDevice bleDevice) {
         // Scan a BLE device that meets the scan rules (main thread)
        mDeviceAdapter.addDevice(bleDevice);
        mDeviceAdapter.notifyDataSetChanged();
    }

    @Override
    public void onScanFinished(List<BleDevice> scanResultList) {
         // Scan finished, list all BLE devices that meet the scan rules
(main thread)
        img_loading.clearAnimation();
        img_loading.setVisibility(View.INVISIBLE);
        btn_scan.setText(getString(R.string.start_scan));
    }
});
Tips:
The scanning and filtering process occurs in a separate thread, so it won't
affect UI operations in the main thread. Ultimately, each callback result
returns to the main thread.
```

- Connect through Device Object

```
Connect using the scanned BleDevice object.
BLEManager.getInstance().connect(bleDevice, new BleGattCallback() {
    @Override
    public void onStartConnect() {
```

```
        // Start connecting
        progressDialog.show();
    }

    @Override
    public void onConnectFail(BleDevice bleDevice, BleException exception)
{
        // Connection failed
        img_loading.clearAnimation();
        img_loading.setVisibility(View.INVISIBLE);
        btn_scan.setText(getString(R.string.start_scan));
        progressDialog.dismiss();
        Toast.makeText(MainActivity.this, getString(R.string.connect_fail),
Toast.LENGTH_LONG).show();
    }

    @Override
    public void onConnectSuccess(BleDevice bleDevice, BluetoothGatt gatt,
int status) {
        // Connection successful, BleDevice is the connected scanner
        progressDialog.dismiss();
        mDeviceAdapter.addDevice(bleDevice);
        mDeviceAdapter.notifyDataSetChanged();
        receive(bleDevice);
    }

    @Override
    public void onDisConnected(boolean isActiveDisConnected, BleDevice
bleDevice, BluetoothGatt gatt, int status) {
        // Connection interrupted, isActiveDisConnected indicates whether
the disconnection method was actively called
        progressDialog.dismiss();

        mDeviceAdapter.removeDevice(bleDevice);
        mDeviceAdapter.notifyDataSetChanged();

        if (isActiveDisConnected) {
            Toast.makeText(MainActivity.this,
getString(R.string.active_disconnected), Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(MainActivity.this,
getString(R.string.disconnected), Toast.LENGTH_LONG).show();
            ObserverManager.getInstance().notifyObserver(bleDevice);
        }
```

```
    }
});
Tips:
- On some phone models, connectGatt must be in the main thread to be
effective. It is highly recommended to perform the connection process in
the main thread.
- Reconnect after connection failure: The framework includes a reconnect
mechanism after connection failure, which can be configured with the number
of reconnect attempts and time intervals. Alternatively, you can manually
call the `connect` method with a delay in the `onConnectFail` callback.
- Reconnect after connection disconnection: You can call the `connect`
method again in the `onDisConnected` callback.
- To ensure a successful reconnection rate, it is recommended to wait for
some time after disconnection before attempting reconnection.
- On some device models, after a connection failure, the device may be
briefly unable to scan. Directly connecting to the device through its
object or MAC address, without scanning, can be done to address this.
```

- Connect through MAC

```
Connect directly using the known device's Mac address.
BLEManager.getInstance().connect(mac, new BleGattCallback() {
    @Override
    public void onStartConnect() {
         // Start connecting
        progressDialog.show();
    }

    @Override
    public void onConnectFail(BleDevice bleDevice, BleException exception)
{
         // Connection failed
        img_loading.clearAnimation();
        img_loading.setVisibility(View.INVISIBLE);
        btn_scan.setText(getString(R.string.start_scan));
        progressDialog.dismiss();
        Toast.makeText(MainActivity.this, getString(R.string.connect_fail),
Toast.LENGTH_LONG).show();
    }

    @Override
    public void onConnectSuccess(BleDevice bleDevice, BluetoothGatt gatt,
int status) {
         // Connection successful, BleDevice is the connected scanner
        progressDialog.dismiss();
        mDeviceAdapter.addDevice(bleDevice);
```

```
        mDeviceAdapter.notifyDataSetChanged();
        receive(bleDevice); // Start data reception
    }

