Hopeland RFID Reader Device- PC

Development Guide

--JAVA

Editor：Paul

Shenzhen Hopeland Technologies Co., Ltd

V 2.11

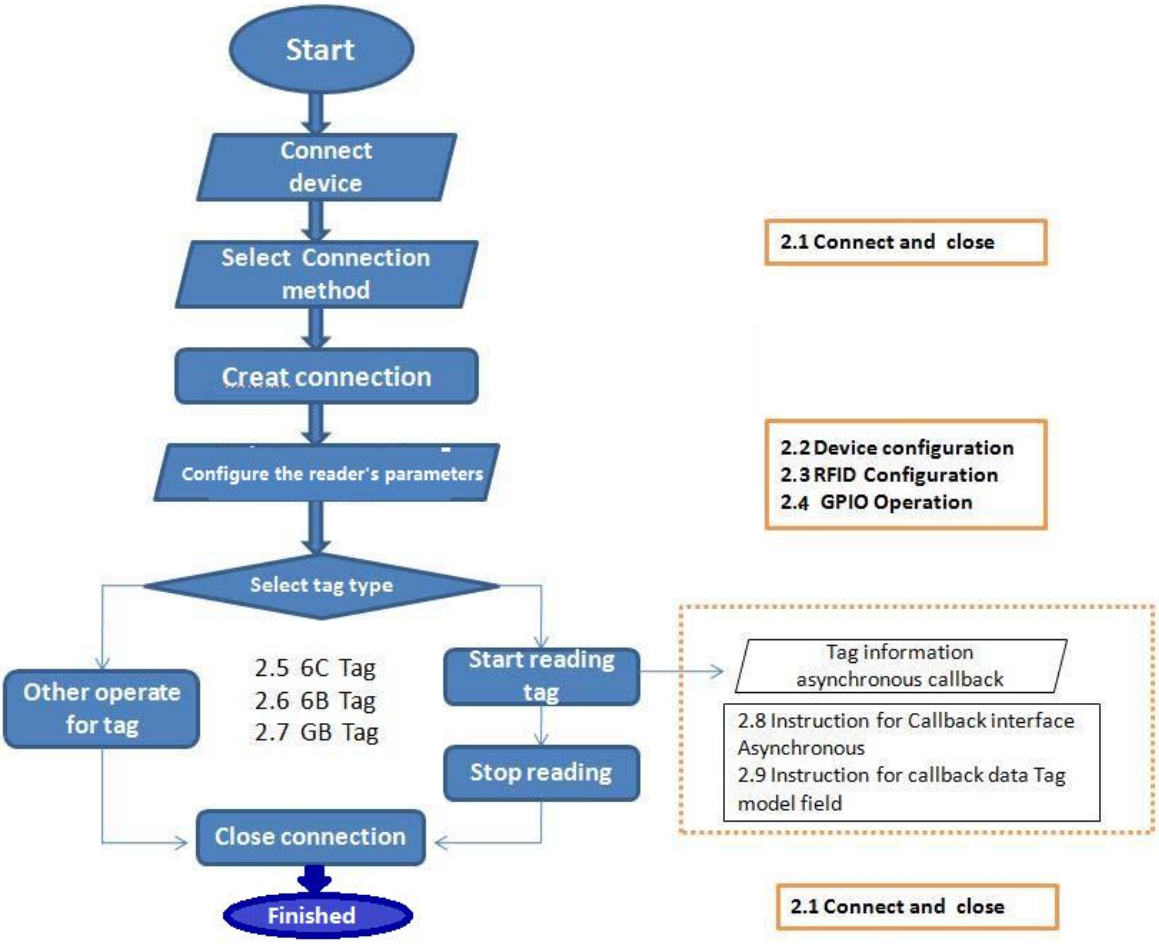
1.Summary

1.1 Summary of content

In order to facilitate the user to carry out the second development, we can provide the function library for JAVA platform. The library is written and encapsulated in the JAVA language into a standard JAR package , development environment is JDK1.8 .

Application development guide will introduce Corresponding technical indicators ,application development instruction and notes , Application interface function description and so on.

1.2 Development Process



1.3 Applicable equipment type

This document lists the API for all RFID devices, the following form lists the functional modules for different type of devices. (For the specific, please see the detailed description for functional modules)

Functional module	Applicable equipment type
Connect device	All devices
Configure device	All devices
RFID Configuration	All devices
GPO Operation	CL7206B Series, CL720C Series, Shine/Sharp/Smart series
6C tag operation	All devices
6B tag operation	All devices
GB tag operation	All devices

1.4 Copyright notice

2.Function description for API function module

2.1 Connect and close

2.1.1 Create a serial port connection

Package	RFIDReader
Function	static Boolean CreateSerialConn(string seriaParam, IAsynchronousMessage log)
Parameter	seriaParam: Serial connection parameter, e.g. "COM1:115200" log: Data callback interface, all tag data will be called back from the interface object.
Return value	True -- succeeded, false --- failed.
Description	1. The connection established by this method, the ID "seriaParam" that is the connection channel is distinguished from the other link channel. 2. log - Data callback interface, for specific please refer to Callback interface descripton
Sample code	IAsynchronousMessage log = new SampleCode(); if(RFIDReader.CreateSerialConn("COM1:115200", log)){ System.out.println("success! ");

	}**else**{
	System.**out**.println("**error!** ");
	}

2.1.2 Create TCP connection

Package	RFIDReader
Function	static Boolean CreateTcpConn(string tcpParam, IAsynchronousMessage log)
Parameter	tcpParam: TCP connection Parameter, e.g. "192.168.1.116:9090" Log: Data callback interface, all tags data will be called back from this interface.
Return	True -- succeeded, false --- failed.
Remark	1. The connection established by this method, the ID "tcpParam" that is the connection channel is distinguished from the other link channel. 2. log - Data callback interface, for specific please refer to Callback interface descripton

Sample code	IAsynchronousMessage log = **new** SampleCode();
	if(RFIDReader.CreateTcpConn("**192.168.1.116:9090**", log)){
	System.**out**.println("**success!** ");
	}**else**{
	System.**out**.println("**error!** ");
	}

2.1.3 Create 485 connection

Package	RFIDReader
Function	static Boolean Create485Conn(string _485Param, IAsynchronousMessage log)
Parameter	_485Param: RS485 connection Parameter, e.g. "1:COM1:115200", 1 as 485 connection address. Log: Data callback interface, all tags data will be called back from this interface.
Return	True -- succeeded, false --- failed.
Remark	1. The connection established by this method, the ID "_485Param" that is the connection channel is distinguished from the other link channel。 2. log - Data callback interface, for specific please refer to Callback interface descripton

Sample code	IAsynchronousMessage log = **new** SampleCode();
	if(RFIDReader.Create485Conn("**1:COM1:115200**", log))
	{
	System.**out**.println("**success!** ");
	}**else**{
	System.**out**.println("**error!** ");
	}

2.1.4 Create USB connection

Package	RFIDReader

Function	static bool CreateUsbConn(string usbDevice, IAsynchronousMessage log)
Parameter	usbDevice: usb connection Parameter Log: Data callback interface, all tags data will be called back from this interface.
Return	True -- succeeded, false --- failed.
Remark	1. We can get the connection parameter "usbDevice" of the USB device list through the function static List<string> GetUsbHidDeviceList() under the same package. 2. log - Data callback interface, for specific please refer to Callback interface descripton

Sample code	The current example does not cyclically obtain the connection parameters of the USB device list. Please change it according to the actual situation during the secondary development of the user.
	IAsynchronousMessage log = new SampleCode();
	List<String> usbDevice = RFIDReader.GetUsbHidDeviceList();
	if(RFIDReader.CreateUsbConn(usbDevice.get(1), log)){
	System.out.println("success! ");
	}else{
	System.out.println("error! ");
	}

2.1.5 Close single connection

Package	RFIDReader
Function	static void CloseConn(string connectID)
Parameter	connectID: connection ID, e.g. "192.168.1.116:9090"
Return	None
Remark	"connectID" is the connection parameter when creates connection

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	RFIDReader.CloseConn(ConnID);
	if(RFIDReader._Config.GetReaderBaseBandSoftVersion(ConnID).equals("") ||
	RFIDReader._Config.GetReaderBaseBandSoftVersion(ConnID) == null){
	System.out.println("Close the connection success! ");
	}else{
	System.out.println("Close the connection failed! ");
	}
	}else{
	System.out.println("error! ");
	}

2.1.6 Close all connections

Package	RFIDReader

Function	static void CloseAllConnect()
Parameter	None
Return	None
Remark	This method will close all created connections.

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	RFIDReader.CloseAllConnect();
	if(RFIDReader._Config.GetReaderBaseBandSoftVersion(ConnID).equals("") ||
	RFIDReader._Config.GetReaderBaseBandSoftVersion(ConnID) == null){
	System.out.println("Close the connection success! ");
	}else{
	System.out.println("Close the connection failed! ");
	}
	}else{
	System.out.println("error! ");
	}

2.1.7 Open TCP server listening

Package	RFIDReader
Function	static Boolean OpenTcpServer(string serverIP,string serverPort,IAsynchronousMessage log)
Parameter	serverIP: the IP been listening serverPort: the port been listening Log: Data callback interface, all tags data will be called back from this interface.
Return	Listen successful or not
Remark	1. Open server listening 2. When device in client mode, will connect related listening port automatically e.g.: testing IP and port under client mode of reader as"IP:192.168.1.75 Port:9090" Code example: OpenTcpServer("192.168.1.75","9090",log); 3. The default connecting IP (client mode) is "192.168.1.1", port is "9090" 4. When the connection is successful, the current connection identity ConnID is returned via the PortConnecting interface for other operation.

