Newland AIDC
SCANNING MADE SIMPLE

RFID MODULE

For generic Android devices

Version: V2.3

Development Guide

Disclaimer

© 2023 Newland Europe BV. All rights reserved.

Please read the manual carefully before using the product and operate it according to the manual. It is advised that you keep this manual for future reference.

Do not disassemble the device or remove the seal label from the device; doing so will void the product warranty provided by Newland Europe BV.

All pictures in this manual are for reference only, and the actual product may differ.

Regarding product modification and update, Newland Europe BV reserves the right to make changes to any software or hardware to improve reliability, function, or design at any time without notice. The information contained herein is subject to change without prior notice.

The products depicted in this manual may include software copyrighted by Newland Europe BV or a third party. The user, corporation or individual shall not duplicate, in whole or in part, distribute, modify, decompile, disassemble, decode, reverse engineer, rent, transfer, or sublicense such software without prior written consent from the copyright holders.

This manual is copyrighted. No part of this publication may be reproduced, distributed, or used in any form without Newland Europe BV's written permission.

Risk Warning Regarding Unauthorized System Updates:
You should use the Newland-provided tool to update this product's system. Modifying system files by installing a third-party ROM system or using any cracking method may result in product malfunction or data loss and void your warranty.

Newland Europe BV reserves the right to make a final interpretation of the statement above.

Newland Europe BV

Rolweg 25, 4104 AV, Culemborg,
The Netherlands
www.newland-id.com

Newland Europe BV is a subsidiary of Newland Digital Technology Co., Ltd. Our general conditions of Purchase, Sale and Delivery are filed with the Record Office of the Chamber of Commerce of Utrecht, The Netherlands.

K.v.K. H.R. Utrecht / Chamber of Commerce Utrecht: Reg. nr. 17109876

Contents

About This Manual ***0***

Development Process ***0***

Importing SDK 0

Registering the Necessary Listeners 1

Connecting to the Module 1

Module Functions 1

How to Develop Functions ***2***

Register Status Listener (Broadcast Receiver) 2

Register Tag Information Listener (Broadcast Receiver) 3

RFID Module Connection and Disconnection 3

Inventory 4

Inventory Trigger 5

Inventory Notification 6

Read Write Lock 6

Parameters Setting 10

Association Classes ***12***

UHFReader 12

TagInfo 13

UHFModuleInfo 14

Revision History ***14***

About This Manual

The document provides guidance on Android application development for Newland RFID functionality, enabling third-party applications to properly operate the RFID module. The document is organized by functional modules, allowing developers to refer to the relevant development solutions for the required functional interfaces.

Development Process

Importing SDK

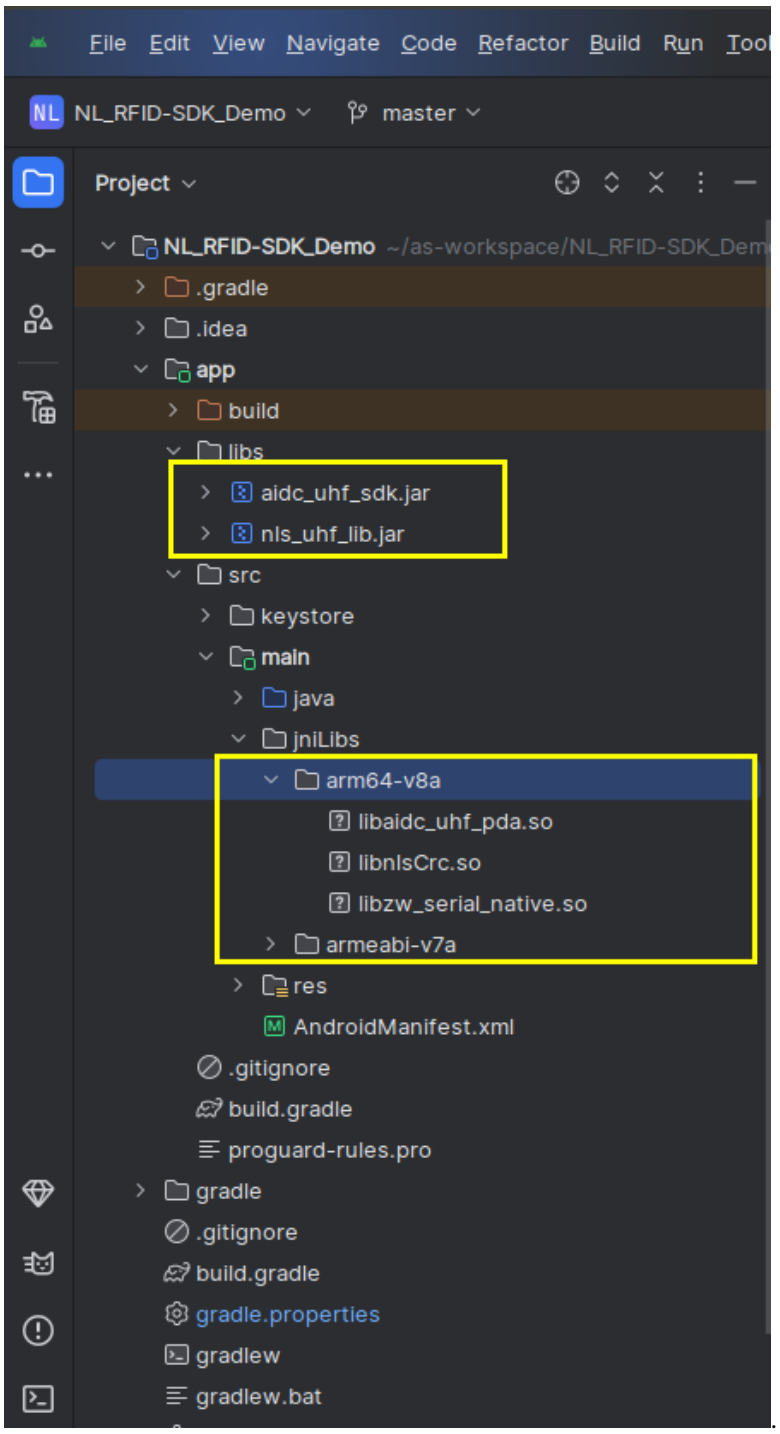
The SDK provides API interfaces based on the Java language. Developers need to import the SDK's JAR and SO library files into the Android project's compilation environment.

