IDENTIUM TECH SOLUTIONS PVT. LTD.

Creating Identities for All

UHF Handheld Reader

SDK & Application Developer Guide

CX1500N Series | Android SDK V1.2 | Complete Integration Reference

Version 1.0 | March 2026

Identium Tech Solutions Private Limited
Plot No. 5, First Floor, Santnagar, East of Kailash, New Delhi – 110065, India
Ph: 011-47147839 / 7011001472 | Email: info@identium.in
Web: identium.in | indiarfidstore.com | rfidindia.in
GSTIN: 07AAGCI6275E1ZH

Table of Contents

Chapter 1: Introduction

1.1 About This Document

This document is the official SDK & Application Developer Guide for the Identium CX1500N UHF Handheld RFID Reader, published by Identium Tech Solutions Private Limited. It covers:

- Android UHF SDK integration (AAR-based)
- All 12 core API functions with working code examples
- Flutter Plugin API and Dart code samples
- Keyboard Emulator App configuration and usage
- Error codes, troubleshooting, and best practices

NOTE: This document is intended for APK/App developers integrating the CX1500N into enterprise Android or Flutter applications. Basic Android development knowledge (Java/Kotlin) is assumed.

1.2 Device Overview – CX1500N UHF Handheld Reader

The CX1500N is a rugged UHF RFID handheld reader designed for industrial and enterprise asset tracking, inventory management, retail, logistics, and supply chain applications.

Specification	Details
Protocol	EPC Gen2 / ISO 18000-6C
Frequency – India	865–867 MHz (RG_PRC band)
Frequency – EU	865–868 MHz (RG_EU3)
Frequency – USA	902–928 MHz (RG_NA)
Read Range	Up to 10 metres (at 33 dBm, optimal conditions)
RF Power	5–33 dBm, adjustable in 1 dBm steps
Antenna	Internal, 1 antenna (antid = 1)
Interface	Internal UART – /dev/ttysWK0
OS Platform	Android 9.0 (API 28) and above
Barcode / QR	Integrated 1D/2D scanner via ScansManager SDK
Memory Banks	Reserved, EPC, TID, USER (EPC Gen2 standard)
RFID Standard	ISO 18000-6C

1.3 SDK Architecture Overview

The SDK is delivered as an AAR (Android Archive) library: UHF_SDK-debug.aar. It exposes four primary components:

Component	Package / Class	Responsibility
UHFEngine	com.pda.uhf.UHFEngine	Hardware power-on/off, module connect/disconnect, start/stop inventory, read/write/lock/kill
UHFParamsOperator	com.pda.uhf.UHFParamsOperator	Configure RF power, Gen2 session, target, region, RF mode, tag filter, embedded data
Reader API	com.uhf.api.cls.Reader	Data models: TagInfo, AntPowerConf, TagFilter_ST, Lock_Obj, Lock_Type, READER_ERR enum
ScansManager	(System Service: "scans")	Barcode/QR scanner control, input mode, alerts, prefix/suffix

1.4 Key Classes and Packages

```
// Core UHF Engine
import com.pda.uhf.UHFEngine;
import com.pda.uhf.UHFParamsOperator;
import com.pda.uhf.UHFUtils;
import com.pda.uhf.UHFSetting;

// Reader API & Callbacks
import com.uhf.api.cls.Reader;
import com.uhf.api.cls.ReadListener;
import com.uhf.api.cls.ReadExceptionListener;

// Inventory Models
import com.pda.uhf.model.InventoryMode;
import com.pda.uhf.model.InventoryParams;

// Parameter Result Models
import com.pda.uhf.model.params.AntPowerConfResult;
import com.pda.uhf.model.params.Gen2SessionResult;
import com.pda.uhf.model.params.Gen2TargetResult;
import com.pda.uhf.model.params.MaxPowerResult;
import com.pda.uhf.model.params.RegionConfResult;
```

Chapter 2: SDK Setup & Integration

2.1 Prerequisites

Requirement	Minimum Version / Value
Android Studio	4.0 or higher
Gradle	7.0+
minSdkVersion	21 (Android 5.0 Lollipop)
targetSdkVersion	33 (Android 13)
Java	Java 8+
Device	Identium CX1500N with Android 9.0+
SDK Files Required	UHF_SDK-debug.aar, android-support-v4.jar

2.2 Adding the SDK to Your Project

Step 1 – Copy SDK Files

Copy the following files into your project's app/libs/ directory:

```
app/
  libs/
    UHF_SDK-debug.aar      <- UHF reader SDK
    android-support-v4.jar <- Support library
```

Step 2 – Configure build.gradle

```
// app/build.gradle
android {
    compileSdkVersion 33
    defaultConfig {
        minSdkVersion 21
        targetSdkVersion 33
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.aar', '*.jar'])
    implementation 'androidx.appcompat:appcompat:1.4.1'
    implementation 'com.google.android.material:material:1.5.0'
}
```

Step 3 – AndroidManifest.xml Permissions

```
<!-- AndroidManifest.xml -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.VIBRATE"/>
```

2.3 Initialization Flow

The CX1500N UHF module must be powered on and connected before any RFID operations can be performed. The correct sequence is:

Step	Method Call	Notes
1. Power On	UHFEngine.getEngine().powerOn()	Must be called first. Powers the RF hardware.
2. Connect	UHFEngine.getEngine().connectModule("/dev/ttysWK0", 1)	UART path is fixed. Returns READER_ERR.
3. Configure	UHFParamsOperator.getInstance().setAntPowerConf()	Set power, session, region, RF mode.
4. Operate	startInventory() / getTagData() / writeTagData()	Perform RFID operations.
5. Disconnect	UHFEngine.getEngine().disconnectModule()	Call in onPause().
6. Power Off	UHFEngine.getEngine().powerOff()	Call in onPause() after disconnect.

```
@Override
protected void onResume() {
    super.onResume();
    // Step 1: Initialize root permissions (required once)
    UHFSetting.initRoot(getApplicationContext());

    // Step 2: Power on the UHF module
    UHFEngine.getEngine().powerOn();

    // Step 3: Connect to the module via internal UART
    Reader.READER_ERR er = UHFEngine.getEngine().connectModule("/dev/ttysWK0", 1);
    if (er == Reader.READER_ERR.MT_OK_ERR) {
        configureDefaultParameters();  // See Section 2.4
        Log.d("UHF", "Reader connected successfully");
    } else {
        Log.e("UHF", "Connection failed: " + er.name());
    }
}

@Override
protected void onPause() {
    // Always disconnect and power off when activity pauses
    UHFEngine.getEngine().disconnectModule();
    UHFEngine.getEngine().powerOff();
    super.onPause();
}
```

