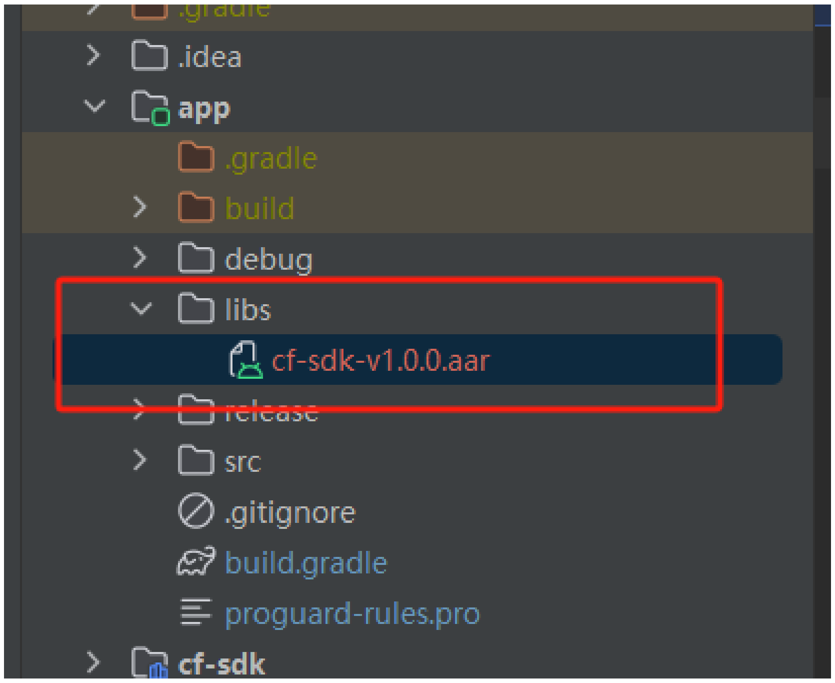
Table of contents

1. Overview

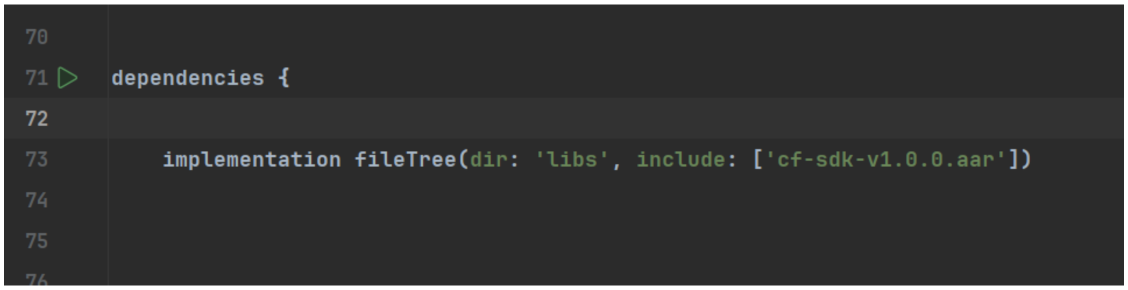
cf-sdk is a development kit for UHF Reader that integrates BLE, serial port, USB, and TCP/IP communication methods under the Android system. Developers can quickly integrate and use the functions of related hardware by accessing cf-sdk. Supports Android 5.0 and above systems.

2. How to use

2.1 Import SDK



Create a new libs folder as shown above, and put cf-sdk in it, then add aar package dependency to the build.gradle file in the project, as shown below.



```
dependencies {

    implementation fileTree(dir: 'libs', include: ['cf-sdk-v1.0.0.aar'])
```

Complete the above steps and refresh the project.

2.2 Using SDK

Before using the SDK, you need to initialize the SDK. Call CfSdk.load() or CfSdk. load(ExecutorService pExecutorService) (Note! Pass in the thread pool. For the normal operation of the SDK function, the number of threads in the thread pool should be greater than or equal to 2) to complete the initialization of the SDK. After completing the initialization action, get the relevant communication objects by calling the CfSdk. get(Class pClass) interface. The input parameters are SdkC.BLE, SdkC. IP_CONN, SdkC. UART, SdkC. USB; they correspond to BLE Bluetooth, TCP/IP, serial port and USB communication methods respectively. Developers can get communication objects as needed according to different hardware devices.

