Newland AIDC
SCANNING MADE SIMPLE

RFID MODULE DEVELOPMENT

For generic Android device

Version: V2.0.0

Quick Start 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

0、About This Manual 2

1、Importing the SDK 2

2、Get start: Connecting the RFID Module 3

Step 1: Register Connection Status Listener 4

Step 2: Connect to RFID Module 4

3、Inventory & Retrieve Tags 5

Step 1: Register a listener to obtain tag data. 6

Step 2: Start Inventory 6

Step 3: Stop the inventory. 7

4、Set the antenna power. 7

5、Output TID (Tag Identifier). 8

6、Output RSSI (Received Signal Strength Indicator). 9

7、End: Disconnect the RFID module. 10

F、Introduction to Related Classes 10

UHFReader 10

Detail Status： 11

Tag bank types： 11

Tag locking area： 11

Tag locking types.： 12

TagInfo 12

Revision History 13

0、*About This Manual*

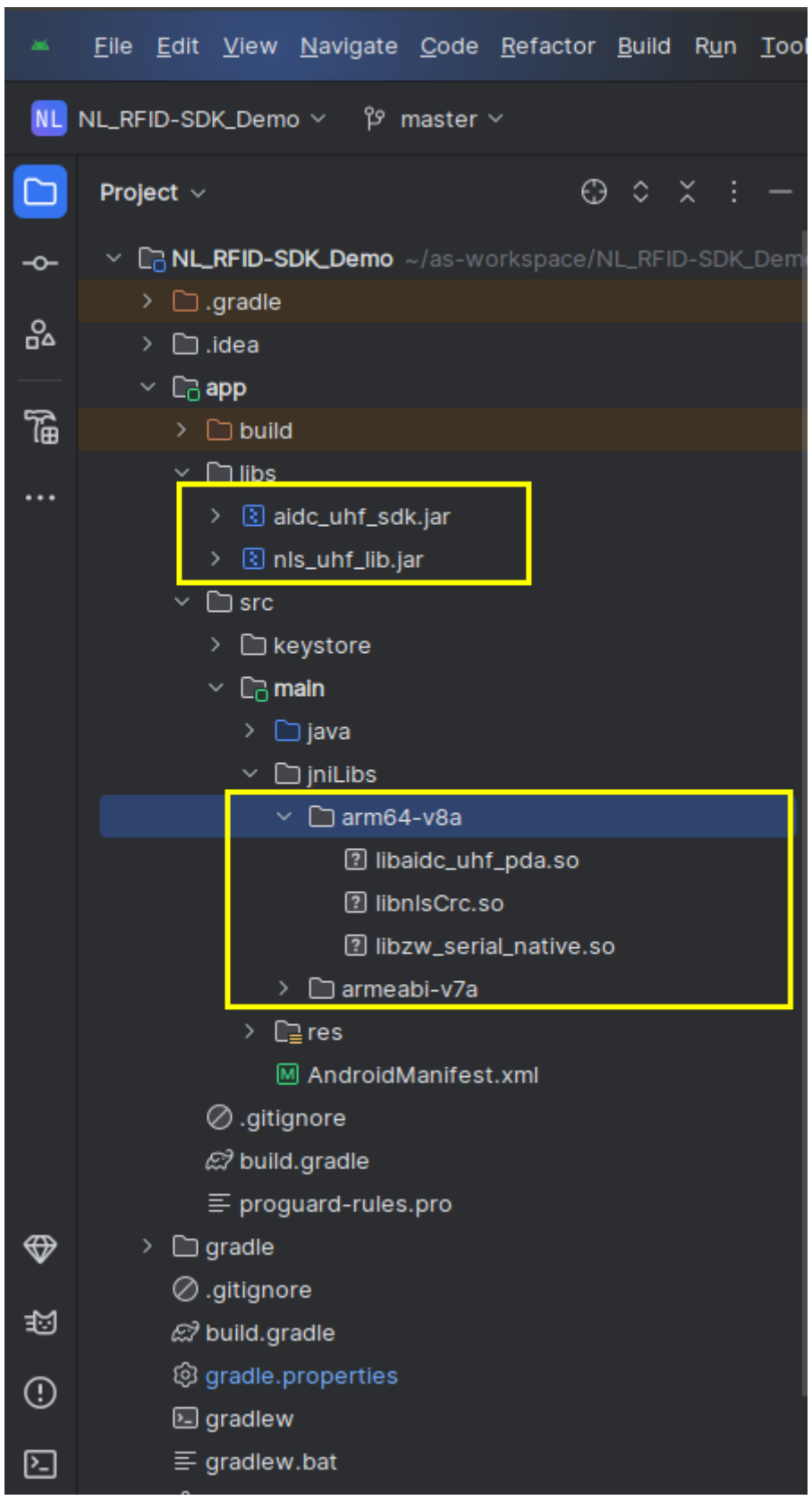
The document provides a development guide for the RFID module functionality on the Android platform. It helps developers quickly get started with RFID function development through logical sequencing and common function settings.

