

Programming Guide

RFID SDK

unitech electronics Documentation
Issue1, revision 4
Jun 2026

Revision History

Release	Revision	Date	Changes
2	0	2025-08-05	Release for unitechRFID V2.0.0.0
2	1	2025-09-02	Change DMService version for EA530 to V1.2.55.5
2	2	2025-12-10	Change DMService version for EA530 to V1.2.56.1
2	3	2026-04-07	Support RG768P
2	4	2026-06-08	Support HT730P Add Chapter 3 and Chapter 4.

Table of Contents

1.	PREFACE	1
1.1	PREREQUISITES.....	2
2.	UNIECHRFID API	3
3.	MIGRATING FROM RG768 TO RG768P	5
3.1	UPDATES TO IREADEREVENTLISTENER.....	6
3.2	UNSUPPORTED APIS.....	6
3.3	NEW FEATURES AVAILABLE IN RG768P	6
4.	DISCOVERING DEVICE AND SDK CAPABILITIES	8
4.1	DISCOVERING READER-LINE CAPABILITIES.....	8
4.1.1	DYNAMIC DISCOVERY.....	8
4.1.2	STATIC DISCOVERING.....	9
4.2	DISCOVERING BARCODE ENGINE CAPABILITIES	9
4.3	DISCOVERING UHF RFID MODULE CAPABILITIES.....	9
4.3.1	IDENTIFYING THE MODULE TYPE	10
4.3.2	QUERYING THE FULL API SUPPORT LIST	10

1. Preface

This document describes the functionalities of the UnitechRFID. The SDK is delivered through the JAVA-based library “unitechRFID.aar” for user with the supported portable Android RFID readers from unitech. Those supported RFID readers use the modules ST25RU3993, RM100, UTE1717 and UTE7180.

The figure below shows the structure of SDK. The *Binary* folder includes the library and one sample application, and the *Source* folder contains the source codes of sample application, and the doc folder contains the javaDoc of the library “unitechRFID.aar”.

Folders & Files	Description
/UnitechRFID	The SDK package
/UnitechRFID/Binary/	The Binary folder contains the RFID Library (AAR) and the dependency apps & sample app.
/UnitechRFID/doc/	The doc folder contains the JAVADOC of the unitechRFID.aar library file. Please load index.html into your browsers to view
/UnitechRFID/Binary/unitechRFID_vx.x.x.x.aar	The unitechRFID.aar library file. Include this library in your Android Studio project to access the API functions.
/UnitechRFID/Binary/DMSERVICE_1.2.8_HT730.apk	The dependency app DMSERVICE required for HT730. Please make sure this APK is installed on the host device before using the unitechRFID.aar
/UnitechRFID/Binary/DMSERVICE_1.2.32_PA768.apk	The dependency app DMSERVICE required for PA768. Please make sure this APK is installed on the host device before using the unitechRFID.aar
/UnitechRFID/Binary/DMSERVICE_1.2.54.3_PA768e.apk	The dependency app DMSERVICE required for PA768e. Please make sure this APK is installed on the host device before using the

	unitechRFID.aar
/UnitechRFID/Binary/DMSERVICE_1.2.56.1_EA530.apk	The dependency app DMSERVICE required for EA530. Please make sure this APK is installed on the host device before using the unitechRFID.aar
/UnitechRFID/Binary/DMSERVICE_1.2.59.1_HT730P.apk	The dependency app DMSERVICE required for HT730P. Please make sure this APK is installed on the host device before using the unitechRFID.aar
/UnitechRFID/Binary/unitechRFIDSample_x.x.x.x.apk /UnitechRFID/Binary/unitechrfidsamplemaui x.x.x.x.apk	The sample application shows the use of the unitechRFID.aar, The source code of this application is also included in this package for studying.
/UnitechRFID/Source/	The source code of the unitechRFIDSample application.

Note: The library “unitechRFID.aar” needs

- **HT730: DMSERVICE V1.2.8_HT730 or later version**
 - **PA768: DMSERVICE V1.2.32_PA768 or later version**
 - **PA768e: DMSERVICE V1.2.54.3_PA768e or later version**
 - **EA530: DMSERVICE V1.2.56.1_EA530 or later version**
 - **HT730P: DMSERVICE V1.2.59.1_HT730P or later version**
- to support. You could get it in the Binary folder.**

Note: The BaseReader(HT730Reader, RG768Reader, RP902Reader, PA768eReader, RP300Reader, EA530Reader, HT730pReader) should be initialized in non-main thread.

Note: The application needs API version 28 or higher.

1.1 Prerequisites

The SDK supports the following RFID Reader peripherals:

- RP902 with an Android 9 to 14.
- HT730 with Android 10.

- RG768 with Android 12 to 14.
- PA768e with Android 14.
- EA530 with Android 15.
- RG768P with Android 12 to 14.
- HT730P with Android 14.

