Nut Android SDK Document

Revision：

|  |  |  |
| --- | --- | --- |
| Time | Version | mark |
| 2016/12/29 | 1.0 | Init |
| 2020/08/19 | 1.1 | Support Beacon |
| 2020/11/06 | 1.2 | Add API usage examples |
| 2020/11/26 | 1.3 | Add DFU manual |
| 2021/05/24 | 1.4 | Add error code |

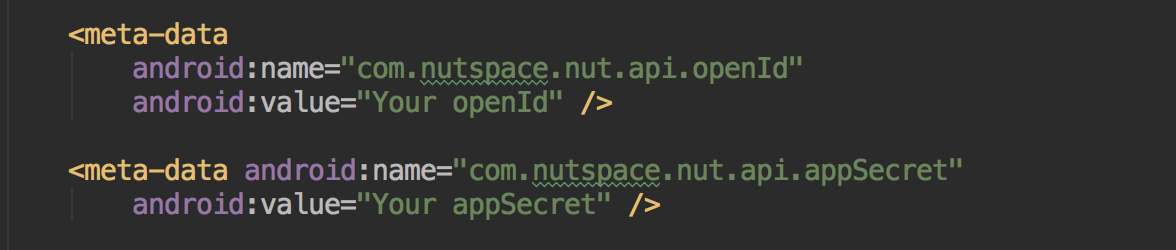
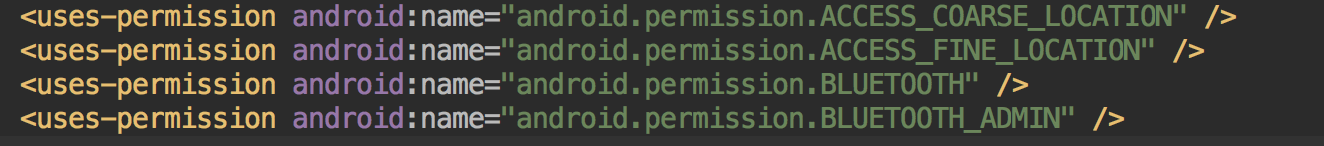
#### Development guide

**1. Development environment**

Recommend Android Studio 2.0 or up；Android OS 4.3 or up.

**2. Configuration manifest and input openId and appSecret**

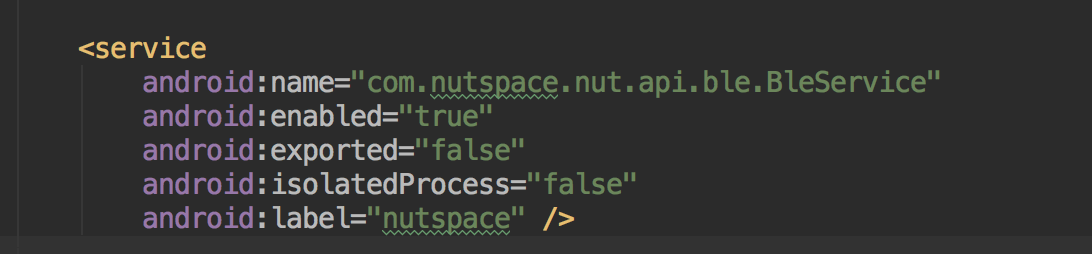
2.1. Manifest configuration includes the addition of permissions, the following permissions are indispensable, fill openId and appSecret, the code example is as follows:



Permission description：

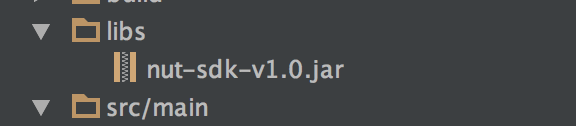
|  |  |
| --- | --- |
| Permission | Use |
| ACCESS\_COARSE\_LOCATION | Scan Nut Device |
| ACCESS\_FINE\_LOCATION | Scan Nut Device |
| BLUETOOTH | BLE |
| BLUETOOTH\_ADMIN | BLE |

2.2. Configuration BleService

In the manifest definition BleService, the code example is as follows:

**3****. Import and init the SDK**

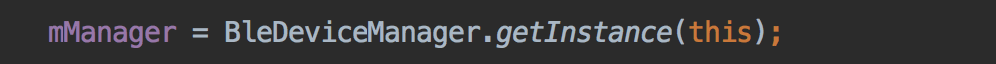
3.1. Import the SDK jar package into the project, for example:



3.2. Initialize BleManager in the component (Activity or Service) and bind BleService.

The code example is as follows:

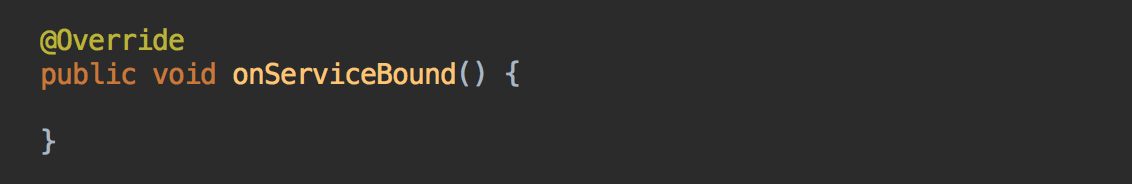
Initialization BleManager：



Bind BleService：



Note: The bind bind () and unbind () methods must appear in pairs

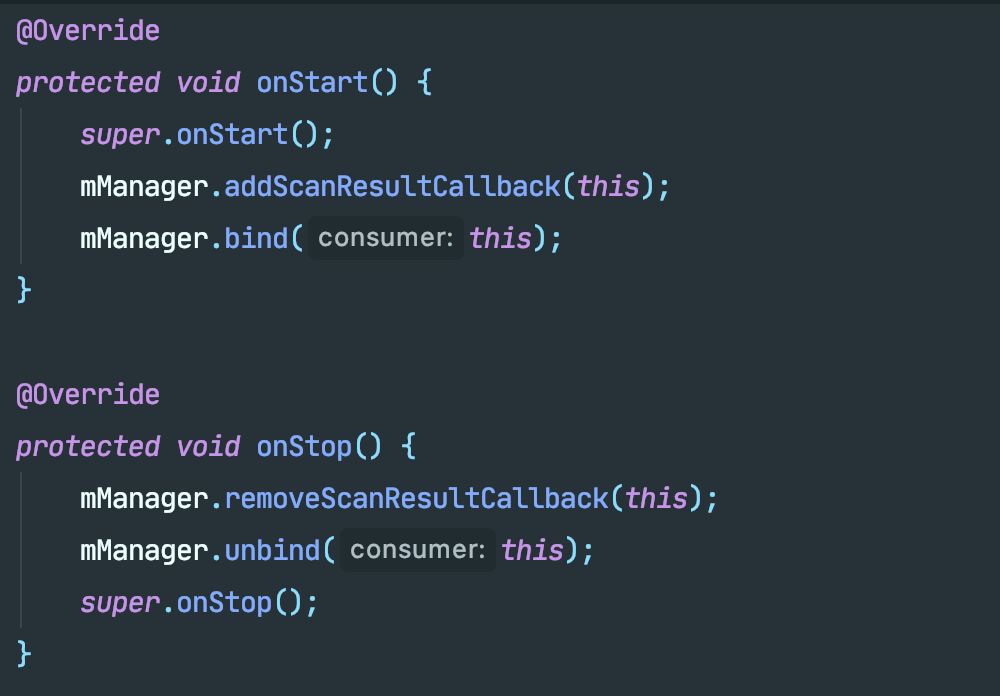
After the binding is successful callback method:

After the binding is successful, you can use the SDK by calling the object mManager.