3. SDK interface introduction

3.1 BleCore: BLE Bluetooth device operation class

```
/**
 * initialization
 *
 * @param pContext Context
 */
public void init(Context pContext);

/**
 * Whether to support Bluetooth module
 *
 * @return true: Support, otherwise
 */
public boolean isSupportBt();

/**
 * Is Bluetooth turned on?
 *
 * @return true: Indicates that it is already opened, otherwise
 */
public boolean isEnabled();
```

```
/**
 * Is it connected
 *
 * @return trueIndicates that the connection, false Disconnect
 */
public boolean isConnect();

/**
 * Start Scan ble device
 *
 * @param pIBtScanCallback Scan result callback interface, scanned ble
device, Returned through this interface
 */
public void startScan(IBtScanCallback pIBtScanCallback);

/**
 * Stop Scanning ble device
 */
public void stopScan();

/**
 * Connecting devices
 *
 * @param pDevice      To connectbledevice
 * @param pContext     Context Object
 * @param pAutoConnect Whether to connect automatically, trueIndicates
automatic connection after disconnection, falseIndicates not to connect
 automatically
 * @return return {@link BluetoothGatt}Object
 */
public BluetoothGatt connectDevice(BluetoothDevice pDevice, Context pCo
ntext, boolean pAutoConnect);

/**
 * Get the current connectionble device
 *
 * @return
 */
public BluetoothDevice getConnectedDevice();

/**
 * Turn notifications on or off
 *
 * @param pServiceUuid Serveuuid
 * @param pNotifyUuid  notifyuuid
 * @param pOpen        true: Turn on notifications, false: Turn off not
ifications
 * @return returnBooleanvalue, trueIndicates that the operation was suc
```

```
cessfully executed, false Indicates that the operation failed
 */
public boolean setNotifyState(UUID pServiceUuid, UUID pNotifyUuid, bool
ean pOpen);


/**
 * Turn notifications on or off
 *
 * @param pServiceUuid Serveuuid
 * @param pNotifyUuid  notifyuuid
 * @param pOpen        true: Turn on notifications, false: Turn off not
ifications
 * @param pCallback    Open result callback
 * @return returnBooleanvalue, trueIndicates that the operation was suc
cessfully executed, falseIndicates that the operation failed
 */
public boolean setNotifyState(UUID pUuid, UUID pNotifyUuid, boolean pOp
en, IOnNotifyCallback pCallback);

/**
 * Set notification callback, The data reported by the BLE device is tr
ansmitted through IOnNotifyCallback Callbacks,
 * If it is empty, it will be a logout callback.
 *
 * @param pCallback {@link IOnNotifyCallback}
 */
public void setOnNotifyCallback(IOnNotifyCallback pCallback);

/**
 * ble Device disconnect callback, if it is empty, it will be a logout
callback
 *
 * @param pIBleDisConnectCallback {@link IBleDisConnectCallback}
 */
public void setIBleDisConnectCallback(IBleDisConnectCallback pIBleDisCo
nnectCallback);

/**
 * Connection completion callback, if empty, it is a logout callback
 *
 * @param pIConnectDoneCallback {@link IConnectDoneCallback}
 */
public void setIConnectDoneCallback(IConnectDoneCallback pIConnectDoneC
allback);

/**
 * Writing Data
 *
 * @param pServiceUuid        Serve uuid
 * @param pCharacteristicUuid Write Featuresuuid
 * @param pCmd                Data Array
 * @return true: Execution is successful, otherwise
 */
public boolean writeData(UUID pServiceUuid, UUID pCharacteristicUuid, b
yte[] pCmd);

/**
 * Reading Data
 *
 * @param pServiceUuid        Serve uuid
 * @param pCharacteristicUuid The characteristics of the data to be rea
d udid
 * @return true: Indicates that the operation was successful, otherwise
 */
public boolean readData(UUID pServiceUuid, UUID pCharacteristicUuid);

/**
 * Disconnect a connected device
 */
public synchronized void disconnectedDevice();
```

BleCore Callback interface description

```
/**
 * scanningbleDevice callback interface

 */
public interface IBtScanCallback {
    /**
     * Scan Results
     *
     * @param pResult Scan Results
     */
    void onBtScanResult(ScanResult pResult);

    /**
     * Scan failed, default implementation required, developers can imp
lement it by themselves
     *
     * @param pErrorCode Error Code
     */
    default void onBtScanFail(int pErrorCode) {
    }
}

/**
 * bleDevice disconnect callback, This callback is triggered when the B
```

```
LE Bluetooth and device are disconnected.
 */
public interface IBleDisConnectCallback {
    void onBleDisconnect();
}

/**
 * Device connection successful callback
 */
public interface IConnectDoneCallback {
    /**
     * @param pB true: The connection is successful, otherwise the conn
ection fails
     */
    void onConnectDone(boolean pB);
}


/**
 * ble Device response callback interface
 */
public interface IOnNotifyCallback {

    /**
     * After the command is parsed, the interface is returned. The comm
and parsing is only applicable to
     * Parsing and processing of return instructions of our company's e
quipment, other manufacturers
     * The data returned by the device needs to be parsed and processed
 by the developer
     *
     * @param pCmdType Instruction Type
     * @param pBleData Dataset
     */
    void onNotify(int pCmdType, BleData pBleData);

    /**
     * Default implementation. Developers who need it can implement it
and process the original data returned by BLE Bluetooth.
     *
     * @param pBytes ble Response raw data
     */
    default void onNotify(byte[] pBytes) {
    }
}
```