1、Importing the SDK

The SDK is based on Java language API calls. Developers need to import the SDK's jar/so library file into the Android application's build environment.

For example, in ***Android Studio***, place the jar file into the "project/app/libs" directory and load it into the build

dependencies:



app/build.gradle dependencies：

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])//, includes: ['*.aar']
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
    implementation 'com.jakewharton:butterknife:8.5.1'
    //annotationProcessor 'com.jakewharton:butterknife-compiler:8.5.1'
    implementation 'com.android.support:multidex:1.0.3'
}
```

2、Get start: Connecting the RFID Module

Basic : Connect the RFID Module before performing any other operations, the RFID module must first be connected.

- ***Step 1:*** Register a connection status listener to monitor the connection process and determine if it is successful.
- ***Step 2:*** Connect the module.

Step 1: Register Connection Status Listener

Register Connection Status BroadcastReceiver

Since the connection is asynchronous, you need to first register a listener to monitor the connection status. This listener will listen for broadcast messages to track the connection status.

```
/*------Register status receiver-----
* Status type：
*    Connected
*    Disconnected
*    Connect failed
*/
//Register broadcast receiver
IntentFilter iFilter = new IntentFilter(UHFManager.ACTOIN_UHF_STATE_CHANGE);
mContext.registerReceiver(mUHFStateReceiver, iFilter);
//Status's Receiver
private BroadcastReceiver mUHFStateReceiver = new BroadcastReceiver() {

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

                if (intent == null)
                        return;
                if (UHFManager.ACTOIN_UHF_STATE_CHANGE.equals(intent.getAction())) {
                        int uhf_state = intent.getIntExtra(UHFManager.EXTRA_UHF_STATE, -1);

                        switch (uhf_state) {

                                case UHFManager.UHF_STATE_CONNECTED:
                                        //Connect success.
                                        break;
                                case UHFManager.UHF_STATE_DISCONNECTED:
                                        //Disconnected.
                                        break;
                                case UHFManager. UHF_STATE_CONNECT_FAIL:
                                        //Connect fail.
                                        break;
                        }
                }
        }
};
```

Step 2: Connect to RFID Module

The communication methods vary for different models of modules, and they are divided into three types: ***Serial***, ***Bluetooth(BLE)***, and ***Network*** communication.

- ***Serial***：Connect via the serial port node path (the path varies depending on the PDA model).

  Serial path(e.g.)：***/dev/ttyS0***

- ***Bluetooth(BLE)***：Connect via the Bluetooth module's MAC address.（eg: a1:b2:c3:d4:e5）。

（Before connecting, need to start BLE　scanning. Once address is found, you can proceed with the connection.）

- ***Network***：Module and PDA device are connected to the same router network, you can connect using the "IP: Port" address.（eg：　192.168.0.101:8058）

Connect to RFID Module

Connect to the RFID module asynchronously, and the connection status can be obtained through Step1.

//Connect by serial：

```
//e.g.:【/dev/ttyS0】
//e.g.：  【/dev/ttyS2】
String mDevPathOrMac = "/dev/ttyS0"; // the path varies depending on the Android device
//Communication type：
int mPluginType = UHFManager.PLUGIN_TYPE_SERIEL;
//RFID model name, be empty.
String mDeviceModel = "";
Context context = getApplicationContext();
UHFManager mUHFMgr = UHFManager.getInstance(context);
//Do connect asynchronously
UHFReader.READER_STATE er = mUHFMgr.asyncConnect(mDevPathOrMac, mPluginType, mDeviceModel, null);
```

//Connect by BLE：