    @Override
    public void onDisConnected(boolean isActiveDisConnected, BleDevice
bleDevice, BluetoothGatt gatt, int status) {
         // Connection interrupted, isActiveDisConnected indicates whether
the disconnection method was actively called
        progressDialog.dismiss();

        mDeviceAdapter.removeDevice(bleDevice);
        mDeviceAdapter.notifyDataSetChanged();

        if (isActiveDisConnected) {
            Toast.makeText(MainActivity.this,
getString(R.string.active_disconnected), Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(MainActivity.this,
getString(R.string.disconnected), Toast.LENGTH_LONG).show();
            ObserverManager.getInstance().notifyObserver(bleDevice);
        }
    }
});
Tips:
- This method can attempt to directly connect to scanners around with the
specified MAC address without scanning.
```

- Scan and Connect

```
After scanning and identifying the first device that meets the scanning
criteria, stop scanning and proceed to connect to that device.
BLEManager.getInstance().scanAndConnect(new BleScanAndConnectCallback() {
        @Override
        public void onScanStarted(boolean success) {
                                // Start scanning (main thread)
        }

        @Override
        public void onScanFinished(BleDevice scanResult) {
                                // Scan finished, the result is the first BLE
device that meets the scan rules; if empty, no device was found (main
thread)
        }

        @Override
```

```
        public void onStartConnect() {
                                // Start connecting (main thread)
        }

        @Override
        public void onConnectFail(BleDevice bleDevice,BleException
exception) {
                                // Connection failed (main thread)
        }

        @Override
        public void onConnectSuccess(BleDevice bleDevice, BluetoothGatt
gatt, int status) {
                                // Connection successful, BleDevice is the
connected BLE device (main thread)(main thread)
        }

        @Override
        public void onDisConnected(boolean isActiveDisConnected, BleDevice
device, BluetoothGatt gatt, int status) {
                                // Connection disconnection, isActiveDisConnected
indicates whether it is actively disconnected (main thread)(main thread)
        }
    });

Tips:
- The scanning and filtering processes take place in the working thread, so
they do not affect the UI operations of the main thread. However, each
callback result returns to the main thread. Connection operations occur in
the main thread.
```

- Stop Scanning

```
During the scanning process, stop the scanning
operation.BLEManager.getInstance().cancelScan();
Tips:
- After calling this method, if the scanning is still ongoing, it will end
immediately and callback to the `onScanFinished` method.
```

BLE Mode: Receiving Data and Sending Commands

➢ Start Receiving Data

```
BLEManager.getInstance().startReceive(
    bleDevice,
    new BleNotifyCallback() {

        @Override
        public void onNotifySuccess() {
            runOnUiThread(new Runnable() {
                  // Notification operation successful
                @Override
                public void run() {
                    //addText(txt, "notify success");
                }
            });
        }

        @Override
        public void onNotifyFailure(final BleException exception) {
            runOnUiThread(new Runnable() {
                  // Notification operation successful
                @Override
                public void run() {
                    //addText(txt, exception.toString());
                }
            });
        }

        @Override
        public void onCharacteristicChanged(byte[] data) {
                 // After the notification is opened, the data sent by the
scanner will appear here
            final String message = new String(data);
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    if(MessageActivity.Instance!=null)
                    {

MessageActivity.Instance.addMessage(bleDevice.getName()+"-
"+bleDevice.getMac(),message);
                    }else {
                        Toast.makeText(MainActivity.this, message,
Toast.LENGTH_LONG).show();
                    }
```

```
                    //addText(txt,
HexUtil.formatHexString(characteristic.getValue(), true));
                }
            });
        }
    });
```

- Stop Receiving (Unsubscribe)

```
BLEManager.getInstance().stopReceive(bleDevice);
```

- Send Scanner Commands

```
BLEManager.getInstance().ScannerCommand(device,
ScannerUtil.CustomBeep(Integer.parseInt(LedBeep_Control.getText().toString(
))),new BleWriteCallback() {

    @Override
    public void onWriteSuccess(final int current, final int total, final
byte[] justWrite) {