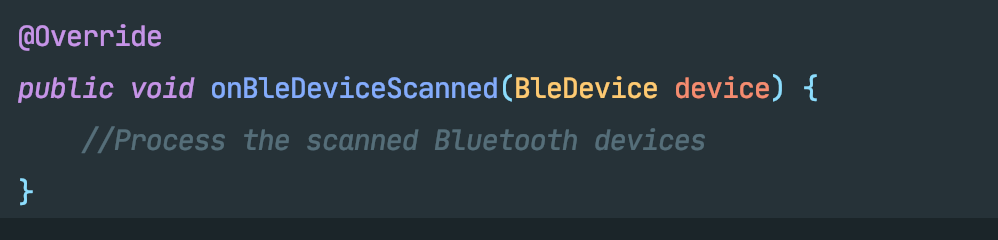
**4. SDK feature usage**

4.1. Scan for Bluetooth devices

Register the scan callback interface ScanResultCallback , and process the scanned devices in onBleDeviceScanned().



Note: You can call the removeScanResultCallback () method to remove a listening event when you do not need to listen for a scan event.

Scan to device after the callback method: 

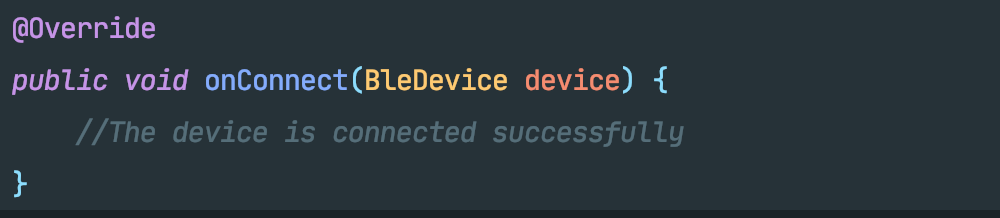
And returns the object of class BleDevice that contains the device information.

4.2. Connecting device

Register the callback interface ConnectStateChangedCallback, then call the connection device method.

connect(Context context, BleDevice device, boolean isAuto) 

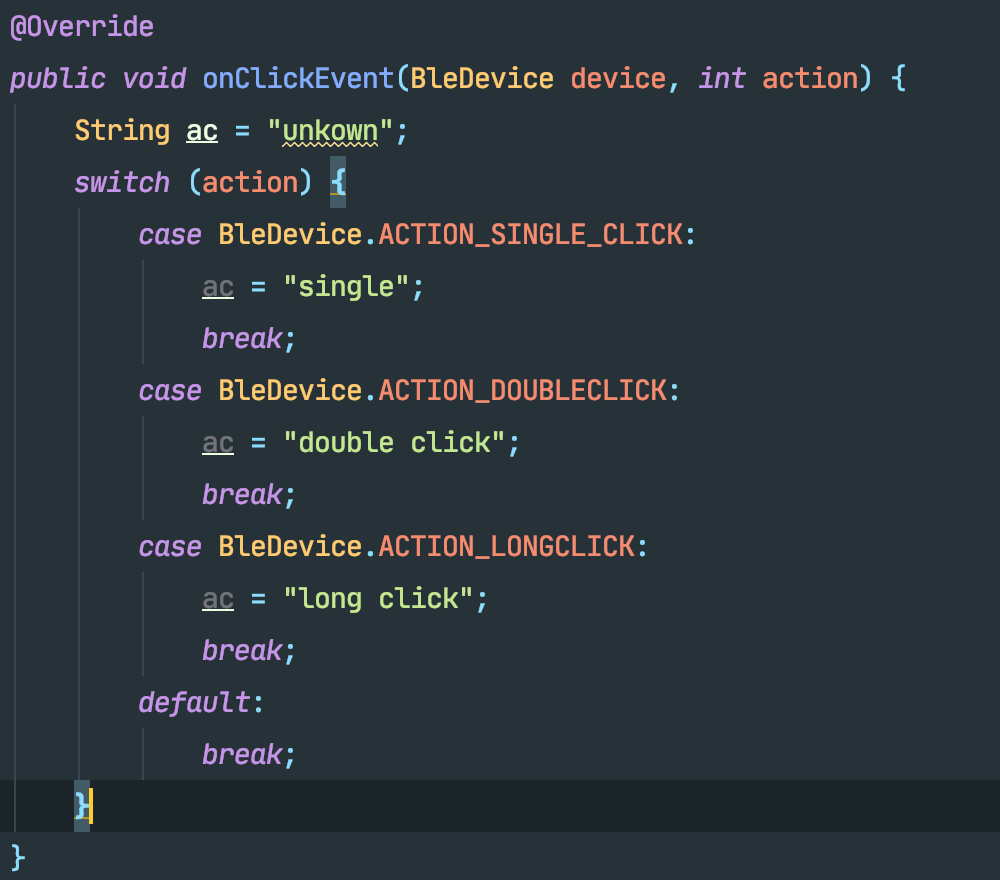
If the device is connected successfully, The device object instance is returned from onConnect(BleDevice device). You can then call other methods of the device