For example, in ***Android Studio***:

- Place the ***JAR*** file in the ***app/libs*** directory.
- Place the ***SO*** files in the ***app/src/main/jniLibs*** directory.
- Then, include them in the compilation dependencies.

Example (in app/***build.gradle***):

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation fileTree(dir: 'src/main/jniLibs', include: ['*.so'])
}
```



Registering the Necessary Listeners

1、Connection & Inventory Status Monitoring:

- The device connection interface operates asynchronously, and status changes are transmitted via broadcasts.
- ***You must first register a status listener*** to receive connection and inventory state updates.

2、Inventory Data Listening:

- Inventory (tag scanning) data is transmitted via broadcasts.
- ***You must first register a tag data listener*** to receive real-time RFID tag information.

Connecting to the Module

To perform RFID-related operations such as inventory, read/write, and parameter settings on the module, you need to first call the API interface to establish a successful connection with the module. Once the connection is established, you can proceed with the desired RFID operations.

Module Functions

After establishing the connection, you can operate the following RFID functions:

- Tag Inventory
- Read, Write, Lock
- Parameters Settings

How to Develop Functions

Register Status Listener (Broadcast Receiver)

To monitor ***connection*** and ***inventory (scanning)*** status events, you must ***register the corresponding listeners*** before performing connection or inventory operations.

Action：com.nlscan.intent.action.ACTOIN_UHF_STATE_CHANGE_EX	
Extra Parameters	Description
uhf_state	Value Type: int Definition: When the module's connection status changes, it will send different status values to indicate the updated state. ***0x01: Connected, indicating a successful connection.*** ***0x02: Disconnected, indicating that the connection with the module has been disconnected.*** ***0x03: Connecting, indicating that the connection process is in progress but not yet completed.*** ***0x04: The inventory function is started.*** ***0x05: Inventory function is stopped.*** ***0x06: Failed to connect to RFID module.***

Sample:

```
//Register status listener
IntentFilter iFilter = new IntentFilter("com.nlscan.intent.action.ACTOIN_UHF_STATE_CHANGE_EX");
mContext.registerReceiver(mUHFStateReceiver, iFilter);

public static final int UHF_STATE_CONNECTED = 0x01;
public static final int UHF_STATE_DISCONNECTED = 0x02;
public static final int UHF_STATE_CONNECTING = 0x03;
public static final int UHF_STATE_START_INVENTORY = 0x04;
public static final int UHF_STATE_STOP_INVENTORY = 0x05;
public static final int UHF_STATE_CONNECT_FAIL = 0x06;

private BroadcastReceiver mUHFStateReceiver = new BroadcastReceiver() {

        @Override
        public void onReceive(Context context, Intent intent) {

                if(intent == null)
                        return ;
                final String STATE_ACTION = "com.nlscan.intent.action.ACTOIN_UHF_STATE_CHANGE_EX";
                if(STATE_ACTION.equals(intent.getAction()))
                {
                        int uhf_state = intent.getIntExtra("uhf_state", -1);
                        switch (uhf_state) {
                                case UHF_STATE_CONNECTING:
                                        //Connecting, indicating that the connection process is in progress but not yet completed.
                                        break;
                                case UHF_STATE_CONNECTED:
                                        //Connected, indicating that the connection is successful
                                        break;
                                case UHF_STATE_DISCONNECTED:
                                        //Disconnected, indicating that the connection with the module has been disconnected
                                case UHF_STATE_START_INVENTORY:
                                        //The inventory function is started
                                        break;
                                case UHF_STATE_STOP_INVENTORY:
                                        //Inventory function is stopped
                                        break;
                                case UHF_STATE_CONNECT_FAIL:
                                        //Failed to connect to RFID module
                                        break;
                        }
                }
```