    }

    @Override
    public void onWriteFailure(final BleException exception) {

    }
});
```

SPP Mode: Scanning and Connecting Devices

- Configure Scanning Rules

```
BleScanRuleConfig scanRuleConfig = new BleScanRuleConfig.Builder()
        .setDeviceName(true, names)   // Only scan devices with specified
broadcast names, optional.
        .setDeviceMac(mac)                // Only scan devices with specified
MAC addresses, optional.
        .setAutoConnect(isAutoConnect)      // AutoConnect parameter during
connection, optional, default is false.
        .setScanTimeOut(10000)             // Scan timeout, optional, default
is 10 seconds.
        .setFilter(true)                  // Set device filtering, optional,
default is true.
        .build();
Manager.getInstance().initScanRule(scanRuleConfig);
```

- Scan

```
SPPManager.getInstance().scan(
        new BleScanCallback()//Scanning callback function
        {
            @Override
            public void onScanStarted(boolean success) {
                //Start scanning (main thread)
                mDeviceAdapter.clearScanDevice();
                mDeviceAdapter.notifyDataSetChanged();
                img_loading.startAnimation(operatingAnim);
                img_loading.setVisibility(View.VISIBLE);
                btn_scan.setText(getString(R.string.stop_scan));
            }

            @Override
            public void onLeScan(BleDevice bleDevice) {
                super.onLeScan(bleDevice);
            }

            @Override
            public void onScanning(BleDevice bleDevice) {
                //Scanned a device that meets the scanning rule (main thread)
                mDeviceAdapter.addDevice(bleDevice);
                mDeviceAdapter.notifyDataSetChanged();
            }

            @Override
            public void onScanFinished(List<BleDevice> scanResultList) {
                //Scanning finished, list all devices that meet the scanning rule
(main thread)
                img_loading.clearAnimation();
                img_loading.setVisibility(View.INVISIBLE);
                btn_scan.setText(getString(R.string.start_scan));
            }
            },
        new SppScanAndConnectCallback() //SDP discovered service callback
function
        {
            @Override
            public void onScanStarted(boolean success) {
                //Start service (main thread)
                mDeviceAdapter.clearScanDevice();
                mDeviceAdapter.notifyDataSetChanged();
                img_loading.startAnimation(operatingAnim);
```

```
                img_loading.setVisibility(View.VISIBLE);
                btn_scan.setText(getString(R.string.stop_scan));
            }

            @Override
            public void onLeScan(BleDevice bleDevice) {
                super.onLeScan(bleDevice);
            }

            @Override
            public void onScanning(BleDevice bleDevice) {
                //Display actively requested SPP device (main thread)
                mDeviceAdapter.addDevice(bleDevice);
                mDeviceAdapter.notifyDataSetChanged();
            }

            @Override
            public void onScanFinished(List<BleDevice> scanResultList) {
                //Stop service, list all actively requested SPP devices (main
thread)
                img_loading.clearAnimation();
                img_loading.setVisibility(View.INVISIBLE);
                btn_scan.setText(getString(R.string.start_scan));
            }

            @Override
            public void onStartConnect() {
                progressDialog.show();
            }//Start connecting

            @Override
            public void onConnectFail(BleDevice bleDevice, BleException
exception) {
                //Connection failed
                img_loading.clearAnimation();
                img_loading.setVisibility(View.INVISIBLE);
                btn_scan.setText(getString(R.string.start_scan));
                progressDialog.dismiss();
                Toast.makeText(MainActivity.this,
getString(R.string.connect_fail), Toast.LENGTH_LONG).show();
            }

            @Override
```

```
            public void onConnectSuccess(BleDevice bleDevice,
BluetoothSocket socket, int status) {
                //Connection successful
                progressDialog.dismiss();
                mDeviceAdapter.addDevice(bleDevice);
                mDeviceAdapter.notifyDataSetChanged();
            }

            @Override
            public void onDataReceiving(final BleDevice bleDevice, byte[]
data) {
                //Receive Data
                final String message = new String(data);
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        if(MessageActivity.Instance!=null)
                        {

MessageActivity.Instance.addMessage(bleDevice.getName()+"-
"+bleDevice.getMac(),message);
                        }else {
                            Toast.makeText(MainActivity.this, message,
Toast.LENGTH_LONG).show();
                        }
                        //addText(txt,
HexUtil.formatHexString(characteristic.getValue(), true));
                    }
                });
            }

