IDTS-UWR6 Watch Reader

Android SDK — Developer & Integration Guide

Wearable UHF RFID reader · BLE (Nordic UART) · Android 6–15, targetSdk 34

Model: IDTS-UWR6 ("Wearable U6 BLE") · **SDK: handheld_v1.1.64.jar**

Document version 1.0 · Demo app: Identium UWR6 1.0.0 (Kotlin)

Every statement checked against the decompiled jar and the vendor demo app

1. Introduction

The **IDTS-UWR6** is Identium's wearable UHF RFID watch reader. A phone or tablet talks to it over **Bluetooth Low Energy**; the vendor SDK `handheld_v1.1.64.jar` (package `com.example.handheld.lib`) wraps that link. This guide explains how the SDK really behaves and how to use it safely, with a reference for every feature. The **Identium UWR6** demo app in this package implements all of it — its `rfid/RfidController.kt` is the reference integration.

The vendor's own API document lists method names only, and several entries don't match the jar (for example it documents `setPower(read, write)`; the jar has `setPower(int)`). Everything here was checked against the jar itself.

1.1 The SDK at a glance

Item	Detail
Main class	`Device` — BLE. A process-wide singleton: `Device.getInstance(context)`.
Unused class	`BluetoothChatService` — a Classic-Bluetooth (SPP) variant; the vendor demo never uses it.
Transport	Nordic UART: service 6e400001-…, write 6e400002-… (no response), notify 6e400003-…
Advertised name	**Wearable U6 BLE…** — the SDK picks the UUIDs from this name (sibling handhelds: Handheld_A2/A5/A6).
Protocol	Outer frame `1B cmd seq len BCC`; RFID commands are M100 frames `BB 00 cmd len … sum 7E` inside.
Listeners	Connection, UHF (tags), Battery, Handle (watch key), Barcode, Upgrade — called on the Bluetooth thread.
Build	Java 8 bytecode; works with AGP 8 / targetSdk 34 / minSdk 23 unchanged.

2. Rules for using the SDK safely

These follow directly from how the jar is written. Breaking them causes timeouts, wrong answers or a link that looks up but does nothing.

SDK fact	What to do
Commands block the caller up to 1 s on **one shared reply slot**; replies are **not matched** to requests.	Run every SDK call on **one** dedicated thread. Leave ~150 ms between commands (~300 ms after a timeout). Sanity-check replies and retry.
Listener callbacks run on the **Bluetooth (binder) thread** — the thread that delivers replies.	Never call the SDK from a listener (it would always time out). Convert the data and hand it to your own thread/flow.
The SDK **does not scan**.	Scan with `BluetoothLeScanner` and pass the MAC to `connect()`.
`connect()` returns true when the attempt *starts*; a failed attempt reports **nothing**.	Use your own timeout (15 s) for `onConnected()`.
`onConnected()` fires even when the UART characteristics weren't found.	Treat the link as up only after a command answers (`getVersion()`).
UUIDs are chosen from the name only if Android has it cached.	Before connecting, call the SDK's own `initUuid("Wearable U6 BLE")` (private; via reflection), or scan first.
Tag-access failures all come back as `CommandException(-1)` after ~1 s.	Show one message: no tag in range, wrong password, or locked memory.
Battery/Barcode listeners must be set, or parsing hits a NullPointerException.	Register all six listeners once (no-ops where unused).
`Device.getInstance()` keeps the first Context forever.	Pass `applicationContext`; one wrapper per process.
No permission checks inside the SDK.	Request BLUETOOTH_SCAN/CONNECT (Android 12+) or Location (6–11) first; use ACTION_REQUEST_ENABLE to turn Bluetooth on.

2.1 Connect sequence

```
val sdk = Executors.newSingleThreadExecutor { Thread(it, "uwr6-sdk") }.asCoroutineDispatcher()
suspend fun connect(mac: String, name: String?) = withContext(sdk) {
    device.disconnect()                     // drop any stale GATT first
    initUuid("Wearable U6 BLE"); name?.let { initUuid(it) }   // reflection on the private method
    connected = CompletableDeferred()
    check(device.connect(mac))              // blocks ~0.9 s
    withTimeout(15_000) { connected.await() }   // completed by onConnected()
    SystemClock.sleep(1_500)                // module warm-up (vendor demo does the same)
    device.version                          // handshake — only a reply proves the link
    device.stopInventory(); device.setSelectMode(1); device.closeBuzzer(0)
}
```

3. Feature reference

3.1 Reading tags

Continuous reading: `inventory()`, re-sent every second while reading (the vendor's cadence); stop with `stopInventory()`. One round: `inventoryOnce()`. Tags arrive on `OnUHFListener.onInventory(pc, epc, rssi)` — one read per call, RSSI in dBm.