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	if(RFIDReader.OpenTcpServer("192.168.1.116", "9090", log)){
	System.out.println("start up success! ");
	}else{
	System.out.println("start up error! ");
	}
	}else{
	System.out.println("error! ");

	}

2.1.8 Close TCP server listening

Package	RFIDReader
Function	static void CloseTcpServer()
Parameter	None
Return	None
Remark	close server listening

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	if(RFIDReader.OpenTcpServer("192.168.1.116", "9090", log)){
	System.out.println("start up success! ");
	}
	RFIDReader.CloseTcpServer();
	if(!RFIDReader.GetServerStartUp()){
	System.out.println("Close successfully! ");
	}else{
	System.out.println("Close failed! ");
	}
	}else{
	System.out.println("error! ");
	}

2.1.9 Get TCP server listening status

Package	RFIDReader
Function	static bool GetServerStartUp()
Parameter	None
Return	True means listening, and false means not listening
Remark	Whether the server is listening

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	if(RFIDReader.GetServerStartUp()){
	System.out.println("Server is listening! ");
	}else{
	System.out.println("Server is not listening! ");
	}
	}else{
	System.out.println("error! ");

	}

2.2 Device configuration

2.2.1 IP configuration

Package	RFIDReader._Config
Function	static int SetReaderNetworkPortParam(String ConnID, String iP, String mask, String gateway)

Parameter	// ConnID: connection identification
	// iP: IP address, e.g.: "192.168.1.116"
	// mask: Subnet Mask, e.g.: "255.255.255.0"
	// gateway: gateway, e.g.: "192.168.1.1"

Return	0:succeeded, other: failed.
Remark	This method will close all created connection.

Sample code	String ConnID = **"192.168.1.116:9090"**;
	IAsynchronousMessage log = **new** SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	//Before configuring the device, you need to ensure that the reader is idle
	RFIDReader._Config.Stop(ConnID);
	if(RFIDReader._Config.SetReaderNetworkPortParam(**"192.168.1.116"**,**"192.168.2.1"**, **"255.255.255.0"**, **"192.168.2.1"**) != 0){
	System.**out**.println(**"IP Configuration failed! ! "**);
	}**else**{
	System.**out**.println(**"IP Successfully configured! "**);
	}
	}**else**{
	System.**out**.println(**"error! "**);
	}

2.2.2 Device IP configuration query

Namespace	RFIDReader._Config
Function	static String GetReaderNetworkPortParam(String ConnID)
Parameter	// ConnID: connection identification
Return	IP address| Subnet Mask |gateway
Remark	

Sample code	String ConnID = **"192.168.1.116:9090"**;
	IAsynchronousMessage log = **new** SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	String Result = RFIDReader._Config.GetReaderNetworkPortParam(ConnID);
	System.**out**.println(Result);
	}**else**{

	System.out.println("error! "); }

2.2.3 Stop instruction

Package	RFIDReader._Config
Function	static int Stop(String ConnID)
Parameter	// ConnID: connection identification
Return	0:succeeded, other:failed
Remark	1. This method will make reader stop current working. 2. Use this method to stop inventorying(cycle reading tags)

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	if(RFIDReader._Config.Stop(ConnID) != 0){
	System.out.println("Stop failed! ");
	}else{
	System.out.println("Stop Success! ");
	}
	}else{
	System.out.println("error! ");
	}

2.2.4 Configure reader time

Namespace	RFIDReader._Config
Function	**static Int32 SetReaderUTC(String ConnID, String param)**
Parameter	// ConnID: connection identification // param： time parameter ”yyyy.MM.dd HH:mm:ss”, eg: ”1970.01.01 00:00:00”
Return	0 means succeed, other means failed.
Remard	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	if(RFIDReader._Config.SetReaderUTC(ConnID,"2020.4.1 9:12:02") != 0){
	System.out.println("failed! ");
	}else{
	System.out.println("Success! ");
	}
	}else{
	System.out.println("error! ");
	}

2.2.5 Inquire reader time

Namespace	RFIDReader._Config
Function	static String GetReaderUTC(String ConnID)
Parameter	// ConnID: connection Identification
Return	Time parameter "yyyy.MM.dd HH:mm:ss", eg: "1970.01.01 00:00:00"
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetReaderUTC(ConnID));
	}else{
	System.out.println("error! ");
	}

2.2.6 serial port configuration

Namespace	RFIDReader._Config
Function	static int SetReaderSerialPortParam(String ConnID, eBaudrate baudRate)

Parameter	// ConnID: connection identification
	// baudRate:
	eBaudrate._9600bps,
	eBaudrate._19200bps,
	eBaudrate._115200bps,
	eBaudrate._230400bps,
	eBaudrate._460800bps.

Return	0 means succeeded, other means failed.
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	if(RFIDReader._Config.SetReaderSerialPortParam(ConnID, eBaudrate._115200bps) != 0){
	System.out.println("failed! ");
	}else{
	System.out.println("Success! ");
	}
	}else{
	System.out.println("error! ");
	}

2.2.7 Serial port configuration query

Namespace	RFIDReader._Config
Function	static eBaudrate GetReaderSerialPortParam(String ConnID)
Parameter	// ConnID: Connection identification
Return	eBaudrate._9600bps, eBaudrate._19200bps, eBaudrate._115200bps, eBaudrate._230400bps, eBaudrate._460800bps.
Remark	None

Name space	RFIDReader._Config
Function	static String GetReaderSerialPortParam2(String ConnID)
Parameter	// ConnID: connection identification
Return	9600 bps,19200 bps,115200 bps,230400 bps,460800 bps
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetReaderSerialPortParam2(ConnID));
	}else{
	System.out.println("error! ");
	}

2.2.8 MAC Address Setting

Namespace	RFIDReader._Config
Function	static int SetReaderMacParam(String ConnID, String param)
Parameter	// ConnID: connection identification // param: MAC address format "00-00-00-00-00-00"
Return	0 means succeed, other value means failed.
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	if(RFIDReader._Config.SetReaderMacParam(ConnID, "6E-7A-1C-AA-FF-0B") != 0){
	System.out.println("failed! ");
	}else{
	System.out.println("Success! ");
	}
	}else{

	System.out.println("error! "); }

2.2.9 MAC Address Query

Namespace	RFIDReader._Config
Function	static String GetReaderMacParam(String ConnID)
Parameter	// ConnID: connection identification
Return	MAC address
Remark	MAC address format "00-00-00-00-00-00"

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetReaderMacParam(ConnID));
	}else{
	System.out.println("error! ");
	}

2.2.10 RS485 Address Setting

Namespace	RFIDReader._Config
Function	static int SetReader485(String ConnID, String param)
Parameter	// ConnID: connection identification // param: 0~255
Return	0 means succeed, other means failed.
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	if(RFIDReader._Config.SetReader485(ConnID, "2") != 0){
	System.out.println("failed! ");
	}else{
	System.out.println("Success! ");
	}
	}else{
	System.out.println("error! ");
	}

2.2.11 RS485 Address Query

Namespace	RFIDReader._Config
Function	static String GetReader485(String ConnID)

Parameter	// ConnID: connection identification
Return	RS485 address
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetReader485(ConnID));
	}else{
	System.out.println("error! ");
	}

2.2.12 DHCP Setting

Namespace	RFIDReader._Config
Function	**public intSetDHCP(String connID, Boolean param)**
Parameter	// ConnID: connection identification // param： true is ON, false is OFF
Return	0 means succeed, other means failed.
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.SetDHCP(ConnID, true));
	}else{
	System.out.println("Failed to create connection!");
	}

2.2.13 DHCP Query

Namespace	RFIDReader._Config
Function	**public Boolean GetDHCP(String ConnID)**
Parameter	// ConnID: connection identification
Return	false means OFF, true means ON
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetDHCP(ConnID));
	}else{
	System.out.println("Failed to create connection!");
	}

2.2.14 Server/Client Mode Setting

Namespace	RFIDReader._Config
Function	static int SetReaderServerOrClient(String ConnID, eWorkMode workMode,String ip,String port)

Parameter	// ConnID: connection identification
	// workMode: eWorkMode.Server , eWorkMode.Client
	ip: e.g. "192.168.1.1"
	port: e.g. "9090"

Return	0 means succeed, other means failed.
Remark	When set the reader to server mode, ip parameter is invalid, you can enter any string.

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	//Client Mode: eWorkMode.Client TCP server mode: eWorkMode.Server
	if(RFIDReader._Config.SetReaderServerOrClient(ConnID, eWorkMode.Client, "192.168.1.12", "8081") != 0){
	System.out.println("failed! ");
	}else{
	System.out.println("Success! ");
	}
	}else{
	System.out.println("error! ");
	}

2.2.15 Server/Client Mode Query

Namespace	RFIDReader._Config
Function	static String GetReaderServerOrClient(String ConnID)
Parameter	// ConnID: connection identification
Return	Server|"Server port" or Client|"Client IP"|"Client port"
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetReaderServerOrClient(ConnID));
	}else{
	System.out.println("error! ");
	}

2.2.16 Reader Information Query

Namespace	RFIDReader._Config

Function	static String GetReaderInformation(String ConnID)
Parameter	// ConnID: connection identification
Return	Embedded Application software version| reader name |reader power-up time
Remark	The read - on time unit is seconds

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetReaderInformation(ConnID));
	}else{
	System.out.println("error! ");
	}

2.2.17 Baseband Software Version Query

Namespace	RFIDReader._Config
Function	static String GetReaderBaseBandSoftVersion(String ConnID)
Parameter	// ConnID: connection identification
Return	Baseband software version number
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetReaderBaseBandSoftVersion(ConnID));
	}else{
	System.out.println("error! ");
	}

2.2.18 Antenna standing wave ratio query

Namespace	RFIDReader._Config
Function	static String GetAntennaStandingWaveRatio(String ConnID, int antNo)
Parameter	// ConnID: connection identification, antNo: Enumeration of antenna Number
Return	Forward power measurement value | backward power measurement value
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetAntennaStandingWaveRatio(ConnID,1));
	}else{
	System.out.println("error! ");
	}

2.2.19 Get data output format

Namespace	RFIDReader._Config
Function	**public String GetDataOutPutFormat (String ConnID)**
Parameter	// ConnID: connection identification
Return	Custom output switch|output format|output data type|frame header data|frame end data
Remark	None