            @Override
            public void onDisConnected(boolean isActiveDisConnected,
BleDevice bleDevice, BluetoothSocket socket, int status) {
                //Disconnect
                progressDialog.dismiss();

                mDeviceAdapter.removeDevice(bleDevice);
                mDeviceAdapter.notifyDataSetChanged();

                if (isActiveDisConnected) {
                    Toast.makeText(MainActivity.this,
getString(R.string.active_disconnected), Toast.LENGTH_LONG).show();
                } else {
```

```
                    Toast.makeText(MainActivity.this,
getString(R.string.disconnected), Toast.LENGTH_LONG).show();
                    ObserverManager.getInstance().notifyObserver(bleDevice);
                }
            }
        });

Tips:
The scanning and filtering processes occur in the working thread, so they
do not affect the UI operations of the main thread. Ultimately, each
callback result returns to the main thread.
```

- Connect via Device Object

```
Connect using the scanned BleDevice
object.SPPManager.getInstance().connect(bleDevice, new SppConnectCallback()
{
    @Override
    public void onStartConnect() {
        progressDialog.show();
    }//Start connecting

    @Override
    public void onConnectFail(BleDevice bleDevice, BleException exception)
{
        //Connection failed
        img_loading.clearAnimation();
        img_loading.setVisibility(View.INVISIBLE);
        btn_scan.setText(getString(R.string.start_scan));
        progressDialog.dismiss();
        Toast.makeText(MainActivity.this, getString(R.string.connect_fail),
Toast.LENGTH_LONG).show();
    }

    @Override
    public void onConnectSuccess(BleDevice bleDevice, BluetoothSocket
socket, int status) {
        //Connection successful
        progressDialog.dismiss();
        mDeviceAdapter.addDevice(bleDevice);
        mDeviceAdapter.notifyDataSetChanged();
    }

    @Override
```

```
    public void onDataReceiving(final BleDevice bleDevice, byte[] data) {
        //Receive Data
        final String message = new String(data);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                if(MessageActivity.Instance!=null)
                {

MessageActivity.Instance.addMessage(bleDevice.getName()+"-
"+bleDevice.getMac(),message);
                }else {
                    Toast.makeText(MainActivity.this, message,
Toast.LENGTH_LONG).show();
                }
                //addText(txt,
HexUtil.formatHexString(characteristic.getValue(), true));
            }
        });
    }

    @Override
    public void onDisConnected(boolean isActiveDisConnected, BleDevice
bleDevice, BluetoothSocket socket, int status) {
        //Disconnect
        progressDialog.dismiss();

        mDeviceAdapter.removeDevice(bleDevice);
        mDeviceAdapter.notifyDataSetChanged();

        if (isActiveDisConnected) {
            Toast.makeText(MainActivity.this,
getString(R.string.active_disconnected), Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(MainActivity.this,
getString(R.string.disconnected), Toast.LENGTH_LONG).show();
            ObserverManager.getInstance().notifyObserver(bleDevice);
        }
    }
});
Tips:
- On some phone models, connectGatt must be in the main thread to be
effective. It is highly recommended to perform the connection process in
the main thread.
```

```
- Reconnect after connection failure: The framework includes a reconnect
mechanism after connection failure, which can be configured with the number
of reconnect attempts and time intervals. Alternatively, you can manually
call the `connect` method with a delay in the `onConnectFail` callback.
- Reconnect after connection disconnection: You can call the `connect`
method again in the `onDisConnected` callback.
- To ensure a successful reconnection rate, it is recommended to wait for
some time after disconnection before attempting reconnection.
- On some device models, after a connection failure, the device may be
briefly unable to scan. Directly connecting to the device through its
object or MAC address, without scanning, can be done to address this.
```

➢ Connect through Mac

```
Connect directly using the known device's Mac address.
SPPManager.getInstance().connect(mac, new SppConnectCallback() {
    @Override
    public void onStartConnect() {
        progressDialog.show();
    }//Start connecting

    @Override
    public void onConnectFail(BleDevice bleDevice, BleException exception)
{
        //Connection failed
        img_loading.clearAnimation();
        img_loading.setVisibility(View.INVISIBLE);
        btn_scan.setText(getString(R.string.start_scan));
        progressDialog.dismiss();
        Toast.makeText(MainActivity.this, getString(R.string.connect_fail),
Toast.LENGTH_LONG).show();
    }