3.2 IpCore: TCP/IP Device Operation

```
/**
 * connect
 *
 * @param pIpAddress ip address
 * @param pPort      Port Number
 * @return true: Indicates that the connection is successful, otherwise
 the connection fails
 */
public boolean connect(String pIpAddress, int pPort);

/**
 * Writing Data
 *
 * @param pBytes data byte Arrays
 * @return true: Write for success, false Write failure
 */
@WorkerThread
public boolean writeData(byte[] pBytes);

/**
 * Child thread receives data,
 *
 * @param pCallback Data callback interface{@link ISocketDataCallback}
 */
public synchronized void receiveData;

/**
 * Disconnect
 */
public void disConnect();
```

IpCore Callback interface description

```
/**
 * socket Data callback
 */
public interface ISocketDataCallback {
    /**
     * @param pBytes socket Return data
     */
    void onDataCallback(byte[] pBytes);
}
```

Discover the device class UdpFounder, discover the device in the LAN by sending UDP broadcasts, and get the instance by calling the UdpFounder. get() function

```java
/**
 * Send a single broadcast
 *
 * @param pData Broadcast data
 * @param pPort Destination port number
 */
public void sendBroadcast(String pData, int pPort);

/**
 * Sending Broadcasts

 *
 * @param pData       Broadcast data
 * @param pTargetPort Destination port number
 * @param pTimeOut    Sending timeout
 */
public void sendBroadcast(byte[] pData, int pTargetPort, int pTimeOut);

/**
 * Stop sending off-broadcast
 */
public void stopSendBroadcast();

/**
 * Receive target port number data
 *
 * @param pPort Destination port number
 */
public synchronized void receiveData(int pPort);

/**
 * Set udp broadcast data callback, set to null for logout callback
 *
 * @param pIUdpDataCallback udp Data return interface
 */
public void setIUdpDataCallback(IUdpDataCallback pIUdpDataCallback);

/**
 * release
 */
public void release();
```

UdpFounder Interface callback description

```java
/**
 * udp Broadcast data callback */
public interface IUdpDataCallback {
    /**
     * @param pPacket udpBroadcast Response Packet
     */
    void onReceiveUdpData(DatagramPacket pPacket);
}
```

3.3 UartCore: Serial port device operation class

```
/**
 * initialization
 *
 * @param pPath     Serial port address
 * @param pBaudRate Baud rate
 * @return true: Initialization successful, false Initialization failed
 */
public boolean init(String pPath, int pBaudRate);

/**
 * Sending Data
 *
 * @param pBytes Byte array of data to be sent
 * @return true: Indicates successful sending, false Indicates that the
 sending failed
 */
public boolean sendData(byte[] pBytes);

/**
 * Receive serial port data
 *
 * @param pSerialDataCallback Serial port data callback interface
 */
public synchronized void receiverData(ISerialDataCallback pSerialDataCa
llback);

/**
 * Stop receiving data
 */
public void stopReceiverData();

/**
 * Release the serial port
 */
public void release();
```

UartCore Release the serial port

```
/**
 * Serial port data callback

 */
public interface ISerialDataCallback {
    /**
     * @param pBytes Serial port data
     */
    void onDataCallback(byte[] pBytes);
}
```

3.4 UsbCore：USB device operation class

```
/**
 * initialization
 *
 * @param pContext Context Object
 */
public void init(Context pContext);

/**
 * Get the pid and vid of all usb devices
 *
 * @return List of pid and vid devices for all devices
 */
public ArrayList<Pair<Integer, Integer>> getAllDevicePidAndVid();

/**
 * Find the target device based on the provided pid and vid
 *
 * @param pPid
 * @param pVid
 * @return Returns the hit USB device, otherwise returns null
 */
public UsbDevice findTargetDevice(int pPid, int pVid);

/**
 * Connecting devices
 *
 * @param pContext          Context Object
 * @param pUsbDevice        Connected target device
 * @param pIUsbConnectDone usbDevice connection completion callback
 */
public void connectDevice(Context pContext, UsbDevice pUsbDevice, IUsbC
onnectDone pIUsbConnectDone);

/**
 * Writing Data
 *
 * @param pData    To send data byte array
 * @param pTimeOUt Timeout Unitms
 * @return true: Write success, otherwise write failure
 */
public boolean writeData(byte[] pData, int pTimeOUt);

/**
```

```
 * Get USB data synchronously
 *
 * @param pTimeOut Retrieval of data timeout
 * @return Returns the data read from USB
 */
public byte[] readDataSync(int pTimeOut);

/**
 * Read USB data asynchronously
 *
 * @param pTimeOut Single read data timeout
 */
public synchronized void readDataAsync(int pTimeOut);

/**
 * Set the USB device data callback, if it is empty, it will be the log
out callback
 *
 * @param pIReadDataCallback
 */
public void setIReadDataCallback(IReadDataCallback pIReadDataCallback);

/**
 * Release USB device
 */
public void release();
```