Note about the Dependency App “DMService”

The DMService application is one of the unitech helper applications that provide the functions of the UnitechSDK. The “unitechRFID.aar” UnitechRFID uses the UnitechSDK for some of its functionalities such as Power Settings. It is recommended that your application can take care of the installation of this DMService APK if the host device does not already have it installed, or if the minimum version required are not correct.

The UnitechRFID requires

- HT730: DMService V1.2.8_HT730 or later (inclusive)
- PA768: DMService V1.2.32_PA768 or later (inclusive)
- PA768e: DMService V1.2.54.3_PA768e or later (inclusive)
- EA530: DMService V1.2.56.1_EA530 or later (inclusive)
- HT730P: DMService V1.2.59.1_HT730P or later (inclusive)

to be installed on the supported host Android devices. The DMService APK files for the supported models can be found from the Binary folder inside the SDK package.

Note about Multithreaded Applications

The BaseReader class should be initialized in non-main thread.

2. uniechRFID API

The TransportBluetooth used to connect to RFID module by Bluetooth. Below is the sample code, and you can also find the details in the SampleFragment.java in the Source folder.

RP902:

```
TransportBluetooth tb = new TrandportBluetooth(DeviceType.RP902, “RP902”,  
“11:22:33:44:55:66”);
```

```
BaseReader baseReader = new RP902Reader(tb);
```

```
baseReader.addListener(this);  
baseReader.setRfidEventListener(this);  
baseReader.connect();
```

RP300:

```
TransportBluetooth tb = new TrandportBluetooth(DeviceType.RP300, "RP300",  
"11:22:33:44:55:66");  
BaseReader baseReader = new RP300Reader(tb, getApplicationContext());  
baseReader.addListener(this);  
baseReader.setRfidEventListener(this);  
baseReader.setBarcodeEventListener(this);  
baseReader.connect();
```

Connect to RFID module with UART. Below is the sample code, and you can also find the details in the SampleFragment.java in the Source folder.

HT730:

```
BaseReader baseReader = new HT730Reader(getApplicationContext());  
baseReader.addListener(this);  
baseReader.setRfidEventListener(this);  
baseReader.connect();
```

RG768 & RG768P:

```
BaseReader baseReader = new RG768Reader(getApplicationContext());  
baseReader.addListener(this);  
baseReader.setRfidEventListener(this);  
baseReader.connect();
```

PA768e:

```
BaseReader baseReader = new PA768eReader(getApplicationContext());  
baseReader.addListener(this);  
baseReader.setRfidEventListener(this);  
baseReader.connect();
```

EA530:

```
BaseReader baseReader = new EA530Reader(getApplicationContext());
```

```
baseReader.addListener(this);  
baseReader.setRfidEventListener(this);  
baseReader.connect();
```

HT730P:

```
BaseReader baseReader = new HT730pReader(getApplicationContext());  
baseReader.addListener(this);  
baseReader.setRfidEventListener(this);  
baseReader.connect();
```

Connect to RFID module with USB. Below is the sample code, and you can also find the details in the SampleFragment.java in the Source folder.

RP300:

```
// Detect Reader  
DetectReader detectReader = new DetectReader(getApplicationContext(), this);  
detectReader.start();
```

```
public void onUsbAttached(UsbDevice device) {  
    TransportUsb transportUsb = new TransportUsb(DeviceType.RP300, device);  
    BaseReader baseReader = new RP300Reader(transportUsb, getApplicationContext());  
    baseReader.addListener(this);  
    baseReader.setRfidEventListener(this);  
    baseReader.setBarcodeEventListener(this);  
    baseReader.connect();  
}
```

Note about Multithreaded Applications

The BaseReader class should be initialized in non-main thread.

3. Migrating from RG768 to RG768P

If you are migrating from RG768 to RG768P, the following modifications are required during application develop:

3.1 Updates to IReaderEventListener

For implementations utilizing IReaderEventListener, the following functions must be added:

- onAntennaStatus
- onLBTStatus
- onFirmwareUpdateProgress
- onBatchDataEvent

3.2 Unsupported APIs

The following APIs do not corresponding counterparts in RG768P and must be modified(getRFMode/setRFMode) or removed:

- RG768Reader.getRfidUhf().getModuleProfile()
- RG768Reader.getRfidUhf().setModuleProfile(int id)

3.3 New Features Available in RG768P

The following APIs represent features specifically available in the RG768P:

RG768Reader APIs

- public String getSerialNumber()
- public void factoryReset()
- public void reboot()
- public double getAmbientTemperature()
- public double getPaTemperature()
- public void setEventNotification(EventNotification eventNotification)
- public InventoryDataFormat[] getInventoryDataFormat()
- public void setInventoryDataFormat(InventoryDataFormat[] inventoryDataFormat)
- public int getAmbientTemperatureProtection()
- public void setAmbientTemperatureProtection(int temperature)
- public int getPaTemperatureProtection()
- public void setPaTemperatureProtection(int temperature)

- `public TemperatureProtectionMode getTemperatureProtectionMode()`
- `public void setTemperatureProtectionMode(TemperatureProtectionMode mode)`

