Ultra High Frequency RFID Data Collection Terminal

GS U20 Development Guide

Disclaimer

This manual can only be provided to qualified customers (individuals or organizations) through attachment in encrypted e-mail, and watermark of customers' e-mail address or company name has been used as signature in the manual. GENERALSCAN reserves the right to investigate legal responsibility against other illegal means to transmission.

GENERALSCAN reserves the right to make changes to the product specifications and this manual without any statements. Customers can contact the technical support of this manual to get the latest version of the product specifications and manual. GENERALSCAN will not take any responsibility for the loss, damage and any other liability caused by the described information in this manual.

Please carefully evaluate whether this product is applicable to your application scenario. For special use, including but not limited to aviation, aerospace, military, medical and life support systems, GENERALSCAN cannot guarantee the applicability and will not take any responsibility.

This manual cannot be used as the basis for the authorization of intellectual property rights (including but not limited to patents, trademarks, software copyrights).

1. Overview ..... - 5 -
1.1. Content Overview ..... - 5 -
1.2. Applicable equipment types ..... - 5 -
1.3. Copyright Statement ..... - 5 -
2. UHF function module function description ..... - 6 -
2.1. UHF development process ..... - 6 -
2.2. Opening and closing connections ..... - 7 -
2.2.1. Open module ..... - 7 -
2.2.2. Close the module ..... - 7 -
2.3. Stop command ..... - 7 -
2.4. Obtain antenna power ..... - 8 -
2.5. Set antenna power ..... - 8 -
2.6. Obtain baseband parameters ..... - 8 -
2.7. Set baseband parameters ..... - 10 -
2.8. Acquiring literacy skills ..... - 11 -
2.9. Obtain RF band ..... - 12 -
2.10. Set RF band ..... - 12 -
2.11. Query tag upload parameters ..... - 12 -
2.12. Set tag upload parameters ..... - 13 -
2.13. Query automatic idle mode ..... - 13 -
2.14. Configuring automatic idle mode ..... - 13 -
2.15. Obtain baseband software version ..... - 14 -
2.16. Transmitting carrier wave ..... - 14 -
2.17. Standing wave detection ..... - 14 -
2.18. Restore factory settings ..... - 15 -
2.19. 6C label operation ..... - 15 -
2.19.1. Reading Tags ..... - 15 -
2.19.2. Writing tags ..... - 21 -
2.19.3. Lock tag ..... - 23 -
2.19.4. Inactivation tag ..... - 24 -
2.20. 6B tag operation ..... - 24 -
2.20.1. Reading Tags ..... - 24 -
2.20.2. Writing tags ..... - 25 -
2.20.3. Locking Tags ..... - 25 -
2.20.4. Lock query ..... - 25 -
2.21. GB tag operation ..... - 26 -
2.21.1. Reading Tags ..... - 26 -
2.21.2. Writing tags ..... - 27 -
2.21.3. Lock tag ..... - 28 -
2.22. GJB tag operation ..... - 29 -
2.22.1. Reading Tags ..... - 29 -
2.22.2. Writing tags ..... - 30 -
2.23. Callback interface IAsynchronousMessage Description ..... - 31 -
2.24. Callback data EPCModel field description ..... - 32 -

2.25. Sample code ..... - 32 -
3. Function description of scaner function ..... - 36 -
3.1. Obtain barcode instance ..... - 36 -
3.2. Open scanner ..... - 36 -
3.3. Close scanner ..... - 36 -
3.4. Trigger scan ..... - 37 -
3.5. Cancel scan ..... - 37 -
4. Other function descriptions ..... - 37 -
4.1. Get SDK version ..... - 37 -
4.2. 6.2 LED related interfaces ..... - 38 -
4.2.1. Obtain LED instance ..... - 38 -
4.2.2. LED display ..... - 38 -
4.3. Key-related interfaces ..... - 38 -
4.4. Enter the background to stop the SDK ..... - 39 -
4.5. Initialize SDK ..... - 39 -
5. Troubleshooting and solving common problems ..... - 40 -

1.Overview

1.1. Content overview

In order to facilitate users' secondary development, we provide function libraries that can run on the android platform. The function library is written in Java language and packaged into a standard JAR package.

The application development guide comprehensively introduces the corresponding technical indicators, application development instructions and precautions, UHF module function description, scanner module function description, etc.

1.2. Applicable equipment type

This document lists the APIs of all RFID devices. The following table lists the functional modules supported by different models (for specific events that need attention, please see the detailed description of the functional modules)

functional module	Applicable equipment type
Connect devices	Full line
Device Configuration	Full line
UHF module function description	Full line
Scanner terminal module function description	Optional barcode function model
6C label operation	Full line
6B label operation	Optional 6B function model
GB tag operation	Optional GB function model
GJB tag operation	Optional GJB function model

Note: When using the SDK, you need to initialize the SDK before you can correctly call other API resources and object instances (see 6.8 Initializing the SDK Interface)

Note : APP needs to add the following permissions (at the same time, it needs to dynamically apply for permissions before calling SDK initialization):

- android.permission.READ_PHONE_STATE

1.3. Copyright statement

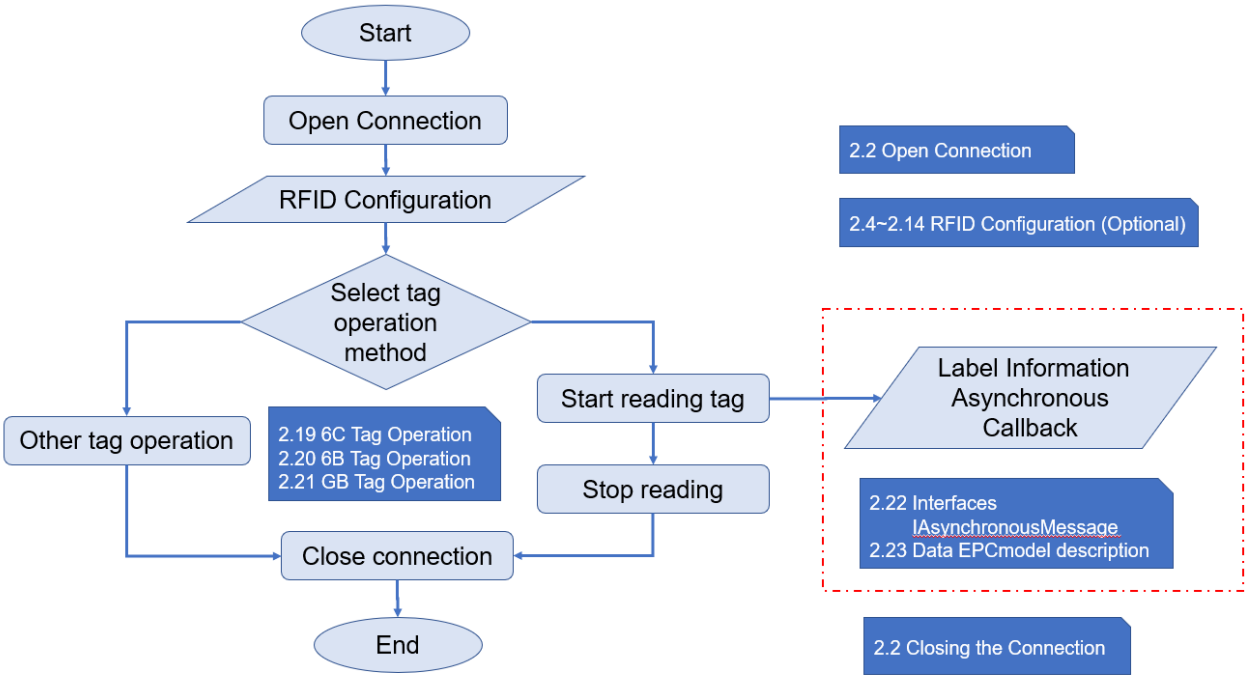
right to pursue legal liability for those who use it for commercial purposes without permission.

