

# **Hopeland RFID Reader Device- PC Development Guide --JAVA**

Editor: Paul

Shenzhen Hopeland Technologies Co., Ltd

V 3.24

<b>1. Summary</b>	<b>- 1 -</b>
1.1 Summary of content	- 1 -
1.2 Development Process	- 1 -
1.3 Applicable equipment type	- 2 -
1.4 Copyright notice	- 2 -
<b>2. Function description for APIs</b>	<b>- 2 -</b>
2.1 Connect and close	- 2 -
2.1.1 Create TCP connection	- 2 -
2.1.2 Close single connection	- 3 -
2.1.3 Close all connections	- 3 -
2.2 Device configuration	- 4 -
2.2.1 Set IP	- 4 -
2.2.2 Get Device IP	- 4 -
2.2.3 Stop Instruction	- 5 -
2.2.4 Set Reader Time	- 5 -
2.2.5 Get reader time	- 6 -
2.2.6 Set Serial Port	- 6 -
2.2.7 Get Serial Port Setting	- 7 -
2.2.8 Set MAC Address	- 7 -
2.2.9 Get MAC Address	- 8 -
2.2.10 Set RS485 Address	- 8 -
2.2.11 Get RS485 Address	- 9 -
2.2.12 Set Server/Client Mode	- 9 -
2.2.13 Get Server/Client Mode	- 10 -
2.2.14 Get Reader Information	- 10 -
2.2.15 Get Baseband Software Version	- 10 -
2.2.16 Get antenna standing wave ratio	- 11 -
2.2.17 Restart the reader	- 11 -
2.2.18 Get reader temperature	- 12 -
2.2.19 Get reader SN	- 12 -
2.2.20 Set hidden light strip switch	- 12 -
2.2.21 Set Buzzer Control	- 13 -
2.2.22 Set Buzzer Switch	- 13 -
2.3 RFID Configuration	- 14 -
2.3.1 Restore Factory Settings	- 14 -
2.3.2 Set Baseband Parameters	- 14 -
2.3.3 Get Baseband Parameter	- 15 -
2.3.4 Set Antenna Power	- 16 -
2.3.5 Get Antenna Power	- 16 -
2.3.6 Set Tag Upload Parameters	- 17 -
2.3.7 Get Tag upload parameters	- 17 -
2.3.8 Get Reader Property	- 18 -
2.3.9 Set RF Band	- 18 -
2.3.10 Get RF Band	- 19 -
2.3.11 Set RF Frequency points	- 19 -
2.3.12 Get RF Frequency points	- 20 -
2.3.13 Set Reader Automatically Idle Mode	- 21 -
2.3.14 Get Reader Automatically Idle Mode	- 21 -
2.3.15 Set Antenna Enable	- 21 -

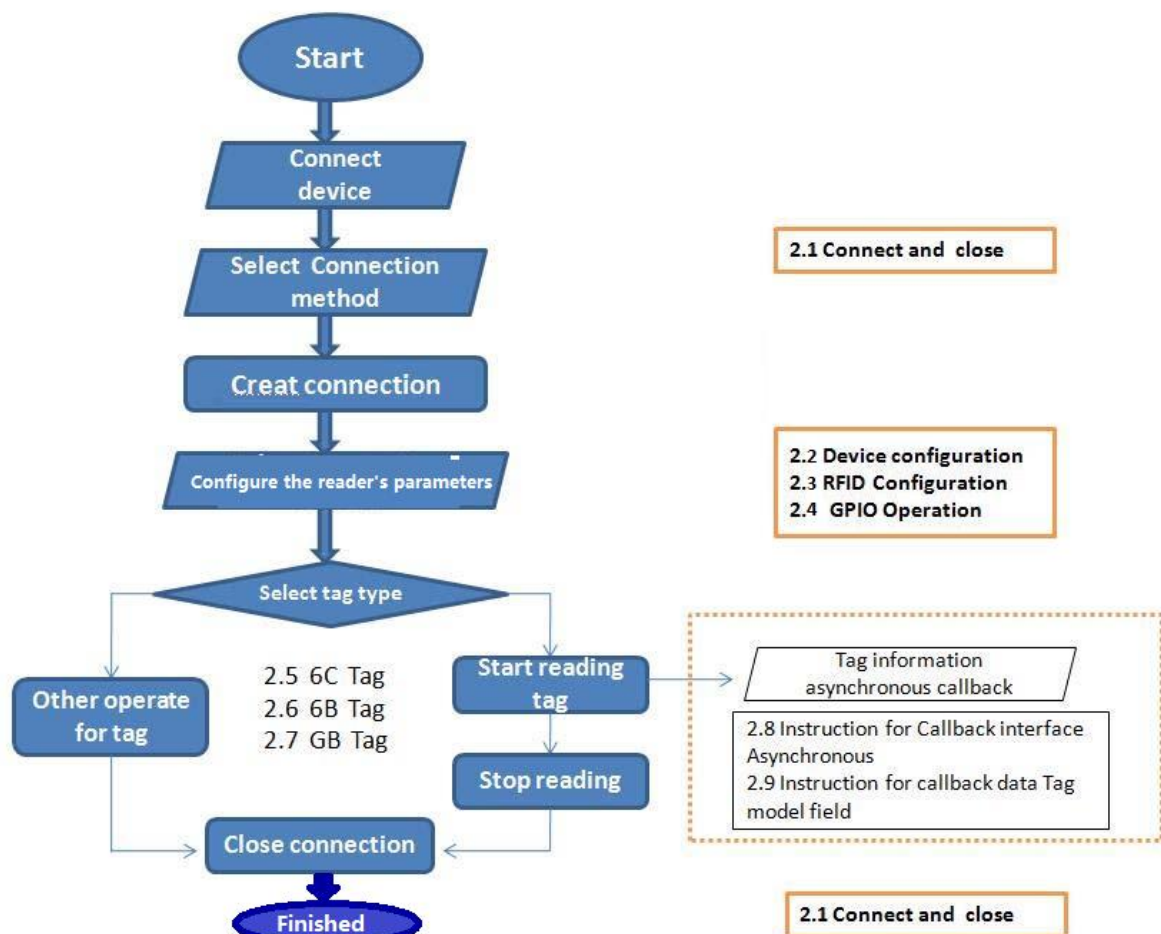
2.3.16 Get Antenna Enable Status .....	- 22 -
2.3.17 Get RFID Temperature .....	- 23 -
2.3.18 Set Expand EPC Baseband Params .....	- 23 -
2.3.19 Get Expand EPC Baseband Params .....	- 24 -
2.3.20 Get Reader Working Status .....	- 25 -
2.4 GPIO Operation .....	- 26 -
2.4.1 Set GPI Trigger Parameters .....	- 26 -
2.4.2 Get GPI Trigger parameters .....	- 28 -
2.4.3 Get GPI Status .....	- 28 -
2.4.4 Set GPO Level .....	- 29 -
2.4.5 Set Wiegand Communication Parameters .....	- 30 -
2.4.6 Get Wiegand Communication Parameters .....	- 31 -
2.5 6C Tag operation .....	- 31 -
2.5.1 read tag .....	- 31 -
2.5.2 Write tag .....	- 38 -
2.5.3 Lock tag .....	- 42 -
2.5.4 Kill tag .....	- 43 -
2.6 6B tag operation .....	- 43 -
2.6.1 Read tag .....	- 43 -
2.6.2 Write tag .....	- 44 -
2.6.3 Lock tag .....	- 44 -
2.6.4 Lock query .....	- 45 -
2.7 GB tag operation .....	- 45 -
2.7.1 Read tag .....	- 45 -
2.7.2 Write tag .....	- 45 -
2.7.3 Lock tag .....	- 46 -
2.7.4 Destroy tag .....	- 46 -
2.8 Callback interface IAsynchronousMessage description .....	- 46 -
2.9 Callback data Tag_Model field description .....	- 47 -
2.10 Callback data GPI_Model field description .....	- 47 -
2.11 Breakpoint resume .....	- 48 -
2.11.1 Set Breakpoint Resume .....	- 48 -
2.11.2 Get Breakpoint Resume Setting .....	- 48 -
2.11.3 Retrieve breakpoint cache .....	- 49 -
2.11.4 Clear breakpoint cache .....	- 49 -
2.12 Antenna number parameter description .....	- 50 -
3. Programming example .....	- 50 -
4. FAQ and Solution .....	- 51 -
Appendix A: 6C tag operation returns the error code .....	- 52 -
Appendix B: 6B tag operation returns the error code .....	- 53 -