```
String mDevPathOrMac = "a1:b2:c3:d4:e5"; //BT MAC
//Communication type：
//【 UHFManager.PLUGIN_TYPE_BLE;】
int mPluginType = UHFManager.PLUGIN_TYPE_BLE;
//RFID model name, be empty
String mDeviceModel = "";
Context context = getApplicationContext();
UHFManager mUHFMgr = UHFManager.getInstance(context);
//Do connect asynchronously
UHFReader.READER_STATE er = mUHFMgr.asyncConnect(mDevPathOrMac, mPluginType, mDeviceModel, null);
```

//Connect by network：

```
String mDevPathOrMac = "192.168.0.101:8058"; //IP and port
//Communication type：
//【UHFManager.PLUGIN_TYPE_NETWORK;】
int mPluginType = UHFManager.PLUGIN_TYPE_NETWORK;
//RFID model name, be empty
String mDeviceModel = "";
Context context = getApplicationContext();
UHFManager mUHFMgr = UHFManager.getInstance(context);
//Do connect asynchronously
UHFReader.READER_STATE er = mUHFMgr.asyncConnect(mDevPathOrMac, mPluginType, mDeviceModel, null);
```

3、Inventory & Retrieve Tags

After successfully connecting the module, you can perform tag ***inventory & retrieval*** to obtain tag data.

Step 1: Register a listener to obtain tag data.

After the inventory starts, tag data is obtained through broadcast listening.

- Broadcast Action：nlscan.intent.action.uhf.ACTION_RESULT_EX
- Broadcast Extra： tag_info

（Data type：Parcelable[],Parcelable object change to TagInfo，For details, please refer to the ***TagInfo*** object section.）

Register a listener for receiving tag data.

1. Obtain tag data through broadcast listening.

2. The returned data is a Parcelable array (***Parcelable[]***).

```
//Register broadcast receiver
final String ACTION_UHF_RESULT_SEND = "nlscan.intent.action.uhf.ACTION_RESULT_EX";
IntentFilter iFilter = new IntentFilter(ACTION_UHF_RESULT_SEND);
mContext.registerReceiver(mUhfBR, iFilter);

//The receiver
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;
                final String EXTRA_TAG_INFO = "tag_info";
                Parcelable[] tagInfos = intent.getParcelableArrayExtra(EXTRA_TAG_INFO);
                for (int i = 0; tags != null && i < tags.length; i++) {
                    //Change to TagInfo
                    TagInfo tag = (TagInfo) tags[i];
                    //Epc byte[]
                     byte[] epcIdBytes = Arrays.copyOfRange(tagInfo.EpcId, 0, tagInfo.Epclen);
                    //Epc hex string（eg: FABCDE1234560 ...）;
                     String epcIdHex = UHFReader.bytes_Hexstr(epcIdBytes);
                }
        }
};
```

Step 2: Start Inventory

Calling the API：The application can start the inventory by calling the API.

1、Start by Calling the API

The application can start the inventory by calling the API.

```
Context context = getApplicationContext();
UHFManager mUHFMgr = UHFManager.getInstance(context);
//Start inventory
UHFReader.READER_STATE er = mUHFMgr.startTagInventory();
if(er == UHFReader.READER_STATE.OK_ERR)
    //Success...
else
    //Fail...
```

Step 3: Stop the inventory.

- By releasing the scan button：The inventory will stop as soon as the Key is released.
- By calling Api.

（Note: After stopping the inventory, some tag data may still be in the process of transmission. You need to wait for a short delay of 100ms to allow the data transfer to complete.）

Stop the inventory by calling the API .

Stop the inventory by calling the API .

```
Context context = getApplicationContext();
UHFManager mUHFMgr = UHFManager.getInstance(context);
//Stop inventory
UHFReader.READER_STATE er = mUHFMgr.stopTagInventory();
if(er == UHFReader.READER_STATE.OK_ERR)
    //Success...
else
    //Fail...
```

4、Set the antenna power.

The higher the antenna power, the more tags can be inventoried, and the faster the performance.

- The parameter is a string in the ***org.json.JSONArray*** format.
- The power data for each antenna is represented as an ***org.json.JSONObject***.
- Antenna count: Single antenna or multiple antennas (default: single antenna, 1).

设置天线功率

1、org.json.JSONArray format

antid：Antenna ID 1 ~ 4 (The number of antennas supported by the RFID module, e.g., the URF520 module supports 4 antennas).

enbale：Enable or disable this antenna.

readPower：Antenna's tag reading power (e.g., inventory).

writePower：Antenna's tag writing power (e.g, writing tag data).