```
        }
};
```

Register Tag Information Listener (Broadcast Receiver)

To ensure ***complete and real-time RFID tag data*** during inventory (scanning), you ***must register a broadcast receiver before*** starting the inventory process. If the listener is registered too late, some tag data may be missed.

Action：nlscan.intent.action.uhf.ACTION_RESULT_EX		
Extra Parameters	Data Type	Description
tag_info	Parcelable[]	Value Type: Parcelable[] (Please refer to the description of the associated class "TagInfo" for reference). Definition: Used to receive broadcasted data during tag inventory.
extra_start_reading_time	long	The start time (in milliseconds) of the current inventory session, which can be used to calculate the duration of the entire inventory process.

Sample:

```
public final static String ACTION_UHF_RESULT_SEND = "nlscan.intent.action.uhf.ACTION_RESULT_EX";
public final static String EXTRA_TAG_INFO = "tag_info";

//Register receiver.
IntentFilter iFilter = new IntentFilter(ACTION_UHF_RESULT_SEND);
mContext.registerReceiver(mUhfBR, iFilter);

private BroadcastReceiver mUhfBR = new BroadcastReceiver() {

        @Override
        public void onReceive(Context context, Intent intent) {
                String action = intent.getAction();
                if (!ACTION_UHF_RESULT_SEND.equals(action))
                        return;

                Parcelable[] tagInfos = intent.getParcelableArrayExtra(EXTRA_TAG_INFO);
                for (Parcelable parcel : tagInfos) {
                        TagInfo tagInfo = (TagInfo) parcel;
                        //To hex string.
                        final String sHexEpcId = UHFReader.bytes_Hexstr(tagInfo.EpcId);
                }

        }
    };
```

RFID Module Connection and Disconnection

1. ***Connection First*** – Before performing any operations (***inventory, read/write/lock, parameter settings***), you ***must*** establish a successful connection to the RFID module.
2. ***Disconnect When Idle*** – To ***save power***, disconnect the RFID module when not in use.

Interface Definition:

Class：com.nlscan.uhf.lib.UHFManager	
Return Value	Method
static UHFManager	***getInstance(Context context)*** ***asyncConnect (String devPathOrMacOrIp,*** ***int pluginType,*** ***String deviceModel,*** ***IUHFConnectStateListener***

	connectStateListener) Description: Connect to RFID Reader Module asynchronously, and the connection status is obtained through a status listener (Refer to section about registering a status listener). Parameters: ***devPathOrMacOrIp: The category of data passed, such as the device's serial port path, BT MAC address, network IP, etc., depends on the connection method of the RFID module.*** ***pluginType: Connection type such as serial port, Bluetooth, network. If the connection type is unknown, the pluginType is set to 0.*** ({@link UHFManager#PLUGIN_TYPE_SERIEL} Series Port) ({@link UHFManager#PLUGIN_TYPE_BLE} Bluetooth) ({@link UHFManager#PLUGIN_TYPE_NETWORK} network) ***deviceModel: RFID module name such as URM500， URF520, or null*** ***connectStateListener: A callback for receiving the connection status result. It is recommended to set it as null and use the broadcasting listener approach for monitoring the connection status.***
boolean	***isConnected()*** ***Description: Checks whether the connection is established. Returns: true - Yes, false - No***
UHFReader.READER_STATE	***disconnect()*** ***Description: Disconnect the R***FID***.*** ***Returns: UHFReader.READER_STATE.OK_ERR - Success*** ***Other values - Failure*** ***(Note: It is necessary to call the disconnect function when exiting the RFID module.)***

Sample:

```
// Connect to RFID modules such as URM_E7， URM500 via serial port
UHFManager mUHFMgr = UHFManager.getInstance(mContext);
mUHFMgr. asyncConnect("/dev/ttyS0:115200",UHFManager.PLUGIN_TYPE_SERIEL,"" ,null);
// use the registered status listener to monitor and receive the connection status results

// Disconnect
UHFReader.READER_STATE er = mUHFMgr.disconnect();
if(er == UHFReader.READER_STATE.OK_ERR)
    // successfully
else
    //fail
```

Inventory

- ***Start Inventory***: Activates continuous scanning mode, where the RFID reader detects nearby tags and transmits data ***via broadcast***.
- ***Stop Inventory***: Ends the scanning process to conserve power or halt operations.
- ***Data Flow***: Tag data is broadcast in real-time—***must register a listener first*** (refer to *Section :* ***Registering the Necessary Listeners***).

Interface Definition:

- ***Start/Stop Tag Inventory***

Class: com.nlscan.uhf.lib.UHFManager	
Return Value	Method
static UHFManager	***getInstance(Context context)*** To get instance
UHFReader.READER_STATE	***startTagInventory()*** To start tag inventory Return value: (@See ***UHFReader.READER_STATE***)
UHFReader.READER_STATE	***stopTagInventory()*** To stop tag inventory Return value: (@See***UHFReader.READER_STATE***)

The inventory data is uploaded through broadcasting, and please refer to section about "Register Tag Information Listener (Broadcast receiver)".

```
//Get UHFManager instance
UHFManager mUHFMgr = UHFManager.getInstance(mContext);

//----Start inventory
UHFReader.READER_STATE er = mUHFMgr.startTagInventory();
if( er == UHFReader.READER_STATE.OK_ERR)
    //success.....
else
    //fail.....
```

It is necessary to call the "stop tag inventory" function when exiting the use of the inventory function.