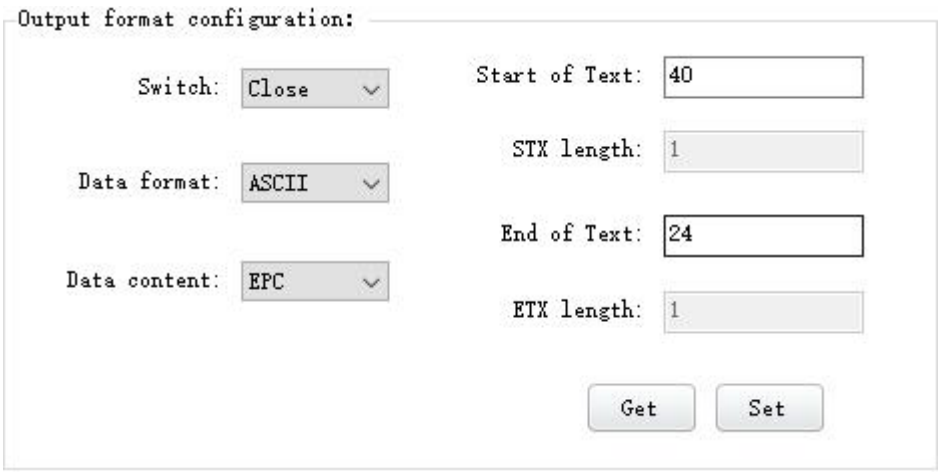
Sample code	String ConnID = **"192.168.1.116:9090"**;
	IAsynchronousMessage log = **new** SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.**out**.println(RFIDReader._Config.GetDataOutPutFormat(ConnID));
	}**else**{
	System.**out**.println(**"Failed to create connection!"**);
	}

2.2.20 Set data output format

Namespace	RFIDReader._Config
Function	**public String SetDataOutPutFormat (string connID,string swith,string outPutFormat,string outDataType,string startData,string endData,string dataStartByte,string dataLen)**
Parameter	**public String SetDataOutPutFormat (string connID,string swith,string outPutFormat,string outDataType,string startData,string endData)**

Return	// ConnID: connection identification
	// switch: Custom output switch. 0-close, 1-open, 2-UDP output
	// outPutFormat:output format, 0-hex, 1-ASCII
	//outDataType: output data type, 0-EPC, 1-TID
	//startData: Start of Text
	//endData: End of Text
	//dataStartByte: Tag data start byte
	//dataLen: Tag data length

Remark	0 means succeed, other means failed.

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if (RFIDReader.CreateTcpConn(ConnID , log))
	{
	String rt = RFIDReader._Config.SetDataOutPutFormat(ConnID ,"1","1","0","E200","7206", “0”, “12”);
	//Custom output switch|output format|output data type|Stat of Text|End of Text
	System.out.println(rt);
	}

2.2.21 Restart the reader

Namespace	RFIDReader._Config
Function	**public String ReSetReader(String ConnID)**
Parameter	// ConnID: connection identification
Return	None
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	RFIDReader._ReaderConfig.ReSetReader(ConnID);
	System.out.println(“Restarting the reader”);
	}else{
	System.out.println("Failed to create connection!");
	}

2.2.22 Get reader temperature

Namespace	RFIDReader._Config

Function	**public String GetReaderTemperature (String ConnID)**
Parameter	// ConnID: connection identification
Return	reader temperature
Remark	None

Sample code	String ConnID = **"192.168.1.116:9090"**;
	IAsynchronousMessage log = **new** SampleCode();
	if (RFIDReader.CreateTcpConn(ConnID , log))
	{
	String rt = RFIDReader._Config.GetReaderTemperature (ConnID);
	System.**out**.println(rt);
	}

2.2.23 Set custom reader id

Namespace	**RFIDReader**._Config
Function	**public String SetCustomCode (String ConnID:)**
Parameter	// ConnID: connection identification // param: String to be set
Return	result
Remark	None

Sample code	String ConnID = **"192.168.1.116:9090"**;
	IAsynchronousMessage log = **new** SampleCode();
	if (RFIDReader.CreateTcpConn(tcp, log))
	{
	String rt = RFIDReader._Config.SetCustomCode(ConnID, “cust_code”);
	System.**out**.println(rt);
	}

2.2.24 Get custom reader id

Namespace	**RFIDReader**._ Config
Function	**public String GetCustomCode (String ConnID)**
Parameter	// ConnID: connection identification
Return	Customised reader ID
Remark	None

Sample code	String ConnID = **"192.168.1.116:9090"**;
	IAsynchronousMessage log = **new** SampleCode();
	if (RFIDReader.CreateTcpConn(tcp, log))
	{
	String rt = RFIDReader._Config.GetCustomCode(ConnID);
	System.**out**.println(rt);
	}

2.2.25 Set Buzzer Control

Namespace	RFIDReader._Config
Function	**public String SetBuzzerControl(String ConnID, String param)**
Parameter	// ConnID: connection identification // param: Refer to the description of the sample code below
Return	0 means succeed, other means failed.
Remark	None
Sample code	/// /// Set Buzzer Control /// /// eg:0001 ==> byte0 (Whether to turn on the buzzer) 00:Buzzer stop, 01:ring byte1(Does it keep ringing) 00: The buzzer rings once, 01:Keeps ringing `RFIDReader._Config.SetBuzzerControl(ConnID, "0100");//Set the buzzer to ring once`

2.2.26 Set Buzzer Switch

Namespace	RFIDReader._ Config
Function	**public String SetBuzzerSwitch (String ConnID,String param)**

Parameter	// ConnID: connection identification
	// param: 0 -- Reader control, that is, the buzzer will sound once the reader reads a tag,
	1 -- PC control is controlled through the SetBuzzerControl interface

Return	0 means succeed, other means failed.
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if (RFIDReader.CreateTcpConn(tcp, log))
	{
	String rt = RFIDReader._Config.SetBuzzerSwitch(ConnID, "1");
	System.out.println(rt);
	}

2.2.27 Get Reader SN

Namespace	RFIDReader._Config
Function	**public String GetSN(String ConnID)**
Parameter	// ConnID: connection identification
Return	Reader ID
Remark	None
Sample code	String ConnID = "192.168.1.116:9090";

	IAsynchronousMessage log = **new** SampleCode();
	if (RFIDReader.CreateTcpConn(tcp, log))
	{
	String rt = RFIDReader._Config.GetSN(ConnID);
	System.**out**.println(rt);
	}

2.3 RFID Configuration

2.3.1 Restore Factory Setting

Package	RFIDReader._Config
Function	static int SetReaderRestoreFactory(String ConnID)
Parameter	// ConnID: connection identification
Return	0 means succeed, other value means failed.
Remark	Restore all settings to factory state.

Sample code	String ConnID = **"192.168.1.116:9090"**;
	IAsynchronousMessage log = **new** SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	if(RFIDReader._Config.SetReaderRestoreFactory(ConnID) != 0){
	System.**out**.println(**"error"**);
	}**else**{
	System.**out**.println(**"Factory reset succeeded! "**);
	}
	}**else**{
	System.**out**.println(**"error! "**);
	}

2.3.2 Baseband Parameter Setting

Package	RFIDReader._Config
Function	static int SetEPCBaseBandParam(String ConnID,int basebandMode,int qValue,int session,int searchType)

Parameter	// ConnID: connection identification
	// basebandMode: EPC Baseband rate (0~255, 255 means AUTO)
	(0-Tari=25us, FM0, LHF=40KHz)
	(1-Tari=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: Inventory flag parameter (0 only with Flag A inventory, 1 only Flag B inventory, 2 turns using with Flag A and Flag B double-sided inventory).

Return	0 means succeed, other value means failed.
Remark	

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	  if(RFIDReader._Config.SetEPCBaseBandParam(ConnID, 255, 4, 1, 2) != 0){
	  System.out.println("failed! ");
	  }else{
	  System.out.println("Success! ");
	  }
	}else{
	  System.out.println("error! ");
	}

2.3.3 Baseband Parameter Query

Namespace	RFIDReader._Config
Function	static String GetEPCBaseBandParam(String ConnID)
Parameter	// ConnID: connection identification
Return	basebandMode|qValue|session|searchType

Remark	// basebandMode: EPCBaseband rate(0~255, 255 means AUTO)
	(0-Tari=25us, FM0, LHF=40KHz)
	(1-Tari=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: Inventory flag parameters (0 only with Flag A inventory, 1 only Flag B inventory, 2 turns using with Flag A and Flag B double-sided inventory).

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	  System.out.println(RFIDReader._Config.GetEPCBaseBandParam(ConnID));
	}else{
	  System.out.println("error! ");
	}

2.3.4 Antenna Power Setting

Package	RFIDReader._Config
Function	static int SetANTPowerParam(String ConnID, HashMap<Integer, Integer> dicPower)
Parameter	// ConnID: connection identification // dicPower: Antenna number and power level key value pairs
Return	0 means succeed, other value means failed.
Remark	Set the power of each antenna of reader

Sample code	// Set the power level of antenna 1 and antenna 2 of the reader to 30
	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	HashMap<Integer, Integer> dicPower = new HashMap<Integer, Integer>();
	dicPower.put(1, 30);
	dicPower.put(2, 30);
	if(RFIDReader._Config. SetANTPowerParam (ConnID, dicPower) != 0){
	System.out.println("failed! ");
	}else{
	System.out.println("Success! ");
	}
	}else{
	System.out.println("error! ");
	}

2.3.5 Antenna Power Query

Namespace	RFIDReader._Config
Function	static String GetANTPowerParam(String ConnID)
Parameter	// ConnID: connection identification
Return	1,Power of antenna 1 & 2,Power of antenna 2 & 3,Power of antenna 3 & 4,Power of antenna 4
Remark	

Namespace	RFIDReader._Config
Function	static String GetANTPowerParam2(String ConnID)
Parameter	// ConnID: connection identification
Return	1,Power of antenna 1 & 2,Power of antenna 2 & 3,Power of antenna 3 & 4,Power of antenna 4
Remark	

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetANTPowerParam2(ConnID));
	}else{

	System.out.println("error! "); }

2.3.6 Tag Upload Parameter Setting

Package	RFIDReader._Config
Function	static int SetTagUpdateParam(String ConnID, int repeatTimeFilter, int RSSIFilter)

Parameter	// ConnID: connection identification
	// repeatTimeFilter: Duplicated tag upload filter time
	// RSSIFilter: RSSI filter

Return	0 means succeed, other value means failed.
Remark	repeatTimeFilter value range 0 ~ 65535 RSSIFilter value range 0 ~ 255

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	if(RFIDReader._Config.SetTagUpdateParam(ConnID, 10, 0) != 0){
	System.out.println("failed! ");
	}else{
	System.out.println("Success! ");
	}
	}else{
	System.out.println("error! ");
	}

2.3.7 Tag upload parameter query

Name space	RFIDReader._Config
Function	static String GetTagUpdateParam(String ConnID)
Parameter	// ConnID: connection identification
Return	repeatTimeFilter|RSSIFilter。