2.4 Default Parameter Configuration

After connecting, configure the default parameters. The following is the recommended configuration for India:

```
private void configureDefaultParameters() {
    // --- Set RF Power (33 dBm = maximum) ---
    Reader.AntPowerConf apcf = new Reader.AntPowerConf();
    apcf.antcnt = 1;  // CX1500N has one antenna
    Reader.AntPower antPower = new Reader.AntPower();
    antPower.antid     = 1;     // Antenna ID (always 1)
    antPower.readPower = 3300;  // 33.00 dBm read power
```

```
    antPower.writePower= 3300;  // 33.00 dBm write power
    apcf.Powers[0] = antPower;
    UHFParamsOperator.getInstance().setAntPowerConf(apcf);

    // --- Set Region (India uses RG_PRC band 865-867 MHz) ---
    UHFParamsOperator.getInstance().setRegionConf(Reader.Region_Conf.RG_PRC);

    // --- Set Gen2 Session (SESSION1 for moderate-large tag counts) ---
    UHFParamsOperator.getInstance().setGen2Session(1);  // SESSION1

    // --- Set Gen2 Target (A to B cycling) ---
    UHFParamsOperator.getInstance().setGen2Target(2);  // A_2_B

    // --- Set RF Mode (105 = default, suitable for most environments) ---
    UHFParamsOperator.getInstance().setGen2TagEncoding(105);
}
```

Chapter 3: Core API Reference

This chapter documents all 12 core functions of the Identium UHF SDK. Each function includes purpose, method signatures, parameters, return values, and a complete working code example.

3.1 Function 1 – Connect to Reader

Powers on the RF hardware and establishes communication with the UHF module over the internal UART interface.

Method	Return Type	Description
UHFEngine.getEngine().powerOn()	void	Powers on the UHF RF hardware
UHFEngine.getEngine().powerOff()	void	Powers off the UHF RF hardware
connectModule(String uartPath, int antCount)	Reader.READER_ERR	Connects to reader module via UART
disconnectModule()	void	Disconnects from the reader module

Parameter	Type	Value	Description
uartPath	String	"/dev/ttysWK0"	Fixed UART path for CX1500N
antCount	int	1	Number of antennas (always 1)

```
// ── Connect to UHF Reader ──────────────────────────
@Override
protected void onResume() {
    super.onResume();
    UHFSetting.initRoot(getApplicationContext()); // Required once
    UHFEngine.getEngine().powerOn();
    Reader.READER_ERR er = UHFEngine.getEngine()
        .connectModule("/dev/ttysWK0", 1);
    if (er == Reader.READER_ERR.MT_OK_ERR) {
        Toast.makeText(this, "UHF Reader Connected",
            Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(this, "Connection Failed: " + er.name(),
            Toast.LENGTH_LONG).show();
    }
}

@Override
protected void onPause() {
    UHFEngine.getEngine().disconnectModule();
    UHFEngine.getEngine().powerOff();
    super.onPause();
}
```

NOTE: If connectModule() fails, ensure powerOn() was called first and the device is a CX1500N. Retry after 500ms on first failure.

3.2 Function 2 – Read Single Tag

Reads one RFID tag from the RF field. Uses SINGLE_TAG inventory mode — ideal when you want to read one tag at a time with minimal interference.

```
// ── Read Single Tag ──────────────────────────────
private void readSingleTag() {
    InventoryParams params = new InventoryParams();
    params.ants          = new int[]{1};
    params.inventoryMode = InventoryMode.SINGLE_TAG;
    params.smartMode     = Reader.IT_MODE.IT_MODE_CT;
    params.timeout       = 50;
    params.isReadTid     = false;
    params.option        = 0;

    ReadListener listener = new ReadListener() {
        @Override
        public void tagRead(List<Reader.TagInfo> list) {
            if (list == null || list.isEmpty()) return;
            Reader.TagInfo tag = list.get(0); // First tag only
            String epc  = Reader.bytes_Hexstr(tag.EpcId);
            int rssi    = tag.RSSI;
            int freq    = tag.Frequency;
            Log.d("UHF", "EPC=" + epc + " RSSI=" + rssi + " Freq=" + freq);
            // Stop after one read
            UHFEngine.getEngine().stopInventory();
        }
    };

    ReadExceptionListener errListener = new ReadExceptionListener() {
        @Override
        public void tagReadException(Reader.READER_ERR er) {
            Log.e("UHF", "Read error: " + er.name());
        }
    };

    Reader.READER_ERR er = UHFEngine.getEngine()
        .startInventory(params, listener, errListener);
    if (er != Reader.READER_ERR.MT_OK_ERR) {
        Log.e("UHF", "Failed to start inventory: " + er.name());
    }
}
```

3.3 Function 3 – Read Multiple Tags (Inventory Mode)

Continuously scans and collects all RFID tags in the RF field. Best for warehouse scanning, stock-takes, and asset tracking.

Mode Enum	Description	Use Case
NORNAL	Standard mode	General use, up to ~100 tags
E7_NEW_FAST	Fast mode for E7 chips	E7-chipset tags, speed priority
SINGLE_TAG	Single tag at a time	One-by-one reading
E7_READ_STOP	Read and stop for E7	E7 chips, stop-after-read
E7_SMART	Intelligent temp-control mode	E7 with thermal management

Mode Enum	Description	Use Case
MULTI_TAG_FAST	Optimised for high tag volumes	Large inventories (RECOMMENDED)

Session	Value	Recommended For
SESSION0	0	< 10 tags, fastest response
SESSION1	1	10–500 tags (DEFAULT)
SESSION2	2	Advanced / multi-reader environments
SESSION3	3	Advanced / multi-reader environments

```
// ── Multi-Tag Inventory ──────────────────────────────
private boolean inventoryRunning = false;

private void startMultiTagInventory() {
    InventoryParams params = new InventoryParams();
    params.ants          = new int[]{1};
    params.inventoryMode = InventoryMode.MULTI_TAG_FAST;
    params.smartMode     = Reader.IT_MODE.IT_MODE_CT;
    params.timeout       = 50;
    params.isReadTid     = false;
    params.option        = METAFLAG.RSSI | METAFLAG.Frequency;

    // Configure session for large tag counts
    UHFParamsOperator.getInstance().setGen2Session(1);  // SESSION1
    UHFParamsOperator.getInstance().setGen2Target(2);   // A_2_B

    ReadListener listener = new ReadListener() {
        @Override
        public void tagRead(List<Reader.TagInfo> list) {
            if (list == null) return;
            for (Reader.TagInfo tag : list) {
                String epc = Reader.bytes_Hexstr(tag.EpcId);
                String tid = Reader.bytes_Hexstr(tag.EmbededData);
                int rssi   = tag.RSSI;
                int freq   = tag.Frequency;
                addTagToList(epc, tid, rssi, freq);
            }
            runOnUiThread(() -> adapter.notifyDataSetChanged());
        }
    };

    Reader.READER_ERR er = UHFEngine.getEngine()
        .startInventory(params, listener, errListener);
    if (er == Reader.READER_ERR.MT_OK_ERR) {
        inventoryRunning = true;
    }
}

private void stopInventory() {
    UHFEngine.getEngine().stopInventory();
    inventoryRunning = false;
}
```

3.4 Function 4 – Read EPC Only

Directly reads the EPC (Electronic Product Code) memory bank from a targeted tag using getTagData().

