鸿陆 Hopeland

Turn on the value of data

Hopeland PDA Device Android

Development Guide

1.Overview

1.1 Introduction

For easier secondary development for programmers, we offer the function library written in Java and packaged into the standard JAR packet based on Android OS.

This Application Development Guide, has a comprehensive description of the corresponding technical specifications, application development instructions and precautions, UHF module, HF module, Barcode scanner module, PSAM module, the serial interface module Function Description and so on.

1.2 Applicable device models

This document lists the detailed API for all related devices, please refer to the following sheet.

Function	Applicable device type
Device connecting	CL7202G3/CL7202K3 series，HY820 series
Device configuration	CL7202G3/CL7202K3 series，HY820 series
UHF RFID	CL7202G3 /CL7202K3 series，HY820 series
HF RFID	CL7202G3 /CL7202K3 series
Barcode Scanner	CL7202G3 /CL7202K3 series，HY820 series
PSAM	CL7202G3 /CL7202K3 series
Serial port	CL7202G3 /CL7202K3 series
6C tag operation	CL7202G3 /CL7202K3 series，HY820 series
6B tag operation	CL7202G3 /CL720K3 series

Note: Since the HY820 series, you need to initialize the SDK in order to call the hardware resources correctly (see 6.8 Initialize the SDK).

Note: APP requires the following permission to be added(Since the HY820 series, you need to dynamically apply for permission before you can call the SDK initialization).

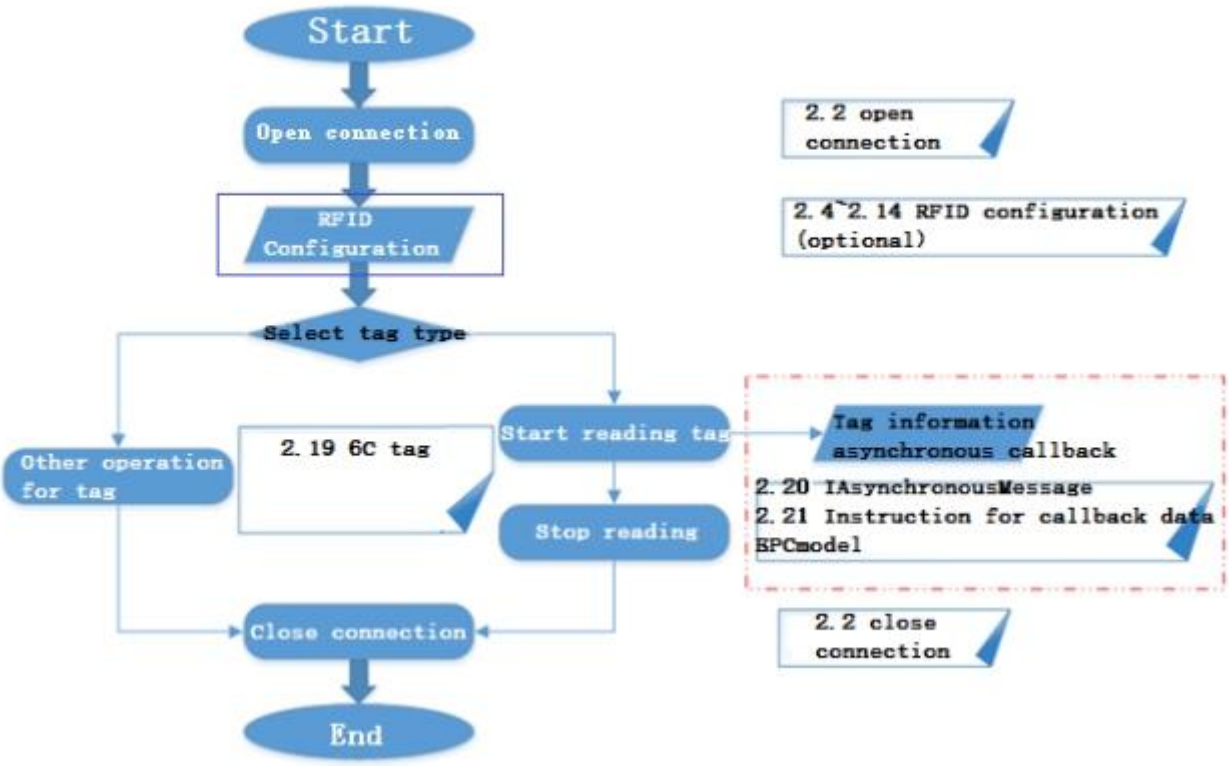
- android.permission.READ_PHONE_STATE

1.3 Copyright

2.UHF Functional module description

JAR packet and so libraries to be loaded(Or directly load pdasdk.aar):
JAR Packet：pdasdk.jar
SO library：libpda.so, libcepri_dev.so

2.1 UHF Development Process



Note: Since the HY820 series, you need to initialize the SDK in order to call the hardware resources correctly (see 6.8 Initialize the SDK).

Note: APP requires the following permission to be added(Since the HY820 series, you need to dynamically apply for permission before you can call the SDK initialization).

- android.permission.READ_PHONE_STATE

2.2 Open and Close connection

Open module

Packet	com.pda.rfid.uhf
Class	UHFReader._Config or UHFReader
Function	`public int OpenConnect(IAsynchronousMessage log)`
Parameter	**log:** data call-back interface, all the data read would be called back from this interface
Return	0: success, 1: failure
Description	1. **log:** data call-back interface, please refer to 2.20; 2. Example code `UHFReader._Config.OpenConnect(log)` 3. The built-in battery(Backup Power) of the K3A / K6 series is specifically designed to power the RFID module.Every time the K3A/K6 series call the read/write/lock/kill command, the SDK will detect the built-in battery level, and automatically switch to the main battery when the built-in battery is less than 15%, and automatically switch to the built-in battery when the built-in battery is greater than 30%. 4. If the K3A/K6 series is powered by the main battery and can only use a maximum RF power of 25dBm. 5. The HY820 series has only the main battery and the RF power can be set at will.

Packet	com.pda.rfid.uhf
Class	UHFReader._Config or UHFReader
Function	`public int OpenConnect(boolean usingBackupPower,IAsynchronousMessage log)`
Parameter	**usingBackupPower:** Whether to use built-in battery power supply **log:** data call-back interface, all the data read would be called back from this interface
Return	0: success, 1: failure
Description	1. **log:** data call-back interface, please refer to 2.20; 2. Example code `UHFReader._Config.OpenConnect(usingBackupPower, log)` 3. The built-in battery(Backup Power) of the K3A / K6 series is specifically designed to power the RFID module. PDA will not automatically switch batteries, the use of primary / built-in power supply is necessary to pay attention to the battery level, It is generally recommended to switch to the main battery when the built-in battery is less than 10%. To obtain the built-in battery power, refer to the Get Built-in Battery Power interface. 4. If the K3A/K6 series is powered by the main battery and can only use a maximum RF power of 25dBm.

	5. Do not use this interface for HY820 series.

Close module

Packet	com.pda.rfid.uhf
Class	UHFReader._Config or UHFReader
Function	**public void CloseConnect()**
Parameter	None
Return	Null
Description	1. call it to close UHF module and the module no longer powered. 2. example code UHFReader._Config.CloseConnect()

2.3 Stop instruction

Packet	com.pda.rfid.uhf
Class	UHFReader._Config or UHFReader
Function	**public String Stop()**
Parameter	None
Return	'0' indicates success, non '0' indicates failure
Description	1.This method will stop the reader from doing everything 2.Stop inventorying(read continually) using this method 3. example code UHFReader._Config.**Stop()**

2.4 Get Antenna Power

Packet	com.pda.rfid.uhf
Class	UHFReader._Config
Function	**public int GetANTPowerParam()**
Parameter	None
Return	RF output power, range is 0~30dBm, -1 indicates failure
Description	1. Get the RF output power of the RFID module 2. example code UHFReader._*Config*.**GetANTPowerParam()**

2.5 Set Antenna Power

Packet	com.pda.rfid.uhf
Class	UHFReader._Config
Function	**public int SetANTPowerParam(int antCount, int powerValue)**

Parameter	// antCount, number of antennas, antenna quantity of CL7202K is 2,HY820 is 1
	For HY820, passing value 1, for CL7202K, passing value 2
	// powerValue, power value to be set, value range 0-30dbm

Return	0 success, non 0 failure

Description	1. Set RF output power 2. K3A example code `UHFReader._Config.SetANTPowerParam(2,28)` 3. HY820 example code `UHFReader._Config.SetANTPowerParam(1,30)`

2.6 Get Baseband Parameters

Packet	com.pda.rfid.uhf
Class	UHFReader._Config
Function	`public String GetEPCBaseBandParam()`
Parameter	None

Return	Return data example: "255|4|0|2", returns empty ("), indicating a failed fetch.
	// basebandMode: EPCBaseband rate(0~255, 255 means AUTO)
	（0-Tari=25us, FM0, LHF=40KHz）
	（1-TTari=25us, Miller4, LHF=250KHz）
	（2-Tari=25us, Miller4, LHF=300KHz）
	（3-Tari=6.25us, FM0, LHF=400KHz）
	（255-Auto）
	// qValue: 0~15, the initial Q value used by the reader。
	// session: 0~3
	// searchType： searchType parameter(0 indicates inventory by Flag A only, 1 indicates inventory by Flag B only, 2 indicates inventory by Flag A and Flag B)