RG768Reader.getRfidUhf() APIs

- `public ResultCode userInventory6c(BankType bank, int offset, int length, String password)`
- `public ResultCode writeBlockMemory6c(BankType bank, int offset, int length, String data, String password)`
- `public int getDwellTime()`
- `public void setDwellTime(int time)`
- `public RFMode getRFMode()`
- `public void setRFMode(RFMode mode)`
- `public boolean getTagFocus()`
- `public void setTagFocus(boolean enable)`
- `public long[] getFixedFreqParams()`
- `public void setFixedFreqParams(long[] frequencies)`
- `public PowerSavingMode getPowerSavingMode()`
- `public void setPowerSavingMode(PowerSavingMode mode)`
- `public int getLbtCfgSetting()`
- `public void setLbtCfgSetting(int value)`
- `public AntennaSwitchingMode getAntennaSwitchingMode()`
- `public void setAntennaSwitchingMode(AntennaSwitchingMode mode)`
- `public FreqSwitchingMode getFreqSwitchingMode()`
- `public void setFreqSwitchingMode(FreqSwitchingMode mode)`
- `public CfgStorageMode getCfgStorageMode()`
- `public void setCfgStorageMode(CfgStorageMode mode)`
- `public void cwOnOff(CWParms param, boolean enable)`
- `public void txRandomData(TxRandomDataParam param)`
- `public ResultCode blockPermaLock(BlockPermaLockParam param)`
- `public void authenticate(AuthenticateParam param)`
- `public boolean getPhaseSetting()`
- `public void setPhaseSetting(boolean enable)`
- `public int getPowerSaveTime()`
- `public void setPowerSaveTime(int timeout)`
- `public Gen2XScanSetting getGen2XScanSetting()`
- `public void setGen2XScanSetting(Gen2XScanSetting setting)`
- `public Gen2XScanIdSetting getGen2XScanIdSetting()`

- `public void setGen2XScanIdSetting(Gen2XScanIdSetting setting)`

Note: To view the latest APIs supported by the device, please refer to Chapter 4.

4. Discovering Device and SDK Capabilities

This chapter explains how developers can use the Unitech RFID SDK to query the capabilities and supported APIs of the currently connected device. Since Unitech offers various RFID readers with different UHF modules, the supported features may vary across different models. Developers should utilize the discovery mechanisms provided by the SDK to dynamically adapt their applications.

4.1 Discovering Reader-Line Capabilities

Reader-level capabilities typically include hardware settings such as the Beeper, vibrator, and Battery status monitoring.

4.1.1 Dynamic Discovery

Once a BaseReader instance is created and a connection is successfully established, you can use the following methods:

```
BaseReader reader = ...; // Obtain your connected Reader instance

// Retrieve the list of all supported parameter IDs
DeviceParamID[] supportedIds = reader.getSupportedParameterIds();

// Check if a specific feature is supported (e.g., Beeper)
boolean isBeeperSupported = false;
for (DevicePaamID id : supportedIds) {
    if (id == DeviceParamID.beeper) {
        isBeeperSupported = true;
        break;
    }
}
```

```
}  
}
```

4.1.2 Static Discovering

If the device model (DeviceType) is known, you can query its supported parameters even before establishing a connection:

```
// Query supported parameters based on the specific DeviceType  
DeviceParamID[] supportedIds = BaseReader.getSupportedParameterIds(DeviceType.RP902);
```

4.2 Discovering Barcode Engine Capabilities

If your device is equipped with a barcode scanning module, you can query supported symbologies or configuration items via the BaseEngine class.

```
BaseEngine engine = reader.getBaseEngine();  
if (engine != null) {  
    // Retrieve the list of supported parameter IDs for the barcode engine  
    EngineParamID[] engineParams = engine.getSupportedParameterIds();  
  
    // Retrieve more detailed parameter information (including version constraints)  
    EngineParam[] details = engine.getSupportedParameters();  
}
```

4.3 Discovering UHF RFID Module Capabilities

The availability of UHF features depends on the specific RFID module integrated into the device (e.g., RM100, ST25RU3993, UTE1717, etc.).

4.3.1 Identifying the Module Type

First, identify the type of the current UHF module using the `BaseUHF` or `BaseReader` instance:

```
BaseUHF uhf = reader.getRfidUhf();  
ModuleUHFTYPE type = uhf.getType(); // Returns type such as ModuleUHFTYPE.RM100
```

4.3.2 Querying the Full API Support List

Each UHF module implementation class provides a static method `getUhfSupportedIds()` that returns all `UhfParamId` supported by that specific module.

```
// Example: Checking supported APIs for the RM100 module  
UhfParamId[] supportedApis = UHFRM100.getUhfSupportedIds();
```

```
// Check if the "Tag Focus" feature is supported  
boolean isTagFocusSupported = false;  
for (UhfParamId id : supportedApis) {  
    if (id == UhfParamId.tagFocus) {  
        isTagFocusSupported = true;  
        break;  
    }  
}
```