```
//----Stop inventory
UHFReader.READER_STATE er = mUHFMgr.stopTagInventory();
if( er == UHFReader.READER_STATE.OK_ERR)
    //success.....
else
    //fail.....
```

Inventory Trigger

Note: （Only supported on Newland PDA）

Description: Set inventory operation triggers, including SCAN key, left/right SCAN key, Trigger on pistol grip.

- ***SCAN Key:*** Use the Scan key on the front panel of the terminal as inventory trigger.
- ***Left/Right SCAN key:*** Use the Scan keys on the left/right side of the terminal as inventory trigger.
- ***Trigger on pistol grip:*** Use the trigger on the pistol grip attached to the terminal as inventory trigger.

Class: com.nlscan.uhf.lib.UHFManager	
Return Value	Method
static UHFManager	***getInstance(Context context)*** To get instance
boolean	***setTrigger(String trigger,boolean on)*** To set trigger Parameters: trigger identifier Main SCAN key： uhf_trigger_mode_main Left/Right SCAN key： uhf_trigger_mode_left/uhf_trigger_mode_right Trigger on the pistol grip： uhf_trigger_mode_black Parameters: on true-enable, false-disable Return value: true: set successfully, false: set failed
boolean	***isTriggerOn(String trigger)*** Whether the scan trigger is enabled or not? Parameters: trigger identifier Main SCAN key： uhf_trigger_mode_main Left/Right SCAN key： uhf_trigger_mode_left/uhf_trigger_mode_right Trigger on the pistol grip： uhf_trigger_mode_black Return value: true: enable, false: disable

Example：

```
//get UHFManager instance
UHFManager mUHFMgr = UHFManager.getInstance(mContext);
```

```
//-- set trigger
 boolean bGunScan = true;
 boolean suc = mUHFMgr.setTrigger("uhf_trigger_mode_black", bGunScan);

//-----get isTriggerOn
 boolean gunTriggerEnable = mUHFMgr.isTriggerOn("uhf_trigger_mode_black");
```

Inventory Notification

Description: Use sound and/or vibration to indicate a good tag read.

- ***Sound***：Use sound to indicate a good tag read.
- ***Vibration***：Use vibration to indicate a good tag read.

Description: Use sound and/or vibration to indicate a good tag read.

- ***Sound***：Use sound to indicate a good tag read.
- ***Vibration***：Use vibration to indicate a good tag read.

Interface definition:

Class: com.nlscan.uhf.lib.UHFManager	
Return Value	Method
static UHFManager	***getInstance(Context context)*** To get instance
boolean	***setPromptSoundEnable(boolean enable)*** To set sound on/off Parameters: enable/disable sound (true: enable, false: disable) Return value: true: set successfully, false: set failed
boolean	***setPromptVibrateEnable(boolean enable)*** To set vibration on/off Parameters: enable/disable vibration (true: enable, false: disable) Return value: true: set successfully, false: set failed
boolean	***isPromptSoundEnable()*** Enable sound or not? Return value: true: enable, false: disable
boolean	***isPromptVibrateEnable()*** Enable vibration or not? Return value: true: enable, false: disable

Example：

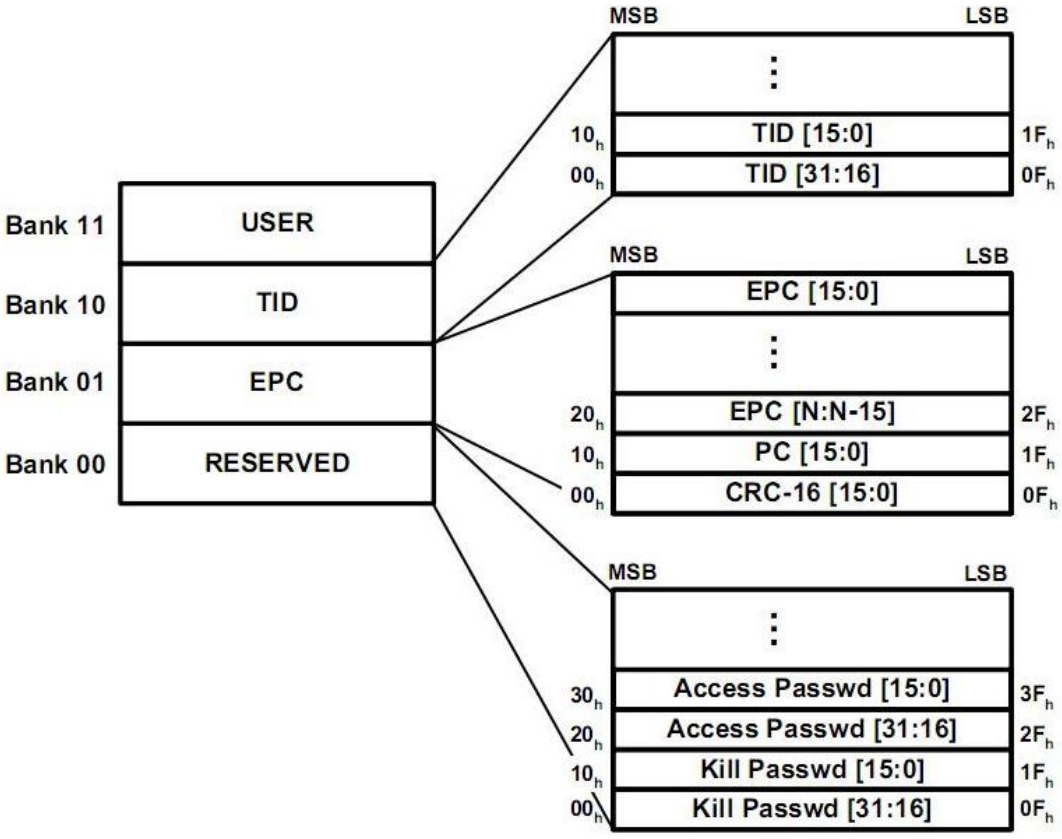
```
// get UHFManager instance
UHFManager mUHFMgr = UHFManager.getInstance(mContext);
//----set inventory notification
boolean sound = true;// sound
boolean vibrate = true;// vibrate
boolean suc = mUHFMgr.setPromptSoundEnable(sound);
suc = mUHFMgr.setPromptVibrateEnable(vibrate);