Description	1. UHF module baseband parameters 2. example code `UHFReader._Config.GetEPCBaseBandParam()`

Packet	com.pda.rfid.uhf
Class	UHF
Function	`public String GetBaseBandX()`
Parameter	

Return	Return data example:
	"1,00000000&2,00000000&3,00000000&4,00000000&5,00000000", return null ("), indicating acquisition failure.
	// The returned data is a set of optional configuration parameters, separated by '&', each optional parameter includes id and parameter, separated by ',', and the parameter is 4 bytes of data (returned in hexadecimal string format)
	// Parameter 1: Big-endian format composes U32
	bit3-bit0: rfu
	bit4: tag focus enable
	bit5: fast id enable
	bit15-bit6: rfu
	bit16: NXP fast id enable
	bit31-bit17: rfu
	// Parameter 2:
	Byte 1: maxQ
	Byte 2: minQ
	Byte 3: tmult
	Byte 4: bit 0 Dynamic start Q enable
	bit1: Force Loop enable
	// Parameter 3:

	Byte 1: Antenna switching mode. 0--Switch immediately without tags, 1--Running out of resistance time
	Byte 2: Number of retries (Switch immediately without tags mode)
	Byte 3-4: Big-endian format composes U16,max antenna resistance time (x10ms)
	// Parameter 4：
	Byte 1: Waiting time between antenna switching(x10ms)
	Byte 2: antenna switching sequence
	Byte 3: Antenna protection threshold (unit is dBm), set to 0 to disable protection.
	Byte 4: reserved
	// Parameter 5：
	Byte 1:LBT working mode
	0: disable
	1: listening only
	2: read tag after listening
	3: read tag after meeting RSSI
	Byte 2: RSSI maximum value
	Byte3-4: reserved

Description	1. Get the baseband expansion parameters of UHF module 2. example code `UHFReader.getInstance().GetBaseBandX()`

2.7 Set Baseband Parameters

Packet	com.pda.rfid.uhf
Class	UHFReader._Config
Function	`public int SetEPCBaseBandParam(int basebandMode,int qValue,int session,int searchType)`

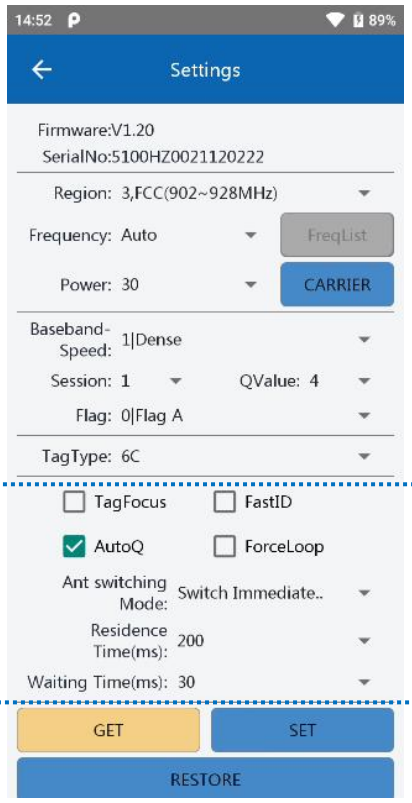
Parameter	// basebandMode: EPCBaseband rate(0~255, 255 means AUTO)
	（0-Tari=25us, FM0, LHF=40KHz）
	（1-TTari=25us, Miller4, LHF=250KHz）
	（2-Tari=25us, Miller4, LHF=300KHz）
	（3-Tari=6.25us, FM0, LHF=400KHz）
	（255-Auto）
	// qValue: 0~15, the initial Q value used by the reader。
	// session: 0~3
	// searchType： searchType parameter(0 indicates inventory by Flag A only, 1 indicates inventory by Flag B only, 2 indicates inventory by Flag A and Flag B)

Return	0 success, non 0 failure
Description	1. UHF module baseband parameters 2. example code `UHFReader._Config.SetEPCBaseBandParam(255,4,0,2)`

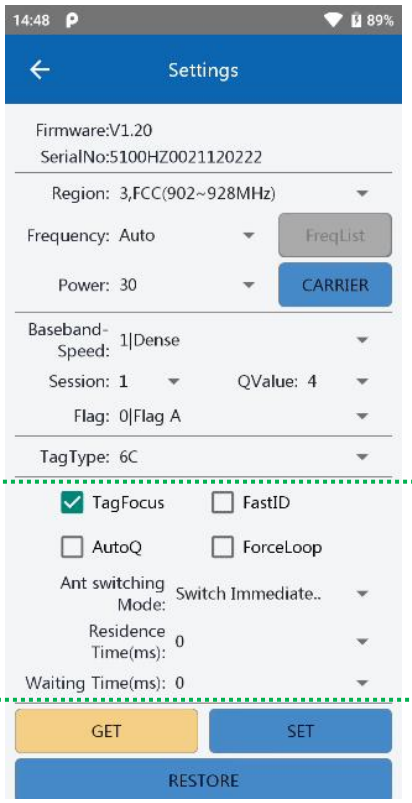
Packet	com.pda.rfid.uhf
Class	UHF
Function	`public String SetBaseBandX(String param)`
Parameter	Example of parameter data: “1,00000000&2,00000000” // The param is a set of optional configuration parameters, separated by ‘&’, each optional parameter includes id and parameter, separated by ‘,’, and the parameter is 4 bytes of data (returned in hexadecimal string format)

	// Parameter 1: Big-endian format composes U32
	bit3-bit0: rfu
	bit4: tag focus enable
	bit5: fast id enable
	bit15-bit6: rfu
	bit16: NXP fast id enable
	bit31-bit17: rfu
	// Parameter 2:
	Byte 1: maxQ
	Byte 2: minQ
	Byte 3: tmult
	Byte 4: bit 0 Dynamic start Q enable
	bit1: Force Loop enable
	// Parameter 3:
	Byte 1: Antenna switching mode. 0--Switch immediately without tags, 1--Running out of resistance time
	Byte 2: Number of retries (Switch immediately without tags mode)
	Byte 3-4: Big-endian format composes U16,max antenna resistance time (x10ms)
	// Parameter 4：
	Byte 1: Waiting time between antenna switching(x10ms)
	Byte 2: antenna switching sequence
	Byte 3: Antenna protection threshold (unit is dBm), set to 0 to disable protection.
	Byte 4: reserved
	// Parameter 5：
	Byte 1:LBT working mode
	0: disable
	1: listening only
	2: read tag after listening
	3: read tag after meeting RSSI
	Byte 2: RSSI maximum value
	Byte3-4: reserved

Return	0 success, non 0 failure
Description	1. This interface is to set the baseband extension parameters of UHF module 2. Modification of the baseband extension parameters of the UHF module may cause the handheld reader to fail to read tags. Please modify it carefully. If you need to modify it, please contact technical support. 3. example code `UHFReader.getInstance().SetBaseBandX("1,00000010")` "01,00000000&02,0F000401&03,00030014&04,03000000&05,00000000" //This parameter corresponds to the following settings.



"01,00000010&02,0F000400&03,00030000&04,00000000&05,00000000" /This parameter corresponds to the following settings



2.8 Get Reader Property

Packet	com.pda.rfid.uhf
Class	UHFReader._Config
Function	**public String GetReaderProperty()**
Parameter	None
Return	Return data example: Minimum transmit power | maximum transmit power | number of antennas | band list | list of RFID protocols returns empty (''), indicating a failed fetch.
Description	1. Output unit is dBm 2. example code `UHFReader._Config.GetReaderProperty()`

2.9 Get RF Frequency Band

Packet	com.pda.rfid.uhf
Class	UHFReader._Config
Function	**public int GetFrequency()**
Parameter	None
Return	-1, Fetch parameter failed 0, GB_920_to_925MHz 1, GB_840_to_845MHz 2, GB_920_to_925MHz_and_GB_840_to_845MHz 3, FCC_902_to_928MHz 4, ETSI_866_to_868MHz
Description	1. Fetch RF frequency band 2. example code `UHFReader._Config.GetFrequency()`

2.10 Set RF Frequency Band

Packet	com.pda.rfid.uhf
Class	UHFReader._Config
Function	**public int SetFrequency(int frequencyIndex)**
Parameter	// frequencyIndex: RF frequency band index 0, GB_920_to_925MHz 1, GB_840_to_845MHz 2, GB_920_to_925MHz_and_GB_840_to_845MHz 3, FCC_902_to_928MHz 4, ETSI_866_to_868MHz
Return	0 success, non 0 failure
Description	1. Set RF band frequency 2. example code `UHFReader._Config.SetFrequency(0)`

2.11 Get Tag upload parameters

Packet	com.pda.rfid.uhf
Class	UHFReader._Config
Function	**public String GetTagUpdateParam()**
Parameter	None

Return	Return data example: "20|90", returns empty ("), indicating a failed fetch.
	// repeatTimeFilter: Duplicate tag upload filter time (Unit: 10ms)
	// RSSIFilter: RSSI Filter

Description	1. Tag upload parameter query 2. example code `UHFReader._Config.GetTagUpdateParam()`

2.12 Tag upload parameter setting

Packet	com.pda.rfid.uhf
Class	UHFReader._Config
Function	**public int SetTagUpdateParam(int repeatTimeFilter, int RSSIFilter)**
Parameter	// repeatTimeFilter: Repeat tag upload filter time (Unit: 10ms) // RSSIFilter: RSSI Filter
Return	0 success, non 0 failure
Description	1. Set tag upload parameter 2. example code `UHFReader._Config.SetTagUpdateParam(2,0)` //Set the repeat tag upload time to 20ms, that is, the same tag is read multiple times within 20ms, and the reader uploads it only once. RSSI filtering is set to 0, that is, RSSI is not required for filtering.