Memory Bank	Constant	Value	Contents
Reserved	BANK.Reserved	0x00	Access password + Kill password
EPC	BANK.EPC	0x01	Electronic Product Code (writable)
TID	BANK.TID	0x02	Tag Identifier (factory-programmed, read-only)
USER	BANK.USER	0x03	User-defined custom data (writable)

```
// ── Read EPC Memory Bank ──────────────────────────
private String readEPC() {
    byte[] epcData  = new byte[12];  // 96-bit EPC = 12 bytes
    byte[] password = {0x00, 0x00, 0x00, 0x00}; // No password

    // Retry up to 3 times
    Reader.READER_ERR er = null;
    for (int i = 0; i < 3; i++) {
        er = UHFEngine.getEngine().getTagData(
            1,                      // Antenna number (always 1 for CX1500N)
            (char) BANK.EPC,        // Bank = EPC (0x01)
            2,                      // Start block: skip PC word (blocks 0-1)
            6,                      // 6 blocks x 2 bytes = 12 bytes = 96 bits
            epcData,                // Output buffer
            password,               // Access password
            (short) 1000            // Timeout in milliseconds
        );
        if (er == Reader.READER_ERR.MT_OK_ERR) break;
    }

    if (er == Reader.READER_ERR.MT_OK_ERR) {
        return Util.bytes2hex(epcData).toUpperCase();
    }
    return null; // Read failed
}
```

3.5 Function 5 – Read EPC + TID

Reads both the EPC and TID memory banks. TID (Tag Identifier) is factory-programmed and unique to each chip — it cannot be modified. Two methods are provided.

Method A – Via Inventory with Embedded Data (concurrent read)

```
// ── Configure TID to be read alongside EPC during inventory ──
private void enableTidDuringInventory() {
    Reader.EmbeddedData_ST edst = new Reader.EmbeddedData_ST();
    edst.bank      = BANK.TID;   // Read from TID bank
    edst.startaddr = 0;          // Start at block 0
    edst.bytecnt   = 12;         // Read 12 bytes (96-bit TID)
    edst.accesspwd = null;       // No access password
    UHFParamsOperator.getInstance().setTagEmbededData(edst);
}

// ── In your InventoryParams, enable EmdData ──────────
params.isReadTid = true;
params.option    = METAFLAG.RSSI | METAFLAG.EmdData;
```

```
// ── In tagRead callback ──────────────────────────────
String epc = Reader.bytes_Hexstr(tagInfo.EpcId);
String tid = Reader.bytes_Hexstr(tagInfo.EmbededData);
Log.d("UHF", "EPC: " + epc + "  TID: " + tid);
```

Method B – Direct TID Read with EPC Filter (targeted)

```
// ── Read TID for a specific tag identified by its EPC ──
private String readTidForTag(String targetEpc) {
    // Step 1: Set EPC filter to target this specific tag
    Reader.TagFilter_ST filter = new Reader.TagFilter_ST();
    byte[] epcBytes = hexToBytes(targetEpc);
    filter.fdata     = epcBytes;
    filter.flen      = epcBytes.length * 8; // Length in bits
    filter.bank      = BANK.EPC;
    filter.startaddr = 32;  // Skip PC word (first 32 bits of EPC bank)
    filter.isInvert  = 0;
    UHFParamsOperator.getInstance().setTagFilter(filter);

    // Step 2: Read TID bank
    byte[] tidData  = new byte[12];
    byte[] password = {0x00, 0x00, 0x00, 0x00};
    Reader.READER_ERR er = null;
    for (int i = 0; i < 3; i++) {
        er = UHFEngine.getEngine().getTagData(
            1, (char) BANK.TID, 0, 6, tidData, password, (short)1000
        );
        if (er == Reader.READER_ERR.MT_OK_ERR) break;
    }

    // Step 3: Clear filter
    UHFParamsOperator.getInstance().setTagFilter(null);

    return er == Reader.READER_ERR.MT_OK_ERR ?
        Util.bytes2hex(tidData).toUpperCase() : null;
}
```

3.6 Function 6 – Read EPC + TID + User Memory

Reads all three user-accessible memory banks sequentially: EPC (product identity), TID (chip identity), and User (application data).

```
// ── Read EPC + TID + User Memory Banks ──────────────
public class TagMemory {
    public String epc;
    public String tid;
    public String user;
    public boolean success;
}

private TagMemory readAllMemoryBanks(String targetEpc) {
    TagMemory result = new TagMemory();
    byte[] password  = {0x00, 0x00, 0x00, 0x00};

    // Set EPC filter to target specific tag
    if (!setEpcFilter(targetEpc)) {
        return result; // Filter failed
    }
```

```
    // Read EPC bank (bank=0x01, startBlock=2, blocks=6 -> 12 bytes)
    byte[] epcData = new byte[12];
    Reader.READER_ERR er = UHFEngine.getEngine()
        .getTagData(1, (char)0x01, 2, 6, epcData, password, (short)1000);
    if (er == Reader.READER_ERR.MT_OK_ERR)
        result.epc = Util.bytes2hex(epcData).toUpperCase();

    // Read TID bank (bank=0x02, startBlock=0, blocks=6 -> 12 bytes)
    byte[] tidData = new byte[12];
    er = UHFEngine.getEngine()
        .getTagData(1, (char)0x02, 0, 6, tidData, password, (short)1000);
    if (er == Reader.READER_ERR.MT_OK_ERR)
        result.tid = Util.bytes2hex(tidData).toUpperCase();

    // Read USER bank (bank=0x03, startBlock=0, blocks=16 -> 32 bytes)
    byte[] userData = new byte[32];
    er = UHFEngine.getEngine()
        .getTagData(1, (char)0x03, 0, 16, userData, password, (short)1000);
    if (er == Reader.READER_ERR.MT_OK_ERR)
        result.user = Util.bytes2hex(userData).toUpperCase();

    // Always clear filter when done
    UHFParamsOperator.getInstance().setTagFilter(null);
    result.success = (result.epc != null);
    return result;
}
```

3.7 Function 7 – Read QR Code / Barcode

Activates the integrated 1D/2D barcode scanner using the ScansManager system service. The CX1500N includes a dedicated scan module with vibration, audio, and LED feedback.

Input Mode	Value	Description
Broadcast	0	Data sent via broadcast intent android.scanservice.action.UPLOAD_BARCODE_DATA
Direct Input	1	Data typed directly into the currently focused text field
Keyboard	2	Simulated keyboard keystrokes to focused field
Single Input	3	Broadcast mode, duplicate scans ignored

```
// ── Barcode / QR Scanner Setup ──────────────────────────
public static final String SCANS_SERVICE = "scans";
private ScansManager mScansManager;
private BroadcastReceiver scanReceiver;

private void initScanner() {
    mScansManager = (ScansManager)
        context.getSystemService(SCANS_SERVICE);

    // Configure scanner feedback
    mScansManager.setInputMode(0);           // Broadcast mode
    mScansManager.setDecodeTipVibrator(true); // Vibrate on scan
    mScansManager.setDecodeTipAudio(true);    // Beep on scan
    mScansManager.setLedNotify(true);         // LED flash
    mScansManager.setExtras(1);              // Append Enter key
    mScansManager.openScan(true);             // Power on scanner
```

```
    // Register broadcast receiver for scan results
    IntentFilter filter = new IntentFilter();
    filter.addAction(
        "android.scanservice.action.UPLOAD_BARCODE_DATA");
    scanReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context ctx, Intent intent) {
            String barcode = intent.getStringExtra("barcode");
            if (barcode != null) {
                onBarcodeScanned(barcode);
            }
        }
    };
    registerReceiver(scanReceiver, filter);
}

private void onBarcodeScanned(String data) {
    Log.d("SCAN", "Scanned: " + data);
    // Process barcode/QR code data here
}

// Trigger scan via software (or use hardware scan key)
private void triggerScan() {
    mScansManager.startScan();
}

@Override
protected void onDestroy() {
    if (scanReceiver != null) unregisterReceiver(scanReceiver);
    mScansManager.openScan(false); // Power off scanner
    super.onDestroy();
}
```

3.8 Function 8 – Set RF Power (Slider-Based)

Adjusts the RF transmission power for read and write operations. A SeekBar (slider) provides intuitive control from 5 dBm to 33 dBm.