4.3. Basic function of device and event callback

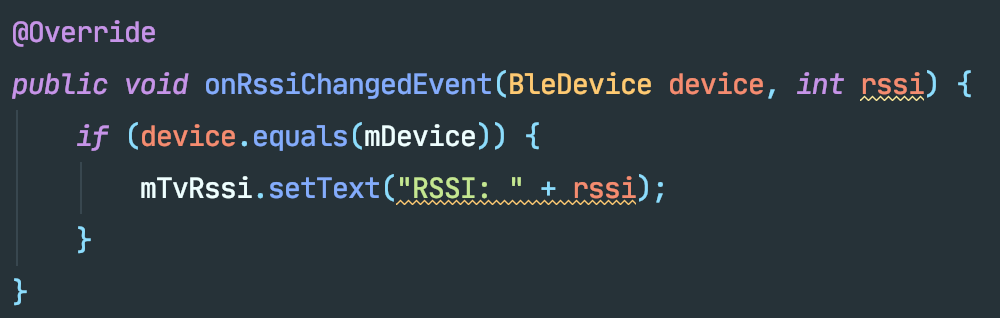
Register the device event callback interface EventCallback (please refer to register connection callback interface) and add interface implementation.

Click / double click the listen device button (the new firmware does not support the long press event)



Read device signal strength value:

mManager.readRssi(device);



Read device power value:

mManager.readBattery(this, mDevice);



Find device / stop looking for device:

mManager.changeRingState(mDevice, BleDevice.STATE\_RING);

mManager.changeRingState(mDevice, BleDevice.STATE\_QUIT);



Turn on and off the device disconnect alert feature:

mManager.enableAntiLost(mDevice, true/false);

4.4. Beacon configuration function (Device need customized firmware to support this API)

4.4.1Enable Beacon feature

After the hardware is turned on and enter the configuration mode, after setting the effective UUID and Major and Minor information of the Beacon, the device will enter the Beacon operating mode after restarting.

//Set the UUID information of Beacon, the string format of UUID needs to adopt standard format, for example: "10102233-4455-6677-8899-AABBCCDDEEFF"

void setBeaconUUID(BleDevice device, String beaconUUID)

//Setting UUID information is complete, result=true means success, false means failure

void onBeaconUUID(BleDevice device, String uuid, boolean result)

//Set the Major and Minor information of Beacon

void setBeaconMajorMinor(BleDevice device, int major, int minor)

//Set Major and Minor information completed, result=true means success, false means failure

void onBeaconMajorMinor(BleDevice device, int major, int minor, boolean result)

4.4.2 Disable the Beacon feature

After the hardware is turned on and enter the configuration mode, set the UUID, Major and Minor of the Beacon to invalid values, and the device will enter the Nut Tag mode after restarting.