UsbCore Callback interface description

```
/**
 * usb Read data callback
 */
public interface IReadDataCallback {
    /**
     * @param pBytes usb Data returned by the device
     */
    void onDataBack(byte[] pBytes);
}

/**
 * usb Connection completion callback

*/
public interface IUsbConnectDone {
    /**
     * @param pB true: Indicates that the connection is successful, oth
erwise the connection fails
     */
    void onUsbConnectDone(boolean pB);
}
```

4. Directive construction and directive types

4.1 Instruction construction: Build corresponding instructions through CmdBuilder

```
/**
* Get the initialization module instructions
*/
public static byte[] buildModuleInitCmd();

/**
* Get the factory reset instructions
*/
public static byte[] buildRebootCmd();

/**
* Get the command to set the RF output power
*
* @param pPower Output power, in: dBm, The value range is: [0, 26]dBm, G
reater than 26 dBm means 26 dBm.
* @param pResv  System reserved field, default0x00;
*/
public static byte[] buildSetPwrCmd(byte pPower, byte pResv);

/**
* Set/read the RF protocol standard supported by the module
*
* @param pOption Command Control Options
*                0x01: Set, followed by 1 Byte of RFID ;
*                0x02: Read, then do not connect to RFID;
*                Other values: invalid;
* @param pRfid   Protocol Options, 0x00: ISO 18000-6C;  0x01: GB/T 29768;
0x02: GJB 7377.1; Currently only supports ISO 18000-6C。
* @return
*/
public static byte[] buildSetOrGetRfidCmd(byte pOption, byte pRfid);

/**
* Get instructions for setting all configurable parameters
*
* @param pBean Configurable parameter bean class, which can be parsed b
y sending a get all parameter instruction
*/
public static byte[] buildSetAllParamCmd(AllParamBean pBean);

/**
* Get all configurable parameters
*/
public static byte[] buildGetAllParamCmd();
```

```
/**
* Get battery level command
*/
public static byte[] buildGetBatteryCapacityCmd();

/**
* Set/get Bluetooth device name command
*
* @param pOption Command Control Options
*                0x01: Set, followed by 1 Byte of RFID ;
*                0x02: Read, then do not connect RFID;
*                Other values: invalid;
* @param pBtName Bluetooth Name
* @return
*/
public static byte[] buildSetOrGetBtNameCmd(byte pOption, String pBtNam
e);

/**
 * Build settings bluetooth output mode directive
 *
 * @param pMode 0x00: Bluetooth HID output; 0x01: Bluetooth transparent
 transmission output
 * @return
 */
public static byte[] buildSetOutputModeCmd(byte pMode)

/**
* Construct command to get Bluetooth output mode
*
* @return
*/
public static byte[] buildGetOutputModeCmd()

/**
* This command reports the start and end status of the key action.
*
* @param pKeyState 0x01: start;
*                  0x02: Finish;
*                  Other values: invalid.
* @return
*/
public static byte[] buildReportKeyStateCmd(byte pKeyState);

/**
* Set the device read mode.
*
```

```
* @param pReadMode Scan Enable Parameters，1Byte。
*                   0x01: Scan head mode ;
*                   0x00: RFID mode;
*                   Other values: invalid;
* @param pRecev     reserve，7Bytes
* @return
*/
public static byte[] buildSetReadModeCmd(byte pReadMode, byte[] pRecev);

/**
* Get the device read mode
*
* @return
*/
public static byte[] buildGetReadModeCmd();

/**
* @param pInvType  Inventory method:
*                   0x00: By time inventory tag, stop the inventory afte
r executing the specified time or after receiving the stop inventory co
mmand;
*                   0x01: Inventory is counted according to the number o
f cycles. The inventory is stopped after executing the specified number
 of polls or receiving the stop inventory command.;
* @param pInvParam Inventory method parameters:
*                   1.If InvType is 0x00:
*                   InvParamIndicates the inventory time, in seconds. If
 the value is 0, it means that the inventory will continue until a stop
 command is received.;
*                   2.If InvType is 0x01:
*                   InvParamIndicates the number of inventory counts, in
 times, the value must be greater than 0;
*/
public static byte[] buildInventoryISOContinueCmd(byte pInvType, int pI
nvParam);

/**
* Stop counting
*
* @return
*/
public static byte[] buildStopInventoryCmd();

/**
* Get device information
*
* @return
*/
```

```
public static byte[] buildGetDeviceInfoCmd();

/**
* Constructing a select tag directive
*
* @param pMask Mask, refers to the EPC number of the tag
* @return
*/
public static byte[] buildSelectMaskCmd(byte[] pMask);

/**
* Read tag data
*
* @param pAccPwd    Access password, used for the tag to enter the safe
 state, the default value is 0x00000000;
* @param pMemBank   The storage area of the tag to be read, the value l
ist is as follows:
*                    0x00: Reserved;  0x01: EPC;  0x02: TID;  0x03: User;
* @param pWordPtr   Points to the read start address of the logical sto
rage area (word);
* @param pWordCount The number of words to be read cannot be 0. The def
ault value is 4. The value range is [1,120]
* @return
*/
public static byte[] buildReadISOTagCmd(
byte[] pAccPwd,
byte pMemBank,
byte[] pWordPtr,
byte pWordCount
);

/**
* Write tag data
*
* @param pAccPwd    Access password, used for the tag to enter the safe
 state, the default value is 0x00000000;
* @param pMemBank   The storage area of the tag to be read, the value l
ist is as follows:
*                    0x00: Reserved;  0x01: EPC;  0x02: TID;  0x03: User;
* @param pWordPtr   Points to the read start address of the logical sto
rage area (words);
* @param pWordCount The number of data words that need to be written to
 the tag (1 word is two bytes), must be greater than 0;
* @param pData      The data to be written to the tag must be an intege
r multiple of words, with a length of 1 to WordCount words.
* @return
*/
public static byte[] buildWriteISOTagCmd(
byte[] pAccPwd,
byte pMemBank,
byte[] pWordPtr,
byte pWordCount,
byte[] pData
);

/**
* Lock tag data
*
* @param pAccPwd Access password, used for the tag to enter the safe st
ate, the default value is 0x00000000
* @param pArea   The area that needs to be locked, the value list is as
 follows:
*                0x00: Kill password area; 0x01: Access password area;
0x02: EPC; 0x03: TID; 0x04: User;
* @param pAction Lock operation type, the value list is as follows:
*                0x00: open; 0x01: permanently open; 0x02: locked; 0x03:
 permanently locked;
*/
public static byte[] buildLockISOTagCmd(byte[] pAccPwd, byte pArea, byt
e pAction);

/**
* Kill Tags
*
* @param pKLENIPwd Four-byte kill key
* @return
*/
public static byte[] buildKlenlISOTagCmd(byte[] pKLENIPwd);

/**
* Get permission parameters
*
* @return
*/
public static byte[] buildGetPermissionParamCmd();

/**
* Set permission parameters
*
* @param pBean Permission parameter bean class
* @return
*/
public static byte[] buildSetPermissionParamCmd(PermissionParamBean pBe
an);
```