//----get current notification status
boolean isSound = mUHFMgr.isPromptSoundEnable();
boolean isVibrate = mUHFMgr.isPromptVibrateEnable();
```

Read Write Lock

Description: You can read, write and lock tags.

RFID Tag Memory Bank Structure:



Tags compliant with ***EPC Class 1 Gen 2/ISO 18000-6C*** standards partition memory into four banks, each with distinct functions and addressing rules. All operations use ***16-bit blocks*** as the minimal unit.

1. Bank 0 (Reserved Memory)

- ***Purpose***: Stores security credentials.
- ***Layout***:
  - ***Access Password***: Block 0–1 (32 bits total).
  - ***Kill Password***: Block 2–3 (32 bits total).
- ***Properties***:
  - Write-lockable (permanently protect passwords).
  - Address example: ***Bank=0, Block=0*** reads the first 16 bits of the Access Password.

2. Bank 1 (EPC Memory)

- ***Purpose***: Stores the Electronic Product Code and metadata.
- ***Layout***:
  - ***CRC-16***: Block 0 (16 bits, checksum for EPC).
  - ***PC (Protocol Control)***: Block 1 (16 bits, encodes EPC length and features).
  - ***EPC Code***: Blocks 2–N (variable length, typically 96 bits = Blocks 2–7).
- ***Note***: EPC length is adjustable (up to 496 bits = Blocks 2–31).

3. Bank 2 (TID Memory)

- ***Purpose***: Factory-programmed unique identifier.
- ***Layout***:
  - ***TID (Tag ID)***: Blocks 0–3 (64 bits, globally unique).
  - ***AFI/DSFID*** (optional): Additional application identifiers.
- ***Properties***: Read-only; used for anti-cloning.

4. Bank 3 (User Memory)

- ***Purpose***: Custom user data storage.
- ***Layout***:
  - Variable capacity (e.g., 32–512 bits = Blocks 0–31).
  - Some tags omit this bank entirely.
- ***Usage***: Store application-specific data (e.g., maintenance logs).

- Read: Read data in a specified memory bank, e.g. read EPC ID in EPC memory bank.
- Write: Write data to a specified memory bank, e.g. write access password to the Reserved Memory Bank.
- Lock: Lock data in a specified memory bank. Enter access password to rewritten the locked tag (Set access password before locking the tag).

  Lock options：

  - Temporary Lock: Lock the selected memory bank temporarily for reading and writing.
  - Permanent Lock: Lock the selected memory bank permanently for reading and writing.

- Destroy: The Kill Command permanently disables an RFID tag, rendering it unreadable and unusable. This is irreversible and requires:
  - Valid Kill Password (32-bit, stored in Bank0, Blocks 2–3).
  - Successful authentication (password match).

Interface definition:

Class: com.nlscan.uhf.lib.UHFManager	
Return Value	Method
static UHFManager	***getInstance(Context context)*** To get instance
byte[]	***GetTagData(int antId,*** ***int bank,*** ***int address,*** ***int blkcnt,*** ***String hexAccesspasswd)*** To obtain data in a specified memory bank Parameter: antId (The antenna ID, support multi-antenna modules. For single-antenna modules: antId = 1). Example: The URF520/URF120 module supports multiple antennas. Parameters: Types of Memory Banks(@See UHFReader.BANK_TYPE) Reserved memory bank: UHFReader.BANK_TYPE.RESERVED EPC memory bank: UHFReader.BANK_TYPE.EPC TID memory bank: UHFReader.BANK_TYPE.TID USER memory bank: UHFReader.BANK_TYPE.USER Parameter: start address, unit: block (16bit in a block) Parameter: blkcnt (Number of blocks to read) Parameter: hexAccesspasswd. If access password is required, (4-byte hexadecimal string, e.g. (0x): 0D0A0305, the length should be a multiple of two), enter it (4-byte data). If not required, pass NULL. Return value: byte[], Byte read.
UHFReader.READER_STATE	***writeTagData(int antId,*** ***int bank,*** ***int address,*** ***byte[] data,*** ***String hexAccesspasswd)*** To write data to each memory bank Parameter: antId (The antenna ID, support multi-antenna modules. For single-antenna modules: antId = 1). Example: The URF520/URF120 module supports multiple antennas. Parameters: Types of Memory Banks(@See UHFReader.BANK_TYPE) Reserved memory bank: UHFReader.BANK_TYPE.RESERVED EPC memory bank: UHFReader.BANK_TYPE.EPC TID memory bank: UHFReader.BANK_TYPE.TID USER memory bank: UHFReader.BANK_TYPE.USER Parameter: start address, unit: block (16bit in a block) Parameter: read-in data (Byte arrays) Parameter: hexAccesspasswd. If access password is required, (4-byte hexadecimal string, e.g. (0x): 0D0A0305, the length should be a multiple of two), enter it（4-byte data）. If not required, pass NULL. Return value: (@See UHFReader.READER_STATE)
UHFReader.READER_STATE	***writeTagEpcEx(int antId,*** ***byte[] epcData,*** ***String hexAccesspasswd)*** To write data to EPC memory bank (EPC memory bank is usually used to store tag initialization info. Locked EPC memory bank is not writable.) Parameter: antId (The antenna ID, support multi-antenna modules. For single-antenna modules: antId = 1). Example: The URF520/URF120 module supports multiple antennas. Parameter: read-in epcData (Byte arrays) Parameter: hexAccesspasswd. If access password is required, (4-byte hexadecimal string, e.g. (0x): 0D0A0305, the length should be a multiple of two), enter it（4-byte data）. If not required, pass NULL. Return value: (@See UHFReader.READER_STATE)

UHFReader.READER_STATE	***LockTag(int antId,*** ***int lockObjects,*** ***int lockTypes,*** ***String hexAccesspasswd)*** To lock data in a specified memory bank Memory areas of a tag can be locked: kill password, access password, EPC (bank 1), TID (bank 2), USER (bank 3). Set access password none-zero before locking tag. Access password is required to lock data. One function call can perform three types of lock operation (unlock/temporary lock/permanent lock) on multiple areas at the same time Parameter: antId (The antenna ID, support multi-antenna modules. For single-antenna modules: antId = 1). Example: The URF520/URF120 module supports multiple antennas. Parameter: lockObject, parameter (@See UHFReader.Lock_Obj) (Select several areas and use "|" to connect.) Parameter: LockType l, parameter (@See UHFReader.Lock_Type) （Select several types based on selected area and use "|" to connect.） Parameter: hexAccesspasswd If access password is required, (4-byte hexadecimal string, e.g. (0x): 0D0A0305, the length should be a multiple of two), enter it (4-byte data). If not required, pass NULL. Return value: (@See UHFReader.READER_STATE)
UHFReader.READER_STATE	***destroyTag (int antId, String hexDestroyPasswd)*** Destroy tag, when the tag is destroyed, it will become invalid and cannot be used again. (Set destroy password before calling this method) Parameter: antId (The antenna ID, support multi-antenna modules. For single-antenna modules: antId = 1). Example: The URF520/URF120 module supports multiple antennas. Parameters: hexDestroyPasswd, destroy password (hexadecimal string), cannot be empty.

Example:

//get UHFManager instance