Power Level	Internal Value	Use Case
5 dBm (min)	500	Very short range, low interference
10 dBm	1000	Short range (< 1m)
20 dBm	2000	Medium range (2-4m)
30 dBm	3000	Long range (7-9m)
33 dBm (max)	3300	Maximum range (DEFAULT, up to 10m)

XML Layout – SeekBar Slider

```
<!-- res/layout/activity_main.xml -->
<LinearLayout android:orientation="vertical">
  <TextView
      android:id="@+id/powerLabel"
      android:text="RF Power: 33 dBm"
      android:textSize="16sp" />
  <SeekBar
```

```
      android:id="@+id/powerSeekBar"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:max="28"
      android:progress="28" />
  <!-- max=28: positions 0-28 map to 5-33 dBm (28 steps x 1 dBm) -->
</LinearLayout>
```

Java Implementation

```
// ── RF Power Control with SeekBar ──────────────────────
private SeekBar powerSeekBar;
private TextView powerLabel;

// Position 0 = 5 dBm (value 500), Position 28 = 33 dBm (value 3300)
private short positionToPower(int position) {
    return (short) ((5 + position) * 100);
}
private int powerToPosition(short power) {
    return power / 100 - 5;
}

private void initPowerSlider() {
    powerSeekBar = findViewById(R.id.powerSeekBar);
    powerLabel   = findViewById(R.id.powerLabel);

    powerSeekBar.setOnSeekBarChangeListener(
        new SeekBar.OnSeekBarChangeListener() {
        @Override
        public void onProgressChanged(SeekBar sb, int progress,
                boolean fromUser) {
            short pwr = positionToPower(progress);
            powerLabel.setText("RF Power: " + (pwr/100) + " dBm");
        }
        @Override public void onStartTrackingTouch(SeekBar sb) {}
        @Override
        public void onStopTrackingTouch(SeekBar sb) {
            applyRFPower(positionToPower(sb.getProgress()));
        }
    });
}

private void applyRFPower(short powerValue) {
    Reader.AntPowerConf apcf = new Reader.AntPowerConf();
    apcf.antcnt = 1;
    Reader.AntPower ap = new Reader.AntPower();
    ap.antid      = 1;
    ap.readPower  = powerValue;
    ap.writePower = powerValue;
    apcf.Powers[0] = ap;
    Reader.READER_ERR er =
        UHFParamsOperator.getInstance().setAntPowerConf(apcf);
    Log.d("UHF", "Set power " + powerValue + " -> " + er.name());
}
```

3.9 Function 9 – Set RF Mode

Configures the RF encoding/modulation mode to optimise performance for different tag types and environments.

Index	Mode Value	Description	Best For
0	203	High sensitivity	Long range, sparse tags
1	111	Dense reader	Many readers in same area
2	220	High speed	High-speed conveyor belts
3	101	Standard	General purpose
4	45	Low interference	Noisy RF environments
5	115	Balanced	Mixed environments
6	112	Dense reader 2	High-density deployments
7	103	Medium speed	Moderate tag counts
8	105	DEFAULT (recommended)	Most standard deployments
9	107	Extended range	Maximum read distance
10	113	High throughput	Very large tag volumes

```
// ── Set RF Mode ──────────────────────────────────────
private static final int[] RF_MODE =
    {203, 111, 220, 101, 45, 115, 112, 103, 105, 107, 113};

// Using a Spinner (dropdown) for RF Mode selection
rfModeSpinner.setOnItemSelectedListener(
    new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent,
            View view, int position, long id) {
        int selectedMode = RF_MODE[position];
        Reader.READER_ERR er = UHFParamsOperator.getInstance()
            .setGen2TagEncoding(selectedMode);
        if (er == Reader.READER_ERR.MT_OK_ERR) {
            Toast.makeText(ctx, "RF Mode: " + selectedMode,
                Toast.LENGTH_SHORT).show();
        }
    }
    @Override
    public void onNothingSelected(AdapterView<?> parent) {}
});

// ── Set Region ──────────────────────────────────────
// India (865-867 MHz): RG_PRC
// Europe (865-868 MHz): RG_EU3
// North America (902-928 MHz): RG_NA
// Open (all bands): RG_OPEN
UHFParamsOperator.getInstance()
    .setRegionConf(Reader.Region_Conf.RG_PRC);

// ── Gen2 Session & Target ──────────────────────────
// Session: 0=few tags, 1=many tags (DEFAULT)
UHFParamsOperator.getInstance().setGen2Session(1);
// Target: 0=A, 1=B, 2=A->B (DEFAULT), 3=B->A
UHFParamsOperator.getInstance().setGen2Target(2);
```

3.10 Function 10 – Write EPC / Write User Memory

Writes data to a tag's EPC or User memory bank. Always set an EPC filter to target the specific tag before writing.

⚠ **WARNING:** ALWAYS set an EPC filter before writing. Writing without a filter will overwrite ANY tag in the RF field.

Write EPC (using writeTagEpcEx – recommended)

```
// ── Write new EPC to a specific tag ──────────────────
private boolean writeEPC(String currentEpc, String newEpc,
                         String passwordHex) {
    // Step 1: Set filter to target only this tag
    if (!setEpcFilter(currentEpc)) return false;

    // Step 2: Prepare data
    byte[] epcBytes = hexToBytes(newEpc);
    byte[] password = hexToBytes(passwordHex); // "00000000" = no pwd

    // Step 3: Write EPC (retry up to 3 times)
    Reader.READER_ERR er = null;
    for (int i = 0; i < 3; i++) {
        er = UHFEngine.getEngine().writeTagEpcEx(
            1,               // Antenna
            epcBytes,        // New EPC data
            epcBytes.length, // Length in bytes
            password,        // Access password
            (short) 1000     // Timeout ms
        );
        if (er == Reader.READER_ERR.MT_OK_ERR) break;
    }

    // Step 4: Clear filter
    UHFParamsOperator.getInstance().setTagFilter(null);
    return er == Reader.READER_ERR.MT_OK_ERR;
}
```