    @Override
    public void onConnectSuccess(BleDevice bleDevice, BluetoothSocket
socket, int status) {
        //Connection successful
        progressDialog.dismiss();
        mDeviceAdapter.addDevice(bleDevice);
        mDeviceAdapter.notifyDataSetChanged();
    }

    @Override
    public void onDataReceiving(final BleDevice bleDevice, byte[] data) {
        //Receive Data
```

```
        final String message = new String(data);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                if(MessageActivity.Instance!=null)
                {

MessageActivity.Instance.addMessage(bleDevice.getName()+"-
"+bleDevice.getMac(),message);
                }else {
                    Toast.makeText(MainActivity.this, message,
Toast.LENGTH_LONG).show();
                }
                //addText(txt,
HexUtil.formatHexString(characteristic.getValue(), true));
            }
        });
    }

    @Override
    public void onDisConnected(boolean isActiveDisConnected, BleDevice
bleDevice, BluetoothSocket socket, int status) {
        //Disconnect
        progressDialog.dismiss();

        mDeviceAdapter.removeDevice(bleDevice);
        mDeviceAdapter.notifyDataSetChanged();

        if (isActiveDisConnected) {
            Toast.makeText(MainActivity.this,
getString(R.string.active_disconnected), Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(MainActivity.this,
getString(R.string.disconnected), Toast.LENGTH_LONG).show();
            ObserverManager.getInstance().notifyObserver(bleDevice);
        }
    }
});
Tips:
- This method can attempt to directly connect to scanners around with the
specified MAC address without scanning.
```

- Stop Scanning

```
During the scanning process, stop the scanning
operation.BLEManager.getInstance().cancelScan();
Tips:
- After calling this method, if the scanning is still ongoing, it will end
immediately and callback to the `onScanFinished` method.
```

SPP Mode: Sending Commands

➢ Send Scanner Commands

```
SPPManager.getInstance().ScannerCommand(device,
ScannerUtil.CustomBeep(Integer.parseInt(LedBeep_Control.getText().toString(
))),new BleWriteCallback() {

    @Override
    public void onWriteSuccess(final int current, final int total, final
byte[] justWrite) {

    }

    @Override
    public void onWriteFailure(final BleException exception) {

    }
});
```

List of Command Methods

ScannerUtil.ConvertByte(String cmd)

instructions: This method is used to encapsulate the scanner's regular character string command into a byte[] command before sending it to the scanner.

```
Parameter:
com.netum.device.instruction.Scanner

/**
```

```java
 * Get software version number
 */
public static final String ReadFirmwareVersion="$SW#VER";

/**
 * Restore to factory settings
 */
public static final String RestoreToFactorySet = "%#IFSNO$B";

/**
 * Read interface settings
 */
public static final String ReadInterfaceSetting = "%#IFSNO$";

//region Operating Modes
/**
 * Normal operating mode
 */
public static final String WorkMode_Normal  = "%#NORMD";
/**
 * Storage operating mode
 */
public static final String WorkMode_Store  = "%#INVMD";
/**
 * Store data upload
 */
public static final String StoreDataUpload  = "%#TXMEM";
/**
 * Clear after storing data upload
 */
public static final String StoreDataUploadClear  = "%#TXMEM#C";
/**
 * Current total stored items
 */
public static final String StoreDataNumber = "%#+TCNT";
/**
 * Total stored items and space occupancy
 */
public static final String StoreDataNumberAndSpaceOccupancy  = "%#+TCNT#";
/**
 * Clear stored barcodes
 */
public static final String StoreDataClear = "%#*NEW*";
/**
```

```
 * Turn off automatic storage mode
 */
public static final String StoreAutoSaveOff = "%AutoSav#Off";
/**
 * Turn on automatic storage mode
 */
public static final String StoreAutoSaveOn = "%AutoSav#On";
//endregion