Without authorization, users are not allowed to add, modify, or delete the content of this document, and they are not allowed to disseminate it through the Internet, CDs, etc. If you fail to do so, you will be responsible for the consequences.

2.UHF module function description

JAR packages and SO libraries that need to be loaded (or load pdasdk.aar directly) :
JAR package: pdasdk.jar
SO library: libpda.so

2.1. UHF development process



Note: Before using the SDK, you need to initialize the SDK before you can call other API resources and object instances correctly (see 6.8 Initializing the SDK Interface)

Note : APP needs to add the following permissions (you need to dynamically apply for permissions before calling SDK initialization):

- android.permission.READ_PHONE_STATE

2.2. Open and close connections

2.2.1. Open module

Bag	com.pda.rfid.uhf
kind	UHFReader._Config or UHFReader
function	**public int OpenConnect(IAsynchronousMessage log)**
parameter	**log:** data callback interface, all tag data will be called back from this interface object.
return value	0 success, 1 failure.
illustrate	1. **log** For data callback interface, please refer to **the callback interface description** for details. 2. Call *example* **U** HFReader._Config.OpenConnect (log **)**

2.2.2. Close module

Bag	com.pda.rfid.uhf
kind	UHFReader._Config or UHFReader
function	**public void CloseConnect()**
parameter	none
return value	none
illustrate	1.Calling this method will shut down the UHF module and no longer provide power to the module. 2.Call *example* UHFReader._Config.CloseConnect (**)**

2.3. Stop command

Bag	com.pda.rfid.uhf
kind	UHFReader._Config or UHFReader
function	**public String Stop()**
parameter	none
return value	" 0 " operation is successful, non- " 0 " operation fails.
illustrate	1. This method will cause the reader to stop all current work. 2. Use this method to stop looping and reading tags. 3. *UHFReader* .getUHFInstance () **.Stop()**

2.4. Acquire antenna power

Bag	com.pda.rfid.uhf
kind	UHFReader._Config
function	**public int GetANTPowerParam()**
parameter	
return value	Power value, value range 0 - 3 3 dBm. -1, indicates that the acquisition failed.
illustrate	1. Obtain the power of the antenna. 2. Call the example *U* HFReader._Config.GetANTPowerParam **(**)

2.5. Set antenna power

Bag	com.pda.rfid.uhf
kind	UHFReader._Config
function	**public int SetANTPowerParam(int antCount, int powerValue)**
parameter	//antCount, the number of antenna numbers, fixed to 1 for all systems // powerValue, the power value to be set, the value range is 0 - 3 3 dBm.
return value	0 means the operation is successful, non-0 means the operation failed.
illustrate	1、Set the antenna power. 2、Call **example** U HFReader._Config.SetANTPowerParam (1,28 **)**

2.6. Get baseband parameters

Bag	com.pda.rfid.uhf
kind	UHFReader._Config
function	**public String GetEPCBaseBandParam ( )**
parameter	

return value	Return data example: " 255|4|0|2 " , return empty (""), indicating failure to obtain.
	// basebandMode: EPC baseband rate (0~255, 255 is 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 starting Q value used by the reader and writer.
	// session: 0~3
	// searchType: Inventory flag parameter (0 only uses Flag A for inventory, 1 only uses Flag B for inventory, 2 uses Flag A and Flag B in turn for double-sided inventory).

illustrate	1、Baseband parameters of UHF module 2、Call example UHFReader._*Config*.**GetEPCBaseBandParam()**

Bag	com.pda.rfid.uhf
kind	UHF
function	**public String GetBaseBandX ( )**
parameter	

return value	Return data example: " 1,00000000&2,00000000&3,00000000&4,00000000 & 5,00000000 ", returns empty (""), indicating that the acquisition failed.
	//The returned data is a set of optional configuration parameters, separated by '&'. The optional parameters include id and parameters, separated by ','. The parameters are 4-byte data (returned in hexadecimal string format)
	// Parameter 1: Big endian format composed of U32
	bit3-bit0: rfu
	bit4: tag focus enable
	bit5: fast id enable
	bit31-bit6: rfu
	// Parameter 2:
	Byte 1: maxQ
	Byte 2: minQ
	Byte 3::tmult
	Byte 4: bit 0 dynamic start Q enable
	bit 1 Force Loop enable
	// Parameter 3:
	Byte 1: Antenna switching mode
	0: No label, switch immediately
	1: The dwell time is exhausted
	Byte 2: Number of retries (immediate switch mode without label)
	Bytes 3-4: Big endian format composed of U16, maximum antenna dwell time (x10ms)
	// Parameter 4:
	Byte 1: Antenna switching waiting time (x10ms)
	Byte 2: Antenna switching step value
	Byte 3: Antenna protection threshold (return loss dBm), set to 0 to disable protection.
	Byte 4 : Reserved
	// Parameter 5:
	Byte 1 : L BT operating mode
	0: disable
	1: Listen only
	2: Listening and card reading
	3: Read the card only after meeting the RSSI
	Byte 2 : RSSI maximum value
	Bytes 3-4: reserved

illustrate	1、Get the baseband extension parameters of the UHF module 2、Modification of the baseband extension parameters of the UHF module will result in the inability to read the card. Please modify it carefully. If you need to modify it, please call technical support. 3、Call example UHFReader.getInstance().**GetBaseBandX()**

2.7. Set baseband parameters

Bag	com.pda.rfid.uhf
kind	UHFReader._Config
function	**public int SetEPCBaseBandParam (int basebandMode, int qValue, int session, int searchType)**

parameter	// basebandMode: EPC baseband rate (0~255, 255 is 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 starting Q value used by the reader and writer.
	// session: 0~3
	// searchType: Inventory flag parameter (0 only uses Flag A for inventory, 1 only uses Flag B for inventory, 2 uses Flag A and Flag B in turn for double-sided inventory).

return value	0 means the operation is successful, non-0 means the operation fails.
illustrate	1、This interface is used to set the baseband parameters of the UHF module. 2、Call **example** *U* HFReader._Config.SetEPCBaseBandParam (255,4,0,2 **)**