```
UHFManager mUHFMgr = UHFManager.getInstance(mContext);
```

//----Read tag data

```
int bank = UHFReader.BANK_TYPE.EPC.value();//EPC Bank
int antId = 1;//Ant ID
int startAddr = 0;// starting address from block 0th
int blkcnt = 2 ;// read two blocks, that is to read 4 bytes.
String hexAccesspasswd = "00000001";// access password（hexadecimal string）, if no access password entered, set null.
byte[] rdata = mUHFMgr.GetTagData(antId,bank, startAddr, blkcnt, sHexPasswd);
```

//----Write tag data

```
int bank = UHFReader. BANK_TYPE.EPC.value();//EPC Bank
int startAddr = 0;// starting address from block 0th
byte[]data = UHFReader.Str2Hex("27099801201000011301");// write data（hexadecimal change to byte[]）
String sHexPasswd = "00000001";// access password（hexadecimal string）, if no access password entered, set null.
int antId = 1;//Ant ID
UHFReader.READER_STATE er = mUHFMgr.writeTagData(antId,bank, startAddr, data, sHexPasswd);
if( er == UHFReader.READER_STATE.OK_ERR )
        //success.....
else
        //fail.....
```

//----Directly write tag EPC data

```
byte[]data = UHFReader.Str2Hex("27099801201000011301");// data（hexadecimal change to byte[]）
String sHexPasswd = "00000001";//access password（hexadecimal string）, if no access password entered, set null.
int antId = 1;//Ant ID
UHFReader.READER_STATE    er = mUHFMgr.writeTagEpcEx(antId,data, sHexPasswd);
if( er == UHFReader.READER_STATE.OK_ERR)
        //success.....
else
        //fail.....