UUID invalid value: FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF

Major invalid value: FFFF

Invalid Minor value: FFFF

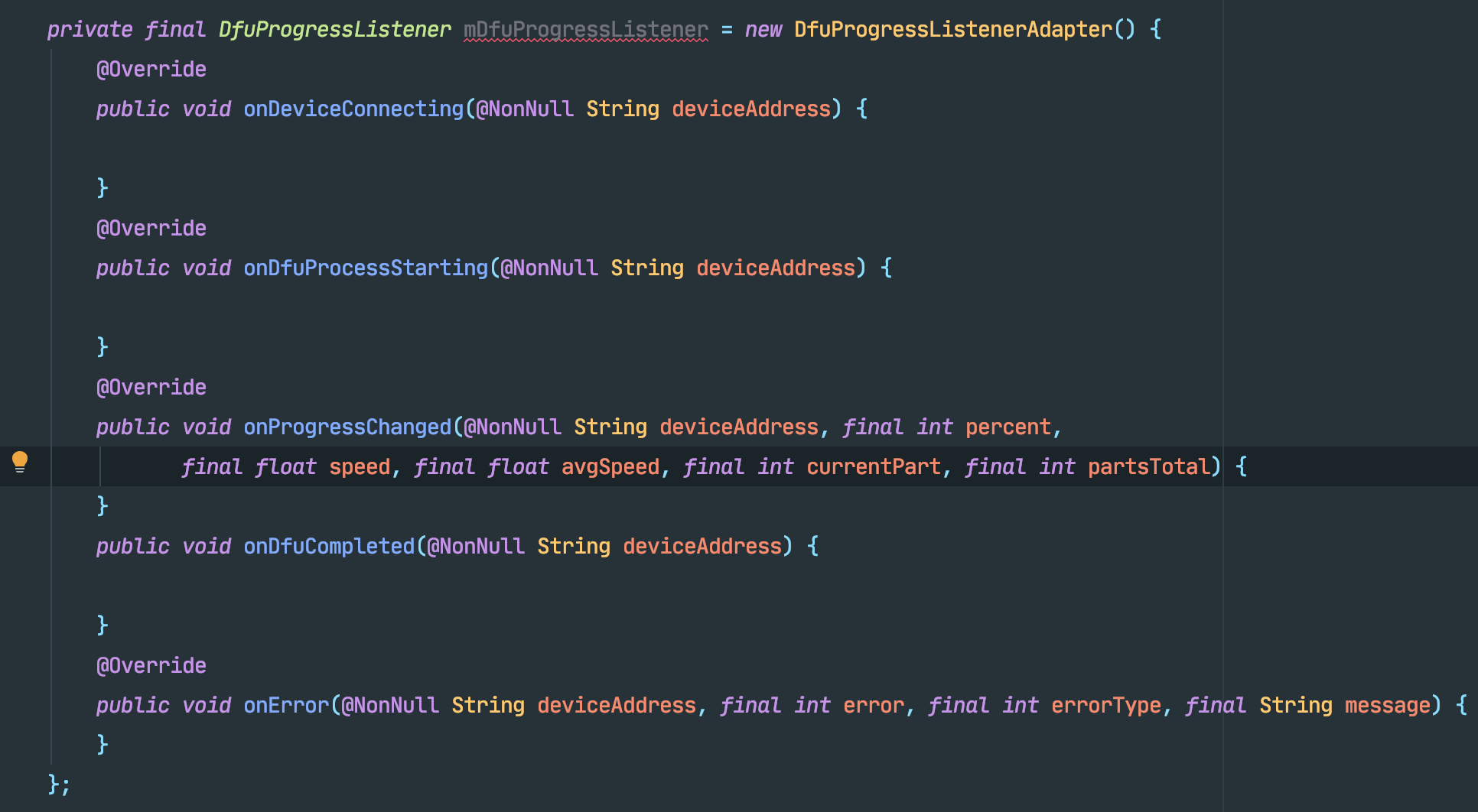
Note: Each time you update the Beacon information, you need to call setBeaconUUID first, and then call setBeaconMajorMinor after completion. Each time you update the Beacon information, you need to update the UUID and MajorMinor. Updating only one of them will cause the Beacon information to be abnormal.