//region Power Management
/**
 * Get current sleep time
 */
public static final String ReadSleeptime  = "$RF#ST";
/**
 * Shut down immediately
 */
public static final String PowerOff  = "$POWER#OFF";
/**
 * Sleep after 1 minute
 */
public static final String SleepTime1Min  = "$RF#ST02";
/**
 * Sleep after 3 minutes
 */
public static final String SleepTime3Min  = "$RF#ST06";
/**
 * Sleep after 5 minutes
 */
public static final String SleepTime5Min  = "$RF#ST10";
/**
 * Sleep after 10 minutes
 */
public static final String SleepTime10Min  = "$RF#ST20";
/**
 * Sleep after 30 minutes
 */
public static final String SleepTime30Min  = "$RF#ST60";
/**
 * Sleep after 1 hour
 */
public static final String SleepTime1Hour  = "$RF#ST<0";
/**
 * Sleep after 2 hours
```

```
 */
public static final String SleepTime2Hour  = "$RF#STH0";
/**
 * Never sleep
 */
public static final String NeverSleep  = "$RF#ST00";
/**
 * Get scanner battery level
 */
public static final String GetVolume  = "%BAT_VOL#";
//endregion

//region Message Alerts
/**
 * Buzzer control - Mute
 */
public static final String BeepMuteVolume = "$BUZZ#0";
/**
 * Buzzer control - High volume
 */
public static final String BeepHighVolume = "$BUZZ#1";
/**
 * Buzzer control - Medium volume
 */
public static final String BeepMiddleVolume = "$BUZZ#2";
/**
 * Buzzer control - Low volume
 */
public static final String BeepLowVolume = "$BUZZ#3";
/**
 * Buzzer control - High pitch
 */
public static final String BeepHighTone = "$BUZZ#4";
/**
 * Buzzer control - Low pitch
 */
public static final String BeepLowTone = "$BUZZ#5";
/**
 * Vibration Motor Disable
 */
public static final String VibrationDisable = "$MOTO#0";
/**
 * Vibration Motor Enable
 */
```

```java
public static final String VibrationEnable = "$MOTO#1";
/**
 * SDK Sound Response Disable
 */
public static final String SDKAckBeepOff = "%ACKBEEP#0";
/**
 * SDK Sound Response Enable
 */
public static final String SDKAckBeepOn = "%ACKBEEP#1";
/**
 * Base Connect Beep Sound Toggle
 */
public static final String BaseConnectBeep = "%ACKBEEP#2";
/**
 * Custom Control Buzzer Command
 */
public static final String BeepCustomOpt = "$BUZZ#B";
/**
 * Custom Control Buzzer Output Time Command
 */
public static final String BeepCustomTime = "$BUZZ#BK";
//endregion

//region RTC Clock
/**
 * Disable Timestamp
 */
public static final String DisableTimeStamp="%RTCSTAMP#0";
/**
 * Enable Timestamp
 */
public static final String EnableTimeStamp="%RTCSTAMP#1";
/**
 * Timestamp Set Date and Time,23/05/04,09:51:00
 */
public static final String TimeStampSetFormat1="%RTCTIME#{0}";
/**
 * Timestamp Set 10-digit Timestamp,1636530504
 */
public static final String TimeStampSetFormat2="%RTCSTAMP#{0}";
//endregion


//region Module Settings
```

```java
/**
 * Get CCD/2D Module Model
 */
public static final String GetCCDModuleType = "%MODULESN#";
/**
 * Set CCD/2D Module Model
 */
public static final String SetCCDModuleType = "%MODULESN#{0}#";
//endregion

//region Scanning Modes
/**
 * Key Hold Scan Mode
 */
public static final String KeyScanMode = "%SCMD#00#";
/**
 * Continuous Scan Mode
 */
public static final String ContinueScanMode = "%SCMD#01#";
/**
 * Key Pulse Scan Mode
 */
public static final String KeyPulseScanMode = "%SCMD#02#";
/**
 * Host Trigger Mode
 */
public static final String HostTriggerMode = "%SCMD#03#";
/**
 * Decode Timeout 3 seconds
 */
public static final String DecodeOvertime3S = "%SCMD#3000D";
/**
 * Decode Timeout 6 seconds
 */
public static final String DecodeOvertime6S = "%SCMD#6000D";
/**
 * Decode Interval Time 0.5 seconds
 */
public static final String IntervalTime05S = "%SCMD#0500I";
/**
 * Decode Interval Time 1.0 seconds
 */
public static final String IntervalTime10S = "%SCMD#1000I";
/**
```