//----Lock/Unlock data
//Temporary lock EPC
UHFReader.Lock_Obj lobj = UHFReader.Lock_Obj.LOCK_OBJECT_BANK1; //EPC Bank
UHFReader.Lock_Type ltyp= UHFReader.Lock_Type.LOCK; // Temporary lock
int antId = 1;//Ant ID
String sHexPasswd = "00000001";// access password（hexadecimal string）
UHFReader.READER_STATE    er = mUHFMgr.LockTag(antId,lobj.value(), ltyp.value(), sHexPasswd);
if( er == UHFReader.READER_STATE. OK_ERR)
        //success.....
else
        //fail.....

//Unlock EPC
UHFReader.Lock_Objlobj =UHFReader.Lock_Obj.LOCK_OBJECT_BANK1; //EPC;
UHFReader.Lock_Typeltyp=UHFReader.Lock_Type.UNLOCK; //Unlock
String sHexPasswd = "00000001";//access password（hexadecimal string）
UHFReader.READER_STATEer = mUHFMgr.LockTag(lobj.value(), ltyp.value(), sHexPasswd);
if( er == UHFReader.READER_STATE. OK_ERR)
        //success.....
else
        //fail.....
UHFReader.Lock_Obj lobj = UHFReader.Lock_Obj.LOCK_OBJECT_BANK1; //EPC Bank
UHFReader.Lock_Type ltyp= UHFReader.Lock_Type.UNLOCK; // Unlock
int antId = 1;//Ant ID
String sHexPasswd = "00000001";// access password（hexadecimal string）
UHFReader.READER_STATE    er = mUHFMgr.LockTag(antId,lobj.value(), ltyp.value(), sHexPasswd);
if( er == UHFReader.READER_STATE. OK_ERR)
        //success.....
else
        //fail.....


//----Destroy Tag
String sHexPasswd = "00000001";// destroy password (hexadecimal string),
int antId = 1;//Ant ID
UHFReader.READER_STATE    er = mUHFMgr.destroyTag(antId,sHexPasswd);
if( er == UHFReader.READER_STATE. OK_ERR)
        //success.....
else
        //fail.....
```

Parameters Setting

Set RFID module parameters like inventory parameters, antenna power parameters, and additional data parameters. Reading and writing tag performance vary with different parameters.

Different RFID models may have different parameters. Pease refer to corresponding RFID Configuration Developer Handbook.

Interface Definition:

Class: com.nlscan.uhf.lib.UHFManager	
Return Value	Method

static UHFManager	***getInstance(Contex context)*** To get instance
UHFReader.READER_STATE	***setParam(String paramKey,*** ***String paramName,*** ***String sValue)*** Parameters setting Parameter: paramKey, function identifier (Please refer to corresponding model for parameter identifier.) Parameter: paramName, parameter name (Please refer to corresponding model for parameter identifier.) Parameter: sValue, Return value: (@See ***UHFReader.READER_STATE***)
String	***getParam(String paramKey,*** ***String paramName,*** ***String sDefaultValue)*** Parameter: paramKey, function identifier (Please refer to corresponding model for parameter identifier.) Parameter: paramName, parameter name (Please refer to corresponding model for parameter identifier.) Parameter: ***sDefaultValue*** Return value:String

Example :

//get UHFManager instance

```
UHFManager mUHFMgr = UHFManager.getInstance(mContext);

// Antenna parameter format
/*[{
        "antid": 1,
        "readPower": 33,
        "writePower": 33,
        "enable": true
}, {
        "antid": 2,
        "readPower": 33,
        "writePower": 33,
        "enable": true
}, {
        "antid": 3,
        "readPower": 33,
        "writePower": 33,
        "enable": false
}, {
        "antid": 4,
        "readPower": 33,
        "writePower": 33,
        "enable": false
}]*/


//----set antenna power
JSONArray jsAntArray = new JSONArray();
try {
        JSONObject jobj = new JSONObject();
        jobj.put("antid", 1);//Ant ID
        jobj.put("readPower", 2700); // read power
        jobj.put("writePower", 2000); // write power
        jobj.put("enable", true); ///true - enable the antenna, false - disable the antenna
        jsAntArray.put(jobj);
} catch (Exception e) {
}
String sAntPowerValue = jsAntArray.toString();
String paramKey = "RF_ANTPOWER";
String paramName = "PARAM_RF_ANTPOWER";
UHFReader.READER_STATE er = mUHFMgr.setParam(paramKey, paramName, sAntPowerValue);
```

```
if( er == UHFReader.READER_STATE.OK_ERR)
        //success.....
else
        //fail.....