Parameter / call	Type & units
inventory() / inventoryOnce() / stopInventory()	fire-and-forget (no reply awaited)
onInventory(pc: ByteArray, epc: ByteArray, rssi: Int)	EPC bytes; rssi e.g. -55
onInventoryStop()	fires immediately on stop — not when the watch confirms

```
device.inventory()                      // then every 1 s while reading: device.inventory()
device.setOnUHFListener(object : OnUHFListener {
    override fun onInventory(pc: ByteArray?, epc: ByteArray?, rssi: Int) {
        if (reading && epc != null) tags.trySend(epc.toHex() to rssi)   // hand off only
    }
    override fun onInventoryStop() = Unit
})
device.stopInventory()                  // close your own gate first; late tags still arrive
```

Note: Wait for ~300 ms without tags before the next command. Avoid `inventoryTimes()` (can stop itself) and `scanBarcode()` (competes for the reply slot). The SDK only resets its tag counter in `inventory()`; the demo resets it before single reads.

3.2 Watch scan key

`OnHandleListener.onDown(key)` / `onUp(key)`. The current firmware sends 131/132 for the scan key (older firmware: 0/1/2/8). The demo toggles reading or does one single read per press.

Parameter / call	Type & units
onDown(key: Int)	key code, from the Bluetooth thread

```
device.setOnHandleListener(object : OnHandleListener {
    override fun onDown(key: Int) { keys.tryEmit(key) }   // decide on your own thread
    override fun onUp(key: Int) = Unit
})
```

Note: Debounce (~250 ms) and ignore presses while a command is running.

3.3 RF power

One power for reading and writing, whole dBm.

Parameter / call	Type & units
getPower(): Int	dBm (sent as centi-dBm)
setPower(dBm: Int)	16–26 dBm (vendor demo range)

```
val p = device.power            // getPower()
device.setPower(24)             // also re-tunes the demodulator: two exchanges, up to ~2 s
```

Note: If setPower fails, read the power back — the first step may already have been applied.

3.4 Region, hopping and channel

Five regions; no India code. For India's 865–867 MHz band use Europe (3), hopping off, channel 0–9.

Parameter / call	Type & units
getRegion() / setRegion(code)	1 China 920–925 · 2 US 902–928 · 3 Europe 865–868 · 4 China 840–845 · 6 Korea
setFrequencyHopping(Boolean)	true = hop over the region's channels
setChannel(index)	frequency = base + index × step (EU 865.1 + 0.2·i MHz; US 902.25 + 0.5·i)

```
device.setRegion(3)
device.setFrequencyHopping(false)
device.setChannel(4)            // 865.9 MHz
```

Note: The watch can't report hopping, and `getChannel()` misreads channel 5 (taken for a battery report). The demo stores the plan per watch together with its region, and re-applies it on connect only while the watch is still on that region.

3.5 Gen2 (Query)

Read-modify-write the Query so the fields you don't change (DR, M, TR, Sel) are kept.

Parameter / call	Type & units
getQuery() / setQuery(q)	session 0–3 · target 0 = A, 1 = B · q 0–15 (fixed Q only)

```
val q = device.query
q.session = 1; q.target = 0; q.q = 4
device.query = q                // setQuery(q)
```

Note: `getTarget()` returns –1 for B (sign-extended) — treat any non-zero value as B.

3.6 Memory read / write

Banks 0 RESERVED, 1 EPC, 2 TID, 3 USER. Addresses and lengths in 16-bit words; passwords 4 bytes.

Parameter / call	Type & units
readTag(bank, startWord, words, pwd4): ByteArray	data only (PC/EPC header stripped)
writeTag(bank, startWord, words, pwd4, data)	keep to ≤ 8 words per call

```
val data = device.readTag(3, 0, 4, pwd4)        // USER, word 0, 4 words
device.writeTag(3, 0, 2, pwd4, byteArrayOf(0x12, 0x34, 0x56, 0x78))
```

Note: Kotlin can't use named arguments on Java methods — pass them positionally.

3.7 Addressing one tag (Select)

To act on a specific tag, send a Select for its EPC before read/write/lock/kill only (select mode 2), then turn it off.

Parameter / call	Type & units
setSelectParam(SelectParam)	target = Query session; action 0 (target A) or 4 (target B); bank 1; ptr 4 bytes
setSelectMode(mode)	0 always · 1 off · 2 before tag operations only

```
val sp = SelectParam().apply {
    setTarget(session); setAction(if (targetB) 4 else 0)
    setMemoryBank(1); setPtr(4); setMaskLength(epc.size); setMask(epc); setTruncate(false)
}
device.setSelectParam(sp); device.setSelectMode(2)
try { /* readTag / writeTag / lockTag / killTag */ } finally { device.setSelectMode(1) }
```

Note: The mask must be whole bytes. Read the Query just before building the Select, so it targets the current session.

3.8 Lock and kill

Lock areas and types use the SDK's constants — not the vendor demo's labels, which are swapped.

Parameter / call	Type & units
lockTag(pwd4, area, type)	area: 0 kill pwd · 1 access pwd · 2 EPC · 3 TID · 4 USER
	type: 0 unlock · 2 lock · 3 permalock (never 1)
killTag(killPwd4)	permanent; 00000000 can't kill

```
device.lockTag(pwd4, 2, 2)     // password-lock the EPC bank
device.killTag(kill4)          // never retry automatically
```

3.9 Watch: battery, buzzer, versions, save

Battery level and charging state; the watch's own buzzer; versions; persisting settings.