```
[
        {
                "antid": 1,
```

```
                "enable": true,
                "readPower": 26,
                "writePower": 27
        },
        {
                "antid": 2,
                "enable": true,
                "readPower": 26,
                "writePower": 27
        },{
                "antid": 3,
                "enable": true,
                "readPower": 26,
                "writePower": 27
        },
        {
                "antid": 4,
                "enable": true,
                "readPower": 26,
                "writePower": 27
        }
]
```

```
// Set the read and write power for Antenna 1.
try {
        final String RF_ANTPOWER = "RF_ANTPOWER";
        final String PARAM_RF_ANTPOWER = "PARAM_RF_ANTPOWER";
        JSONArray jsItemArray = new JSONArray();
        JSONObject jsItem = new JSONObject();
        jsItem.put("antid", 1);// Antenna 1
        jsItem.put("enable", true);//Enable
        jsItem.put("readPower", 30);//Power level：30db
        jsItem.put("writePower", 30);// Power level：30db
        jsItemArray.put(jsItem);

        Context context = getApplicationContext();
        UHFManager mUHFMgr = UHFManager.getInstance(context);
        //Do setting
        UHFReader.READER_STATE er = mUHFMgr.setParam(RF_ANTPOWER, PARAM_RF_ANTPOWER, jsItemArray.toString());
        if(er == UHFReader.READER_STATE.OK_ERR)
            //Success
        else
            //Fail
} catch (Exception e) {
        e.printStackTrace();
}
```

5、Output TID (Tag Identifier).

TID is the Tag Identifier, which is the unique identification ID of the tag.

By default, TID data is not output during inventory. You need to configure the TID partition to be output in an appended manner with the tag object's TagInfo ***EmbeddedData*** field. For details, refer to the ***TagInfo*** object section.

TID 数据输出
1、During inventory, TID data is stored in the ***EmbeddedData*** field of the tag object ***TagInfo*** (by default, it is not output). 2、Enable TID data output settings. Variant：org.json.JSONObject's string bank：2 = TID Bank

startaddr：0 = Start address

bytecnt：12 = Byte count will reading

```
{
    "bank": 2,
    "startaddr": 0,
    "bytecnt": 12
}
```

```
Context context = getApplicationContext();
UHFManager mUHFMgr = UHFManager.getInstance(context);
int bank = UHFReader.BANK_TYPE.TID.value();//TID Bank
int startaddr = 0;// Start address
int bytecnt = 12;// Byte count
try {
        JSONObject jsItem = new JSONObject();
        jsItem.put("bank", bank);
        jsItem.put("startaddr", startaddr);
        jsItem.put("bytecnt", bytecnt);
        String sValue = jsItem.toString();

        final String TAG_EMBEDEDDATA = "TAG_EMBEDEDDATA";
        final String PARAM_TAG_EMBEDEDDATA = "PARAM_TAG_EMBEDEDDATA";

        //Do setting
        UHFReader.READER_STATE er = mUHFMgr.setParam(TAG_EMBEDEDDATA, PARAM_TAG_EMBEDEDDATA, sValue);
        if(er == UHFReader.READER_STATE.OK_ERR)
            //Success
        else
            //Fail

} catch (Exception e) {
}
```

6、Output RSSI (Received Signal Strength Indicator).

RSSI (Received Signal Strength Indicator) is the signal strength of the tag's recognition, represented in negative values. The higher the value, the better the signal. For example, ***-40*** has a stronger signal than ***-50***.

（During inventory, RSSI is stored in the ***RSSI*** field of the ***TagInfo*** object. For details on the parameters, please refer to the ***TagInfo*** object section.）

（By default, RSSI data may not be output during inventory. You need to enable it to be output.）

To output RSSI during inventory

Call the API to enable RSSI output during inventory.