Remark	// repeatTimeFilter: Duplicated tag upload filter time (Unit:10ms)
	// RSSIFilter: RSSI Filter
	repeatTimeFilter Value range 0 ~ 65535
	RSSIFilter Value range 0 ~ 255

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetTagUpdateParam(ConnID));
	}else{
	System.out.println("error! ");

	}

2.3.8 Reader read and write capabilities

Name space	RFIDReader._Config
Function	static String GetReaderProperty(String ConnID)
Parameter	// ConnID: connection identification
Return	Minimum transmit power | maximum transmit power | number of antennas | band list | list of RFID protocols
Remark	Output unit is dB

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetTagUpdateParam(ConnID));
	}else{
	System.out.println("error! ");
	}

2.3.9 RF Band setting

Name space	RFIDReader._Config
Function	static int SetReaderRF(String ConnID, eRF_Range eRF_Range)

Parameter	// ConnID: connection identification
	// eRF_Range:
	eRF_Range.GB_920_to_925MHz
	eRF_Range.GB_840_to_845MHz
	eRF_Range.GB_920_to_925MHz_and_GB_840_to_845MHz
	eRF_Range.FCC_902_to_928MHz
	eRF_Range.ETSI_866_to_868MHz

Return	0 means succeed, other value means failed.
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	if(RFIDReader._Config.SetReaderRF(ConnID, eRF_Range.GB_920_to_925MHz) != 0){
	System.out.println("failed! ");
	}else{
	System.out.println("Success! ");
	}
	}else{
	System.out.println("error! ");
	}

2.3.10 RF Band query

Name space	RFIDReader._Config
Function	static eRF_Range GetReaderRF(String ConnID)
Parameter	// ConnID: connection identification
Return	eRF_Range.GB_920_to_925MHz eRF_Range.GB_840_to_845MHz eRF_Range.GB_920_to_925MHz_and_GB_840_to_845MHz eRF_Range.FCC_902_to_928MHz eRF_Range.ETSI_866_to_868MHz eRF_Range.JP_916_to_921MHz eRF_Range.TW_922_to_927MHz eRF_Range.ID_923_to_925MHz eRF_Range.RUS_866_to_867MHz
Remark	None

Name space	RFIDReader._Config
Function	static String GetReaderRF2(String ConnID)
Parameter	// ConnID: connection identification
Return	GB_920_to_925MHz，GB_840_to_845MHz，GB_920_to_925MHz_and_GB_840_to_845MHz，FCC_902_to_928MHz，ETSI_866_to_868MHz，JP_916_to_921MHz，TW_922_to_927MHz，ID_923_to_925MHz，RUS_866_to_867MHz
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetReaderRF2(ConnID));
	}else{
	System.out.println("error! ");
	}

2.3.11 RF Frequency setting

Name space	RFIDReader._Config
Function 1	static Int32 SetReaderWorkFrequency_GB920_to_925MHz(String ConnID, eWF_Mode wfMode, List<eGB920_to_925MHz> ListGB920_to_925MHz)
Function 2	static Int32 SetReaderWorkFrequency_GB_840_to_845MHz(String ConnID, eWF_Mode wfMode, List<eGB_840_to_845MHz> ListGB_840_to_845MHz)
Function 3	static Int32 SetReaderWorkFrequency_GB_920_to_925MHz_and_GB_840_to_845MHz(String ConnID, eWF_Mode wfMode, List<eGB_920_to_925MHz_and_GB_840_to_845MHz> ListGB_920_to_925MHz_and_GB_840_to_845MHz)
Function 4	static Int32 SetReaderWorkFrequency_FCC_902_to_928MHz(String ConnID, eWF_Mode

	wfMode, List<eFCC_902_to_928MHz> ListFCC_902_to_928MHz)
Function 5	static Int32 SetReaderWorkFrequency_ETSI_866_to_868MHz(String ConnID, eWF_Mode wfMode, List<eETSI_866_to_868MHz> ListETSI_866_to_868MHz)
Function 6	static Int32 SetReaderWorkFrequency_JP_916_to_921MHz(String ConnID, eWF_Mode wfMode, List<eJP_916_to_921MHzz> ListJP_916_to_921MHz)
Function 7	static Int32 SetReaderWorkFrequency_TW_922_to_927MHz(String ConnID, eWF_Mode wfMode, List<eTW_922_to_927MHz> ListTW_922_to_927MHz)
Function 8	static Int32 SetReaderWorkFrequency_ID_923_to_925MHz(String ConnID, eWF_Mode wfMode, List<eID_923_to_925MHz> ListID_923_to_925MHz)
Function 9	static Int32 SetReaderWorkFrequency_RUS_866_to_867MHz(String ConnID, eWF_Mode wfMode, List<eRUS_866_to_867MHz> ListRUS_866_to_867MHz)
Parameter	// ConnID: connection identification
Return	0 means succeed, other value means failed.
Remark	None

2.3.12 RF Frequency query

Name space	RFIDReader._Config
Function	static String GetReaderWorkFrequency(String ConnID)
Parameter	// ConnID: connection identification
Return	Mode|frequency|frequency point value Like: Auto|GB_920_to_925MHz|920.625,920.875
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetReaderWorkFrequency(ConnID));
	}else{
	System.out.println("error! ");
	}

2.3.13 Reader automatically idle mode setting

Name space	RFIDReader._Config
Function	static int SetReaderAutoSleepParam(String ConnID, BooleanSwitch, String time)
Parameter	// ConnID: connection identification, Switch:ON/OFF, time:idle time
Return	0 means succeed, other value means failed.
Remark	Switch: true is ON, false is OFF idle time unit is 10ms
Sample code	String ConnID = "192.168.1.116:9090"; IAsynchronousMessage log = new SampleCode();

	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	  if(RFIDReader._Config.SetReaderAutoSleepParam(ConnID, true,"100") != 0){
	  System.out.println("failed! ");
	  }else{
	  System.out.println("Success! ");
	  }
	}else{
	  System.out.println("error! ");
	}

2.3.14 Reader automatically idle mode query

Name space	RFIDReader._Config
Function	static String GetReaderAutoSleepParam(String ConnID)
Parameter	// ConnID: connection identification
Return	Close or Open|"idle time"
Remark	idle time unit is 10ms

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	  System.out.println(RFIDReader._Config.GetReaderAutoSleepParam(ConnID));
	}else{
	  System.out.println("error! ");
	}

2.3.15 Antenna enable setting

Name space	RFIDReader._Config
Function	static int SetReaderANT(String ConnID, int antNum)
Parameter	// ConnID: Connection mark, antNum: Antenna number enumeration
Return	0 means succeed, other value means failed.
Remark	Specify antenna 1 and antenna 2 together, example eAntennaNo._1|eAntennaNo._2

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	  if(RFIDReader._Config.SetReaderANT(ConnID, 3) != 0){
	  System.out.println("failed! ");
	  }else{
	  System.out.println("Success! ");
	  }
	}else{
	  System.out.println("error! ");

	}

2.3.16 Antenna enabled query

Name space	RFIDReader._Config
Function	static int GetReaderANT(String ConnID)
Parameter	// ConnID: connection identification
Return	The sum of the each enabled antenna value, refer to 2.12
Remark	None

Name space	RFIDReader._Config
Function	static String GetReaderANT2(String ConnID)
Parameter	// ConnID: connection identification
Return	The antenna port number that has enabled, multiple antenna numbers separate with "," like 1,6,8
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetReaderANT2(ConnID));
	}else{
	System.out.println("error! ");
	}

2.3.17 Get RFID Temperature

Name space	RFIDReader._Config
Function	static String GetRFIDTemperature (String ConnID)
Parameter	// ConnID: connection identification
Return	Temperature
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if (RFIDReader.CreateTcpConn(tcp, log))
	{
	String rt = RFIDReader._Config.GetRFIDTemperature (ConnID);
	System.out.println(rt);
	}

2.3.18 Set Expand EPC Baseband Param

Name space	RFIDReader._Config
Function	**public String SetEPCBaseExpandBandParam(String ConnID, String param)**

Parameter	// ConnID: connection identification
	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
	bit31-bit6: rfu
	// Parameter 2:
	Byte 1: maxQ
	Byte 2: minQ
	Byte 3: tmult
	Byte 4: bit 0 Dynamic start Q 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
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if (RFIDReader.CreateTcpConn(tcp, log))
	{
	String rt = RFIDReader._Config.SetEPCBaseExpandBandParam (ConnID,"1,00000000&2,00000000");
	System.out.println(rt);
	}

2.3.19 Get Expand EPC Baseband Param

Name space	RFIDReader._Config
Function	**public String GetEPCBaseExpandBandParam(String ConnID)**
Parameter	// ConnID: connection identification

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
	bit31-bit6: rfu
	// Parameter 2:
	Byte 1: maxQ
	Byte 2: minQ
	Byte 3: tmult
	Byte 4: bit 0 Dynamic start Q 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

Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if (RFIDReader.CreateTcpConn(tcp, log))
	{
	String rt = RFIDReader._Config.GetEPCBaseExpandBandParam (ConnID,);
	System.out.println(rt);
	}

2.4 GPIO Operation

2.4.1 GPI Trigger parameter configuration

Namespace	RFIDReader._Config
Function	static Int32 SetReaderGPIParam(String ConnID, eGPI GPINum, eTriggerStart triggerStart,eTriggerCode triggerCode, eTriggerStop triggerStop, String DelayTime,Boolean isUpload)

Parameter	// ConnID: connection identification
	
	// GPINum: eGPI._1,eGPI._2,eGPI._3,eGPI._4;
	
	//triggerStart: eTriggerStart.OFF,eTriggerStart.Low_level,eTriggerStart.High_level,
	eTriggerStart.Rising_edge,eTriggerStart.Falling_edge,eTriggerStart.Any_edge;
	
	//triggerCode: triggerCode.Single_Antenna_read_EPC,
	triggerCode.Single_Antenna_read_EPC_and_TID,
	triggerCode.Double_Antenna_read_EPC,
	triggerCode.Double_Antenna_read_EPC_and_TID,
	triggerCode.Four_Antenna_read_EPC,
	triggerCode.Four_Antenna_read_EPC_and_TID;
	
	//triggerStop: eTriggerStop.OFF,eTriggerStop.Low_level,eTriggerStop.High_level,
	eTriggerStop.Rising_edge,eTriggerStop.Falling_edge,eTriggerStop.Any_edge,
	eTriggerStop.Delay;
	