Bag	com.pda.rfid.uhf
kind	UHF
function	**public String SetBaseBandX (String param)**

parameter	Parameter data example: " 1,00000000&2,00000000 "
	// The parameter is a set of optional configuration parameters. Multiple sets of parameters are separated by '&'. The optional parameters include id and parameters, separated by ','. The parameter is 4-byte data (returned in hexadecimal string format);
	// Parameter 1: Big endian format composed of U32
	bit3-bit0: rfu
	bit4: tag focus enable
	bit5: fast id enable
	bit31-bit6: rfu
	// Parameter 2:
	Byte 1: maxQ

	Byte 2: minQ
	Byte 3::tmult
	Byte 4: bit 0 dynamic start Q enable
	bit 1 Force Loop enable
	// Parameter 3:
	Byte 1: Antenna switching mode
	0: No label, switch immediately
	1: Use up the dwell time
	Byte 2: Number of retries (immediate switch mode without label)
	Bytes 3-4: Big endian format composed of U16, maximum antenna dwell time (x10ms)
	// Parameter 4:
	Byte 1: Antenna switching waiting time (x10ms)
	Byte 2: Antenna switching step value
	Byte 3: Antenna protection threshold (return loss dBm), set to 0 to disable protection.
	Byte 4 : reserved
	// Parameter 5:
	Byte 1 : L BT operating mode
	0: disable
	1: Listen only
	2: Listening and card reading
	3: Read the card only after meeting the RSSI
	Byte 2 : RSSI maximum value
	Bytes 3-4: reserved

return value	" 0 " The operation is successful, other operations fail.
illustrate	1、This interface is for setting the baseband extension parameters of the UHF module. 2、Modification of the baseband extension parameters of the UHF module will result in the inability to read the card. Please modify it carefully. If you need to modify it, please call technical support. 3、Call example UHFReader.getInstance().SetBaseBandX("1,000000 1 0")

2.8. Acquire literacy skills

Bag	com.pda.rfid.uhf
kind	UHFReader._Config
function	**public String GetReaderProperty()**
parameter	
return value	Return data example: " Minimum transmit power | Maximum transmit power | Number of antennas | Frequency band list | RFID protocol list "

illustrate	1. Obtain the reading and writing capabilities of the reader. 2. Call example UHFReader._*Config*.**GetReaderProperty()**

2.9. Get RF band

Bag	com.pda.rfid.uhf
kind	UHFReader._Config
function	**public int GetFrequency()**
parameter	
return value	-1, acquisition failed. 0, national standard 920~925MHz 1. National standard 840~845MHz 2. National standard 840~845MHz and 920~925MHz 3. FCC, 902~928MHz 4. ETSI, 866~868MHz
illustrate	1、Get the RF band. 2、Call example UHFReader._*Config*.**GetFrequency()**

2.10. Set RF band

Bag	com.pda.rfid.uhf
kind	UHFReader._Config
function	**public int SetFrequency(int frequencyIndex)**
parameter	// frequencyIndex: RF band index 0, national standard 920~925MHz 1. National standard 840~845MHz 2. National standard 840~845MHz and 920~925MHz 3. FCC, 902~928MHz 4. ETSI, 866~868MHz
return value	0 means the operation is successful, non-0 means the operation failed.
illustrate	1、Set the RF band. 2、Call *example* UHFReader._*Config*.**SetFrequency**(0)

2.11. Query tag upload parameters

Bag	com.pda.rfid.uhf
kind	UHFReader._Config
function	**public String GetTagUpdateParam()**
parameter	
return value	Return data example: " 20|90 "

	// repeatTimeFilter: Repeat tag upload filter time , time unit: 10ms // RSSIFilter: RSSI filtering
illustrate	1、Query the upload parameters of the tag. 2、Call example UHFReader._*Config*.**GetTagUpdateParam()**

2.12. Set tag upload parameters

Bag	com.pda.rfid.uhf
kind	UHFReader._Config
function	**public int SetTagUpdateParam(int repeatTimeFilter, int RSSIFilter)**
parameter	// repeatTimeFilter: Repeat tag upload filter time , time unit: 10ms // RSSIFilter: RSSI filtering
return value	0 means the operation is successful, non-0 means the operation failed.
illustrate	1、Set the upload parameters of the tag. 2、Call **example** UHFReader._*Config*.**SetTagUpdateParam(**2,0**)**

2.13. Query automatic idle mode

Bag	com.pda.rfid.uhf
kind	UHFReader._Config
function	**public String GetReaderAutoSleepParam()**
parameter	
return value	Return data example: " 1|20 " , return empty ("") or "Failed to get!" , indicating failure to obtain. 1: Automatic idle mode enabled. 0--Turn off automatic idle mode; 1--Enable automatic idle mode. 20: The time to automatically enter the idle state, 0~65535, time unit: 10ms.
illustrate	1、Query the automatic idle mode. 2、Call example UHFReader._*Config*.**GetReaderAutoSleepParam()**

2.14. Configure automatic idle mode

Bag	com.pda.rfid.uhf
kind	UHFReader._Config
function	**public int SetReaderAutoSleepParam(boolean** isOpen, **int** time **)**
parameter	// isOpen: enable automatic idle mode // time: idle time, time unit: 10ms.
return value	0 means the operation is successful, non-0 means the operation failed.

illustrate	1、Configure automatic idle mode. 2、Call example UHFReader._*Config*.**SetReaderAutoSleepParam(**true,20**)**

2.15. Get the baseband software version

Bag	com.pda.rfid.uhf
kind	UHFReader._Config
function	**public String GetReaderBaseBandSoftVersion()**
parameter	
return value	Baseband software version, such as " V2.15 " .
illustrate	1、Get the baseband software version. 2、Call example UHFReader._*Config*.**GetReaderBaseBandSoftVersion()**

2.16. Transmit carrier

Bag	com.pda.rfid.uhf
kind	UHFReader._Config
function	**public int Set_0101_00(byte antenna_Port , byte frequency_Number)**
parameter	// **antenna_Port** : Antenna port (1, antenna 1; 2, antenna 2)) // **frequency_Number** : the index value of the frequency of the current frequency band (generally fill in 0)
return value	0 means the operation is successful, non-0 means the operation failed.
illustrate	1、Transmit carrier wave. (used to detect standing waves at antenna ports) 2、Call example UHFReader._*Config*.**Set_0101_00(**(byte)1,(byte)0**)**

2.17. Standing wave detection

Bag	com.pda.rfid.uhf
kind	UHFReader._Config
function	**public String Get_0101_05()**
parameter	none
return value	0 means the operation is successful, non-0 means the operation failed.
illustrate	1、Standing wave detection. (for detecting antennas) 2、Call example UHFReader._*Config*.**Get_0101_05()**

2.18. Factory reset

Bag	com.pda.rfid.uhf
kind	UHFReader._Config
function	**public int SetReaderRestoreFactory()**
parameter	none
return value	0 means the operation is successful, non-0 means the operation fails.
illustrate	1. Restore all settings to the factory state of the device. 2. Call *example* UHFReader.*_Config*.**SetReaderRestoreFactory()**