4.2 Command type: Use the CmdType class to reference the command type

```
/**
 * Inventory Label
 */
public static final int TYPE_INVENTORY = 0x0001;
/**
 * Stop Inventory tags
 */
public static final int TYPE_STOP_INVENTORY = 0x0002;
/**
 * Read Tag
 */
public static final int TYPE_READ_TAG = 0x0003;
/**
 * Write Tag
 */
public static final int TYPE_WRITE_TAG = 0x0004;
/**
 * Lock Tag
 */
public static final int TYPE_LOCK_TAG = 0x0005;
/**
 * Kill Tag
 */
public static final int TYPE_KILL_TAG = 0x0006;
/**
 * Select Tag
 */
public static final int TYPE_SELECT_MASK = 0x0007;
/**
 * Module initialization
 */
public static final int TYPE_MODULE_INIT = 0x0050;
/**
 * Restore factory settings
 */
public static final int TYPE_REBOOT = 0x0052;
/**
 * Network parameters Interface parameters
 */
public static final int TYPE_REMOTE_NET_PARA = 0x005F;
/**
 * Get device information
 */
public static final int TYPE_GET_DEVICE_INFO = 0x0070;
/**
 * Set all parameters
 */
```

```java
public static final int TYPE_SET_ALL_PARAM = 0x0071;
/**
 * Get all parameters
 */
public static final int TYPE_GET_ALL_PARAM = 0x0072;
/**
 * Set or get permission parameters
 */
public static final int TYPE_GET_OR_SET_PERMISSION = 0x0076;
/**
 * Get battery capacity
 */
public static final int TYPE_GET_BATTERY_CAPACITY = 0x0083;
/**
 * Set or get bluetooth name
 */
public static final int TYPE_SET_OR_GET_BT_NAME = 0x0086;
/**
 * Output mode
 */
public static final int TYPE_OUT_MODE = 0x0088;
/**
 * Key State
 */
public static final int TYPE_KEY_STATE = 0x0089;
```