	//isUpload: when the triggerStop is etriggerStop.off, whether upload the sensor status. Ture means upload, false means do not upload.
	Eg: SetReaderGPIParam("192.168.1.116:9090",eGPI._2,eTriggerStart.Low_level,
	triggerCode.Double_Antenna_read_EPC_and_TID,eTriggerStop.Dealy,"100",true);
	
	The current method configures the No. 2 GPI port triggers at low level and executes the trigger code,Delay after 1000ms. Now the triggerStop is not OFF, parameter isUpload is invalid,it's no effect no matter you fill either true or false

Return	0 means succeed, other value means failed.
Remark	● Stop the delay time:unit is 10ms (Valid when the trigger stop condition is delay) For details, please refer to "RFID Reader Management Software" For specific GPI trigger parameter callback , pls refer to Callback interface GPIControlMsg();
Sample code	String ConnID = **"192.168.1.116:9090"**; IAsynchronousMessage log = **new** SampleCode(); **if**(RFIDReader.CreateTcpConn(ConnID, log)) {

	if(RFIDReader._Config.SetReaderGPIParam(ConnID,eGPI._1,eTriggerStart.High_level,
	eTriggerCode.Double_Antenna_read_EPC_and_TID,eTriggerStop.OFF,"",true) != 0){
	System.out.println("failed! ");
	}else{
	System.out.println("Success! ");
	}
	}else{
	System.out.println("error! ");
	}

2.4.2 GPI Trigger parameter query

Name space	RFIDReader._Config
Function	static String GetReaderGPIParam(String ConnID,eGPI GPINum)

Parameter	// ConnID: connection identification
	// GPONum： GPI._1,GPI._2,GPI._3,GPI._4
	Eg: GetReaderGPIParam("192.168.1.116:9090",GPI._1);

Return	" GPI port number | Trigger start condition | Trigger execution instruction | Trigger stop condition | Stop delay time|upload flag" Eg： "GPI1|Low level|Double Antenna read EPC|Delay|100|OFF"
Remark	● Stop the delay time:unit is 10ms (Valid when the trigger stop condition is delay) Upload flag: ON means upload, OFF means do not upload. For details, please refer to "RFID Reader Management Software"

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetReaderGPIParam(ConnID,eGPI._1));
	}else{
	System.out.println("error! ");
	}

2.4.3 GPI Status query

Name space	RFIDReader._Config
Function	static String GetReaderGPIState(String ConnID)
Parameter	// ConnID: connection identification Eg: GetReaderGPIState("192.168.1.116:9090");
Return	Eg: "1,Low & 2,High", This return value means: # 1 GPI port is currently at low level, # 2 GPI port is currently at a high level.
Remark	The method is to actively get the current GPI level status, unrelated with trigger event! For specific GPI trigger parameter callback , pls refer to Callback interface_ GPIControlMsg();
Sample code	String ConnID = "192.168.1.116:9090";

	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.out.println(RFIDReader._Config.GetReaderGPIState(ConnID));
	}else{
	System.out.println("error! ");
	}

2.4.4 GPO Level operation

Name space	RFIDReader._Config
Function	static Int32 SetReaderGPOState(String ConnID, HashMap<eGPO, eGPOState> dicState)

Parameter	// ConnID: connection identification
	// dicState: GPO Number and level corresponding key value pairs
	Eg: SetReaderGPOState("192.168.1.116:9090",new HashMap<eGPO, eGPOState> {{1,0},{2,1}});
	This method is to set the #1 GPO port to Low(0)level and set #2 GPO port to High(1)level .

Return	0 means succeed, other value means failed.
Remark	None

Sample code	// Set GPO port 1 and 2 to high level.
	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	HashMap<eGPO, eGPOState> dicState = new HashMap<eGPO, eGPOState>();
	dicState.put(eGPO._1, eGPOState._High);
	dicState.put(eGPO._2, eGPOState._High);
	if (RFIDReader._Config.SetReaderGPOState(ConnID, dicState) != 0) {
	System.out.println("failed!");
	} else {
	System.out.println("Success!");
	}
	
	}else{
	System.out.println("error! ");
	}

2.4.5 Wiegand communication parameter setting

Name space	RFIDReader._Config
Function	static int SetReaderWG(String ConnID, eWiegandSwitch wiegandSwitch, eWiegandFormat wiegandFormat, eWiegandDetails param)

Parameter	// ConnID: connection identification
	// eWiegandSwitch:
	eWiegandSwitch.Close,

	eWiegandSwitch.Open.
	// eWiegandFormat:
	eWiegandFormat.Wiegand26,
	eWiegandFormat.Wiegand34,
	eWiegandFormat.Wiegand66
	// eWiegandDetails:
	eWiegandDetails.end_of_the_EPC_data,
	eWiegandDetails.end_of_the_TID_data.
	
	Eg: SetReaderWG("192.168.1.116:9090",eWiegandSwitch.Open,
	eWiegandFormat.Wiegand26,eWiegandDetails.end_of_the_TID_data);
	The method is to turn the Wiegand switch on, communication format is Wiegand26 ,Specifies to transmit TID end data

Return	0 means succeed, other value means failed.
Remark	None

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	if(RFIDReader._Config.SetReaderWG(ConnID, eWiegandSwitch.Open,
	eWiegandFormat.Wiegand26, eWiegandDetails.end_of_the_EPC_data) != 0){
	System.out.println("failed! ");
	}else{
	System.out.println("Success! ");
	}
	}else{
	System.out.println("error! ");
	}

2.4.6 Get Wiegand communication parameter

Namespace	RFIDReader._Config
Function	static String GetReaderWG(String ConnID)
Parameter	// ConnID: connection identification
Return value	// Return value: Wiegand function switch | format | data content Eg: If return value is"Open|Wiegand66|end_of_the_EPC_data" The return value indicates that the Wiegand switch is open， communication format is Wiegand 66, Specifies to transmit EPC end data.
Description	None
Sample code	String ConnID = "192.168.1.116:9090"; IAsynchronousMessage log = new SampleCode(); if(RFIDReader.CreateTcpConn(ConnID, log)) { System.out.println(RFIDReader._Config.GetReaderWG(ConnID));

	}**else**{
	System.**out**.println("**error!** ");
	}

2.5 6C Tag operation

2.5.1 read tag

Package	RFIDReader._Tag6C

Function 1	static int GetEPC(String ConnID, int antNum, eReadType readType)
	// only read EPC
	// ConnID: connection identification
	// antNum: Antenna number enumeration.
	Appoint Antenna 1 and 2 working at same time:3, Please refer to the antenna numbering parameter
	// readType: single--- read one time, inventory---Keep reading until it receives the stop command.

Function2	static int GetEPC(String ConnID, int antNum, int readType, String accessPassword) // accessPassword: Tag access password(8 Hexadecimal number string)

Function3	static int GetEPC_MatchEPC(String ConnID, int antNum, int readType, String sEPC)
	// Match EPC, Read EPC
	// sEPC : To be matched EPC value(Hexadecimal string)

Function4	static int GetEPC_MatchEPC(String ConnID, int antNum, int readType, String sEPC, int matchWordStartIndex)
	// match EPC, Read EPC
	// matchWordStartIndex: match Data starting index

Function5	static int GetEPC_MatchEPC(String ConnID, int antNum, int readType, String sEPC, int matchWordStartIndex, String accessPassword) // accessPassword: Tag access password(8 Hexadecimal number string)

Function6	static int GetEPC_MatchTID(String ConnID, int antNum, int readType, String sTID)
	// match TID, Read EPC
	// sTID: To be matched TID value(Hexadecimal string)

Function7	static int GetEPC_MatchTID(String ConnID, int antNum, int readType, String sTID, int matchWordStartIndex)
	// match TID, Read EPC
	// matchWordStartIndex: match Data starting index

Function8	static int GetEPC_MatchTID(String ConnID, int antNum, int readType, String sTID, int matchWordStartIndex, String accessPassword) // accessPassword: Tag access password(8 Hexadecimal number string)
Function9	static int GetEPC_TID(String ConnID, int antNum, int readType) // Read EPC and TID at same time
Function10	static int GetEPC_TID(String ConnID, int antNum, int readType, String accessPassword)

	// accessPassword: Tag access password(8 Hexadecimal number string)
Function11	static int GetEPC_TID_MatchEPC(String ConnID, int antNum, int readType, String sEPC) // match EPC, Read EPC and TID
Function12	static int GetEPC_TID_MatchEPC(String ConnID, int antNum, int readType, String sEPC, int matchWordStartIndex) // match EPC, Read EPC and TID
Function13	static int GetEPC_TID_MatchEPC(String ConnID, int antNum, int readType, String sEPC, int matchWordStartIndex, String accessPassword) // accessPassword: Tag access password(8 Hexadecimal number string)
Function14	static int GetEPC_TID_MatchTID(String ConnID, int antNum, int readType, String sTID) // match TID, Read EPC and TID
Function15	static int GetEPC_TID_MatchTID(String ConnID, int antNum, int readType, String sTID, int matchWordStartIndex) // match TID, Read EPC and TID
Function16	static int GetEPC_TID_MatchTID(String ConnID, int antNum, int readType, String sTID, int matchWordStartIndex, String accessPassword) // accessPassword: Tag access password(8 Hexadecimal number string)

Function17	static int GetEPC_TID_UserData(String ConnID, int antNum, int readType,int readStart, int readLen)
	// Read EPC, TID and UserData
	// readStart , read user area's starting index
	// readLen, read user area's length (unit: Word)