//----get antenna power
//load “reader transmit power JSONArray ”
String paramKey = "RF_ANTPOWER";
String paramName = "PARAM_RF_ANTPOWER";
String sJsonValue = mUHFMgr.getParam(paramKey, paramName, "");
JSONArray jsArray = new JSONArray(sValue);
int len = jsArray.length();
for(int i =0 ;i < len;i++)
{
        JSONObject jobj = jsArray.optJSONObject(i);
        int antid = jobj.optInt("antid");//Ant ID
        short readPower = (short)jobj.optInt("readPower");// read power
        short writePower = (short)jobj.optInt("writePower");// write power
        boolean antEnable = jobj.optBoolean("enable"); // true - enable the antenna, false - disable the antenna
}
```

Association Classes

UHFReader

It provides string-processing method and different types of enumeration.

Class: ***com.nlscan.uhf.lib.UHFReader***

Universal Method:

Return Value	Method
static String	***bytes_Hexstr(byte[] bArray)*** To convert bytes to hexadecimal strings Parameter: bArray (byte array) Return Value: hexadecimal strings e.g.(0x): 0A02
static byte[]	***Str2Hex(String hexStr)*** To convert hexadecimal strings to binary. This conversion only supports hexadecimal string length within 600 characters. Parameter: hexStr hexadecimal strings (The length of string should be a multiple of two, as two characters convert to one byte and each character respectively represents the first 4 bits and the last 4 bits in one byte. e.g.: (0x):0A02) Return value: byte array
static String	***Hex2Str(byte[] buf)*** To convert bytes to hexadecimal strings Parameter: buf (byte buffer) Return value: hexadecimal strings, e.g.: (0x):0A02

Enumeration of Interfaces Calling Returned State:

Enumeration：UHFReader.READER_STATE	
Member	Description
OK_ERR	Operate successfully.
IO_ERR	Network or serial connection failed. Power off reader and reinitialize it.
INTERNAL_DEV_ERR	Internal device issue. Power off reader and reinitialize it.
CMD_FAILED_ERR	Operation failed.

CMD_NO_TAG_ERR	No tag detected.
OP_NOT_SUPPORTED	Operation is not supported.
INVALID_PARA	Invalid parameters.
INVALID_READER_HANDLE	Invalid reader parameters.
HARDWARE_ALERT_ERR_BY_NO_ANTENNAS	No antenna detected.
HARDWARE_ALERT_ERR_BY_HIGH_TEMPERATURE	Reader temperature is too high.
HARDWARE_ALERT_ERR_BY_UNKNOWN_ERR	Unknown error.
OP_EXECING	Operation is currently executing.
UNKNOWN_READER_TYPE	Unknown reader type.
OP_INVALID	Invalid operation.

Enumeration of Tag Memory Banks:

Enumeration: UHFReader.BANK_TYPE	
Member	Description
RESERVED	Reserved memory bank (stores access password and kill password).
EPC	EPC memory bank (stores EPC code)
TID	TID memory bank (tag ID). It is globally unique serial number.
USER	USER memory bank. It is extendable and Different brand tags have different memory capacities. And many tags do not have USER memory bank.

Enumeration of Tag Lock Area:

Enumeration: UHFReader.Lock_Obj	
Member	Description
LOCK_OBJECT_KILL_PASSWORD	Lock kill password area
LOCK_OBJECT_ACCESS_PASSWD	Lock access password area
LOCK_OBJECT_BANK1	Lock EPC memory bank
LOCK_OBJECT_BANK2	Lock TID memory bank
LOCK_OBJECT_BANK3	Lock USER memory bank

Enumeration of Lock Types:

Enumeration: HFReader.Lock_Obj	
Member	Description
UNLOCK	Unlock
LOCK	Temporary lock (Available to unlock)
PERM_LOCK	Permanent lock (NOT available to unlock)

TagInfo

Class of tag info includes data read from tag like EPC ID, RSS (signal strength) and etc.

Class: com.nlscan.uhf.lib.TagInfo		
Member	Type	Description
AntennaID	byte	Which antenna read the tag?
Frequency	int	Which frequency read the tag?
TimeStamp	int	Time stamp (unit ms) that identified when tag was read (Relative to the moment that command was issued.).
EmbededData	byte[]	Additional data
EmbededDatalen	int	Additional data's valid length
EpcId	byte[]	EPC code
EpcId	int	EPC 's valid length
PC	byte[]	PC
CRC	byte[]	CRC
ReadCnt	int	Number of times that tag has been read.
RSSI	int	Received tag signal strength

protocol	SL_TagProtocol	Tag Protocol (Enumeration)
		TagInfo.SL_TagProtocol{
		SL_TAG_PROTOCOL_NONE(0),
		SL_TAG_PROTOCOL_ISO180006B(3),
		SL_TAG_PROTOCOL_GEN2(5),
		SL_TAG_PROTOCOL_ISO180006B_UCODE(6),
		SL_TAG_PROTOCOL_IPX64(7),
		SL_TAG_PROTOCOL_IPX256(8);
		}

UHFModuleInfo

RFID Module Information

Class: com.nlscan.uhf.lib.UHFModuleInfo		
Member	Type	Description
moduleType	String	Module Information

.

Revision History

Version	***Description***	***Date***
V1.0.0	Initial release	June 12, 2023
V1.0.1	Added support for URM500 and URF520; Added antId parameter description and others	Oct.25,2024
V2.1	1. Code Highlighting	May.8,2025

	2. Delete keywords for UR90 3. Add a detailed explanation of label storage partitioning	
V2.2	1.add “Inventory Trigger” 2.add “Inventory Notification”	Nov.12,2025
V2.3	1.TagInfo field ***EmbededDatalen, Epclen***	Apl.23,2026

SCANNING MADE SIMPLE

Newland AIDC EMEA
+31 (0) 345 87 00 33
info@newland-id.com

Rolweg 25
4104 AV Culemborg
The Netherlands