2.13 Get Auto Idle Mode

Packet	com.pda.rfid.uhf
Class	UHFReader._Config
Function	**public String GetReaderAutoSleepParam()**
Parameter	None
Return	Return data example: "1|20", returns empty (") or "Failed to get!" , indicating a failed fetch. 1: Auto idle mode enabled // 0-- turn off automatically idle mode;1- enables automatically idle mode. 20: the time remaining in the idle state, 0~65535, time unit: 10ms.
Description	1. Automatically idle mode query 2. Example code `UHFReader._Config.GetReaderAutoSleepParam()`

2.14 Set Auto Idle Mode

Packet	com.pda.rfid.uhf
Class	UHFReader._Config
Function	**public int SetReaderAutoSleepParam(boolean** isOpen,**int** time**)**
Parameter	// isOpen: Set automatic idle mode // time： idle time, unit is 10ms
Return	0 success, non 0 failure
Description	1. Set automatically idle mode 2. Example code UHFReader._*Config*.**SetReaderAutoSleepParam(**true,20**)** // Set Auto idle setting, set True, time 20*10ms, means when the reader doesn't detect any tag in 3 round inventories (about 20ms), it will rest 20*10=200ms, then back to read tag again.

2.15 Get Baseband Software Version

Packet	com.pda.rfid.uhf
Class	UHFReader._Config
Function	**public String GetReaderBaseBandSoftVersion()**
Parameter	None
Return	Baseband software version, like "V2.1.5"
Description	1. Baseband software version Query 2. Example code UHFReader._*Config*.**GetReaderBaseBandSoftVersion()**

2.16 Transmitting carrier

Packet	com.pda.rfid.uhf
Class	UHFReader._Config
Function	**public int Set_0101_00(byte antenna_Port, byte frequency_Number)**
Parameter	// antenna_Port： antenna no.(1,antenna 1; 2,antenna 2) // frequency_Number:The index value of the frequency of the current band (You can always fill in 0)
Return	0 success, non 0 failure
Description	1. Transmitting carrier.(used to detect standing-wave of antenna port) 2. Example code UHFReader._*Config*.**Set_0101_00(**(byte)1,(byte)0**)**

2.17 Detect standing-wave of antenna port

Packet	com.pda.rfid.uhf
Class	UHFReader._Config
Function	`public String Get_0101_05()`
Parameter	None
Return	0 success, non 0 failure
Description	1. Standing wave detection (for antenna detection) 2. Example code `UHFReader._Config.Get_0101_05()`

2.18 Restore Factory Settings

Packet	com.pda.rfid.uhf
Class	UHFReader._Config
Function	`public int SetReaderRestoreFactory()`
Parameter	None
Return	0 success, non 0 failure
Description	1. Restore all settings to factory state. 2. Example code `UHFReader._Config.SetReaderRestoreFactory()`

2.19 6C Tag Operation

2.19.1 Read tag

Packet	com.pda.rfid.uhf
Class	UHFReader._Tag6C

Function 1	`int GetEPC(int antNum, int readType)`
	// read EPC only
	// antNum: Antenna number enumeration.
	For example: Specify antenna 1 and antenna 2 to working at same time: 3
	Attention!!! In CL7202K3 series device, we designed 2-UHF antennas working mode specially, and for best performance, please fix both antennas in working at same time, so need fix the antNum:3.
	In HY820 only 1 UHF antenna, so antNum:1
	// readType: reading type enumeration(0 -- single or 1 -- cycle continuous)

Function 2	`int GetEPC(int antNum, int readType, String accessPassword)` // accessPassword: tag access password

Function 3	`int GetEPC_MatchEPC(int antNum, int readType, String sEPC)`
	// match EPC to read EPC
	// sEPC : matched EPC value (Hexadecimal string)

Function 4	`int GetEPC_MatchEPC(int antNum, int readType, String sEPC, int matchWordStartIndex)`
	// match EPC to read EPC
	// matchWordStartIndex:Starting index of the tag memory bank to be matched (unit: Word)

Function 5	`int GetEPC_MatchEPC(int antNum, int readType, String sEPC, int matchWordStartIndex, String accessPassword)` // accessPassword: tag access password

Function 6	`int GetEPC_MatchTID(int antNum, int readType, String sTID)`
	// match TID to read EPC
	// sTID: matched TID value (Hexadecimal string)

Function 7	`int GetEPC_MatchTID(int antNum, int readType, String sTID, int matchWordStartIndex)`
	// match TID to read EPC
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)

Function 8	`int GetEPC_MatchTID(int antNum, int readType, String sTID, int matchWordStartIndex, String accessPassword)` // accessPassword: tag access password
Function 9	`int GetEPC_TID(int antNum, int readType)` // simultaneously read EPC and TID
Function 10	`int GetEPC_TID_MatchEPC(int antNum, int readType, String sEPC)` // match EPC to simultaneously read EPC and TID
Function 11	`int GetEPC_TID_MatchEPC(int antNum, int readType, String sEPC, int matchWordStartIndex)` // match EPC to simultaneously read EPC and TID
Function 12	`int GetEPC_TID_MatchEPC(int antNum, int readType, String sEPC, int matchWordStartIndex, String accessPassword)` // accessPassword: tag access password
Function 13	`int GetEPC_TID_MatchTID(int antNum, int readType, String sTID)` // match TID to simultaneously read EPC and TID
Function 14	`int GetEPC_TID_MatchTID(int antNum, int readType, String sTID, int matchWordStartIndex)` // match TID to simultaneously read EPC and TID
Function 15	`int GetEPC_TID_MatchTID(int antNum, int readType, String sTID, int matchWordStartIndex, String accessPassword)` // accessPassword: tag access password

Function 16	`int GetEPC_TID_UserData(int antNum, int readType, int readStart, int readLen)`
	// simultaneously read EPC, TID and UserData
	(Attention!!! In regular applications, usually no need call this function to read 3

	area data at same time. Or else will lower working efficiency, and need longer time to finish one time reading.)
	// readStart: Starting index for reading UserData area
	// readLen: Data length for reading UserData area (unit: Word)

Function 17	`int GetEPC_TID_UserData_MatchEPC(int antNum, int readType, int readStart, int readLen, String sEPC)` // match EPC to simultaneously read EPC,TID and UserData
Function 18	`int GetEPC_TID_UserData_MatchEPC(int antNum, int readType,int readStart, int readLen, String sEPC, int matchWordStartIndex)` // match EPC to simultaneously read EPC,TID and UserData
Function 19	`int GetEPC_TID_UserData_MatchEPC(int antNum, int readType, int readStart, int readLen, String sEPC, int matchWordStartIndex, String accessPassword)` // accessPassword: tag access password
Function 20	`int GetEPC_TID_UserData_MatchTID(int antNum, int readType, int readStart, int readLen, String sTID)` // match TID, simultaneously read EPC, TID and UserData
Function 21	`int GetEPC_TID_UserData_MatchTID(int antNum, int readType,int readStart, int readLen, String sTID, int matchWordStartIndex)` // match TID to simultaneously read EPC, TID and UserData
Function 22	`int GetEPC_TID_UserData_MatchTID(int antNum, int readType, int readStart, int readLen, String sTID, int matchWordStartIndex, String accessPassword)` // accessPassword: tag access password

Function 23	`int GetEPC_EpcData(int antNum, int readType, int readStart, int readLen)`
	//read EPC and EPCData
	//readStart: The starting index for reading EPCData
	//readLen: Data length for reading EPCData (unit: Word)

Function 24	`int GetEPC_EpcData_MathcEPCData(int antNum, int readType, int readStart, int readLen, String matchEPCData, int matchBitStartIndex)`
	//match EPCData to simultaneously read EPC and EPCData
	// readStart: The starting index for reading EPCData
	// readLen: Data length for reading EPCData (unit: Word)
	// matchEPCData: Data to be matched
	// matchBitStartIndex:The starting index of the matching data(unit: Bit)

Function 25	`int GetEPC_TID_EpcData(int antNum, int readType, int readStart, int readLen)`
	// simultaneously read EPC, TID and EPCData
	// readStart: The starting index for reading EPCData
	// readLen: Data length for reading EPCData (unit: Word)