1. **DFU API description**

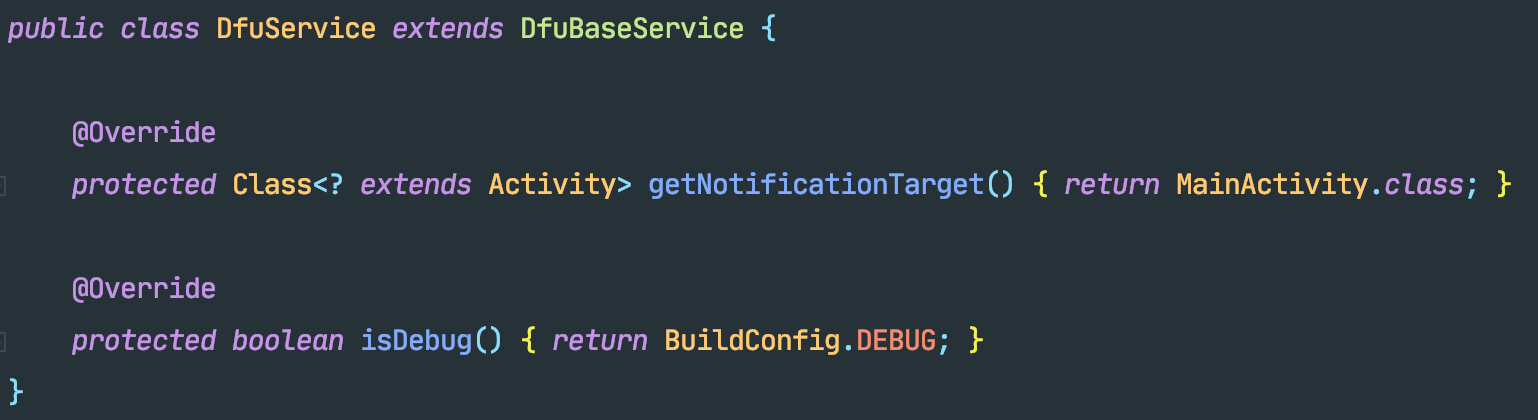
The DfuProgressListener are implemented. The DfuProgressListener handles DFU status changes、error handling and DFU firmware sending process.

Note: You need to call registerProgressListener(this, mDfuProgressListener) in onRsume() and onPause() to register this interface.





5.2. To inherit DfuBaseService and implement DfuService, you need to return the Activity that implements DFU processing in the getNotificationTarget method.



5.3. Initialize DfuServiceInitiator, initialize the firmware object, and then start the upgrade process.



5.3 Introduction to equipment DFU process

5.3.1, turn on DFU mode

When the device enters the TAG working mode, connect the device and execute switchToDFUMode. The device will restart and then enter the DFU working mode. At this time, the Bluetooth broadcast name of the device is "DfuTarg", and the Mac address or DeviceId of the device will be incremented by 1.

5.3.2、Scan the device and start the upgrade

After discovering the device in DFU mode, call performDFUUpload to start the DFU upgrade process. DfuServiceInitiator will automatically connect to the device, send the firmware file, and the device will automatically restart after the sending is completed. At this time, the DFU process is completed.

**FAQ**

The above functions are all implemented through the Bluetooth read-write API of Android system. Due to the limitation of system API execution, Bluetooth read-write and other operations can't be executed simultaneously. Each operation execution needs a certain time (hundreds of milliseconds) to be sent to peripheral devices. If it is executed in parallel, the Android blue tooth protocol stack will directly discard the API execution. To be safe, you can put the operation into the execution queue, take it out of the queue every 1-2s and perform the Bluetooth operation.

**6、Interface introduction：**

|  |  |
| --- | --- |
| Interface | Features |
| ScanResultCallback | Monitor the results of the scanning device |
| EventCallback | Monitor rssi value changes, call device, device key count, battery changes |
| ConnectStateChangedCallback | Listen to device connection status |
| BeaconResultCallback | Listen to device set beacon configuration |

The code snippet in the document is available in the SDK Demo

**7. Common Issues**

1、Common error code

|  |  |
| --- | --- |
| Error Code | Description |
| 0x08(8) | Android BLE error: connection timeout. |
| 0x13(19) | Android BLE error: connection terminate by peer user. |
| 0x16(22) | Android BLE error: connection terminated by local host. |
| 0x03E(62) | Android BLE error: connection fail to establish. |
| 0x85(133) | Android BLE error: Bluetooth gatt error. |
| 0x0101(257) | Android BLE error: no connection to cancel |

2、For the above errors, the App needs to call Nut SDK BleDeviceManager.disconnect() to actively perform device disconnection and reclaim system resources, then perform a Bluetooth scan, wait for the Bluetooth device to be scanned, execute the BleDeviceManager.connect() method, and retry the connection operation.