```
final String INV_FIELD_RSSI = "INV_FIELD_RSSI";
final String PARAM_INV_FIELD_RSSI = "PARAM_INV_FIELD_RSSI";
Context context = getApplicationContext();
UHFManager mUHFMgr = UHFManager.getInstance(context);
//Enable output RSSI
UHFReader.READER_STATE er = mUHFMgr.setParam(INV_FIELD_RSSI, PARAM_INV_FIELD_RSSI, "1");//1:Enable, 0:Disable
if(er == UHFReader.READER_STATE.OK_ERR)
    //Success...
else
    //Fail...
```

7、End: Disconnect the RFID module.

When it is confirmed that the RFID function will not be used for a long time (e.g., when exiting the application), the connection to the RFID module should be disconnected.

（A prolonged connection will increase power consumption and accelerate hardware aging of the module.）

Disconnect the RFID module connection.
Call the API to disconnect. Upon disconnection, a status broadcast indicating the module has been disconnected will be received. For status reception, refer to the "***Connecting to the RFID Module***" section on status listening.
Context context = getApplicationContext(); UHFManager mUHFMgr = UHFManager.getInstance(context); *//Disconnect* UHFReader.READER_STATE er = mUHFMgr.disconnect(); ***if***(er == UHFReader.READER_STATE.OK_ERR) *//Success...* ***else*** *//Fail...*

F、Introduction to Related Classes

UHFReader

Provides string processing methods and various enumeration types.

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

Methods：

返回值	方法
static String	***bytes_Hexstr(byte[] bArray)*** Convert a byte array to a hexadecimal string. Parameter: ***bArray*** (Byte Array) Return value: Hexadecimal string, e.g., ***(0x): 0A02***.
static byte[]	***Str2Hex(String hexStr)*** Convert a hexadecimal string to binary byte data. Currently, this function supports conversion for hexadecimal strings with a length of no more than 600 characters. ***Parameter:*** ***hexStr*** - Hexadecimal string (the length must be an even number because two hexadecimal characters are converted into one binary byte, representing the high and low four bits of a byte). Example: ***(0x): 0A02*** ***Return value:*** byte[].

static String	***Hex2Str(byte[] buf)*** Convert byte data into a hexadecimal string. ***Parameter:*** ***buf*** - Byte data. ***Return value:*** Hexadecimal string, e.g., ***(0x): 0A02***.

Detail Status：

Enum*：*UHFReader.READER_STATE	
Members	Description
OK_ERR	Success
IO_ERR	It is disconnected, need to reConnect.
INTERNAL_DEV_ERR	f there is an issue with the internal device, the reader must be shut down and re-initialized.
CMD_FAILED_ERR	Fail
CMD_NO_TAG_ERR	No tag found
OP_NOT_SUPPORTED	Unsupported operation.
INVALID_PARA	Invalid parameter
INVALID_READER_HANDLE	Invalid reader parameter
HARDWARE_ALERT_ERR_BY_NO_ANTENNAS	No antena faound
HARDWARE_ALERT_ERR_BY_HIGH_TEMPERATURE	The reader's temperature is too high.
HARDWARE_ALERT_ERR_BY_UNKNOWN_ERR	Uknwon
OP_EXECING	Executing
UNKNOWN_READER_TYPE	Unknown reader
OP_INVALID	Invalid operation

Tag bank types：

枚举：***UHFReader.BANK_TYPE***	
Members	Description
RESERVED	Reserved area (stores the access password and destroy password).
EPC	EPC Area (Electronic Product Code Area).
TID	TID Area (Tag ID), contains a globally unique serial number.
USER	USER Area, the capacity is variable and differs between tag brands. Many tags also do not have this partition.

Tag locking area：

枚举：***UHFReader.Lock_Obj***	
Members	Description
LOCK_OBJECT_KILL_PASSWORD	Lock the destroy password area.
LOCK_OBJECT_ACCESS_PASSWD	Lock the access password area.
LOCK_OBJECT_BANK1	Lock the EPC area.
LOCK_OBJECT_BANK2	Lock the TID area.

LOCK_OBJECT_BANK3	Lock the USER area.

Tag locking types.：

枚举：***UHFReader.Lock_Obj***	
Members	Description
UNLOCK	Unlock
LOCK	Temporary lock (can be unlocked).
PERM_LOCK	Permanent lock (cannot be unlocked).

TagInfo

The encapsulation class for tag data obtained during inventory, containing various data of the read tags, such as EPC ID, RSSI (signal strength), etc.

Class*：*com.nlscan.uhf.lib.TagInfo		
Members	Data type	Description
AntennaID	byte	The antenna that reads the tag.
Frequency	int	The frequency from which the tag is read.
TimeStamp	int	The timestamp when the tag is read, in milliseconds (relative to the time the command was issued).
EmbededData	byte[]	Embedded data (additional data).
EpcId	byte[]	EPC
PC	byte[]	PC
CRC	byte[]	CRC
ReadCnt	int	The number of times the tag has been read.
RSSI	int	The signal strength of the received tag.

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);
		}

Revision History

Version	Modifications	Date
V2.0	First version	Apr 8, 2025.

SCANNING MADE SIMPLE

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

Rolweg 25
4104 AV Culemborg
The Netherlands