Function 26	`int GetEPC_TID_EpcData_MathcEPCData(int antNum, int readType, int readStart, int readLen, String matchEPCData, int`

	matchBitStartIndex)
	//match EPCData to simultaneously read EPC ,TID and EPCData
	// readStart: The starting index for reading EPCData
	// readLen: Data length for reading EPCData (unit: Word)
	// matchEPCData: Data to be matched
	// matchBitStartIndex:The starting index of the matching data(unit: Bit)

Function 27	**public int GetEPC_MatchUserData(int antNo, int readType, String sMatchWord,int matchWordStartIndex, String accessPassword)**
	//match UserData to read EPC
	// sMatchWord: Data to be matched
	// matchWordStartIndex:Starting index of the tag memory bank to be matched (unit: Word)
	//accessPassword: tag access password

Function 28	**public int GetEPC_TID_MatchUserData(int antNo, int readType, String sMacthword, int matchWordStartIndex, String accessPassword) ;**
	// match UserData to simultaneously read EPC and TID
	// sMatchWord: Data to be matched
	// matchWordStartIndex:Starting index of the tag memory bank to be matched (unit: Word)
	//accessPassword: tag access password

Function 29	**public int GetEPC_TID_UserData_MatchUserData(int antNo, int readType,int readStart, int readLen, String sMatchWord, int matchWordStartIndex,String accessPassword)**
	// match UserData to simultaneously read EPC ,TID and UserData
	// readStart: Starting index for reading UserData area
	// readLen: Data length for reading UserData area (unit: Word)
	// sMatchWord: Data to be matched
	// matchWordStartIndex:Starting index of the tag memory bank to be matched (unit: Word)
	//accessPassword: tag access password

Function 30	**public int GetRFMicron_Temperature(int antNo, int readType)**
	// Read RFMicron S3 temperature tag.
	// After reading the tag data, convert it to temperature value by the following function.
	`public float RFMicron_ConvertTemperature(EPCModel model)`

Function 31	**public int GetCAB_Temperature(int antNo, int readType)**
	// Read CAB temperature tag.
	// After reading the tag data, convert it to temperature value by the following function.
	`public float CAB_ConvertTemperature(EPCModel model)`

Function 32	**public int GetEM_Temperature(int antNo, int readType)**
	// Read EM temperature tag.
	// After reading the tag data, convert it to temperature value by the following

	function. `public float EM_ConvertTemperature(EPCModel model)`

Function 33	**`public int GetEPC_G2V2(int antNo, int readType, G2V2AuthenticateModel g2v2, String accessPassword)`**
	// read G2V2 data
	// g2v2: G2V2 Authenticate parameters
	//accessPassword: tag access password

Function 34	**`public int GetEPC_G2V2(int antNo, int readType, G2V2AuthenticateModel g2v2)`**
	// read G2V2 data
	// g2v2: G2V2 Authenticate parameters

Function 35	**`public int GetEPC_G2V2_MatchEPC(int antNo, int readType, G2V2AuthenticateModel g2v2,String sEPC, int matchWordStartIndex, String accessPassword)`**
	// match EPC to read G2V2 data
	// g2v2: G2V2 Authenticate parameters
	/ sEPC: Data to be matched
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)
	//accessPassword: tag access password

Function 36	**`public int GetEPC_G2V2_MatchEPC(int antNo, int readType, G2V2AuthenticateModel g2v2, String sEPC, int matchWordStartIndex)`**
	// match EPC to read G2V2 data
	// g2v2: G2V2 Authenticate parameters
	/ sEPC: Data to be matched
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)

Function 37	**`public int GetEPC_G2V2_MatchTID(int antNo, int readType, G2V2AuthenticateModel g2v2, String sTID, int matchWordStartIndex, String accessPassword)`**
	// match TID to read G2V2 data
	// g2v2: G2V2 Authenticate parameters
	/ sTID: Data to be matched
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)
	//accessPassword: tag access password

Function 38	**`public int GetEPC_G2V2_MatchTID(int antNo, int readType, G2V2AuthenticateModel g2v2, String sTID, int matchWordStartIndex)`**
	// match TID to read G2V2 data
	// g2v2: G2V2 Authenticate parameters
	/ sTID: Data to be matched
	// matchWordStartIndex: Starting index of the tag memory bank to be matched

	(unit: Word)

Function 39	**public int GetLTU_Temperature(int antNo, int readType)**
	// read Johar LTU31/32 temperature tag
	// After reading the tag data, the following interface must be called in the callback to check the validity of the data
	public boolean LTU_VerifyTemperature(EPCModel model)
	// After reading the valid tag data, use the following interface to convert to temperature value
	public float LTU_ConvertTemperature(EPCModel model)

Function 40	**public int GetEPC_ReservedData(int antNum, int readType, int readStart, int readLen)**
	//read EPC and reserved data
	// antNum: Antenna number enumeration.
	// readType: reading type enumeration(0 -- single or 1 -- cycle continuous)
	// readStart: Reserved area read tart address (0 or 2,unit is word)
	// readLen: Reserved area read length (max. 4 word)

Function 41	**public int GetEPC_ReservedData(int antNum, int readType, int readStart, int readLen, String accessPassword)** // accessPassword: tag access password

Function 42	**public int GetEPC_ReservedData_MacthEPC(int antNum, int readType, int readStart, int readLen, String sEPC)**
	// match EPC to read EPC and reserved data
	// sEPC: matched EPC value (Hexadecimal string)

Function 43	**public int GetEPC_ReservedData_MacthEPC(int antNum, int readType, int readStart, int readLen, String sEPC, String accessPassword)** // accessPassword: tag access password

Function 44	**public int GetEPC_ReservedData_MacthTID(int antNum, int readType, int readStart, int readLen, String sTID, int matchWordStartIndex)**
	// match TID to simultaneously read EPC, reserved data
	// sTID: matched TID value (Hexadecimal string)
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)

Function 45	**public int GetEPC_ReservedData_MacthTID(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String sTID, int matchWordStartIndex, String accessPassword)** // accessPassword: tag access password

Function 46	**public int GetEPC_ LightKX2005X (int antNo, int readType, String accessPassword)**
	//Light up the KX2005X chip tag
	//accessPassword : tag access password

Function 47	**public int GetEPC_ LightKX2005X (int antNo, int readType)**

	//Light up the KX2005X chip tag

Function 48	**public int GetEPC_ LightKX2005X _MatchEPC(int antNo, int readType,String sEPC, int matchWordStartIndex, String accessPassword)**
	//Matching EPC to light up the KX2005X chip tag
	/ sEPC: matched EPC value (Hexadecimal string)
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)
	//accessPassword: tag access password

Function 49	**public int GetEPC_ LightKX2005X _MatchEPC(int antNo, int readType, String sEPC, int matchWordStartIndex)**
	//Matching EPC to light up the KX2005X chip tag
	/ sEPC: matched EPC value (Hexadecimal string)
	// matchWordStartIndex:Starting index of the tag memory bank to be matched (unit: Word)

Function 50	**public int GetEPC_ LightKX2005X _MatchTID(int antNo, int readType,String sTID, int matchWordStartIndex, String accessPassword)**
	//Matching TID to light up the KX2005X chip tag
	/ sTID: matched TID value (Hexadecimal string)
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)
	//accessPassword: tag access password

Function 51	**public int GetEPC_ LightKX2005X _MatchTID(int antNo, int readType, String sTID, int matchWordStartIndex)**
	//Matching TID to light up the KX2005X chip tag
	/ sTID: matched TID value (Hexadecimal string)
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)

Parameter	Check each functions description
Return	'0' indicates success, no '0' indicates failure, error codes refer to "Appendix A"
Description	1. Detailed returned value , check appendix 2. Stop cycle reading please use the "stop" command 3. Example code `UHFReader._Config.GetEPC(3,1)` // For K3A/K6, read EPC continuously until receiving stop command 4. Example code `UHFReader._Config.GetEPC_TID(1,1)`//For HY820, read EPC and TID continuously until receiving stop command 5. Example code `UHFReader._Config.GetEPC_TID_UserData(1,1,0,6)` //For HY820, read EPC , TID and UserData continuously until receiving stop command, starting index for reading UserData area is 0, data length for reading UserData area is 6,unit is Word

2.19.2Write tag

2.18.2.1 Write EPC area

Packet	com.pda.rfid.uhf
Class	UHFReader._Tag6C

Function 1	**int WriteEPC(int antNum, int sWriteData)**
	// antNum: Antenna number enumerate
	// sWriteData: data to-be-writen (Hexadecimal string)
	Simultaneously appoint antenna1 and antenna2 working, antNum: 3
	Attention!!! In CL7202K3 series device, we designed 2-UHF antenna working mode specially, and for best performance, please fix both antennas in working at same time, so need fix the antNum:3.
	In HY820 only 1 UHF antenna, so antNum:1

Function 2	**int WriteEPC_MatchEPC(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex)**
	// match EPC to write EPC area
	// sMatchData: EPC data to-be-matched
	// matchWordStartIndex :Starting index of the tag memory bank to be matched (unit: Word)

Function 3	**int WriteEPC_MatchEPC(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)**
	// match EPC to write EPC area
	// accessPassword tag access password

Function 4	**int WriteEPC_MatchTID(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex)**
	// match TID to write EPC area
	// sMatchData: TID data to-be-matched
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)