Function18	static int GetEPC_TID_UserData(String ConnID, int antNum, int readType,int readStart, int readLen, String accessPassword) // accessPassword: Tag access password(8 Hexadecimal number string)
Function19	static int GetEPC_TID_UserData_MatchEPC(String ConnID, int antNum, int readType,int readStart, int readLen, String sEPC) // match EPC, Read EPC,TID and UserData
Function20	static int GetEPC_TID_UserData_MatchEPC(String ConnID, int antNum, int readType,int readStart, int readLen, String sEPC, int matchWordStartIndex) // match EPC, Read EPC,TID and UserData
Function21	static int GetEPC_TID_UserData_MatchEPC(String ConnID, int antNum, int readType,int readStart, int readLen, String sEPC, int matchWordStartIndex, String accessPassword) // accessPassword: Tag access password(8 Hexadecimal number string)
Function22	static int GetEPC_TID_UserData_MatchTID(String ConnID, int antNum, int readType,int readStart, int readLen, String sTID) // match TID, Read EPC,TID and UserData
Function23	static int GetEPC_TID_UserData_MatchTID(String ConnID, int antNum, int readType,int readStart, int readLen, String sTID, int matchWordStartIndex) // match TID, Read EPC,TID and UserData
Function24	static int GetEPC_TID_UserData_MatchTID(String ConnID, int antNum, int readType,int readStart, int readLen, String sTID, int matchWordStartIndex, String accessPassword) // accessPassword: Tag access password(8 Hexadecimal number string)
Function25	**public int GetEPC_ReservedData(String ConnID, eAntennaNo antNum, eReadType**

	readType, int readStart, int readLen)
	//Read EPC and password
	// ConnID: connection identification
	// antNum: Antenna number enumeration.
	Appoint Antenna 1 and 2 working at same time:3, Please refer to the antenna numbering parameter
	// readType: single--- read one time, inventory---Keep reading until it receives the stop command.
	// readStart: Reserved area read start address (0 or 2)
	// readLen: Reserved area read length (max 4)

Function26	**public int GetEPC_ReservedData(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String accessPassword)** // accessPassword: Tag access password(8 Hexadecimal number string)

Function27	**public int GetEPC_ReservedData_MacthEPC(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String sEPC)**
	// match EPC to read EPC, Reserved data
	// sEPC: To be matched EPC value(Hexadecimal string)

Function28	**public int GetEPC_ReservedData_MacthEPC(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String sEPC, String accessPassword)** // accessPassword: Tag access password(8 Hexadecimal number string)

Function29	**public int GetEPC_ReservedData_MacthTID(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String sTID, int matchWordStartIndex)**
	// match TID to read EPC, Reserved data
	// sTID: To be matched TID value(Hexadecimal string)
	// matchWordStartIndex: match Data starting index

Function30	**static 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 Hexadecimal number string)
Function31	**public int GetEPC_ReservedData_TID(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen)** // Read EPC, Reserved data and TID
Function32	**public int GetEPC_ReservedData_TID(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String accessPassword)** // accessPassword: Tag access password(8 Hexadecimal number string)

Function33	**public int GetEPC_ReservedData_TID_MacthEPC(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String sEPC)**
	// match EPC to read EPC, Reserved data and TID
	// sEPC: To be matched EPC value(Hexadecimal string)

Function34	**public int GetEPC_ReservedData_TID_MacthEPC(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String sEPC , String accessPassword)** // accessPassword: Tag access password(8 Hexadecimal number string)
Function35	**public int GetEPC_ReservedData_TID_MacthTID(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String sTID, int matchWordStartIndex)**

	// match TID to read EPC, Reserved data and TID
	// sTID: To be matched TID value(Hexadecimal string)
	// matchWordStartIndex: match Data starting index

Function36	**public int GetEPC_ReservedData_TID_MacthTID(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String sTID, int matchWordStartIndex , String accessPassword)** // accessPassword: Tag access password(8 Hexadecimal number string)

Function37	**public int GetRFMicron_Temperature(String ConnID, int antNo, int readType)**
	// Read RFMicron S3 temperature tag
	// After reading the tag data, it is converted into a temperature value through the following interface
	public float RFMicron_ConvertTemperature(EPCModel model)

Function38	**public int GetCAB_Temperature(String ConnID, int antNo, int readType)**
	// Read CAB temperature tag
	// After reading the tag data, it is converted into a temperature value through the following interface
	public float CAB_ConvertTemperature(EPCModel model)

Function39	**public int GetEM_Temperature(String ConnID, int antNo, int readType)**
	// Read EM temperature tag
	// After reading the tag data, it is converted into a temperature value through the following interface
	public float EM_ConvertTemperature(EPCModel model)

Function40	**public int GetEPC_G2V2(String ConnID, int antNo, int readType, G2V2AuthenticateModel g2v2, String accessPassword)**
	//read EPC and G2V2 data
	// g2v2 EPC G2V2 Authenticate parameters
	G2V2AuthenticateModel: Set the C2G2 authentication parameters
	//_AuthMethod: TAM Authentication Method
	//_CustomData: TAM Custom Data
	//_KeyID: TAM KeyID
	//_Profile: memory parameter. 0,EPC area; 1, TID area; 2, user area
	//_Offset: Define the first address of the custom data block.
	//_BlockCount: Block Count
	//_ProtMode: Specifies the mode of operation that shall be used for the encipherment and/or protection of the custom data
	//G2V2Authenticate(): Converts the decimal parameters to a hexadecimal String, returning a String
	
	//accessPassword: Tag access password(8 Hexadecimal number string)

Function41	**public int GetEPC_G2V2(String ConnID, int antNo, int readType, G2V2AuthenticateModel g2v2)**
	//read EPC and G2V2 data
	// g2v2 EPC G2V2 Authenticate parameters

Function42	**public int GetEPC_G2V2_MatchEPC(String ConnID, int antNo, int readType,**

	G2V2AuthenticateModel g2v2, String sEPC, int matchWordStartIndex, String accessPassword)
	//match EPC to read EPC and G2V2 data
	// g2v2 EPC G2V2 Authenticate parameters
	/ sEPC: To be matched EPC value(Hexadecimal string)
	// matchWordStartIndex: match Data starting index
	//accessPassword: Tag access password(8 Hexadecimal number string)

Function43	**public int GetEPC_G2V2_MatchEPC(String ConnID, int antNo, int readType, G2V2AuthenticateModel g2v2, String sEPC, int matchWordStartIndex)**
	//match EPC to read EPC and G2V2 data
	// g2v2 EPC G2V2 Authenticate parameters
	/ sEPC:To be matched EPC value(Hexadecimal string)
	// matchWordStartIndex: match Data starting index

Function44	**public int GetEPC_G2V2_MatchTID(String ConnID, int antNo, int readType, G2V2AuthenticateModel g2v2, String sTID, int matchWordStartIndex, String accessPassword)**
	// match TID to read EPC and G2V2 data
	// g2v2 EPC G2V2 Authenticate parameters
	/ sTID: To be matched TID value(Hexadecimal string)
	// matchWordStartIndex: match Data starting index
	//accessPassword: Tag access password(8 Hexadecimal number string)

Function45	**public int GetEPC_G2V2_MatchTID(String ConnID, int antNo, int readType, G2V2AuthenticateModel g2v2, String sTID, int matchWordStartIndex)**
	// match TID to read EPC and G2V2 data
	// g2v2 EPC G2V2 Authenticate parameters
	/ sTID: To be matched TID value(Hexadecimal string)
	// matchWordStartIndex: match Data starting index

Function46	**public int GetLTU_Temperature(int antNo, int readType)**
	// Read LTU31/32 chip 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 tag data, it is converted into a temperature value through the following interface
	public float LTU_ConvertTemperature(EPCModel model)

Parameter	Please refer to Function Remark.
Return	0 means succeed, other value means failed.. Appendix A
Remark	1. Please refer to the appendix for detailed return value description 2. Using the Stop command to stop the inventory-read 3. Difference between single-read and inventory-read: single-read will automatically stop reading the tags after reading the tags for once, and the inventory-read will only stop reading the tags after calling the stop function. 4. The common feature of single and inventory read: After the last tag data is uploaded, API will notify PC side through asynchronous callback that tag upload finished. See 2.8 for details in

	the callback interface specification of OutPutTagsOver();

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	RFIDReader._Config.Stop(ConnID);
	RFIDReader._Tag6C.GetEPC_MatchTID(ConnID,1,eReadType.Inventory,
	"E28011052000308565F90157");
	
	RFIDReader._Config.Stop(ConnID);
	RFIDReader._Tag6C.GetEPC_MatchTID(ConnID,1,eReadType.Inventory,
	"E28011052000308565F90157",0);
	
	RFIDReader._Config.Stop(ConnID);
	RFIDReader._Tag6C.GetEPC_MatchTID(ConnID,1,eReadType.Inventory,
	"E28011052000308565F90157",0,"00000001");
	RFIDReader._Config.Stop(ConnID);
	}else{
	System.out.println("error! ");
	}
	
	// Tag data callback interface
	public void OutPutTags(Tag_Model arg0) {
	System.out.println("EPC: "+ arg0._EPC + " TID: " + arg0._TID + " Userdata:" +
	arg0._UserData + " ReaderName: " + arg0._ReaderName);
	}

2.5.2 Write tag

2.5.2.1 Write EPC

Package	RFIDReader._Tag6C

Function1	static int WriteEPC(String ConnID, int antNum, String sWriteData)
	// ConnID: connection identification
	// antNum: Antenna number enumeration.
	Appoint antenna 1 and 2 working at same time; e.g.: 3, Please refer to the antenna numbering parameter
	// sWriteData: data to be written (Hexadecimal string)

Function2	static int WriteEPC_MatchEPC(String ConnID, int antNum, String sWriteData, String sMatchData, int matchWordStartIndex)
	// match EPC, Write EPC
	// sMatchData, EPC data to be matched
	// matchWordStartIndex, match Data starting index

Function3	static int WriteEPC_MatchEPC(String ConnID, int antNum, String sWriteData, String

	sMatchData, int matchWordStartIndex, String accessPassword)
	// match EPC, Write EPC
	// accessPassword, Tag access password(8 Hexadecimal number string)

Function4	static int WriteEPC_MatchTID(String ConnID, int antNum, String sWriteData, String sMatchData, int matchWordStartIndex)
	// match TID, Write EPC
	// sMatchData, TID data to be matched
	// matchWordStartIndex, match Data starting index

Function5	static int WriteEPC_MatchTID(String ConnID, int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword) // match TID, Write EPC
Parameter	Please refer to Function Remark.
Return	0 means succeed, other value means failed. Appendix A
Remark	1.Suggest to use matched ID to write tag, means to use "Function4"and "Function5". 2. Please refer to the appendix for detailed return value description

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	RFIDReader._Config.Stop(ConnID);
	
	if(RFIDReader._Tag6C.WriteEPC_MatchTID(ConnID,1,"E002100SF001","E280110520003 08565F90157",0,"00000001") != 0){
	System.out.println("Failed to write to EPC! ");
	}else{
	System.out.println("Successfully written to EPC! ");
	}
	}else{
	System.out.println("error! ");
	}