2.19. 6C label operation

2.19.1. read tag

Bag	com.pda.rfid.uhf
kind	UHFReader._Tag6C

Function 1	**int GetEPC(int antNum, int readType)**
	//Read only EPC
	// antNum: Antenna port.
	Example of antenna operation when specifying antenna 1 and antenna 2 at the same time: 3// readType: read type enumeration (single or loop)

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 and read EPC
	// sEPC: EPC value to match (hexadecimal string)

Function 4	**int GetEPC_MatchEPC(int antNum, int readType, String sEPC, int matchWordStartIndex)**
	// Match EPC and read EPC
	// matchWordStartIndex: starting index of matching data

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 and read EPC
	// sTID: TID value to match (hexadecimal string)

Function 7	int **GetEPC_MatchTID(int antNum, int readType, String sTID, int**

	matchWordStartIndex)
	// Match TID and read EPC
	// matchWordStartIndex: starting index of matching data

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)** //Read EPC and TID simultaneously
Function 10	**int GetEPC_TID_MatchEPC(int antNum, int readType, String sEPC)** // Match EPC and read EPC and TID at the same time
Function 11	**int GetEPC_TID_MatchEPC(int antNum, int readType, String sEPC, int matchWordStartIndex)** // Match EPC and read EPC and TID at the same time
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 and read EPC and TID at the same time
Function 14	**int GetEPC_TID_MatchTID(int antNum, int readType, String sTID, int matchWordStartIndex)** // Match TID and read EPC and TID at the same time
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)**
	// Read EPC, TID and UserData at the same time
	// readStart reads the starting index of the user area
	// readLen reads the length of the user area (unit: Word)

Function 17	**int GetEPC_TID_UserData_MatchEPC(int antNum, int readType, int readStart, int readLen, String sEPC)** // Match EPC and read EPC, TID and UserData at the same time
Function 18	**int GetEPC_TID_UserData_MatchEPC(int antNum, int readType,int readStart, int readLen, String sEPC, int matchWordStartIndex)** // 匹配 EPC 同时读 EPC、TID 和 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 and read EPC, TID and UserData at the same time
Function 21	**int GetEPC_TID_UserData_MatchTID(int antNum, int readType, int readStart, int readLen, String sTID, int matchWordStartIndex)**

	// Match TID and read EPC, TID and UserData at the same time
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 area
	// readStart reads the starting index of the EPCData area
	// readLen reads the length of the EPCData area (unit: Word)

Function 24	**int GetEPC_EpcData_MathcEPCData(int antNum , int readType, int readStart, int readLen, String matchEPCData, int matchBitStartIndex)**
	//Match EPCData and read the EPC and EPCData areas at the same time
	// readStart reads the starting index of the EPCData area
	// readLen reads the length of the EPCData area (unit: Word)
	// data content matched by matchEPCData
	// matchBitStartIndex matches the data starting bit (unit: Bit)

Function 25	**int GetEPC_TID_EpcData(int antNum , int readType, int readStart, int readLen)**
	//Read EPC, TID and EPCData areas
	// readStart reads the starting index of the EPCData area
	// readLen reads the length of the EPCData area (unit: Word)

Function 26	**int GetEPC_TID_EpcData_MathcEPCData(int antNum , int readType, int readStart, int readLen, String matchEPCData, int matchBitStartIndex)**
	//Match EPCData and read EPC, TID and EPC Data areas at the same time
	// readStart reads the starting index of the EPC Data area
	// readLen reads the length of the EPC Data area (unit: Word)
	// data content matched by match EPC Data
	// matchBitStartIndex matches the data starting bit (unit: Bit)

Function 27	**public int GetEPC_MatchUserData(int antNo, int readType, String sMatchWord,**
	int matchWordStartIndex, String accessPassword)
	//Match user area and read EPC at the same time
	// match Word matching data content
	// match Word StartIndex matches the data starting bit (unit: word)
	//accessPassword access password

Function 28	**public int GetEPC_TID_MatchUserData(int antNo, int readType, String sMacthword, int matchWordStartIndex, String accessPassword) ;**
	//Match user area and read EPC and TID at the same time
	/ match Word matching data content
	// match Word StartIndex matches the data starting bit (unit: word)
	//accessPassword 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 user area and read EPC, TID and user area at the same time
	// readStart reads the starting index of the user area
	// readLen reads the length of the user area (unit: Word)
	/ match Word matching data content
	// match Word StartIndex matches the data starting bit (unit: word)
	//accessPassword access password

Function 30	**public int GetRFMicron_Temperature(int antNo, int readType)**
	//Read RFMicron S3 temperature tag
	// After reading the tag data, convert it into a temperature value through the following
	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 into a temperature value through the following
	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 into a temperature value through the following
	public float EM_ConvertTemperature(EPCModel model)

Function 33	**public int GetEPC_G2V2(int antNo, int readType, G2V2AuthenticateModel g2v2, String accessPassword)**
	//Read G2V2 authentication data
	// g2v2 EPC G2V2 Authenticate parameters
	//accessPassword access password

Function 34	**public int GetEPC_G2V2(int antNo, int readType, G2V2AuthenticateModel g2v2)**
	//Read G2V2 authentication data
	// g2v2 EPC 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 authentication data
	// g2v2 EPC G2V2 Authenticate parameters
	/sEPC matching data content
	// match Word StartIndex matches the data starting bit (unit: word)
	//accessPassword access password

Function 36	**public int GetEPC_G2V2_MatchEPC(int antNo, int readType, G2V2AuthenticateModel g2v2, String sEPC, int matchWordStartIndex)**
	//Match EPC to read G2V2 authentication data
	// g2v2 EPC G2V2 Authenticate parameters
	/sEPC matching data content
	// match Word StartIndex matches the data starting bit (unit: word)

Function 37	**public int GetEPC_G2V2_Match TID (int antNo, int readType, G2V2AuthenticateModel g2v2, String s TID , int matchWordStartIndex, String accessPassword)**
	// Match TID and read G2V2 authentication data
	// g2v2 EPC G2V2 Authenticate parameters
	/sTID matching data content
	// match Word StartIndex matches the data starting bit (unit: word)
	//accessPassword access password

Function 38	**public int GetEPC_G2V2_Match TID (int antNo, int readType, G2V2AuthenticateModel g2v2, String s TID , int matchWordStartIndex)**
	//Match EPC to read G2V2 authentication data
	// g2v2 EPC G2V2 Authenticate parameters
	/sTID matching data content
	// match Word StartIndex matches the data starting bit (unit: word)

Function 39	**public int Get LTU _Temperature(int antNo, int readType)**
	//Read LTU31/32 temperature tag
	//After reading the tag data, you must call the following interface in the callback to check the validity of the data
	public boolean LTU_VerifyTemperature(EPCModel model)
	// After reading the valid tag data, convert it into a temperature value through the following
	public float LTU_ConvertTemperature(EPCModel model)

Function 40	**public int GetEPC_ReservedData(int antNum, int readType, int readStart, int readLen)**
	// Read reserved area
	// antNum : Antenna number
	// readType: reading method
	// readStart : reserved area reading start bit (0 or 2)
	// readLen: reserved area read length (maximum 4)

Function 41	**public int GetEPC_ReservedData(int antNum, int readType, int readStart, int readLen, String accessPassword)** // accessPassword: tag access password (8-digit hexadecimal string)