Write User Memory (using writeTagData)

```
// ── Write to USER memory bank ────────────────────────
private boolean writeUserMemory(String targetEpc,
        String hexData, int startBlock, String passwordHex) {
    if (!setEpcFilter(targetEpc)) return false;

    byte[] writeData = hexToBytes(hexData);
    byte[] password  = hexToBytes(passwordHex);

    Reader.READER_ERR er = null;
    for (int i = 0; i < 3; i++) {
        er = UHFEngine.getEngine().writeTagData(
            1,               // Antenna
            (char) BANK.USER,// Bank: USER (0x03)
            startBlock,      // Start block (0-based)
            writeData,       // Data bytes
            writeData.length,// Data length
            password,        // Access password
            (short) 1000     // Timeout ms
        );
        if (er == Reader.READER_ERR.MT_OK_ERR) break;
    }

    UHFParamsOperator.getInstance().setTagFilter(null);
```

```
    return er == Reader.READER_ERR.MT_OK_ERR;
}

// ── Helper: Set EPC Filter ──────────────────────────
private boolean setEpcFilter(String epc) {
    if (epc == null || epc.length() % 2 != 0) return false;
    Reader.TagFilter_ST filter = new Reader.TagFilter_ST();
    byte[] epcBytes    = hexToBytes(epc);
    filter.fdata       = epcBytes;
    filter.flen        = epcBytes.length * 8; // bits
    filter.bank        = BANK.EPC;
    filter.startaddr   = 32; // Skip PC word
    filter.isInvert    = 0;
    Reader.READER_ERR er =
        UHFParamsOperator.getInstance().setTagFilter(filter);
    return er == Reader.READER_ERR.MT_OK_ERR;
}
```

3.11 Function 11 – Lock Tag

Locks specific memory banks or passwords to prevent unauthorised modification. Three lock types are available: Unlock, Lock (reversible), and Permanent Lock (irreversible).

Lock Object Enum	Bank/Area Locked
LOCK_OBJECT_ACCESS_PASSWD	Access password memory
LOCK_OBJECT_KILL_PASSWORD	Kill password memory
LOCK_OBJECT_BANK1	EPC memory bank
LOCK_OBJECT_BANK2	TID memory bank
LOCK_OBJECT_BANK3	User memory bank

Lock Type	Effect	Reversible?
BANK1_UNLOCK	Remove lock from EPC bank	Yes
BANK1_LOCK	Lock EPC bank with password	Yes (with password)
BANK1_PERM_LOCK	Permanently lock EPC bank	NO – IRREVERSIBLE
BANK3_UNLOCK	Remove lock from User bank	Yes
BANK3_LOCK	Lock User bank with password	Yes (with password)
BANK3_PERM_LOCK	Permanently lock User bank	NO – IRREVERSIBLE

⚠ **WARNING:** PERM_LOCK operations are permanently irreversible. The tag memory can never be written again. Always show a strong confirmation dialog before executing.

```
// ── Lock a Tag's EPC Bank ──────────────────────────
private void lockTag(String targetEpc, Reader.Lock_Obj lockObj,
                     Reader.Lock_Type lockType, String passwordHex) {
    // Show confirmation dialog first!
    new AlertDialog.Builder(this)
        .setTitle("Confirm Lock Operation")
```

```
        .setMessage("Lock tag " + targetEpc + "? " +
            (lockType.name().contains("PERM") ?
            "\nWARNING: This is PERMANENT!" : ""))
        .setPositiveButton("Lock", (d, w) -> {
            performLock(targetEpc, lockObj, lockType, passwordHex);
        })
        .setNegativeButton("Cancel", null)
        .show();
}

private void performLock(String targetEpc, Reader.Lock_Obj lockObj,
                         Reader.Lock_Type lockType, String pwdHex) {
    if (!setEpcFilter(targetEpc)) return;

    byte[] password = hexToBytes(pwdHex);
    Reader.READER_ERR er = UHFEngine.getEngine().lockTag(
        1,                          // Antenna
        (byte) lockObj.value(),     // Lock object
        (short) lockType.value(), // Lock type
        password,                   // Access password
        (short) 1000                // Timeout ms
    );

    UHFParamsOperator.getInstance().setTagFilter(null);

    String msg = er == Reader.READER_ERR.MT_OK_ERR ?
        "Lock Successful" : "Lock Failed: " + er.name();
    Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();
}
```

3.12 Function 12 – Destroy (Kill) Tag

Permanently disables a tag. Once killed, the tag will never respond to any RFID command again. This operation is completely irreversible.

⚠ **WARNING:** KILL is IRREVERSIBLE. The tag is permanently disabled — it cannot be reactivated. A kill password must be set on the tag for kill to succeed. The default kill password 0x00000000 may cause kill to fail on some tags.

Method	Signature
killTag	Reader.READER_ERR killTag(int antId, byte[] killPassword, short timeout)

```
// ── Destroy (Kill) a Tag ──────────────────────────────
private void killTag(String targetEpc, String killPasswordHex) {
    // MANDATORY: Show irreversible-action warning dialog
    new AlertDialog.Builder(this)
        .setTitle("⚠ IRREVERSIBLE ACTION")
        .setMessage(
            "Killing tag: " + targetEpc + "\n\n" +
            "This will PERMANENTLY disable the tag.\n" +
            "It cannot be reactivated. Proceed?")
        .setIcon(android.R.drawable.ic_dialog_alert)
        .setPositiveButton("KILL TAG", (dialog, which) -> {
            performKillTag(targetEpc, killPasswordHex);
        })
        .setNegativeButton("Cancel", null)
        .show();
```

```
}

private void performKillTag(String targetEpc, String pwdHex) {
    // Target specific tag with EPC filter
    if (!setEpcFilter(targetEpc)) {
        Toast.makeText(this, "Filter failed",
            Toast.LENGTH_SHORT).show();
        return;
    }

    byte[] killPassword = hexToBytes(pwdHex);
    Reader.READER_ERR er = UHFEngine.getEngine().killTag(
        1,            // Antenna number
        killPassword, // Kill password (4 bytes)
        (short) 1000  // Timeout ms
    );

    UHFParamsOperator.getInstance().setTagFilter(null);

    if (er == Reader.READER_ERR.MT_OK_ERR) {
        Toast.makeText(this,
            "Tag Killed - Permanently Disabled",
            Toast.LENGTH_LONG).show();
    } else {
        Toast.makeText(this,
            "Kill Failed: " + er.name() +
            "\nCheck kill password.",
            Toast.LENGTH_LONG).show();
    }
}
```

Chapter 4: Flutter Plugin Documentation

4.1 Overview

The Identium UHF Flutter Plugin wraps the native Android UHF SDK (AAR) and exposes a clean Dart API via Flutter's MethodChannel and EventChannel. This enables cross-platform Flutter applications running on the CX1500N to control all UHF reader features.