2.5.2.2 Write Userdata

Package	RFIDReader._Tag6C

Function1	static int WriteUserData(String ConnID, int antNum, String sWriteData,int offset)
	// ConnID: connection identification
	// antNum: Antenna number enumeration.
	// sWriteData: data to be written (Hexadecimal string)

Function2	static int WriteUserData_MatchEPC(String ConnID, int antNum, String sWriteData, int offset,String sMatchData, int matchWordStartIndex)
	// match EPC, Write EPC
	// sMatchData, EPC data to be matched (Hexadecimal string)
	// matchWordStartIndex, match Data starting index

Function3	static int WriteUserData_MatchEPC(String ConnID, int antNum, String sWriteData, int offset,String sMatchData, int matchWordStartIndex, String accessPassword) // match EPC, Write EPC

	// accessPassword, Tag access password

Function4	static int WriteUserData_MatchTID(String ConnID, int antNum, String sWriteData, int offset,String sMatchData, int matchWordStartIndex)
	// match TID, WriteEPC
	// sMatchData, TID data to be matched (Hexadecimal string)
	// matchWordStartIndex, match Data starting index

Function5	static Int32 WriteUserData_MatchTID(String ConnID, int antNum, String sWriteData, String sMatchData, Int32 matchWordStartIndex, String accessPassword) // match TID, Write EPC
Parameter	Please refer to Function Remark.
Return	0 means succeed, other value means failed. Appendix A
Remark	1. Suggest to use matched ID to write tag, means "Function4" and "Function5". 2. Please refer to the appendix for detailed return value description

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	RFIDReader._Config.Stop(ConnID);
	if(RFIDReader._Tag6C.WriteUserData_MatchTID(ConnID,1,"E002100SF001",0,"E28011052000308565F90157",0,"00000001") != 0){
	System.out.println("Write failed! ");
	}else{
	System.out.println("Write successful! ");
	}
	}else{
	System.out.println("error! ");
	}

2.5.2.3 Write password

Package	RFIDReader._Tag6C

Function1	static int WriteAccessPassWord(String ConnID, int antNum, String sWriteData)
	// Write Tag access password
	// sWriteData: password content (8 Hexadecimal number string data)

Function2	static int WriteAccessPassWord(String ConnID, int antNum, String sWriteData, String accessPassword)
	// Write Tag access password
	// accessPassword: Original Tag access password (8 Hexadecimal number string data)

Function3	static int WriteAccessPassWord_MatchTID(String ConnID, int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)
	// Write Tag access password
	// sMatchData: TID data to be matched
	// matchWordStartIndex: match Data starting index

Function4	static int WriteDestroyPassWord(String ConnID, int antNum, String sWriteData) // Write the kill password
Function5	static int WriteDestroyPassWord(String ConnID, int antNum, String sWriteData, String accessPassword)

	// Write the destroying password // accessPassword: Original Tag access password (8 Hexadecimal number string data)

Function6	static int WriteDestroyPassWord_MatchTID(String ConnID, int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)
	// Write the kill password
	// sMatchData: TID data to be matched
	// matchWordStartIndex: match Data starting index

Parameter	Please refer to Function Remark.
Return	0 means succeed, other value means failed. Appendix A
Remark	

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	RFIDReader._Config.Stop(ConnID);
	if(RFIDReader._Tag6C.WriteAccessPassWord_MatchTID(ConnID,1,"00000002", "E28011052000308565F90157",0,"00000001") != 0){
	System.out.println("Write failed! ");
	}else{
	System.out.println("Write successful! ");
	}
	}else{
	System.out.println("error! ");
	}

2.5.3 Lock tag

Package	RFIDReader._Tag6C

Function1	static int Lock(String ConnID, int antNum, eLockArea lockArea, eLockType lockType)
	// ConnID: connection identification
	// antNum: antenna number
	// lockArea: lock area enumeration
	// lockType: lock type enumeration

Function2	static int Lock_MatchEPC(String ConnID, int antNum, eLockArea lockArea, eLockType lockType, String sMatchData, int matchWordStartIndex)
	// sMatchData: EPC data to be matched (Hexadecimal string)
	// matchWordStartIndex: match Data starting index

Function3	static int Lock_MatchEPC(String ConnID, int antNum, eLockArea lockArea, eLockType lockType, String sMatchData, int matchWordStartIndex, String accessPassword) // accessPassword: Tag access password

Function4	static int Lock_MatchTID(String ConnID, int antNum, eLockArea lockArea, eLockType lockType, String sMatchData, int matchWordStartIndex)
	// sMatchData: TID data to be matched(Hexadecimal string)
	// matchWordStartIndex: match Data starting index

Function5	static int Lock_MatchTID(String ConnID, int antNum, eLockArea lockArea, eLockType lockType,

	String sMatchData, int matchWordStartIndex, String accessPassword) // accessPassword: Tag access password
Parameter	// refer to above method Remark
Return	0 means succeed, other value means failed. Appendix A
Remark	

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	RFIDReader._Config.Stop(ConnID);
	
	if(RFIDReader._Tag6C.Lock_MatchTID(ConnID,1,eLockArea.epc,eLockType.Lock,"E28011052000308565F90157",0,"00000001") != 0){
	System.out.println("failure! ");
	}else{
	System.out.println("success! ");
	}
	}else{
	System.out.println("error! ");
	}

2.5.4 Kill Tag

Package	RFIDReader._Tag6C

Function1	static int Destroy(String ConnID, int antNum, String destroyPassword)
	// ConnID: connection identification
	// antNum: antenna number
	// destroyPassword: kill password (Hexadecimal string)

Function2	static int Destroy_MatchEPC(String ConnID, int antNum, String destroyPassword, String sMatchData, int matchWordStartIndex)
	// sMatchData: EPC data to be matched (Hexadecimal string)
	// matchWordStartIndex: match Data starting index

Function3	static int Destroy_MatchTID(String ConnID, int antNum, String destroyPassword, String sMatchData, int matchWordStartIndex)
	// sMatchData: TID data to be matched (Hexadecimal string)
	// matchWordStartIndex: match Data starting index

Parameter	// refer to above method Remark
Return	0 means succeed, other value means failed.Appendix A
Remark	

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	RFIDReader._Config.Stop(ConnID);
	
	if(RFIDReader._Tag6C.Destroy_MatchTID(ConnID,1,"00000002","E28011052000308565

	F90157",0) != 0){
	System.out.println("failure! ");
	}else{
	System.out.println("success! ");
	}
	}else{
	System.out.println("error! ");
	}

2.6 6B tag operation

2.6.1 Read tag

Package	RFIDReader._Tag6B

Function 1	static int Get6B(String ConnID, int antNum, int readType, e6BReaderContent readerContent)
	// ConnID: connection identification
	// antNum: Antenna number enumeration.
	Appoint Antenna 1 and 2 working at same time; e.g.: 3 Please refer to the antenna numbering parameter
	// ReadType: read type enumeration (single or inventory)
	// readerContent: read content enumeration

Function 2	static int Get6B_UserData(String ConnID, int antNum, int readType, e6BReaderContent readerContent, int readStart, int readLen)
	// readStart: User data area starting address
	// readLen: byte length of the user area

Function 3	static int Get6B_UserData_MatchTID(String ConnID, int antNum, int readType, e6BReaderContent readerContent, int readStart, int readLen, String sMatchData) // sMatchData: TID data to be matched
Parameter	// refer to above method Remark
Return	0 means succeed, other value means failed.
Remark	Please refer to the appendix for error code.Appendix B

2.6.2 Write tag

Package	RFIDReader._Tag6B
Function	static int Write6B(String ConnID, int antNum, String sTID, int startIndex, String sWriteData)

Parameter	// ConnID: connection identification
	// antNum: Antenna number enumeration.
	Appoint Antenna 1 and 2 working at same time; e.g.: 3 Please refer to the antenna numbering parameter
	// sTID: TID data to be matched(Hex.number string)
	// sWriteData: User data to be written((Hex.number string)

Return	0 means succeed, other value means failed.
Remark	Please refer to the appendix for error code Appendix B

2.6.3 Lock tag

Package	RFIDReader._Tag6B
Function	static int Lock6B(String ConnID, int antNum, String sTID, int lockIndex)

Parameter	// ConnID: connection identification
	// antNum: Antenna number enumeration.
	Appoint Antenna 1 and 2 working at same time; e.g.: 3 Please refer to the antenna numbering parameter
	// sTID: TID data to be matched(Hex.number string)
	// lockIndex: The starting address of the region to be locked

Return	0 means succeed, other value means failed.
Remark	Please refer to the appendix for error code Appendix B

2.6.4Lock query

Package	RFIDReader._Tag6B
Function	static int GetLock6B_State(String ConnID, int antNum, String sTID, int lockIndex)

Parameter	// ConnID: connection identification
	// antNum: Antenna number enumeration.
	Appoint Antenna 1 and 2 working at same time; e.g.: 3 Please refer to the antenna numbering parameter
	// sTID: TID data to be matched(Hex.number string)
	// lockIndex: The starting address of the region that be locked

Return	Query result (0 success, 1 failure) | query status (0 unlocked, 1 locked)
Remark	None

2.7 GB tag operation

2.7.1Read tag

Package	RFIDReader._TagGB

Function	static int GetGB(String ConnID, int antNum, int readType)
	// read only EPC
	// ConnID: connection identification
	// antNum: Antenna number enumeration.
	Appoint Antenna 1 and 2 working at same time; e.g.: 3 Please refer to the antenna numbering parameter
	// ReadType: read type enumeration (single or cyclical reading)

Parameter	refer to above method Remark
Return	0 means succeed, other value means failed.
Remark	Please refer to the appendix for error code

2.7.2 Write tag

link	RFIDReader._TagGB
Function	static int WriteGB(String ConnID, int antNum, String sWriteData)

Parameter	// ConnID: Connection identification
	// antNum: Antenna number enumeration.
	Appoint Antenna 1 and 2 working at same time; e.g.: 3 Please refer to the antenna numbering parameter
	// sTID: TID Data to be matched (Hexadecimal data string)
	// sWriteData: User data to be written((Hex.number string)

Return Value	0 means succeed, other value means failed.
Description	Please refer to the appendix for the error code