Function 42	**public int GetEPC_ReservedData_MacthEPC(int antNum, int readType, int readStart, int readLen, String sEPC)**
	// Match EPC read reserved area
	// sEPC: matching EPC code

Function 43	**public int GetEPC_ReservedData_MacthEPC(int antNum, int readType, int readStart, int readLen, String sEPC , String accessPassword)** // accessPassword: tag access password (8-digit hexadecimal string)

Function 44	**public int GetEPC_ReservedData_MacthTID(int antNum, int readType, int readStart, int readLen, String sTID, int matchWordStartIndex)**
	// Match TID read reserved area
	// sTID: matching TID code
	// matchWordStartIndex: match the starting 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 (8-digit hexadecimal string)

Function 46	**public int GetEPC_ LightKX2005X (int antNo, int readType, String accessPassword)**
	//Light up the KX 2005X label
	//accessPassword access password

Function 47	**public int GetEPC_ LightKX2005X (int antNo, int readType)** //Light up the KX 2005X label

Function 48	**public int GetEPC_ LightKX2005X _MatchEPC(int antNo, int readType, String sEPC, int matchWordStartIndex, String accessPassword)**
	//Match EPC to light up KX 2005X label
	/sEPC matching data content
	// match Word StartIndex matches the data starting bit (unit: word)
	//accessPassword access password

Function 49	**public int GetEPC_ LightKX2005X _MatchEPC(int antNo, int readType, String sEPC, int matchWordStartIndex)**
	//Match EPC to light up KX 2005X label
	/sEPC matching data content
	// match Word StartIndex matches the data starting bit (unit: word)

Function 50	**public int GetEPC_ LightKX2005X _Match TID (int antNo, int readType, String s TID , int matchWordStartIndex, String accessPassword)**
	// Match the TID to light up the KX 2005X label
	/sTID matching data content
	// match Word StartIndex matches the data starting bit (unit: word)
	//accessPassword access password

Function 51	**public int GetEPC_ LightKX2005X _Match TID (int antNo, int readType, String s TID , int matchWordStartIndex)**
	// Match the TID to light up the KX 2005X label
	/sTID matching data content
	// match Word StartIndex matches the data starting bit (unit: word)

Function 52	**int GetEPC_TID Data (int antNum, int readType, int readLen)**
	//Read EPC and TID at the same time , TID length can be set
	// readLen reads the length of the EPCData area (unit: Word)

parameter	See the description of each function.
return value	0 means the operation is successful, non-0 means the operation fails. Stopwatch
illustrate	1.For detailed return value description, please refer to the appendix. 2.To stop cyclic reading, please use the "stop command" 3.Call **example** *U* HFReader._Config.GetEPC (3,1 **)** 4.Call **example** *U* HFReader._Config.GetEPC_TID (3,1 **)** 5.Call **example** *U* HFReader._Config.GetEPC_TID_UserData (3,1,0,6 **)**

2.19.2. Write tags

2.18.2.1Write EPC area

Bag	com.pda.rfid.uhf
kind	UHFReader._Tag6C

Function 1	**int WriteEPC(int antNum, int sWriteData)**
	// antNum: Antenna port.
	// sWriteData: Data to be written (hexadecimal string)
	Example of antenna operation when specifying antenna 1 and antenna 2 at the same time: 3

Function 2	**int WriteEPC_MatchEPC(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex)**
	// Match EPC and write EPC area
	// sMatchData EPC data to be matched
	// matchWordStartIndex matches the starting index of the data

Function 3	**int WriteEPC_MatchEPC(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)**
	// Match EPC and write EPC area
	// accessPassword The access password of the tag

Function 4	**int WriteEPC_MatchTID(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex)**
	// Match TID and write EPC area
	// sMatchData TID data to be matched
	// matchWordStartIndex matches the starting index of the data

Function 5	**int WriteEPC_MatchTID(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)** // Match TID and write EPC area
parameter	See the description of each function.
return value	0 for success, non-zero for failure. Stopwatch
illustrate	1.It is recommended to use matching ID when writing tag data, that is, use "Function 4" and "Function 5". 2.For detailed return value description, please refer to the appendix. 3.Call example UHFReader._*Config*.**WriteEPC_MatchTID(**3,"12345678","E2003412012CFB000B26F75C",0**)**

2.18.2.2 Write Userdata area

Bag	com.pda.rfid.uhf
kind	UHFReader._Tag6C
Function 1	**int WriteUserData(int antNum, String sWriteData)** // antNum: Antenna port.

	// sWriteData: Data to be written (hexadecimal string) Example of antenna operation when specifying antenna 1 and antenna 2 at the same time: 3

Function 2	**int WriteUserData_MatchEPC(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex)**
	// Match EPC and write EPC area
	// sMatchData EPC data to be matched (hexadecimal string)
	// matchWordStartIndex matches the starting index of the data

Function 3	**int WriteUserData_MatchEPC(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)**
	// Match EPC write user area
	// accessPassword The access password of the tag

Function 4	**int WriteUserData_MatchTID(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex)**
	// Match TID and write user area
	// sMatchData TID data to be matched (hexadecimal string)
	// matchWordStartIndex matches the starting index of the data

Function 5	**int WriteUserData_MatchTID(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)** // Match TID and write user area

Function 6	**int WriteUserData_MatchTID(int antNum, String sWriteData, int writeWordStartIndex , String sMatchData, int matchWordStartIndex, String accessPassword)**
	// Match TID and write user area
	// writeWordStartIndex writes the user area word offset

parameter	See the description of each function.
return value	0 for success, non-zero for failure. Stopwatch
illustrate	4.It is recommended to use matching ID when writing tag data, that is, use "Function 4" and "Function 5". 5.For detailed return value description, please refer to the appendix. 6.Call example UHFReader._*Config*.**WriteUserData_MatchTID(**3,"ABCD4321","E2003412012CFB000B26F75C",0**)**

2.18.2.3 Write password area

Bag	com.pda.rfid.uhf
kind	UHFReader._Tag6C

Function 1	**int WriteAccessPassWord(int antNum, String sWriteData)**
	//Write tag access password
	// sWriteData: Password content (8-digit hexadecimal string data)

Function 2	**int WriteAccessPassWord(int antNum, String sWriteData, String accessPassword)** //Write tag access password

	// accessPassword: original tag access password (8-digit hexadecimal string data)

Function 3	**int WriteAccessPassWord_MatchTID(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)**
	//Write tag access password
	// sMatchData: TID data to be matched
	// matchWordStartIndex: starting index of matching data

Function 4	**int WriteDestroyPassWord(int antNum, String sWriteData)** //Write tag destruction password

Function 5	**int WriteDestroyPassWord(int antNum, String sWriteData, String accessPassword)**
	//Write tag destruction password
	// accessPassword: original tag access password (8-digit hexadecimal string data)