4.2 Plugin Architecture

```
Flutter App (Dart Layer)
         |
         | MethodChannel: 'identium_uhf_plugin'
         | EventChannel:  'identium_uhf_plugin/events'
         |
         v
UHFPlugin.java (Android Native Bridge)
         |
         v
UHFEngine + UHFParamsOperator (SDK .aar)
         |
         v
CX1500N Hardware (UART /dev/ttysWK0)
```

4.3 Plugin Setup

pubspec.yaml

```
dependencies:
  flutter:
    sdk: flutter
  identium_uhf_plugin:
    path: ../identium_uhf_plugin  # Local plugin path
    # OR from pub.dev once published:
    # identium_uhf_plugin: ^1.0.0
```

Android build.gradle (plugin's android folder)

```
// identium_uhf_plugin/android/build.gradle
android {
    compileSdkVersion 33
    dependencies {
        implementation fileTree(dir: 'libs',
            include: ['*.aar', '*.jar'])
    }
}
```

4.4 Dart API Reference

```
import 'package:identium_uhf_plugin/identium_uhf_plugin.dart';

final IdentiumUHFPlugin uhf = IdentiumUHFPlugin();

// ── Connection ──────────────────────────────────
bool connected = await uhf.connect();
await uhf.disconnect();
```

```
// ── Read Single Tag ──────────────────────────────
UHFTag? tag = await uhf.readSingleTag(timeoutMs: 2000);
if (tag != null) print('EPC: ${tag.epc}  RSSI: ${tag.rssi}');

// ── Multi-Tag Inventory ──────────────────────────
uhf.startInventory(
  mode: InventoryMode.multiTagFast,
  readTid: true,
  onTagRead: (List<UHFTag> tags) {
    for (var t in tags) {
      print('EPC: ${t.epc}  TID: ${t.tid}  RSSI: ${t.rssi}');
    }
  },
  onError: (String error) => print('Error: $error'),
);
await uhf.stopInventory();

// ── Read Operations ──────────────────────────────
String? epc = await uhf.readEPC(password: '00000000');

Map<String, String>? epcTid = await uhf.readEPCAndTID(
  password: '00000000');

Map<String, String>? all = await uhf.readAllBanks(
  targetEpc: 'E2801160600002000000001',
  password: '00000000',
  userBlockCount: 16,
);
print('EPC: ${all?["epc"]} TID: ${all?["tid"]} USR: ${all?["user"]}');

// ── RF Power & Mode ──────────────────────────────
await uhf.setRFPower(dbm: 33);   // 5–33 dBm
await uhf.setRFMode(modeIndex: 8); // Default mode

// ── Write Operations ─────────────────────────────
bool ok = await uhf.writeEPC(
  targetEpc: 'E2801160600002000000001',
  newEpc:    'E2801160600002000000002',
  password:  '00000000',
);

bool written = await uhf.writeUserMemory(
  targetEpc:  'E2801160600002000000001',
  hexData:    '48656C6C6F576F726C6421',
  startBlock: 0,
  password:   '00000000',
);

// ── Lock ─────────────────────────────────────────
bool locked = await uhf.lockTag(
  targetEpc:  'E2801160600002000000001',
  lockObject: LockObject.epcBank,
  lockType:   LockType.lock,
  password:   '00000000',
);

// ── Kill ─────────────────────────────────────────
bool killed = await uhf.killTag(
  targetEpc:    'E2801160600002000000001',
  killPassword: '12345678',
);

// ── Barcode / QR ─────────────────────────────────
```

```
String? barcode = await uhf.scanBarcode(timeoutMs: 5000);
```

4.5 Data Models

```
// ── UHFTag model ──────────────────────────────────────
class UHFTag {
  final String  epc;
  final String? tid;
  final String? userData;
  final int     rssi;
  final int     frequency;
  int           count;

  UHFTag({
    required this.epc,
    this.tid,
    this.userData,
    required this.rssi,
    required this.frequency,
    this.count = 1,
  });
}

// ── Enumerations ──────────────────────────────────────
enum InventoryMode {
  normal, e7NewFast, singleTag,
  e7ReadStop, e7Smart, multiTagFast
}
enum LockObject {
  accessPassword, killPassword,
  epcBank, tidBank, userBank
}
enum LockType { unlock, lock, permanentLock }
```

4.6 Flutter UI Example – Full Inventory Screen

```
class InventoryScreen extends StatefulWidget {
  @override
  _InventoryScreenState createState() => _InventoryScreenState();
}

class _InventoryScreenState extends State<InventoryScreen> {
  final uhf    = IdentiumUHFPlugin();
  List<UHFTag> tags = [];
  bool  isRunning   = false;
  double rfPower    = 33;  // dBm

  @override
  void initState() {
    super.initState();
    uhf.connect();
  }

  @override
  void dispose() {
    uhf.stopInventory();
    uhf.disconnect();
    super.dispose();
  }

  void toggleInventory() {
    if (isRunning) {
```

```
    uhf.stopInventory();
  } else {
    setState(() => tags.clear());
    uhf.startInventory(
      mode: InventoryMode.multiTagFast,
      readTid: false,
      onTagRead: (newTags) =>
        setState(() => tags.addAll(newTags)),
      onError: (e) => ScaffoldMessenger.of(context)
        .showSnackBar(SnackBar(content: Text(e))),
    );
  }
  setState(() => isRunning = !isRunning);
}

@override
Widget build(BuildContext context) => Scaffold(
  appBar: AppBar(
    title: Text('Identium UHF Inventory'),
    backgroundColor: Color(0xFF1A3C6E),
  ),
  body: Column(children: [
    Padding(
      padding: EdgeInsets.all(16),
      child: Column(children: [
        Text('RF Power: ${rfPower.round()} dBm',
          style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold)),
        Slider(
          min: 5, max: 33, divisions: 28,
          value: rfPower,
          activeColor: Color(0xFF2E75B6),
          onChanged: (v) {
            setState(() => rfPower = v);
            uhf.setRFPower(dbm: v.round());
          },
        ),
      ]),
    ),
    ElevatedButton.icon(
      icon: Icon(isRunning ? Icons.stop : Icons.play_arrow),
      label: Text(isRunning ? 'Stop' : 'Start Inventory'),
      style: ElevatedButton.styleFrom(
        backgroundColor:
          isRunning ? Colors.red : Color(0xFF1A3C6E)),
      onPressed: toggleInventory,
    ),
    Padding(
      padding: EdgeInsets.all(8),
      child: Text('Tags Found: ${tags.length}',
        style: TextStyle(fontSize: 16)),
    ),
    Expanded(
      child: ListView.builder(
        itemCount: tags.length,
        itemBuilder: (ctx, i) => Card(
          child: ListTile(
            title: Text(tags[i].epc,
              style: TextStyle(fontFamily: 'monospace')),
            subtitle: Text(
              'RSSI: ${tags[i].rssi} dBm  |  Freq: ${tags[i].frequency} kHz'),
          ),
        ),
      ),
    ),
  ]),
);
```

```
}
```

Chapter 5: Keyboard Emulator App

5.1 Overview

The Identium Keyboard Emulator App allows the CX1500N to act as a virtual keyboard. When a tag is scanned or a barcode is read, the data is automatically sent as keyboard keystrokes to any focused application on the device — or to a connected host via Bluetooth/USB HID. No special software is required on the host.

NOTE: This is ideal for integrating with legacy ERP systems, Excel, POS terminals, or any application that accepts keyboard input.