5. Instruction parsing return class

5.1 CmdData

The data returned by the interface is returned through this class, and the returned data type is known by calling the getDataType() function

```java
/**
 * CmdDataThe type of data included
 */
private GeneralBean mGeneralBean;
private TagInfoBean mTagInfoBean;
private AllParamBean mAllParamBean;
private BatteryCapacityBean mBatteryCapacityBean;
private DeviceNameBean mDeviceNameBean;
private DeviceInfoBean mDeviceInfoBean;
private OutputModeBean mOutputModeBean;
private PermissionParamBean mPermissionParamBean;
private TagOperationBean mTagOperationBean;
private KeyStateBean mKeyStateBean;
private RemoteNetParaBean mRemoteNetParaBean;


/**
```

```
 * Get the type of the current assigned data by calling getDataType() *
 * @return Returns the current data type object
 */
public Object getData()

/**
 * Get data type
 *
 * @return Returns the class object of the data type

 */
public Class<?> getDataType()
```

5.2 GeneralBean

Command status general class, used to record response status and respon se information

```
/**
 * Status Code
 */
public int mStatus;
/**
 * Status Information

 */
public String mMsg;
```

5.3 AllParamBean

All parameter classes, through which all configuration parameters are o btained and set

```
/**
 * The communication address of the device is 0x00 by default. This add
ress cannot be 0xFF. If it is set to 0xFF, the read/write module will r
eturn parameter error information.
 */
public byte mAddr;

/**
 * Protocol standard specification for equipment radio frequency RFID,
 * 0x00: ISO 18000-6C;
 * 0x01: GB/T 29768; 0x02:
 * GJB 7377.1;
 * Currently only supports ISO 18000-6C。
 */
public byte mRFIDPRO;

/**
```

```
 * The working mode of the device, the default value is 0
 * 0  answer mode
 * 1  active mode
 * 2  trigger mode
 */
public byte mWorkMode;

/**
 * The communication interface of the device, the default value is 0x80,
 the specific meaning is as follows:
 * 0x80     RS232
 * 0x40     RS485
 * 0x20     RJ45
 * 0x10     WiFi
 * 0x01     USB
 * 0x02     keyboard
 * 0x04     CDC_COM
 */
public byte mInterface;

/**
 * Serial port baud rate, the default value is 4, the specific meaning
is as follows:
 * 0  9600bps
 * 1  19200 bps
 * 2  38400 bps
 * 3  57600 bps
 * 4  115200 bps
 */
public byte mBaudrate;

/**
 * Configuration parameters of the Wiegand data output interface, the d
efault value is 0x00, the specific meaning is as follows:
 * WGSet      Bit7              Bit6     Bit5        Bit4    Bit3  Bit2
      Bit1  Bit0
 * Bit Definition   0: Turn off Wiegand output    0: wg26   0: Low posi
tion first   spare    spare     spare     spare     spare
 * -----------1: Enable Wiegand output    1: wg34   1: High position fi
rst
 */
public byte mWGSet;

/**
 * The number of all antennas on the device. The bit indicates the ante
nna to be used. If the corresponding bit value is 1, it means that the
antenna is used, and if the value is 0, it means that the antenna is no
t used.;
 * Starting from the lowest bit, bit 0 represents antenna No. 1, bit 1
```

```
represents antenna No. 2, and so on, up to 8 antennas can be represente
d.;
 * Different modules support different antennas, depending on the speci
fic situation;
 * The default value is 0x01, indicating antenna No. 1.
 */
public byte mAnt;

/**
 * RFID frequency-related parameters of the device, used to select the
frequency band and the upper and lower frequency limits in each frequen
cy band, length 8Bytes */
public RfidFreq mRfidFreq;

/**
 * The RFID output power of the device, in dBm, with a range of [0, 33]
 dBm. Other values are invalid.
 */
public byte mRfidPower;

/**
 * The device wants to access the storage area of the tag.
 * 0x00: reserve;
 * 0x01 (default): EPC storage area;
 * 0x02: TID area;
 * 0x03: USER area;
 * 0x04: EPC+TID;
 * 0x05: EPC+USER;
 * 0x06: EPC+TID+USER;
 * 其This value is reserved. If other values appear in the command, a m