Function 6	**int WriteDestroyPassWord_MatchTID(int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)**
	//Write tag destruction password
	// sMatchData: TID data to be matched
	// matchWordStartIndex: starting index of matching data

parameter	See the description of each function.
return value	0 for success, non-zero for failure. Stopwatch
illustrate	Call example UHFReader._*Config*.**WriteAccessPassWord_MatchTID**(3,"ABCD4321","E200341 2012CFB000B26F75C",0,"00000000")

2.19.3. lock tag

Bag	com.pda.rfid.uhf
kind	UHFReader._Tag6C

Function 1	**int Lock(int antNum, int lockArea, int lockType)**
	//antNum: antenna port
	// lockArea: lock area: 0, fire password area 1, access password area 2, EPC area 3, TID area 4, user data area
	// lockType: lock type: 0, unlock 1, lock 2, permanently unlock 3, permanently locked

Function 2	**int Lock_MatchEPC(int antNum, int lockArea, int lockType, String sMatchData, int matchWordStartIndex)**
	// sMatchData: EPC data to be matched (hexadecimal string)
	// matchWordStartIndex: word starting address of matching data

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 string) // matchWordStartIndex: word starting address of matching data
Function 5	**int Lock_MatchTID(int antNum, int lockArea, int lockType, String sMatchData, int matchWordStartIndex, String accessPassword)** // accessPassword: tag access password
parameter	// See method description above
return value	0 means the operation is successful, non-0 means the operation failed. Stopwatch
illustrate	

2.19.4. inactivation tag

Bag	com.pda.rfid.uhf
kind	UHFReader._Tag6C

Function 1	**int Destroy(int antNum, String destroyPassword)**
	//antNum: antenna port
	// destroyPassword: destroy 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: word starting address of matching data

Function 3	**int Destroy_MatchTID(int antNum, String destroyPassword, String sMatchData, int matchWordStartIndex)**
	// sMatchData: TID data to be matched (hexadecimal string)
	// matchWordStartIndex: word starting address of matching data

parameter	// See method description above
return value	0 means the operation is successful, non-0 means the operation failed. Stopwatch
illustrate	

2.20. 6B label operation

2.20.1. read tag

Bag	com.pda.rfid.uhf
kind	UHF
function	**p ublic abstract String Get6B(String param)**
parameter	Param : Antenna port | Continuous/single reading | Reading content (0 - 1 - 2) | 1, user data reading parameters & 2, TID to be matched For example, the G series reads TID: " 1 |1|1|1,000F"
return value	If the String contains " 0 " , the operation succeeds; if the String does not contain "

	0 " , the operation fails.
illustrate	1、Read the 6B label. 2、Call example UHF *CLReader* = UHFReader.getUHFInstance(); *CLReader*.**Get6B(**3+"|1" + "|1" + "|" + "1,000F"**)**

2.20.2. write tags

Bag	com.pda.rfid.uhf
kind	UHF
function	**p ublic abstract String Write6B(String param)**
parameter	Param : Antenna port | TID of the tag to be written | Starting address | Data content
return value	If the String contains " 0 " , the operation succeeds; if the String does not contain " 0 " , the operation fails.
illustrate	1、Write 6B tag. 2、Refer to the sample code.

2.20.3. Lock tag

Bag	com.pda.rfid.uhf
kind	UHF
function	**p ublic abstract String Lock6B(String param)**
parameter	Param : Antenna port | Tag to be locked TID | Lock address
return value	If the String contains " 0 " , the operation succeeds; if the String does not contain " 0 " , the operation fails.
illustrate	Lock 6B tag.

2.20.4. Lock query

Bag	com.pda.rfid.uhf
kind	UHF
function	**public abstract String GetLock6B(String param)**
parameter	Param : Antenna port|Lock tag TID|Lock address
return value	If the String contains " 0 " , the operation succeeds; if the String does not contain " 0 " , the operation fails.
illustrate	Lock 6B tag.

2.21. GB tag operation

2.21.1. read tag

Bag	com.pda.rfid.uhf
kind	UHFReader._TagGB

Function 1	**int GetGB(int antNum, int readType)**
	//Read only EPC
	// antNum: Antenna port.
	Example of antenna operation when specifying antenna 1 and antenna 2 at the same time: 3
	// readType: read type enumeration (single or loop)

Function 2	**int GetGB(int antNum, int readType, String accessPassword)** // accessPassword: tag access password

Function 3	**int GetGB_MatchEPC(int antNum, int readType, String sEPC, int matchWordStartIndex, String accessPassword)**
	// Match EPC and read EPC
	// sEPC: EPC value to match (hexadecimal string)
	// matchWordStartIndex: starting index of matching data
	// accessPassword: tag access password

Function 4	**int GetGB_MatchTID(int antNum, int readType, String sTID, int matchWordStartIndex, String accessPassword)**
	// Match TID and read EPC
	// sTID: TID value to match (hexadecimal string)
	// matchWordStartIndex: starting index of matching data
	// accessPassword: tag access password

Function 5	**int GetGB_TID(int antNum, int readType)** //Read EPC and TID simultaneously

Function 6	**int GetGB_TID_MatchEPC(int antNum, int readType, String sEPC, int matchWordStartIndex, String accessPassword)**
	// Match EPC and read EPC and TID at the same time
	// accessPassword: tag access password

Function 7	**int GetGB_TID_MatchTID(int antNum, int readType, String sTID, int matchWordStartIndex, String accessPassword)**
	// Match TID and read EPC and TID at the same time
	// accessPassword: tag access password

Function 8	**int GetGB_TID_UserData(int antNum, int readType, int userChildArea, int readStart, int readLen)**
	// Read EPC, TID and UserData at the same time
	// userChildArea user sub-area number (0x3X, X is the sub-area number)
	// readStart reads the starting index of the user area
	// readLen reads the length of the user area (unit: Word)

Function 9	**int GetGB_TID_UserData_MatchEPC(int antNum, int readType, int userChildArea, int readStart, int readLen, String sEPC, int matchWordStartIndex, String accessPassword)**
	// Match EPC and read EPC, TID and UserData at the same time
	// accessPassword: tag access password

Function 10	**int GetGB_TID_UserData_MatchTID(int antNum, int readType, int userChildArea, int readStart, int readLen, String sTID, int matchWordStartIndex, String accessPassword)**
	// Match TID and read EPC, TID and UserData at the same time
	// accessPassword: tag access password

parameter	See the description of each function.
return value	0 means the operation is successful, non-0 means the operation fails. Stopwatch
illustrate	1.For detailed return value description, please refer to the appendix. 2.To stop cyclic reading, please use the "stop command" 3.Call **example** *U*HFReader._Config.GetGB (3,1 **)** 4.Call **example** *U*HFReader._Config.GetGB_TID (3,1 **)** 5.Call example UHFReader._Config . *GetGB_TID_UserData* **(** 3,0x31,1,0,6 **)**

2.21.2. write tags

Bag	com.pda.rfid.uhf
kind	UHFReader._TagGB

Function 1	**int WriteGB(int antNum, int sWriteData)**
	//Write to EPC
	// antNum: Antenna port.
	// sWriteData: Data to be written (hexadecimal string)
	Example of antenna operation when specifying antenna 1 and antenna 2 at the same time: 3