```
 * Immediate Scan (n=1~7) S
 */
public static final String SoftTrigger = "%SCANTM#{0}#";
//endregion

//region Keyboard Settings
/**
 * Read Currently Selected Country Language
 */
public static final String ReadCurrentSelectedLanguage = "$LAN#";
/**
 * US English Keyboard
 */
public static final String KeyboardSelectedLanguage_EN = "$LAN#EN";
/**
 * French Keyboard
 */
public static final String KeyboardSelectedLanguage_FR = "$LAN#FR";
/**
 * German Keyboard
 */
public static final String KeyboardSelectedLanguage_GE = "$LAN#GE";
/**
 * Turkish Q Keyboard
 */
public static final String KeyboardSelectedLanguage_TK = "$LAN#TK";
/**
 * Turkish F Keyboard
 */
public static final String KeyboardSelectedLanguage_TF = "$LAN#TF";
/**
 * Portuguese Keyboard
 */
public static final String KeyboardSelectedLanguage_PT = "$LAN#PT";
/**
 * Spanish Keyboard
 */
public static final String KeyboardSelectedLanguage_ES = "$LAN#ES";
/**
 * Czech Keyboard
 */
public static final String KeyboardSelectedLanguage_CS = "$LAN#CS";
/**
 * Italian Keyboard
```

```
 */
public static final String KeyboardSelectedLanguage_IT = "$LAN#IT";
/**
 * Belgian French Keyboard
 */
public static final String KeyboardSelectedLanguage_FB = "$LAN#FB";
/**
 * Brazilian Portuguese Keyboard
 */
public static final String KeyboardSelectedLanguage_PB = "$LAN#PB";
/**
 * Canadian French Keyboard (Traditional)
 */
public static final String KeyboardSelectedLanguage_FC = "$LAN#FC";
/**
 * Croatian Keyboard
 */
public static final String KeyboardSelectedLanguage_HR = "$LAN#HR";
/**
 * Slovak Keyboard
 */
public static final String KeyboardSelectedLanguage_SK = "$LAN#SK";
/**
 * Danish Keyboard
 */
public static final String KeyboardSelectedLanguage_DA = "$LAN#DA";
/**
 * Finnish Keyboard
 */
public static final String KeyboardSelectedLanguage_FI = "$LAN#FI";
/**
 * Hungarian Keyboard
 */
public static final String KeyboardSelectedLanguage_HU = "$LAN#HU";
/**
 * Latin American (Spanish) Keyboard
 */
public static final String KeyboardSelectedLanguage_EL = "$LAN#EL";
/**
 * Dutch Keyboard
 */
public static final String KeyboardSelectedLanguage_NL = "$LAN#NL";
/**
 * Norwegian Keyboard
```

```
 */
public static final String KeyboardSelectedLanguage_NO = "$LAN#NO";
/**
 * Polish Keyboard
 */
public static final String KeyboardSelectedLanguage_PL = "$LAN#PL";
/**
 * Serbian (Latin) Keyboard
 */
public static final String KeyboardSelectedLanguage_SR = "$LAN#SR";
/**
 * Slovenian Keyboard
 */
public static final String KeyboardSelectedLanguage_SL = "$LAN#SL";
/**
 * Swedish Keyboard
 */
public static final String KeyboardSelectedLanguage_SV = "$LAN#SV";
/**
 * Swiss German Keyboard
 */
public static final String KeyboardSelectedLanguage_DS = "$LAN#DS";
/**
 * UK English Keyboard
 */
public static final String KeyboardSelectedLanguage_UK = "$LAN#UK";
/**
 * Japanese Keyboard
 */
public static final String KeyboardSelectedLanguage_JP = "$LAN#JP";
/**
 * Thai Keyboard
 */
public static final String KeyboardSelectedLanguage_TH = "$LAN#TH";
/**
 * ALT Universal Keyboard
 */
public static final String KeyboardSelectedLanguage_AG = "$LAN#AG";
/**
 * ALT Single-byte Special Character Keyboard
 */
public static final String KeyboardSelectedLanguage_RU = "$LAN#RU";
//endregion
```