essage indicating parameter error will be returned.
 */
public byte mInquiryArea;

/**
 * Initial Q value used when querying EPC tags,
 * The Q value should be set so that the number of tags in the field is
 approximately equal to 2Q。
 * The default value of Q is 4, and the range of Q is 0 to 15.
 * If other values appear in the command, a message indicating paramete
r error will be returned.
 */
public byte mQValue;

/**
 * Session value used when querying EPC tags,
 * The default value is 0, and the range is [0, 3]. Other values,
 * A message about parameter errors will be returned.
 * 0  Session use S0
 * 1  Session use S1
 * 2  Session use S2
 * 3  Session use S3
 */
public byte mSession;

/**
 * The starting address of the tag storage area that the device wants t
o access, unit: Byte,
 * Default value: 0x00:
 * When accessing the EPC area, 0x00 indicates the starting address of
the EPC number segment except the CRC and PC segments.;
 * When accessing other storage areas, 0x00 indicates the starting addr
ess of the storage area.
 */
public byte mAcsAddr;

/**
 * The data length that the device wants to access the tag storage area,
 unit: Byte, default value: 0x00.
 */
public byte mAcsDataLen;

/**
 * Filter time, within this value time after successfully reading a tag
 data,
 * Filter out tags with the same data. The unit is: S,
 * The value range is: [0, 255], other values are invalid; the default
value is 0, no filtering.
 */
public byte mFilterTime;

/**
 * The query duration after the device receives the trigger signal, the
 unit is: S, the default value is 1, the value range is: [0, 255], othe
r values are invalid.
 */
public byte mTriggerTime;

/**
 * The duration of the buzzer beeping after the device is successfully
executed, unit: 10ms,
 * The value range is: [0, 255], other values are invalid; the default
value is 1,
 * When it is 0, it means the buzzer does not sound.
 */
public byte mBuzzerTime;
```

```
/**
 * Query interval, unit: 10ms, value range: [0, 255], other values are
invalid, default is 1.
 */
public byte mPollingInterval;

/**
 * Frequency band range of each country:
 * 0x00:  User customized according to needs;
 * 0x01: US [902.75~927.25]
 * 0x02: Korea [917.1~923.5]
 * 0x03: EU [865.1~868.1]
 * 0x04: JAPAN [952.2~953.6]
 * 0x05: MALAYSIA [919.5~922.5]
 * 0x06: EU3 [865.7~867,5]
 * 0x07: CHINA_BAND1 [840.125~844.875]
 * 0x08: CHINA_BAND2 [920.125~924.875]
 */
public byte mREGION;

/**
 * Length two digits
 * Integer part of the starting frequency in megahertz; for example, 92
0.125MHz, STRATFREI = 920 = 0x0398, high byte = 0x03, low byte = 0x98;
 */
public byte[] mSTRATFREI;

/**
 * Length two digits
 * Fractional part of the starting frequency in megahertz; for example,
 920.125MHz, STRATFRED = 125, high byte = 0x00, low byte = 0x7D

 */
public byte[] mSTRATFRED;

/**
 * Length two digits
 * Frequency step (KHz), please refer to the calculation formula of eac
h frequency band; for example, 125KHz, STEPFRE = 125, high byte = 0x00,
 low byte = 0x7D;
 */
public byte[] mSTEPFRE;

/**
 * Number of channels;
 */
public byte mCN;
```

5.4 BatteryCapacityBean

Battery power class, get device power information

```
/**
 * Battery percentage
 */
public byte mBatteryCapacity;
```

5.5 DeviceInfoBean

Device information class, get device information

```
/**
 * Hardware version
 */
public String mHwVer;
/**
 * Firmware version
 */
public String mFirmVer;
/**
 * Serial Number
 */
public String mSn;
/**
 * RFID module version
 */
public String mRFIDModeVer;
/**
 * RFID module name
 */
public String mRFIDModeName;
/**
 * RFID module serial number
 */
public String mRFIDModeSn;
```

5.6 DeviceNameBean

Device Name Class

```
/**
 * Operation Type
 * 0x01: set
 * 0x02: read
 */
public byte mOption;
/**
 * device name
 */
public String mDeviceName;
```

5.7 KeyStateBean

  Button Status Class

```
/**
 * Button Status
 * 0x01: start;
 * 0x02: finish;
 */
public byte mKeyState;
```