Function 5	**int WriteEPC_MatchTID(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)** // match TID to write EPC area
Parameter	Check each function description
Return	'0' indicates success, no '0' indicates failure, error codes refer to "Appendix A"
Description	1. It is recommended to write tag data using matching ID to write, that is, to use "function 4" and "function 5". 2. Detailed returned value see appendix statement 3. Example code `UHFReader._Config.WriteEPC_MatchTID(1,"12345678","E2003412012CFB000B26F75C",0)` // For HY820, write new EPC data "12345678" to the EPC memory bank of the tag whose TID is "E2003412012CFB000B26F75C" `UHFReader._Config.WriteEPC_MatchTID(3,"12345678","E200341201`

	2CFB000B26F75C",0) // For K3A/k6, write new EPC data "12345678" to the EPC memory bank of the tag whose TID is "E2003412012CFB000B26F75C"

2.18.2.2 Write Userdata area

Packet	com.pda.rfid.uhf
Class	UHFReader._Tag6C

Function 1	**int WriteUserData(int antNum, String sWriteData)**
	// antNum: Antenna number enumerate
	// sWriteData: data to-be-writen (Hexadecimal string)
	
	Simultaneously appoint antenna1 and antenna2 working example: 3
	
	Attention!!! In CL7202K3 series device, we designed 2-UHF antenna working mode specially, and for best performance, please fix both antennas in working at same time, so need fix the antNum:3. In HY820 only 1 UHF antenna, so antNum:1

Function 2	**int WriteUserData_MatchEPC(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex)**
	// match EPC to write UserData area
	// sMatchData: EPC data to-be-matched (Hexadecimal string)
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)

Function 3	**int WriteUserData_MatchEPC(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)**
	// match EPC to write UserData area
	// accessPassword: tag access password

Function 4	**int WriteUserData_MatchTID(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex)**
	// match TID to write UserData area
	// sMatchData: TID data to-be-matched (Hexadecimal string)
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)

Function 5	**int WriteUserData_MatchTID(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)** // match TID to write UserData area

Function 6	**int WriteUserData_MatchTID(int antNum, String sWriteData, int writeWordStartIndex, String sMatchData, int matchWordStartIndex, String accessPassword)**
	// match TID to write UserData area
	// writeWordStartIndex: Write user data word offset

Parameter	See each function description
Return	'0' indicates success, no '0' indicates failure, error codes refer to "Appendix A"
Description	1. Suggest write tag data by match ID namely use "function 4" and "function 5"

	2. Detailed returned value see appendix statement 3. Example code `UHFReader._Config.WriteUserData_MatchTID(1,"ABCD4321","E2003412012CFB000B26F75C",0)` //For HY820, write user data "ABCD4321" to the User memory bank of the tag whose TID is "E2003412012CFB000B26F75C",starting index of user memory bank to be written is 0 `UHFReader._Config.WriteUserData_MatchTID(3,"ABCD4321","E2003412012CFB000B26F75C",0)` //For K3A/k6, write user data "ABCD4321" to the User memory bank of the tag whose TID is "E2003412012CFB000B26F75C",starting index of user memory bank to be written is 0

2.18.2.3 Write Password area

Packet	com.pda.rfid.uhf
Class	UHFReader._Tag6C

Function 1	`int WriteAccessPassWord(int antNum, String sWriteData)`
	// write tag access password
	// sWriteData: password content (8bit Hexadecimal string)

Function 2	`int WriteAccessPassWord(int antNum, String sWriteData, String accessPassword)`
	// write tag access password
	// accessPassword: original tag access password（8bit Hexadecimal string）

Function 3	`int WriteAccessPassWord_MatchTID(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)`
	// match TID to write tag access password
	// sMatchData: TID data to-be-matched
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)

Function 4	`int WriteDestroyPassWord(int antNum, String sWriteData)` // write tag destroy password

Function 5	`int WriteDestroyPassWord(int antNum, String sWriteData, String accessPassword)`
	// write tag Kill password
	// accessPassword: tag access password（8bit Hexadecimal string）

Function 6	`int WriteDestroyPassWord_MatchTID(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)`
	// match TID to write tag Kill password
	// sMatchData: TID data to-be-matched
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)

Parameter	Check each function description

Return	'0' indicates success, no '0' indicates failure, error codes refer to "Appendix A"
Description	Example code `UHFReader._Config.WriteAccessPassWord_MatchTID(1,"ABCD4321","E2003412012CFB000B26F75C",0,"00000000")` // For HY820, match TID "E2003412012CFB000B26F75C" to write new access password "ABCD4321" to this matched tag with the original access password "00000000" `UHFReader._Config.WriteAccessPassWord_MatchTID(3,"ABCD4321","E2003412012CFB000B26F75C",0,"00000000")` // For K3A/k6, match TID "E2003412012CFB000B26F75C" to write new access password "ABCD4321" to this matched tag with the original access password "00000000"

2.19.3Lock tag

Packet	com.pda.rfid.uhf
Class	UHFReader._Tag6C

Function 1	`int Lock(int antNum, int lockArea, int lockType)`
	// antNum: antenna number. For K3A/K6 it is 3, for HY820 it is 1.
	// lockArea: "0"- killing password area, "1"- access password area, "2"- EPC area, "3"-TID area, "4"- UserData area
	// lockType: "0"-unlock; "1"-lock; "2"-permanent unlock; "3"-permanent lock

Function 2	`int Lock_MatchEPC(int antNum, int lockArea, int lockType, String sMatchData, int matchWordStartIndex)`
	// sMatchData: EPC data to-be-matched (Hexadecimal string)
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)

Function 3	`int Lock_MatchEPC(int antNum, int lockArea, int lockType, String sMatchData, int matchWordStartIndex, String accessPassword)` // accessPassword: tag access password

Function 4	`int Lock_MatchTID(int antNum, int lockArea, int lockType, String sMatchData, int matchWordStartIndex)`
	// sMatchData: TID data to-be-matched (Hexadecimal character string)
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)

Function 5	`int Lock_MatchTID(int antNum, int lockArea, int lockType, String sMatchData, int matchWordStartIndex, String accessPassword)` // accessPassword: tag access password
Parameter	Check above method description
Return	'0' indicates success, no '0' indicates failure, "Appendix A"
Description	

2.19.4 Kill tag

Packet	com.pda.rfid.uhf
Class	UHFReader._Tag6C

Function 1	**int Destroy(int antNum, String destroyPassword)**
	// antNum: antenna number
	// destroyPassword: Kill password(Hexadecimal string)

Function 2	**int Destroy_MatchEPC(int antNum, String destroyPassword, String sMatchData, int matchWordStartIndex)**
	// sMatchData: EPC data to-be-matched (Hexadecimal string)
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)

Function 3	**int Destroy_MatchTID(int antNum, String destroyPassword, String sMatchData, int matchWordStartIndex)**
	// sMatchData: TID data to-be-matched (Hexadecimal string)
	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)

Parameter	Check above method description
Return	'0' indicates success, no '0' indicates failure, "Appendix A"
Description	

2.20 6B Tag Operation

2.20.1Read tag

Packet	com.pda.rfid.uhf
Class	UHF
Function	**public abstract String Get6B(String param)**
Parameters	Param: antNum | readType -- 0 , single or 1, cycle continuous|Read contents -- 0,read 6B TID only or 1,read 6B TID+ user data or 2,read user data only | 1,user data read parameter -- Byte 0： user data start byte address , Byte 1： user data byte length & 2,The TID code of a 6B tag to be matched. Example "1|1|1|1,000F"
Return	'0' indicates success, not '0' indicates failure
Description	1. Read 6B tag 2. Example code. UHF *CLReader* = UHFReader.getUHFInstance(); *CLReader*.**Get6B(**"1"+"|1" + "|1" + "|" + "1,000F"**)** // use ant 1 to continuous read TID + user data, the start address of the user data to be read is 00, length is 0F.