5.2 How It Works

Step	Action
1	CX1500N scans an RFID tag or barcode using physical scan key or button
2	App retrieves EPC / TID / barcode data from SDK
3	Data is formatted (prefix + data + suffix)
4	App uses ScansManager keyboard mode (setInputMode(2)) to send as keystrokes
5	Host application receives data as if typed on a keyboard

5.3 Configuration Options

Setting	Options	Default
Data Type	EPC / TID / EPC+TID / EPC+USER / Barcode	EPC
Output Format	HEX / ASCII / Decimal	HEX
Prefix	Any text string	(none)
Suffix	Enter / Tab / None / Custom string	Enter
Case	UPPERCASE / lowercase	UPPERCASE
Separator (EPC+TID)	Space / Comma / Tab / Custom	Space
Char delay	0–100 ms between characters	5 ms
Duplicate filter	Ignore / Allow	Ignore (3s window)
Sound on read	On / Off	On
Vibrate on read	On / Off	On
LED on read	On / Off	On

5.4 Android Implementation

```
// ── Keyboard Emulator Core ──────────────────────────
public class KeyboardEmulatorManager {
    private ScansManager mScansManager;
    private String prefix = "";
    private String suffix = "\n"; // Enter key

    public void init(Context ctx) {
        mScansManager = (ScansManager)
            ctx.getSystemService("scans");
        // Set keyboard simulation mode
        mScansManager.setInputMode(2); // Keyboard mode
        mScansManager.setExtras(1);    // Append Enter
        mScansManager.setDecodeTipAudio(true);
        mScansManager.setDecodeTipVibrator(true);
        mScansManager.setLedNotify(true);
        mScansManager.openScan(true);
    }

    // Called when a UHF tag is read
    public void sendTagAsKeystrokes(String epc) {
        // Format: prefix + EPC + suffix
        String output = prefix + epc.toUpperCase() + suffix;
        // In keyboard mode (setInputMode(2)),
        // ScansManager injects this as keystrokes automatically
        // For RFID data (not from scan key), use AccessibilityService
        // or InputMethodService for HID injection
        Log.d("KBD", "Sending: " + output);
    }

    public void setPrefix(String p) { this.prefix = p; }
    public void setSuffix(String s) { this.suffix = s; }
}
```

5.5 App Screens Reference

Main Screen

UI Element	Description
Connection Status LED	Green = Connected, Red = Disconnected
Mode Selector	UHF RFID / Barcode / Both
Data Format	EPC / TID / EPC+TID / EPC+USER
RF Power Slider	5 dBm to 33 dBm real-time adjustment
Prefix Field	Text to prepend to every scan result
Suffix Selector	Enter / Tab / None / Custom
Start / Stop Button	Toggle continuous inventory mode
Last Scanned	Shows most recent EPC or barcode
Count	Total tags scanned in current session

Settings Screen

Setting Group	Options Available
RF Configuration	Power (5-33 dBm), RF Mode (11 options), Region (India/EU/USA/Open)
Gen2 Parameters	Session (0-3), Target (A/B/A-B/B-A), Inventory Mode (6 options)
Data Options	Read TID toggle, Read User Memory toggle, User block count
Notifications	Beep on/off, Vibrate on/off, LED on/off
Format	Prefix, Suffix, Separator, Case (UPPER/lower)
Advanced	Duplicate filter window (1-10s), Char delay (0-100ms)

Chapter 6: Error Codes Reference

All SDK methods return Reader.READER_ERR enum values. Always check return values and handle errors gracefully.

Error Enum	Meaning	Recommended Action
MT_OK_ERR	Success	Continue processing
MT_ERR_NORESP	No tag response received	Move tag closer, retry up to 3x
MT_ERR_TIMEOUT	Operation timed out	Increase timeout, retry
MT_ERR_PARAM	Invalid parameter passed	Check all parameter values
MT_ERR_LOCKED	Tag memory bank is locked	Provide correct access password
MT_ERR_ACCESS	Wrong access password	Verify password; default is 00000000
MT_ERR_KILL_ZERO	Kill password is zero / not set	Set a non-zero kill password first
MT_ERR_POWEROFF	Module not powered on	Call powerOn() before connectModule()
MT_ERR_CONNECT	Module not connected	Call connectModule() before operations
MT_ERR_FILTERLEN	Filter data length invalid	Ensure filter length is multiple of 2 chars
MT_ERR_OVERFLOWED	Data overflow / buffer too small	Increase buffer size for read operation
MT_ERR_ANTENNA	Antenna error	Check hardware; restart module

```
// ── Standard Error Handling Pattern ──────────────────
private boolean executeWithRetry(Callable<Reader.READER_ERR> op,
                                 int maxRetries) {
    Reader.READER_ERR er = null;
    for (int attempt = 0; attempt < maxRetries; attempt++) {
        try {
            er = op.call();
            if (er == Reader.READER_ERR.MT_OK_ERR) return true;
            Log.w("UHF", "Attempt " + (attempt+1) +
                " failed: " + er.name());
        } catch (Exception e) {
            Log.e("UHF", "Exception: " + e.getMessage());
        }
    }
    Log.e("UHF", "All retries failed. Last error: " +
        (er != null ? er.name() : "unknown"));
    return false;
}
```

Chapter 7: Best Practices & Troubleshooting

7.1 Performance Best Practices

Scenario	Recommendation
< 10 tags	Use SESSION0 — fastest response, minimal collision handling
10–500 tags	Use SESSION1 (default) — balanced performance
> 500 tags	Use MULTI_TAG_FAST + SESSION1 — optimised for density
Write / Lock	Always set EPC filter first — prevents accidental writes to wrong tags
Background	Call powerOff() in onPause() — saves significant battery
Read consistency	Retry up to 3x — RF conditions fluctuate; 1 failure is normal
Range vs Speed	Higher power = longer range but slower throughput — tune for use case
TID reading	Use embedded data method for TID — avoids second round-trip per tag

7.2 Common Issues & Solutions

Symptom	Likely Cause	Solution
No tags detected	Low RF power or wrong region	Increase power to 3300; set RG_PRC for India
Inconsistent reads	Interference / RF mode mismatch	Try RF mode 105; reduce power slightly
Write always fails	Tag locked or out of range	Check lock status; bring tag within 20cm
Connect fails	powerOn() not called first	Always call powerOn() before connectModule()
TID shows empty/null	EmdData flag not set	Set isReadTid=true and call setTagEmbededData()
Scanner not working	Wrong input mode or not powered	Call openScan(true); setInputMode(0)
Kill fails	Kill password = 00000000	Default kill pwd may be locked; set a non-zero one
Filter fails	EPC length not even hex chars	Ensure EPC string length is even (2 chars = 1 byte)
App crashes on start	Root permission not initialized	Call UHFSetting.initRoot(context) in onCreate()