Function 2	**int WriteGB(int antNum, int writeArea, int sWriteData)**
	// antNum: Antenna port.
	//writeArea: write area: 0x10: EPC, 0x20: label security area, 0x3X user area (X is sub-area), other values are not supported
	// sWriteData: Data to be written (hexadecimal string)
	Example of antenna operation when specifying antenna 1 and antenna 2 at the same time: 3

Function 3	**int WriteGB_MatchTID(int antNum, int writeArea, String sWriteData, String sMatchData, int matchWordStartIndex)**
	// Match TID and write EPC area
	//writeArea: write area: 0x10:EPC, 0x3X user area (X is sub-area), other values are not supported
	// sMatchData TID data to be matched
	// matchWordStartIndex matches the starting index of the data

Function 4	**int WriteGB_MatchTID(int antNum, int writeArea, String sWriteData, String**

	sMatchData, int matchWordStartIndex, String accessPassword)
	// Match TID and write EPC area
	//writeArea: write area: 0x10: EPC, 0x20: label security area, 0x3X user area (X is sub-area), other values are not supported
	// sMatchData TID data to be matched
	// matchWordStartIndex matches the starting index of the data
	// accessPassword: Tag write password

parameter	See the description of each function.
return value	0 for success, non-zero for failure. Stopwatch
illustrate	1.It is recommended to use matching ID when writing tag data, that is, use "Function 3" and "Function 4". 2.For detailed return value description, please refer to the appendix. 3.Call example UHFReader._*Config*.**WriteGB_MatchTID(**3,0x10,"12345678","E2003412012CFB000B26F75C",0**)**

2.21.3. lock tag

Bag	com.pda.rfid.uhf
kind	UHFReader._TagGB

Function 1	**int Lock GB(int antNum, int lockAreaGB, int lockTypeGB)**
	// lock tag
	// antNum: Antenna port.
	// lockAreaGB: area to be locked, 0x10:EPC, 0x3X user area (X is sub-area)
	// lockTypeGB lock type
	0x00, readable and writable.
	0x01, readable but not writable.
	0x02, not readable but writable.
	0x03, cannot be read or written.
	0x11, the security mode is set to do not require authentication; this operation area must be a label security area.
	0x12, the security mode is set to require authentication and no secure communication is required; this operation area must be a label security area.
	0x13, the security mode is set to require authentication and secure communication; this operation area must be a label security area.
	
	Example of antenna operation when specifying antenna 1 and antenna 2 at the same time: 3

Function 2	**int Lock GB_MatchTID(int antNum, int lockAreaGB, int lockTypeGB, String sMatchData, int matchWordStartIndex)**
	// Match TID lock tag
	// sMatchData TID data to be matched

	// matchWordStartIndex matches the starting index of the data

Function 3	**int Lock GB_MatchTID(int antNum, int lockAreaGB, int lockTypeGB, String sMatchData, int matchWordStartIndex, String accessPassword)**
	// Match TID lock tag
	// sMatchData TID data to be matched
	// matchWordStartIndex matches the starting index of the data
	// accessPassword: Tag write password

parameter	See the description of each function.
return value	0 for success, non-zero for failure. Stopwatch
illustrate	1.It is recommended to use matching ID when writing tag data, that is, use "Function 3" 2.For detailed return value description, please refer to the appendix.

2.22. GJB tag operations

2.22.1. read tag

Bag	com.pda.rfid.uhf
kind	UHFReader._TagGJB

Function 1	**int GetGJB(int antNum, int readType)**
	//Read only EPC
	// antNum: Antenna port.
	Example of antenna operation when specifying antenna 1 and antenna 2 at the same time: 3
	// readType: read type enumeration (single or loop)

Function 2	**int GetGJB(int antNum, int readType, String accessPassword)** // accessPassword: Tag reading password

Function 3	**int GetGJB_MatchEPC(int antNum, int readType, String sEPC, int matchWordStartIndex, String accessPassword)**
	// Match EPC and read EPC
	// sEPC: EPC value to match (hexadecimal string)
	// matchWordStartIndex: starting index of matching data
	// accessPassword: Tag reading password

Function 4	**int GetGJB_MatchTID(int antNum, int readType, String sTID, int matchWordStartIndex, String accessPassword)**
	// Match TID and read EPC
	// sTID: TID value to match (hexadecimal string)
	// matchWordStartIndex: starting index of matching data
	// accessPassword: tag access password

Function 5	**int GetGJB_TID(int antNum, int readType)** //Read EPC and TID simultaneously

Function 6	**int GetGJB_TID_MatchEPC(int antNum, int readType, String sEPC, int matchWordStartIndex, String accessPassword)**
	// Match EPC and read EPC and TID at the same time
	// accessPassword: tag access password

Function 7	**int GetGJB_TID_MatchTID(int antNum, int readType, String sTID, int matchWordStartIndex, String accessPassword)**
	// Match TID and read EPC and TID at the same time
	// accessPassword: tag access password

Function 8	**int GetGJB_TID_UserData(int antNum, int readType, int readStart, int readLen)**
	// Read EPC, TID and UserData at the same time
	// readStart reads the starting index of the user area
	// readLen reads the length of the user area (unit: Word)

Function 9	**int GetGJB_TID_UserData_MatchEPC(int antNum, int readType, int readStart, int readLen, String sEPC, int matchWordStartIndex, String accessPassword)**
	// Match EPC and read EPC, TID and UserData at the same time
	// accessPassword: tag access password

Function 10	**int GetGJB_TID_UserData_MatchTID(int antNum, int readType, int readStart, int readLen, String sTID, int matchWordStartIndex, String accessPassword)**
	// Match TID and read EPC, TID and UserData at the same time
	// accessPassword: tag access password

parameter	See the description of each function.
return value	0 means the operation is successful, non-0 means the operation failed. Stopwatch
illustrate	1.For detailed return value description, please refer to the appendix. 2.To stop cyclic reading, please use the "stop command" 3.Call **example** UHFReader._*Config*.**GetGB(**3,1**)** 4.Call **example** UHFReader._*Config*.**GetGB_TID(**3,1**)** 5.Call example UHFReader._*Config*.**GetGB_TID_UserData(**3,1,0,6**)**

2.22.2. write tags

Bag	com.pda.rfid.uhf
kind	UHFReader._TagGB

Function 1	**int WriteGJB(int antNum, int sWriteData)**
	//Write to EPC
	// antNum: Antenna port.
	// sWriteData: Data to be written (hexadecimal string)
	Example of antenna operation when specifying antenna 1 and antenna 2 at the same time: 3

Function 1	**int WriteGJB(int antNum, int writeArea, int sWriteData)**
	// antNum: Antenna port.
	//writeArea: write area: 0x10:EPC, 0x3X user area (X is sub-area), other values are not supported
	// sWriteData: Data to be written (hexadecimal string)
	Example of antenna operation when specifying antenna 1 and antenna 2 at the same time: 3

Function 4	**int WriteGJB_MatchTID(int antNum, int writeArea, String sWriteData, String sMatchData, int matchWordStartIndex)**
	// Match TID and write EPC area
	//writeArea: write area: 0x10:EPC, 0x3X user area (X is sub-area), other values are not supported
	// sMatchData TID data to be matched
	// matchWordStartIndex matches the starting index of the data