2.7.3 Lock tag

Link	RFIDReader._TagGB
Function	static int LockGB(String ConnID, int antNum, eLockAreaGB LockAreaGB, eLockTypeGB LockTypeGB)

Parameter	// ConnID: Connection identification
	// antNum: Antenna number.
	Appoint Antenna 1 and 2 working at same time; e.g.: 3 Please refer to the antenna numbering parameter

Return Value	0 means succeed, other value means failed.
Description	Please refer to the appendix for the error code

2.7.4 Destroy tag

Link	RFIDReader._TagGB
Function	static int DestoryGB(String ConnID, int antNum, String destroyPassword)

Parameter	// ConnID: Connection identification
	// antNum: Antenna number.
	Appoint Antenna 1 and 2 working at same time; e.g.: 3 Please refer to the antenna numbering parameter
	// destroyPassword: kill password

Return Value	0 means succeed, other value means failed.
Description	None

2.8 Callback interface IAsynchronousMessage description

```
// Asynchronous callback information interface
public interface IAsynchronousMessage
{
    // Output tag information callback    -- All the tag data is callback from this method  (key point)
    void WriteDebugMsg(String msg);
    void WriteLog(String msg);
    void PortConneting(String connID);
    void PortClosing(String connID);
    void OutPutTags(Models.Tag_Model tag);
    void OutPutTagsOver();
    void GPIControlMsg(GPI_model gpiModle);
}
```

Callback method	Description
WriteDebugMsg	Print API internal process debugging information.
WriteLog	API record log callback（not yet open）.
PortConneting	TCP server mode, the client connection callback. When the connection ID is obtained from the callback, the reader device can be controlled by the connection ID.
PortClosing	When the device connection is disconnected, the API will call back the connection ID, indicating that the device with the current connection ID has been disconnected
OutPutTags	Output tag information callback, whether it is a single read, cycle read or get the tag data in the cache within the reader, callback interface are the same. Note: All the tag label data are asynchronous callback in the API, do not handle the complex logic inside the callback to ensure that the cache inside the API in time to clear.
OutPutTagsOver	No matter it is single read or cycle read, after the last tag is uploaded, there will be a sync end signal upload, indicates the end of the current read tag action.
GPIControlMsg	When there is a GPI trigger event occur after the GPI trigger parameter is turned on, the function will call back the GPI port number where the current event is located, and the level status information.

2.9 Callback data Tag_Model field description

Field	Description
_ReaderName	The reader connection identification, representing the data read from which reader, example:" 192.168.1.116:9090"
_ReaderSN	The SN of the reader, which takes effect only after the interface for obtaining the reader SN is called after the connection is established
_TagType	Tag type, "6c","6b","gb" 3 types.
_EPC	Tag EPC data，hexadecimal string.

_PC	Tag PC value
_ANT_NUM	The antenna number of the tag is uploaded
_RSSI	RSSI value
_TID	Tag TID value, Hexadecimal string.
_UserData	Tag user data area value, hexadecimal string.
_TagetData	Tag password area value, hexadecimal string.

2.10 Callback data GPI_Model field description

Field	Description
_ReaderName	The reader connection identification, representing the data read from which reader, example:" 192.168.1.116:9090"
GpiIndex	GPI port no., starting from 1, 1 for GPI1, and so on.
GpiState	0 is low, 1 is high.
StartOrStop	0 starts for the trigger, 1 stops for the trigger.
UTC	Infrared trigger UTC time, byte[] type, length is 8, The first 4 is UTC seconds and the last 4 is UTC microseconds
UTC_Time	Infrared trigger UTC time, string type, format is "yyyy.MM.dd HH:mm:ss", e.g.:"1970.01.01 00:00:00"

2.11 Breakpoint resume

2.11.1 Configuration

Package	RFIDReader._Config
Function	static int SetBreakPointUpload(String ConnID, bool Switch)

Parameter	// ConnID: connection identification
	// Switch: false:close, true:open
	e.g.: SetBreakPointUpload("192.168.1.116:9090",false);
	This method is to close the breakpoint resume function

Return	0 means succeed, other value means failed.
Remark	Enable this function, the ReadTime field in the Tag Model will be effective

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	RFIDReader._Config.Stop(ConnID);
	if(RFIDReader._Config.SetBreakPointUpload(ConnID,true) != 0){
	System.out.println("failure! ");
	}else{
	System.out.println("success! ");
	}

	}**else**{
	System.**out**.println("**error!** ");
	}

2.11.2 Query

Package	RFIDReader._Config
Function	static String GetBreakPointUpload(String ConnID)
Parameter	// ConnID: connection identification e.g.: GetBreakPointUpload("192.168.1.116:9090");
Return	Open or Close
Remark	None

Sample code	String ConnID = "**192.168.1.116:9090**";
	IAsynchronousMessage log = **new** SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	RFIDReader._Config.Stop(ConnID);
	System.**out**.println(RFIDReader._Config.GetBreakPointUpload(ConnID));
	}**else**{
	System.**out**.println("**error!** ");
	}

2.11.3 Query breakpoint cache

Package	RFIDReader._Config
Function	static String GetBreakPointCacheTag(String ConnID)
Parameter	// ConnID: connection identification
Return	Success: exist cache info, Null: no cache info, Receive Over: data returned completely
Remark	When the PC and the device link layer is interrupted, the data read will be stored in the cache of the reader, (cache support max 5000 times tag reading record, if over this reading times, will use FIFO mode to iterate cache), when this method is called, reader will upload cache data when breakpoint, and the ReadTime field in the Tag Model will be effective.

Sample code	String ConnID = "**192.168.1.116:9090**";
	IAsynchronousMessage log = **new** SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	System.**out**.println(RFIDReader._Config.GetBreakPointCacheTag(ConnID));
	}**else**{
	System.**out**.println("**error!** ");
	}

2.11.4 Clear breakpoint cache

Package	RFIDReader._Config

Function	static int ClearBreakPointCache(String ConnID)
Parameter	// ConnID: connection identification
Return	0 means succeed, other value means failed.
Remark	When the PC and the device link layer is interrupted, the data read will be stored in the cache of the reader, When this method is called, the cache when the reader is interrupted will be cleared.

Sample code	String ConnID = "192.168.1.116:9090";
	IAsynchronousMessage log = new SampleCode();
	if(RFIDReader.CreateTcpConn(ConnID, log)) {
	RFIDReader._Config.Stop(ConnID);
	if(RFIDReader._Config.ClearBreakPointCache(ConnID) != 0){
	System.out.println("failure! ");
	}else{
	System.out.println("success! ");
	}
	}else{
	System.out.println("error! ");
	}

2.12 Antenna number parameter description

- Regarding the tag read, write, lock, kill operation of the antenna number parameter: antNum. The function is to specify whether an antenna or multiple antennas for a reader work.
- antNum = 1, antNum = 2, antNum = 4, antNum = 8 respectively means: antenna 1, antenna 2, antenna 3, antenna 4
- antNum = 16, antNum = 32, antNum = 64, antNum = 128 respectively means: antenna 5, antenna 6, antenna 7, antenna 8

- While specifying multiple antennas to work with antNum for their total value, for instance:

Specify antenna 1+ antenna 2 to work: antNum = eAntennaNo._1.GetNum() + eAntennaNo._2.GetNum() = 3

Specify antenna 1+ antenna 2+ antenna 3 work: antNum = eAntennaNo._1.GetNum() + eAntennaNo._2.GetNum() + eAntennaNo._3 = 7

Specify the antenna 1+ antenna 2+ antenna 3+ antenna 4 work: antNum = eAntennaNo._1.GetNum() + eAntennaNo._2.GetNum() + eAntennaNo._3.GetNum() + eAntennaNo._4.GetNum() = 15

3.Programming example

JAVA Code: read 6C tag example

```
public class Exsample implements IAsynchronousMessage  {
```

```
// exsample
public static void main(String[] args) {
	try {
		Scanner sc = new Scanner(System.in);
		String connID = "COM6:115200";
		sc.next();
		Exsample mc = new Exsample();
		// CLReader.CreateTcpConn("192.168.1.116:9090", mc);	// TCP connect
		if(CLReader.CreateSerialConn(connID, mc)){		// SerialPort Connect
			System.out.println("connection success...");
			CLReader.Stop(connID);				// Stop

		}else{
			System.out.println("connection failure!");
		}
		// CLReader._Tag6C.GetEPC_TID(connID, 1, 1);	// ant1
		// CLReader._Tag6C.GetEPC_TID(connID, 3, 1);	// ant1 + ant2
		CLReader._Tag6C.GetEPC_TID(connID, 15, 1);	// ant1 + ant2 + ant3 + ant4
		System.out.println("Reading...");
		String readKey = sc.next();
		CLReader.Stop(connID);

	} catch (Exception ee) {
		System.out.println("Exception:  " + ee.getMessage());
	}

}

@Override
public void OutPutTags(TagModel model) {
	// output tag
	System.out.println("EPC: "+ model._EPC + " TID: " + model._TID);
}
```

4.FAQ and Solution

Question	Solution
Device couldn't work normally	1. Check power light normal or not. 2. If normal, there should be some sound when power on.
Serial port couldn't work normally	1. Check connection cable connecting normal or not . 2. If conditional, use another device to check this cable normal or not. 3. Try to use RJ45 to communicate.

	4. Default baud rate: 115200.
RJ45 couldn't work normally	1. Check LED working normal or not. 2. To use Ping instruction to check cable working normal or not 3. Try serial port connection, inquiry IP correct by Demo 4. Default IP and port: "192.168.1.116:9090"

Appendix A: 6C tag operation returns the error code

Read Error code table:

Code	Remark
0	Configure: succeeded
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 Error code table:

Code	Remark
0	Write succeed
1	Antenna port parameter error
2	Choose parameter error
3	Write parameter error
4	CRC correcting error
5	Power not 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

Lock Error code table:

Code	Remark
0	Lock succeeded
1	Antenna port parameter error
2	Choose parameter error
3	Write parameter error
4	CRC correcting 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 Error code table:

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

Appendix B: 6B tag operation returns the error code

Read Error code table:

Code	Remark
0	Configure succeeded
1	Antenna port Parameter error
2	reading content parameter error
3	user data area parameter error
4	Other error

Write Error code table:

Code	Remark
0	Write succeeded
1	Antenna port parameter error
2	Write parameter error
3	Other error

Lock Error code table:

Code	Remark
0	Lock succeeded
1	Other error