5.8 OutputModeBean

Output Mode Class

```
/**
 * Operation Type

 * 0x01 set
 * 0x02 get
 */
public byte mOption;
/**
 * output mode
 * 0x00: bluetooth HID output
 * 0x01: Bluetooth Transmit Out
 */
public byte mMode = -1;
```

5.9 PermissionParamBean

Permission parameter class

```
/**
 * Command control option, 0x01 set, 0x02 read, other values are invali
d.
 */
public byte mOption;
/**
 * :Password function enable parameter, length 1 byte. 0x01 enables, 0x
00 disables. Default 0x00.
 */
public byte mCodeEN;
/**
 * The access password of the tag is 4 bytes long. Default value is [0x
00,0x00,0x00,0x00].
 */
public byte[] mCodes;
```

```
/**
 * Mask function enable parameter, length 1 byte. 0x01 enables, 0x00 di
sables. Default 0x00.
 */
public byte mMaskEN;
/**
 * Mask start address, length 1 byte, unit byte. Default 0x00.
 */
public byte mStartAdd;
/**
 * Mask length, length 1 byte, unit byte, maximum 12. Default 0x00.
 */
public byte mMaskLen;
/**
 * Mask data, length 31 bytes. When the mask length is less than 31 byt
es, the following bytes are supplemented with 0.
 * The default data is 0x00.
 */
public byte[] mMaskData;
/**
 * :Mask condition, length 1 byte.
 * 0x00: Password or mask matches;
 * 0x01:Password and mask both match.
 * Default 0x00.
 */
public byte mMaskCondition;
```

5.10 RemoteNetParaBean

LAN device discovery, response data type

```
/**
 * Action Item
 * 0x01: set;  0x02: get */
public byte mOption;
/**
 * 4 bytes
 * device IP addree, If the IP address is 192.168.1.1, the data is [0xC
0, A8, 0x01, 0x01]
 */
public byte[] mIpAddr;
/**
 * 6 bytes
 * Mac address, if multiple devices work in the same field, please use
different MAC addresses;
 * example: mac[6]={0x00,0x08,0xdc,0x11,0x11,0x11}, macaddress is 00-08
-DC-11-11-11
 */
public byte[] mMacAddr;
/**
```

```
 * 2 bytes
 * Listening port, the value is [0,65536], the default is 5000;
 */
public byte[] mPort;
/**
 * 4 bytes
 * Subnet mask, default is [0xFF,0xFF,0xFF,0x00]
 */
public byte[] mNetMask;
/**
 * 4 bytes
 * Default gateway, default is [0xC0,0xA8,0x01,0x01]

 */
public byte[] mGateWay;
```

5.11 TagInfoBean

Tag Information Class

```
/**
 * RSSI of tag ACK response, in dBm, signed number, negative numbers us
e the two's complement format;
 */
public int mRSSI;
/**
 * From which antenna port the tag data is received. The value range is:
 1~4, representing antennas 1 to 4 respectively.
 */
public int mAntenna;
/**
 * The channel from which the tag data is received. The value starts fr
om 0, 0 represents channel 0, 1 represents channel 1, and so on.;
 */
public int mChannel;
/**
 * The length of the tag's EPC number (bytes);
 */
public int mEPCLen;
/**
 * The EPC number of the label;
 */
public byte[] mEPCNum;
```

5.12 TagOperationBean

Tag operation response class

```
  /**
 * Tag Status
 */
```

```
public byte mTagStatus;
/**
 * From which antenna port the tag data is received. The value range is:
1~4, representing antennas 1 to 4 respectively.
 */
public byte mAntenna;
/**
 * 2bytes
 * CRC data in tag response data
 */
public byte[] mCRC;
/**
 * 2bytes
 * PC data in tag response data */
public byte[] mPC;
/**
 * The length of the tag's EPC number (bytes)
 */
public byte mEPCLen;
/**
 * The EPC number of the label;
 */
public byte[] mEPCNum;

//The following two fields have values only when reading tags
/**
 * The number of tag data words successfully read;
 */
public byte mWordCount;
/**
 * N bytes
 * The tag data successfully read has a length of WordCount×2 bytes
 */
public byte[] mData;
```

6. Appendix

Appendix A. Definition of STATUS

STATUS	Error description
0x00	Execution successful (this only means that the module successfully received the tag response data. If there is a tag execution status in the tag response, it should be further determined whether the tag execution status is correct)
0x01	The parameter value is incorrect or out of range, or the module does not support the parameter value.
0x02	Command execution failed due to an internal error in the module (setting frequency or setting power)

STATUS	Error description
0x03	reserve
0x12	The entire inventory command is executed
0x13	No tags were inventory
0x14	Tag response timeout
0x15	Demodulation tag response error
0x16	Protocol authentication failed
0x17	Wrong password
0xFF	No more data