2.20.2 Write tag

Packet	com.pda.rfid.uhf
Class	UHF
Function	**public abstract String Write6B(String param)**
Parameters	Param: antNum |The TID code of a 6B tag to be written|start address|content to be written
Return	'0' indicates success, not '0' indicates failure
Description	Write a 6B tag

2.20.3 Lock tag

Packet	com.pda.rfid.uhf
Class	UHF
Function	**public abstract String Lock6B(String param)**
Parameters	Param: antNum|The TID code of a 6B tag to be locked|the address to be locked
Return	'0' indicates success, not '0' indicates failure
Description	Lock a 6B tag

2.20.4 Get Lock Status

Packet	com.pda.rfid.uhf
Class	UHF
Function	**public abstract String GetLock6B(String param)**
Parameters	Param:antNum|The TID code of a 6B tag that locked|the address to be checked
Return	'0' indicates success, not '0' indicates failure
Description	Get lock status of a 6B tag

2.21 Callback interface IAsynchronousMessage

```
// asynchronous callback interface
public interface IAsynchronousMessage
{
    // callback of output tags data -- all the tags data would be called back from this method(very important)
    void OutPutEPC(EPCModel model);
}
```

2.22 Call-back data "EPCModel" Field Description

Field	Description

_TagType	Tag type,"6C","6B","GB" **(Notes: usually now most tags for application only under ISO18000-6C, and our handheld device RFID function closed 6B,and GB-new chinese standard for international market, so you only follow above 6C tag functions enough for your programming.)**
_EPC	EPC data, hex string
_PC	PC value
_ANT_NUM	Antenna Number
_RSSI	RSSI value
_Result	tag data read result
_TID	TID data, hex string
_UserData	User area data, hex string
_TagetData	Reserved area data, hex string

2.23 Sample Code

The simple call sample code is as follows (see the sample project for details):

```
package com.example.myapplication;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;

import android.Manifest;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.util.Log;
import android.view.KeyEvent;
import android.view.View;

import com.pda.rfid.EPCModel;
import com.pda.rfid.IAsynchronousMessage;
import com.pda.rfid.uhf.UHF;
import com.pda.rfid.uhf.UHFReader;
import com.port.Adapt;

import java.util.logging.Logger;

public class MainActivity extends AppCompatActivity implements IAsynchronousMessage {

    private static final String TAG = "Demo";
    private static final int REQUEST_READ_PHONE_STATE = 1;

    private boolean isOpened = false;
    private boolean isReading = false;
```

```
    private void initView() {
        Adapt.init(this);
        isOpened = UHFReader.getUHFInstance().OpenConnect(this);
        if (!isOpened) {
            Log.d(TAG, "open UHF failed!");
            // TODO failed opend UHF
        }

        // Set base band auto mode, q=1, session=1, flag = 0 flagA
        UHFReader._Config.SetEPCBaseBandParam(255, 0, 1, 0);
        // set ant 1 power to 20dBm
        UHFReader._Config.SetANTPowerParam(1, 20);
    }
    private void checkPermission() {
        //
        if (ActivityCompat.checkSelfPermission(this, Manifest.permission.READ_PHONE_STATE)
                != PackageManager.PERMISSION_GRANTED) {
            //request permission
            ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.READ_PHONE_STATE}, REQUEST_READ_PHONE_STATE);
        } else {
            initView();
        }
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        checkPermission();
    }

    @Override
    protected void onDestroy() {
        UHFReader.getUHFInstance().CloseConnect();
        super.onDestroy();
    }

    // read button onClick handle
    public void onRead(View v) {

        if (!isOpened) {
            return ;
```

```
        }
        if (isReading) {
            return ;
        }
        // start reading 6C tags using ant 1 in cycle continuous reading mode
        isReading = UHFReader._Tag6C.GetEPC(1, 1) == 0;
    }

    // stop button onClick handle
    public void onStop(View v) {
        if (!isOpened) {
            return ;
        }
        if (!isReading) {
            return;
        }
        UHFReader.getUHFInstance().Stop();
        isReading = false;
    }

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        Log.d(TAG, "onKeyDown keyCode = " + keyCode);
        if ((Adapt.DEVICE_TYPE_HY820 == Adapt.getDeviceType() && (keyCode == KeyEvent.KEYCODE_F9
/* RFID Handle button*/ || keyCode == 285  /* Left shortcut*/ || keyCode == 286  /* Right shortcut*/))
                || ((Adapt.getSN().startsWith("K3")) && (keyCode == KeyEvent.KEYCODE_F1 || keyCode
== KeyEvent.KEYCODE_F5))
                || ((Adapt.getSN().startsWith("K6")) && (keyCode == KeyEvent.KEYCODE_F1 || keyCode
== KeyEvent.KEYCODE_F5))) { // Press the handle button
            onRead(null);
        }
        return super.onKeyDown(keyCode, event);
    }

    @Override
    public boolean onKeyUp(int keyCode, KeyEvent event) {
        Log.d(TAG, "onKeyUp keyCode = " + keyCode);
        if ((Adapt.DEVICE_TYPE_HY820 == Adapt.getDeviceType() && (keyCode == KeyEvent.KEYCODE_F9
/* RFID Handle button*/ || keyCode == 285  /* Left shortcut*/ || keyCode == 286  /* Right shortcut*/))
                || ((Adapt.getSN().startsWith("K3")) && (keyCode == KeyEvent.KEYCODE_F1 || keyCode
== KeyEvent.KEYCODE_F5))
```

```
                || ((Adapt.getSN().startsWith("K6")) && (keyCode == KeyEvent.KEYCODE_F1 || keyCode
== KeyEvent.KEYCODE_F5))) { // release the handle button
            onStop(null);
        }
        return super.onKeyUp(keyCode, event);
    }

    @Override
    public void OutPutEPC(EPCModel epcModel) {
        Log.d(TAG, " EPC: " + epcModel._EPC
                            + " TID: " + epcModel._TID
                            + " UserData:" + epcModel._UserData);
        // TODO save the data and process in other thread
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
@NonNull int[] grantResults) {
        if (requestCode == REQUEST_READ_PHONE_STATE) {
            initView();
        }
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}
```

3.HF RFID operation

JAR packet and so library need to be loaded:
JAR Packet：pdasdk.jar
SO library：libpda.so,libcepri_dev.so

3.1 Open module

Packet	com.pda.rfid.hf
Class	HF
Function	`public abstract bool OpenConnect(IAsynchronousMessage log)`
Parameter	log: Data callback interface, all the tag data will be callback from the interface object.
Return	True: successful; False: failed
Description	1, log Data callback interface, please refer to the “callback interface description” of the specific understanding. 2, such as : `HF hfReader = HFReader.getHFInstance();` `hfReader.OpenConnect(this)` 3, refer to the sample code.

3.2 Close module

Packet	com.pda.rfid.hf
Class	HF
Function	`public abstract void CloseConnect()`
Parameter	None
Return	None
Description	1, call this method will turn off the HF module, and the module is no longer powered. 2, such as `hfReader.CloseConnect()`

3.3 Get module information

Packet	com.pda.rfid.hf
Class	HF
Function	`public abstract String GetReaderInformation ()`
Parameter	None

Return	Application processor software version | HF reader name | power on time
Description	None

3.4 Configure serial parameter

Packet	com.pda.rfid.hf
Class	HF
Function	public abstract String SetReaderSerialPortParam (String baud)
Parameter	(Baudrate) 0,9600 bps|1,19200 bps|2,115200 bps|3,230400 bps|4,460800bps
Return	"0 | configuration successful" "1 | failed, does not support this baudrate"
Description	

3.5 ISO15693

Read the card serial number

Packet	com.pda.rfid.hf
Class	HF
Function	public abstract String GetReaderInformation (String readType, String afi)
Parameter	readType: Continuous / single read afi:00~FF
Return	"0 | read success" "1 | module busy" "2 | parameter error" "3 | other - error"
Description	e.g.: *hfReader*.GetISO15693SN("00", "00"); Refer to the sample code

Read ISO15693 card

Packet	com.pda.rfid.hf
Class	HF
Function	public abstract String ReadISO15693 (String readMode , String startBlock , String blockNum)
Parameter	readMode: read mode startBlock: start block blockNum: continue reading block quantity
Return	"0 | read success" "1 | module busy" "2 | parameter error" "3 | other - error"
Description	e.g. :*hfReader*.ReadISO15693("00", "00", "00");

Write ISO15693 card

Packet	com.pda.rfid.hf
Class	HF
Function	public abstract String WriteISO15693 (String writeParam , String startBlock , String blockNum , String writeData)

Parameter	writeParam: write paramter startBlock: start block blockNum: block number (quantity) writeData: the data to be written
Return	"0 | write success" "1 | module busy" "2 | parameter error" "3 | other error"
Description	e.g.: `hfReader.WriteISO15693("00", "00", "00", "11223344"); write block` e.g.: `hfReader.WriteISO15693("01", "", "", "01"); write AFI` e.g.: `hfReader.WriteISO15693("02", "", "", "01"); write DSFID`

Lock ISO15693 card

Packet	com.pda.rfid.hf
Class	HF
Function	`public abstract String Lock ISO15693 (String lockSet, String block)`
Parameter	lockSet: lock the setting block: lock block
Return	"0 | success" "1 | module busy" "2 | parameter error" "3 | other error"
Description	e.g.:`hfReader.LockISO15693("01", "");  lock AFI` `hfReader.LockISO15693("02", "");  lock DSFID`

Get ISO15693 card state

Packet	com.pda.rfid.hf
Class	HF
Function	`public abstract String GetState ISO15693 (String state)`
Parameter	State: status switching
Return	
Description	None

3.6 ISO14443A

Get card SN.