```
//region Character Settings
/**
 * Clear Format
 */
public static final String PrefixSuffixHideClearFormat = "$DATA#0";
/**
 * Allow Suffix Output
 */
public static final String AllowSuffixOutput = "$DATA#1";
/**
 * Allow Prefix Output
 */
public static final String AllowPrefixOutput = "$DATA#2";
/**
 * Allow Hide Barcode Suffix
 */
public static final String AllowHidBarcodeSuffix = "$DATA#3";
/**
 * Allow Hide Barcode Content
 */
public static final String AllowHidBarcodeContent = "$DATA#4";
/**
 * Allow Hide Barcode Prefix
 */
public static final String AllowHidBarcodePrefix = "$DATA#5";
//endregion
```

ScannerUtil.SoftTrigger(int second)

instructions:This method is used to obtain the byte[] command that controls the scanner to perform the scanning action.

```
Parameter:
        Range 1-7 seconds
```

ScannerUtil.CustomBeep(int level)

instructions: This method is used to obtain the byte[] command that customizes the buzzer vibration of the scanner.

```
Parameter:
         * @param level
        * Range 0-26
```

SDK sound(n=0x30~0x4A)	"$BUZZ#Bn"	see right beep/led table
e.g. SDK sound 0	"$BUZZ#B0"	1 high short beep
e.g. SDK sound 26	"$BUZZ#BJ"	High-high-low-low beep

Beep / LED Action	Value
1 high short beep	0
2 high short beeps	1
3 high short beeps	2
4 high short beeps	3
5 high short beeps	4
1 low short beep	5
2 low short beeps	6
3 low short beeps	7
4 low short beeps	8
5 low short beeps	9
1 high long beep	10
2 high long beeps	11
3 high long beeps	12
4 high long beeps	13
5 high long beeps	14
1 low long beep	15
2 low long beeps	16
3 low long beeps	17
4 low long beeps	18
5 low long beeps	19
Fast warble beep	20
Slow warble beep	21
High-low beep	22
Low-high beep	23
High-low-high beep	24
Low-high-low beep	25
High-high-low-low beep	26

ScannerUtil. CustomBeepTime(int time,int type,int frequency)

instructions: This method is used to obtain the byte[] command that customizes the buzzer vibration of the scanner.

```
Parameter:
* @param time Range 10-2540 ms
* @param type Range 0-2
*            0:beep+vibration
*            1:only beep
*            2:only vibration
* @param frequency Range 100-5200 Hz
```

Continue beep/vibration	"$BUZZ#BKttnff"	tt=0x02~FF, beep 10ms~2.54s; n=0~2, (0x30~0x32) 0. beep+vibration 1. only beep, 2. only vibration. ff=0x01~FF(120~5200HZ), freqence=100+ff*20.
振动0.6s $BUZZ#BK40264		
		3300hz蜂鸣+振动0.6s $BUZZ#BK400A0

ScannerUtil. SetTimeStamp(Date date)

instructions: This method is used to generate the byte[] command for updating the timestamp in the scanner.

```
Parameter:
* Set the scanner timestamp format 2.
* @param date
* Set time
* @return
```

Usage Example

instructions:Please refer to the quick start guide for pairing and data interaction first.

Send software trigger command

- API: Manager.getInstance().ScannerCommand(device, ScannerUtil.SoftTrigger (Integer.parseInt(LedBeep_Control.getText().toString())),new BleWriteCallback());
- Parameter:second
- Sample:

```
SPPManager.getInstance().ScannerCommand(device, ScannerUtil.SoftTrigger
(Integer.parseInt(LedBeep_Control.getText().toString())),new
BleWriteCallback() {

    @Override
    public void onWriteSuccess(final int current, final int total, final
byte[] justWrite) {

    }

    @Override
    public void onWriteFailure(final BleException exception) {

    }
});
```

Send custom beep vibration command

- API：MainScannerSdk.changeTimeInterval
- Parameter

  level=26;
- Sample:

```
SPPManager.getInstance().ScannerCommand(device,
ScannerUtil.CustomBeep(Integer.parseInt(LedBeep_Control.getText().toString(
))),new BleWriteCallback() {

    @Override
    public void onWriteSuccess(final int current, final int total, final
byte[] justWrite) {

    }

    @Override
    public void onWriteFailure(final BleException exception) {

    }
});
```