Parameter / call	Type & units
getBattery(): IntArray	[level 0–100, charging 0/1] — also pushed to OnBatteryListener while idle
openBuzzer() / closeBuzzer(0)	fire-and-forget; can't be read back — re-apply on connect
getVersion() / getRfidVersion() / getMcuVersion()	the last two start with a type byte (0x01 / 0x04)
saveParameters()	persists the module settings on the watch

```
val (level, charging) = device.battery.let { it[0] to (it[1] == 1) }
device.closeBuzzer(0)
device.saveParameters()
```

Note: Avoid `getdemodulator()` and `sendConnect()` (their replies are never delivered) and `upgrade()` without the vendor's firmware procedure.

4. The Identium UWR6 demo app

The Kotlin app (**Identium UWR6**) has the same UI and functions as the Identium MUBR01 app, built on this SDK:

- **Inventory** — start/stop from the app or the watch key; live de-duplicated list with reads, RSSI and first/last-seen time; phone beep; **Export Excel / PDF** (real .xlsx, EPCs as text, timestamps as Excel date-times).
- **Tag Ops** — read/write any bank, optional EPC filter and access password, lock/unlock/permalock (confirmed), kill (confirmed).
- **Settings** — reader mode (Active = watch key, Passive = app only) and watch-key action; watch buzzer; RF power; region with hopping or a fixed channel (per watch, re-applied on connect); Gen2; save to the watch; RFID Keyboard setup.
- **Info** — watch / UHF module / MCU versions, battery, Bluetooth address, last watch-key code.
- **RFID Keyboard** — an input method that types scanned EPCs into any app, sharing the app's single watch connection.

4.1 How the SDK is wrapped

- **One class touches the SDK** (`RfidController`); everything else uses plain Kotlin types and flows.
- **One SDK thread** — every call runs on `uwr6-sdk`; listeners only hand data over.
- **Spacing and checks** — a quiet gap before each exchange, plausibility checks on replies, retries for reads and setters, never for kill.
- **One connection per process** — `ReaderHub` is shared by the screens and the keyboard; a stale connect attempt can never tear down a newer link.
- **Reading sessions** — each start gets a session id and the stop only closes its own session; a Stop that arrives while a start is still queued cancels it.
- **Link loss** — the SDK's disconnect callback, the adapter state, the system GATT state and a once-a-minute battery heartbeat (three misses = lost).

5. Known SDK limitations & workarounds

Limitation (in the jar)	Workaround (in the demo app)
No request/response matching; one reply slot	Single SDK thread, gaps between exchanges, reply checks + retries
No connect timeout; onFailed never fires (BLE)	15 s app timeout; getVersion() handshake
UUIDs chosen from the cached name only	Preset via the SDK's own initUuid(); short targeted scan for unknown devices
Tag-access error codes dropped	One clear message: no tag / wrong password / locked
Any reply starting 0x05 is treated as a battery report	Battery values trusted only when no other command is waiting
getChannel() fails on channel 5; hopping can't be read	Plan stored per watch with its region; re-applied on connect
Single reads can turn continuous after 5000 tags	Counter reset before each single read
connectGatt uses TRANSPORT_AUTO	None needed so far; switch to a Kotlin Nordic-UART transport if GATT 133 appears on hardware

6. To confirm on hardware

The integration follows the jar and the vendor demo exactly, but no physical IDTS-UWR6 was available. Check these on the first real watch:

- The advertised name starts with **Wearable U6 BLE**, and connecting is reliable (no GATT 133).
- The scan key sends codes **131/132** (Info → Last watch key shows what arrives).
- One `inventory()` keeps reading for at least a second (re-sent every second), and the tag rate.
- Battery level is 0–100 % and charging is reported.
- Power range 16–26 dBm (the Chinese API document says 17–27) and the factory default.
- What `saveParameters()` persists, and what the buzzer commands do on the watch.
- The Select-based EPC filter addresses exactly one tag for read/write/lock/kill.
- How fast a switched-off or out-of-range watch is detected.

7. Troubleshooting

Symptom	Cause	Fix
SecurityException on scan/connect	Runtime permission missing	Request BLUETOOTH_SCAN/CONNECT (12+) or Location (6–11)
Scan finds nothing on Android 6–11	Location switch off	Ask the user to turn Location on
Connect never completes	Watch off / out of range; no SDK timeout	App-side 15 s timeout; power-cycle the watch
Connected but every command times out	Wrong UUIDs (name not cached)	Preset initUuid("Wearable U6 BLE") before connect
Commands return someone else's answer	Overlapping commands / no gap	One SDK thread; gaps; reply checks
Every call from a listener times out	SDK called on the Bluetooth thread	Hand data to your own thread first
getBattery() always times out	No OnBatteryListener registered	Register all six listeners

© Identium Tech Solutions Pvt. Ltd. · IDTS-UWR6 Android SDK Developer & Integration Guide v1.0 · The vendor library handheld_v1.1.64.jar is used unchanged.