Packet	com.pda.rfid.hf
Class	HF
Function	`public abstract String GetISO14443ASN (String readType ,String mode)`
Parameter	readType: Continuous / single read Mode: Request mode
Return	"0 | configuration successful" "1 | read mode error" "2 | request mode error" "3 | configured for continuous reading" "4 | other error"

Description	`setParam = "52"; //setParam = "26";`
	`e.g.:hfReader.GetISO14443ASN("0", setParam);`

ISO14443A (Mifare) hang up

Packet	com.pda.rfid.hf
Class	HF
Function	**public abstract String HaltISO14443A ()**
Parameter	None
Return	"0 | configuration successful" "1 | module busy (continuous reading)" "| configuration failed"
Description	e.g.: *hfReader*. HaltISO14443A();

ISO14443A (Mifare) card reset

Packet	com.pda.rfid.hf
Class	HF
Function	**public abstract String ResetISO14443A (String mode)**
Parameter	Mode: request mode
Return	"0 | configuration successful" "1 | module busy (continuous reading)" "| configuration failed"
Description	e.g.: *fReader*. ResetISO14443A ("0");

ISO14443A RATS command

Packet	Com.pda.rfid.hf
Class	HF
Function	**public abstract String RATSiso14443a (String tag)**
Parameter	tag: Card identifier, range of values 0x00~0x0E
Return	"0 | read success" "1 | module busy" "2 | other error"
Description	e.g.:*hfReader*. RATSiso14443a ("0");

ISO14443A PPS protocol and parameter choose

Packet	com.pda.rfid.hf
Class	HF
Function	**public abstract String PPSiso14443a (String cardRecv ,String cardSend)**
Parameter	cardRecv: card receive cardSend: card send
Return	"0 | configuration successful" "1 | read mode error" "2 | request mode error" "3 | configured for continuous reading" "4 | other error"
Description	None

ISO14443A APDU command

Packet	com.pda.rfid.hf
Class	HF
Function	**public abstract String APDUiso14443a (String order)**
Parameter	order: APDU Instruction string

Return	"0 | read successful" "1 | module busy" "2 | return error" "3 | other failed"
Description	e.g.: *hfReader*. APDUiso14443a("00 84 00 00 08");

ISO14443A Deselect command

Packet	com.pda.rfid.hf
Class	HF
Function	**public abstract String DeselctISO14443A ()**
Parameter	None
Return	"0 | read successful" "1 | module busy" "2 | return error" "3 | other failed"
Description	e.g.: *hfReader*. DeselctISO14443A ();

3.7 Mifare

Mifare card reading

Packet	com.pda.rfid.hf
Class	HF
Function	**public abstract String ReadMifare (String type, String block, String dataLen, String keyComfi, String keyType, String key)**
Parameter	type: 0: read user area, 1: read wallet block: the data block to be read dataLen: Specifies the length of the read data: Mifare* UltraLight 4 bytes; Wallet operation 4 bytes; other 16 bytes keyComfi: key authentication mode, keyType: key type key: keyword
Return	"0 | read success" "1 | module busy" "2 | parameter error" "3 | read failure" "4 | password wrong" "5 | block format error" "6 | other error"
Description	None

Mifare card writing

Packet	com.pda.rfid.hf
Class	HF
Function	**public abstract String WriteMifare (String type, String block, String keyComfi, String data, String keyType, String key)**
Parameter	type: 0: read user area, 1: read wallet block: the data block to be read keyComfi: key authentication mode keyType: key type key: keyword
Return	"0 | Write successful" "1 | Write parameter error" "2 | Access password error" "3 | Other error"
Description	None

Mifare card block operation

Packet	com.pda.rfid.hf
Class	HF
Function	**public abstract String BlockCommandMifare (String mode, String block, String data, String transBlock, String keyComfi, String keyType, String key)**
Parameter	mode: mode word block: card block number data: data transBlock: tranfering block keyComfi: key certificate method keyType:key type key: key password
Return	"0 | Write successful" "1 | Write parameter error" "2 | Access password error" "3 | Other error"
Description	Now

4.Barcode function

JAR packet and so library need to be loaded(Or directly load pdasdk.aar):
JAR packet: pdasdk.jar
so library: libpda.so, libcepri_dev.so, libIAL.so, libSDL.so, libbarcodereader.so, libbarcodereader43.so, libbarcodereader44.so

4.1 Get scanner instance

Packet	com.pda.scanner;
Class	ScanReader
Function	public static Scanner getScannerInstance()
Parameter	None
Return	Return instance when success, otherwise null
Description	

4.2 Open scanner

Packet	com.pda.scanner;
Class	Scanner
Function	public abstract boolean open(Context ctx);
Parameter	Context ctx: refer to context information

Return	True for success, false for failure
Description	

4.3 Close scanner

Packet	com.pda.scanner;
Class	Scanner
Function	public abstract void close();
Parameter	None
Return	Null
Description	

4.4 Trigger scan

Packet	com.pda.scanner;
Class	Scanner
Function	public abstract byte[] decode(int timout);
Parameter	int timout, unit:millisecond
Return	Success return barcode data, failure return null
Description	

Packet	com.pda.scanner;
Class	Scanner
Function	public abstract byte[] decode();
Parameter	None
Return	Success return barcode data, failure return null
Description	Default timeout is 3 seconds

4.5 Cancel scan

Packet	com.pda.scanner;
Class	Scanner
Function	public abstract void cancel();
Parameter	None
Return	None
Description	Only forced cancel needs to call this function

5.Serial ports functions instruction

Serial ports including:
Infrared: "IRDA"
RS485: "RS485"
ESAM: "ESAM"
JAR packet and so library need to be loaded(Or directly load pdasdk.aar):
JAR packet: pdasdk.jar
so library: libpda.so , libcepri_dev.so

5.1 Get serial port instance

Packet	`package com.pda.com;`
Class	`COMReader`
Function	`public static COM getCOMInstance(String name);`
Parameter	String name: The name of the serial port need for instance,there are the following categories. Infrared: "IRDA" RS485: "RS485" ESAM: "ESAM"
Return	Success return a serial port instance, Failure return null
Description	None

5.2 Open Serial port

Packet	`package com.pda.com;`
Class	`COM`
Function	`public abstract boolean open(String param);`
Parameter	String param: serial port parameters, Format is "bps:parity:databits:stopbits" Like: "19200:N:8:1"
Return	Success return true， failure return false
Description	None

5.3 Close Serial port

Packet	`package com.pda.com;`
Class	`COM`
Function	`public abstract void close();`
Parameter	None
Return	None
Description	None

5.4 Send data

Packet	`package com.pda.com;`
Class	`COM`
Function	`public abstract int send(byte[] buf, int offset, int length);`
Parameter	byte[] buf: buffer of the data to be sent int offset: offset of the data to be sent int length: length of the data to be sent
Return	Successfully returns the actual sent data length, and failed to return a negative number
Description	Return immediately, without waiting for completion of data transmission

Packet	`package com.pda.com;`
Class	`COM`
Function	`public abstract int send(byte[] buf);`
Parameter	byte[] buf: the data to be sent
Return	Successfully returns the actual sent data length, and failed to return a negative number
Description	Return immediately, without waiting for completion of data transmission

5.5 Receive Data

Packet	`package com.pda.com;`
Class	`COM`
Function	`public abstract int recv(byte[] buf, int offset, int length);`
Parameter	byte[] buf: buffer of the data to be received int offset: offset of the data to be received int length: data length expect to be received
Return	Successfully returns the actual length of data received, and failed to return a negative number

Description	Return immediately, without waiting for acceptance

Packet	`package com.pda.com;`
Class	`COM`
Function	`public abstract int recv(byte[] buf);`
Parameter	byte[] buf: buffer of the data to be received
Return	Successfully returns the actual length of data received, and failed to return a negative number
Description	Return immediately, without waiting for acceptance

5.6 Clear received buffer

Packet	`package com.pda.com;`
Class	`COM`
Function	`public abstract void clear_input();`
Parameter	None
Return	None
Description	None

6.Other functions interface

JAR packet and so library need to be loaded(Or directly load pdasdk.aar):
JAR package: pdasdk.jar
so Library: libpda.so, libcepri_dev.so

6.1 Get SDK version

Packet	package com.port;
Class	Adapt
Function	public static String **getVersion()**
Parameter	None
Return	Returns the version string, such as: 1.1.4
Description	Example code `Adapt.getVersion()`

6.2 LED Interface

6.2.1 Get LED instance

Packet	`package com.port;`
Class	`Adapt`
Function	`public static LEDManager getLedmanagerInstance()`
Parameter	None
Return	Return the LED management instance
Description	None

6.2.2 Get LED display

Packet	`package com.port;`
Class	`LEDManager`
Function	`public abstract void show(int r, int g, int b)`
Parameter	int r: Red Brightness 0-255 int g: Green Brightness 0-255 int b: Blue Brightness 0-255
Return	Null
Description	Red, green and blue leds are not always available on the device. Please refer to the corresponding model

6.3 Key Interface

Note 1: We can use the standard key processing interface for developing APP, and use the private key processing interface for writing Service

Note 2: The private key processing interface is limited to 7202K series and 7202G series

Definition of key values:

Key description	**Private key interface key definition**(KeyManager.XX)	**Standard key interface key definition**				
		7202K3	7202G3	HL7202G7	HL7202G9	HY820
F1 key on keyboard	KEY_F1	X	139	X	139	X
F2 key on keyboard	KEY_F2	X	140	X	140	X
F3 key on keyboard	KEY_F3	X	141	X	X	X
F4 key on keyboard	KEY_F4	X	142	X	X	X
Device left upper shortcut key	KEY_L1	X	X	133	X	X
Device left lower shortcut key	KEY_L2	X	X	132	X	285
Device right upper shortcut key	KEY_R1	X	X	X	X	X

Device right lower shortcut key	KEY_R2	X	X	134	X	286
Handle button shortcut key	KEY_TRIG	135	X	X	X	139
Device scan shortcut key	KEY_SCAN	X	X	X	X	140

For HY820

Key description	Key value
Device left lower shortcut key	285
Device right lower shortcut key	286
Handle trigger button UHF shortcut key (Switch RFID/Barcode Switching Key to RFID)	139
Handle trigger button SCAN shortcut key (Switch RFID/Barcode Switching Key to SCAN)	140

```
/** Key code constant: F9 key. */
public static final int KEYCODE_F9              = 139;
/** Key code constant: F10 key. */
public static final int KEYCODE_F10             = 140;
/** Key code constant: F11 key. */
```

6.3.1 Get Key management instance

Packet	`package com.port;`
Class	`Adapt`
Function	`public static KeyManager getKeymanagerInstance()`
Parameter	None
Return	Returns the key management instance
Description	None

6.3.2 Key registration call back

Packet	`package com.port;`
Class	`KeyManager`

Function	`public boolean setKeyHandler(KerHandler h)`
Parameter	KerHandler h: Key handling callback
Return	Setting result
Description	The callback is a special key callback, non KeyEvent defined, Special keys are not available in all devices, some devices have only some of them.