Function 5	**int WriteGJB_MatchTID(int antNum, int writeArea, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)**
	// Match TID and write EPC area
	//writeArea: write area: 0x10:EPC, 0x3X user area (X is sub-area), other values are not supported
	// sMatchData TID data to be matched
	// matchWordStartIndex matches the starting index of the data
	// accessPassword: Tag write password

parameter	See the description of each function.
return value	0 for success, non-zero for failure. Stopwatch
illustrate	3.It is recommended to use matching ID when writing tag data, that is, use "Function 4" and "Function 5". 4.For detailed return value description, please refer to the appendix. 5.Call example UHFReader._*Config*.**WriteEPC_MatchTID(**3,"12345678","E2003412012CFB000B26F75C",0**)**

2.23. Callback interface IAsynchronousMessage description

```
// Asynchronous callback information interface
public interface IAsynchronousMessage
    {

// Output tag information callback-- all tag data is called back from this method (emphasis)
void OutPutEPC( EPCModel model);
```

}

2.24. Callback data EPCModel field description

Field	illustrate
_TagType	label types: " 6C " , " 6B " , and " GB " .
_EPC	Tag EPC data, hexadecimal string.
_PC	Tag PC value
_ANT_NUM	Antenna number uploaded this time
_RSSI	RSSI value
_Result	Tag data reading results
_TID	Tag TID value, hexadecimal string.
_UserData	Tag user data area value, hexadecimal string.
_TagetData	The label retains the data area value, a hexadecimal string.

2.25. Sample code

The simple calling sample code is as follows (see the call examples 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 read 6C in ant 1 in loop 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 (keyCode == KeyEvent.KEYCODE_F9 /* RFID 扳机*/ || keyCode == 285    /*
左快捷*/ || keyCode == 286    /*  右快捷*/) { //  按下扳机
            onRead(null);
        }
        return super.onKeyDown(keyCode, event);
    }
```

```
    @Override
    public boolean onKeyUp(int keyCode, KeyEvent event) {
        Log.d(TAG, "onKeyUp keyCode = " + keyCode);
        if (keyCode == KeyEvent.KEYCODE_F9 /* RFID扳机*/ || keyCode == 285    /* 左快
捷*/ || keyCode == 286    /* 右快捷*/) { // 放开扳机
            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.Function description of the scanner fucntion

JAR packages and SO libraries that need to be loaded (or load pdasdk.aar directly) :
JAR package: pdasdk.jar
SO library: libpda.so

3.1. Get barcode instance

Bag	com.pda.scanner;
kind	ScanReader
function	public static Scanner getScannerInstance()
parameter	none
return value	If the operation is successful, barcode instance will be returned, if it fails, null will be returned.
illustrate	

3.2. Open scanner

Bag	com.pda.scanner;
kind	Scanner
function	public abstract boolean open(Context ctx);
parameter	Context ctx: application context information
return value	Returns true if the operation is successful and false if it fails.
illustrate	

3.3. Close scanner

Bag	com.pda.scanner;
kind	Scanner
function	public abstract void close();
parameter	none
return value	none
illustrate	

3.4. Trigger scan

Bag	com.pda.scanner;
kind	Scanner
function	public abstract byte[] decode(int timout);
parameter	int timout timeout time, in milliseconds
return value	Barcode data is returned successfully, null is returned if failed.
illustrate	

Bag	com.pda.scanner;
kind	Scanner
function	public abstract byte[] decode();
parameter	none
return value	Barcode data is returned successfully, null is returned if failed.
illustrate	Default is 3 seconds timeout

3.5. Cancel scan

Bag	com.pda.scanner;
kind	Scanner
function	public abstract void cancel();
parameter	none
return value	none
illustrate	This function needs to be called only for forced cancellation

4.Other function descriptions

JAR packages and SO libraries that need to be loaded (or load pdasdk.aar directly) :
JAR package: pdasdk.jar
SO library: libpda.so

4.1. Get SDK package

Bag	package com.port;

kind	Adapt
function	public static String **getVersion()**
parameter	none
return value	Returns the version string, such as: 3.39
illustrate	Calling example Adapt.getVersion ( )

4.2.6.2 LED related interfaces

4.2.1. Get LED instance

Bag	package com.port;
kind	Adapt
function	public static LEDManager getLedmanagerInstance()
parameter	none
return value	Return to LED management instance
illustrate	none

4.2.2. LED display

Bag	package com.port;
kind	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 value	none
illustrate	The device does not necessarily have red, green, and blue LEDs. For details, please refer to the corresponding model.

4.3. Button related interface

Key value definition:

Button description	key value
Shortcut keys on the lower left side of the machine	289
Shortcut keys on the lower right side of the machine	290
Machine handle shortcut keys	139
Machine scanning shortcut keys	140

4.4. Enter the background to stop the SDK

Bag	package com.port;
kind	Adapt
function	public static void enablePauseInBackGround(Context ctx)
parameter	Context ctx APP context
return value	none
illustrate	If you need the SDK to stop working after the APP enters the background, you can call this interface in the first Activity

4.5. Initialize SDK

Bag	package com.port;
kind	Adapt
function	public static boolean init(Context context)
parameter	Context ctx APP context
return value	none
illustrate	Before using SDK , you need to call this interface in the APP for initialization before you can call other API resources and object instances correctly. This port must be called in the first interface . The calling sequence is to first dynamically apply for the permission android.permission.READ_PHONE_STATE permission. After passing the permission , call this interface, and then call other interfaces or obtain other object instances in the SDK. The wrong order will cause the sdk not running properly.

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

5.Troubleshooting and solving common problems

question	Troubleshooting and Solving
UHF cannot set parameters	1.Check whether the module is in the status of reading tags cyclically. If it is in the status of reading tags, please call the Stop() method to stop the current working status of the UHF module and then set the parameters.
PDA battery drains too quickly	1.Please make sure that the UHF module is turned on when the software needs to use it. If the UHF module is kept turn-on by the software, the power will be consumed too quickly and it will generate heat. 2.If possible, you can replace a battery to confirm whether the problem is with the battery itself. 3.Depending on the application usage , please do not read the card continuously for a long time
UHF does not work properly after standby	1.The PDA will turn off the power of all modules after standby, and will power on again after the standby is restored. Please turn the UHF module back on after the standby is restored.
UHF function returns -1	1. Incorrect parameters was passed in or abnormal execution of operations ; 2. Whether the module is damaged or the physical connection is abnormal or the battery is seriously damaged . 3. When the device is in the card reading state, it sends card reading/configuration instructions. At this time, card reading should be stopped first.
JNI issue	For the android studio project, please put the library file under the jniLibs file. If the so of armeabi-v7 is used in the system, please manually copy the so in armeabi to armeabi-v7.
All interfaces work abnormally	1. Please confirm that you have dynamically applied for the android.permission.READ_PHONE_STATE permission 2. Please confirm the calling sequence of Adapt.init() before using the sdk API or creating/getting the sdk object.