7.3 Recommended Configuration for India

```
// ── India-Optimised Configuration ──────────────────
private void configureForIndia() {
    // Region: India uses 865-867 MHz (mapped to RG_PRC)
    UHFParamsOperator.getInstance()
```

```
        .setRegionConf(Reader.Region_Conf.RG_PRC);

    // RF Mode: 105 is stable and works well in India
    UHFParamsOperator.getInstance()
        .setGen2TagEncoding(105);

    // Session 1: Good for 10-500 tags
    UHFParamsOperator.getInstance().setGen2Session(1);

    // Target A->B: Standard cycle
    UHFParamsOperator.getInstance().setGen2Target(2);

    // Power: 33 dBm (maximum for longest range)
    Reader.AntPowerConf apcf = new Reader.AntPowerConf();
    apcf.antcnt = 1;
    Reader.AntPower ap = new Reader.AntPower();
    ap.antid = 1; ap.readPower = 3300; ap.writePower = 3300;
    apcf.Powers[0] = ap;
    UHFParamsOperator.getInstance().setAntPowerConf(apcf);
}
```

Chapter 8: Quick Reference Card

8.1 Connection Lifecycle

```
onResume():  powerOn() -> connectModule() -> configure()
onPause():   disconnectModule() -> powerOff()
```

8.2 Memory Bank Constants

Bank	Constant	Hex	Description
Reserved	BANK.Reserved	0x00	Access & Kill passwords (read/write with password)
EPC	BANK.EPC	0x01	Electronic Product Code — main identifier (writable)
TID	BANK.TID	0x02	Chip unique ID — factory programmed (read-only)
USER	BANK.USER	0x03	User-defined application data (writable)

8.3 Key API Method Signatures

```
// Connection
void        powerOn()
void        powerOff()
READER_ERR  connectModule(String uartPath, int antCount)
void        disconnectModule()

// Inventory
READER_ERR  startInventory(InventoryParams p, ReadListener rl, ReadExceptionListener rel)
void        stopInventory()

// Read
READER_ERR  getTagData(int ant, char bank, int startBlock,
            int blockCount, byte[] data, byte[] pwd, short timeout)

// Write
READER_ERR  writeTagData(int ant, char bank, int startBlock,
            byte[] data, int len, byte[] pwd, short timeout)
READER_ERR  writeTagEpcEx(int ant, byte[] epc, int len,
            byte[] pwd, short timeout)

// Lock & Kill
READER_ERR  lockTag(int ant, byte lockObj, short lockType,
            byte[] pwd, short timeout)
READER_ERR  killTag(int ant, byte[] killPwd, short timeout)

// Parameters
READER_ERR  setAntPowerConf(AntPowerConf conf)
READER_ERR  setRegionConf(Region_Conf region)
READER_ERR  setGen2Session(int session)
READER_ERR  setGen2Target(int target)
READER_ERR  setGen2TagEncoding(int rfMode)
READER_ERR  setTagFilter(TagFilter_ST filter)  // null = clear
READER_ERR  setTagEmbededData(EmbeddedData_ST edst)
```

8.4 RF Power Quick Reference

SeekBar Position	Power (dBm)	Internal Value	Approx Range
0	5 dBm	500	< 0.5 m
5	10 dBm	1000	~1 m
10	15 dBm	1500	~2 m
15	20 dBm	2000	~3-4 m
20	25 dBm	2500	~5-6 m
25	30 dBm	3000	~7-9 m
28	33 dBm	3300	Up to 10 m (MAX)

Appendix A: Complete API Reference

A.1 UHFEngine Methods

Method	Returns	Description
powerOn()	void	Powers on UHF RF module
powerOff()	void	Powers off UHF RF module
connectModule(String path, int antCnt)	READER_ERR	Establishes UART connection
disconnectModule()	void	Closes UART connection
startInventory(InventoryParams, ReadListener, ExceptionListener)	READER_ERR	Starts async tag scanning
stopInventory()	void	Stops ongoing inventory
getTagData(ant, bank, start, count, buf, pwd, timeout)	READER_ERR	Reads tag memory bank
writeTagData(ant, bank, start, data, len, pwd, timeout)	READER_ERR	Writes to tag memory bank
writeTagEpcEx(ant, epc, len, pwd, timeout)	READER_ERR	Writes new EPC to tag
lockTag(ant, lockObj, lockType, pwd, timeout)	READER_ERR	Locks tag memory bank
killTag(ant, killPwd, timeout)	READER_ERR	Permanently kills tag

A.2 UHFParamsOperator Methods

Method	Returns	Description
setAntPowerConf(AntPowerConf)	READER_ERR	Set read/write RF power
getAntPowerConf()	AntPowerConfResult	Get current RF power settings
setRegionConf(Region_Conf)	READER_ERR	Set frequency region
getRegionConf()	RegionConfResult	Get current region
setGen2Session(int)	READER_ERR	Set Gen2 session (0-3)
getGen2Session()	Gen2SessionResult	Get current session
setGen2Target(int)	READER_ERR	Set Gen2 target (0-3)
getGen2Target()	Gen2TargetResult	Get current target
setGen2TagEncoding(int)	READER_ERR	Set RF mode encoding
setTagFilter(TagFilter_ST)	READER_ERR	Set tag filter (null=clear)
setTagEmbededData(EmbeddedData_ST)	READER_ERR	Configure embedded data read

Appendix B: Supported Barcode Formats

B.1 1D Barcodes

Format	Description
Code 128	High-density alphanumeric — widely used in logistics
Code 39	Alphanumeric — automotive and defence applications
Code 93	Compact alphanumeric variant of Code 39
EAN-13	European Article Number — retail products globally
EAN-8	Compact EAN for small product packaging
UPC-A	Universal Product Code — North American retail
UPC-E	Compressed UPC for small items
Interleaved 2/5	Numeric only — warehousing and distribution
Codabar	Numeric — libraries, blood banks, FedEx
MSI/Plessey	Numeric — inventory and shelf labels

B.2 2D Barcodes

Format	Description
QR Code	Most common 2D — URLs, payments, product info
Data Matrix	Small footprint — PCBs, medical devices, small items
PDF417	Stacked linear — ID cards, transport tickets
Aztec Code	Transport ticketing — railway, airline boarding passes
MaxiCode	UPS postal sorting — hexagonal matrix
MicroQR	Compact QR variant for constrained spaces

Appendix C: Regulatory Information

The CX1500N UHF Handheld Reader operates in the UHF ISM (Industrial, Scientific, Medical) frequency band. The device has been designed for compliance with the following regulatory standards:

Certification / Regulation	Applicable Region	Notes
WPC / DoT India	India	865–867 MHz band; comply with Wireless Planning & Coordination Wing

Certification / Regulation	Applicable Region	Notes
EPC Gen2 v2	Global	ISO 18000-6C standard compliance
CE Marking	European Union	865–868 MHz ETSI EN 302 208
FCC Part 15	North America	902–928 MHz; operate within authorised power limits
BIS	India	Bureau of Indian Standards certification

Operators are responsible for ensuring compliance with local regulations regarding RF power limits and frequency usage. Identium Tech Solutions Pvt. Ltd. recommends consulting with local regulatory authorities before deployment in new geographies.

Identium Tech Solutions Private Limited
Plot No. 5, First Floor, Santnagar, East of Kailash, New Delhi – 110065, India
Ph: 011-47147839 / 7011001472 | Email: info@identium.in
Web: identium.in | indiarfidstore.com | rfidindia.in
GSTIN: 07AAGCI6275E1ZH