6.4 Attributes Interface

6.4.1 Get Properties instance

Packet	`package com.port;`
Class	`Adapt`
Function	`public static PropertiesManager getPropertiesInstance()`
Parameter	None
Return	Returns the properties management instance
Description	None

6.4.2 Get Android version

Packet	`package com.port;`
Class	`PropertiesManager`
Function	`public abstract String getVersion();`
Parameter	None
Return	Return android version number
Description	None

6.4.3 Get SN number

Packet	`package com.port;`
Class	`PropertiesManager`
Function	`public abstract String` **`getSN();`**
Parameter	None
Return	return 16 bytes SN number
Description	Example code `Adapt.getPropertiesInstance().`**`getSN()`**

6.4.4 Check whether the device supports related modules

Packet	`package com.port;`
Class	`PropertiesManager`
Function	`public abstract boolean` **`support`**`(String strDevice)`
Parameter	String strDevice: device type string
Return	Return whether the device is supported
Description	"UHF" "backupPower" // built-in battery specially designed for powering RFID module "HF" "1DSCANNER" "2DSCANNER" "PSAM" "IRDA" "RS232" "RS485" Example code `Adapt.getPropertiesInstance().support("UHF")`

6.4.5 Get device model

Packet	`package com.port;`
Class	`PropertiesManager`
Function	`public abstract String getDeviceModel(String strDevice);`
Parameter	String strDevice: device type string
Return	Return Device Type
Description	"UHF" "backupPower" "HF" "1DSCANNER" "2DSCANNER" "PSAM" "IRDA" "RS232" "RS485"

6.5 Power related interfaces

6.5.1 Get a power management instance

Packet	package com.port;
Class	Adapt
Function	public static PowerManager getPowermanagerInstance()
Parameter	None
Return	Return the power management instance
Description	None

6.5.2 Get the backup battery power (Only for models with built-in battery, CL7202K3/HL7202K6 Series)

Packet	package com.port;
Class	PowerManager
Function	public abstract int getBackupPowerSOC();
Parameter	None
Return	Return power value from 0 to 100 value
Description	(only for CL7202K3 / HL7202K6 serials)

6.6 Get Model type

Packet	package com.port;
Class	Adapt
Function	public static int getDeviceType()
Parameter	None
Return	Returns the model code, see Adapt.DEVICE_TYPE_XX definition.
Description	None

6.7 Stop the SDK after entering the background

Packet	package com.port;

Class	Adapt
Function	public static void enablePauseInBackGround(Context ctx)
Parameter	Context ctx APP context
Return	None
Description	If the app needs the SDK to stop working when it enters the background, you can call this interface in the first Activity.

6.8 SDK Initialize the SDK

Packet	package com.port;
Class	Adapt
Function	public static boolean init(Context context)
Parameter	Context ctx APP context
Return	None
Description	Starting from the HY820 series, it is necessary to call this interface in the APP for initialization to correctly call other API resources and object instances. This interface must be called in the first interface, the calling sequence is 1. First dynamically apply for permission android.permission.READ_PHONE_STATE, 2. After passing, call this interface, and then call other interfaces or obtain other object instances in the SDK The wrong order will cause the SDK to not run normally.

Use reference	`//First interface call`
	`@Override`
	`public void onCreate(Bundle savedInstanceState) {`
	
	`  super.onCreate(savedInstanceState);`
	`  setContentView(R.layout.item_main);`
	
	`  Adapt.init(this);`
	
	`  // TOOD Other`
	
	`}`

7.PSAM function

JAR packet and so library need to be loaded:
JAR package: pdasdk.jar
so Library: libpda.so, libcepri_dev.so

7.1 Get a PSAM instance

Packet	com.pda.psam;
Class	PSAMReader
Function	public static PSAM getPSAMInstance(int id)
Parameter	id PSAM socket number, starting from 0, i.e. PSAM card 1 is 0, PSAM card 2 is 1.
Return	A successful operation returns a PSAM instance, a failed operation returns null.
Description	private PSAM psamReader = PSAMReader.getPSAMInstance(0);

7.2 Open PSAM

Packet	com.pda.psam;
Class	PSAM
Function	public abstract byte[] open(String param);
Parameter	String param: Baud rate. Currently only "9600" is supported.
Return	A successful operation returns ATR, a failed operation returns null.

Description	PSAM psamReader = PSAMReader.getPSAMInstance(0);
	byte[] atr = psamReader.open("9600");
	// TODO something
	//...
	psamReader.close();

7.3 Close PSAM

Packet	com.pda.psam;
Class	PSAM
Function	public abstract void close();
Parameter	None
Return	Null
Description	

7.4 Send and receive data

Packet	com.pda.psam;
Class	PSAM
Function	public abstract byte[] xfer(byte[] apdu, int offset, int length);
Parameter	byte[] apdu: APDU command cache

	int offset: APDU Cache Location int length: APDU Command Length
Return	Success returns apdu answer, failure returns null
Description	The APDU command format is CLA INS P1 P2 P3 DATA The APDU return format is DATA SW1 SW2

Packet	`com.pda.psam;`
Class	`PSAM`
Function	`public abstract byte[] xfer(byte[] apdu);`
Parameter	byte[] apdu: APDU command cache
Return	Success returns apdu answer, failure returns null

Description	byte[] apdu = new byte[]{0x00, (byte)0x84, 0x00, 0x00, 0x04}; // get 4 bytes rand
	PSAM psamReader = PSAMReader.getPSAMInstance(0);
	byte[] atr = psamReader.open("9600");
	// get 4 bytes rand
	byte[] rtAPDU = psamReader.xfer(apdu); // will be return " xx xx xx xx 90 00" if success
	psamReader.close();

8. Troubleshooting and solutions

Problem	Troubleshoot and resolve
UHF unable to set parameters	Check whether the module in a state of loop read tags, if it is in a state reading the tags, please first call the Stop() method to stop the current UHF module working condition, then set the parameters.
PDA power consumption too fast	1. Please make sure to turn on the UHF module only when the software needs to use it, if the software keeps the UHF module on, the power consumption will be too fast and it will get hot. 2. Replace another battery If possible to confirm whether it is the battery's problem. 3. Depending on the application, do not read the tags for a long time continuously.
Not work properly after the UHF standby	The PDA will power off all modules after standby and power back on only after standby is restored. Please re-open the UHF module after standby recovery.
UHF Function returns -1	1. Incorrect incoming parameters or abnormal execution of operations. 2. The module is damaged or the physical connection is abnormal or the battery is seriously damaged. 4. The device is sending a card reading/configuration command when it is in a tag reading state, in which case the tag reading should be stopped first.
JNI issue	For android studio project, please put the library file into jniLibs folder, if you use so from armeabi-v7 in your system, please copy the so from armeabi to armeabi-v7 manually.
HY820 all interfaces	1. Make sure you have dynamically applied the

working abnormally	android.permission.READ_PHONE_STATE 2. Make sure that Adapt.init() is called before using the sdk API or creating/getting sdk objects

Appendix A: The 6C tag operation returned error codes

Read tag error codes:

Code	Remark
0	Configuration successful
1	Antenna port Parameter error
2	Choosing tag Parameter error
3	TID parameter error
4	user data area parameter error
5	reserved area parameter error
6	Other parameter error

Write tag error codes:

Code	Remark
0	Write successfully
1	Antenna port parameter error
2	Choose parameter error
3	Write parameter error
4	CRC checksum error
5	Power not enough
6	data block overflow
7	data block locked
8	Access password error
9	Other tag error
10	Tag lost
11	Instruction error

Lock tag error codes:

Code	Remark
0	Lock successfully
1	Antenna com port parameter error
2	Choose parameter error
3	Write parameter error

4	CRC checksum error
5	Power no enough
6	data block overflow
7	data block is locked
8	Access password error
9	Other tag error
10	Tag lost
11	Reader instruction error

Kill tag error codes:

Code	Remark
0	Kill successfully
1	Antenna port parameter error
2	Choose parameter error
3	CRC checksum error
4	Power not enough
5	Kill Password error
6	Other tag error
7	Tag lost
8	Instruction error