# 1.Summary

## 1.1 Summary of content

In order to facilitate the user to carry out the secondary 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	CL7206A Series, CL7206B Series, CL720C Series,HH340/380,HF340/380,HZ340/380
Configure device	All devices
RFID Configuration	All devices
GPO Operation	CL7206B Series, CL720C Series,HH340/380,HF340/380,HZ340/380
6C tag operation	All devices
6Btag operation	All devices
GB operation	All devices

## 1.4 Copyright notice

All contents of this document, including text, pictures are original. Our company reserves the right to pursue its legal liability without permission or unauthorized use in business users.

Unauthorized, users are not allowed to add, modify, delete the contents of this document, Not allowed to spread by internet, CD-ROM etc. If there is a violation, the consequences are self-confident.

# 2.Function description for APIs

## 2.1 Connect and close

For the serial port, USB, Bluetooth or other special interfaces for Android devices, please refer to the document Hopeland RFID Reader Development Guide(Android version ) - PC - Android\_V1.11.

### 2.1.1 Create TCP connection

Package	RFIDReader
Function	static Boolean CreateTcpConn(string tcpParam, IAsynchronousMessage log)
Parameter	tcpParam: TCP connection Parameter, eg:"192.168.1.116:9090" log: Data callback interface, all tags data will be called back from this interface.

<b>Return</b>	True : succeeded, error: failed.
<b>Remark</b>	1. The connection established by this method, "tcpParam" The ID that is the connection channel is distinguished from the other link channel. 2. <b>log:</b> Data callback interface, Please refer to 2.8 <b>callback interface Remark</b> for more information
<b>Example code</b>	<pre> IAsynchronousMessage log = new SampleCode(); if(RFIDReader.CreateTcpConn("192.168.1.116:9090", log)){     System.out.println("Connection created successfully"); }else{     System.out.println("Connection created failed"); } </pre>

## 2.1.2 Close single connection

<b>Package</b>	RFIDReader
<b>Function</b>	static void CloseConn(string connectID)
<b>Parameter</b>	connectID: connection ID, eg:"192.168.1.116:9090"
<b>Return</b>	None
<b>Remark</b>	"connectID" is the connection parameter when creates connection
<b>Example code</b>	<pre> 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 successfully");     }else{         System.out.println("Close the connection failed");     } }else{     System.out.println("Connection created failed"); } </pre>

## 2.1.3 Close all connections

<b>Package</b>	RFIDReader
<b>Function</b>	static void CloseAllConnect()
<b>Parameter</b>	None
<b>Return</b>	None
<b>Remark</b>	This method will close all created connections.
<b>Example code</b>	<pre> String ConnID = "192.168.1.116:9090"; IAsynchronousMessage log = new SampleCode(); if(RFIDReader.CreateTcpConn(ConnID, log)) {     RFIDReader.CloseAllConnect();      if(RFIDReader._Config.GetReaderBaseBandSoftVersion(ConnID).equals(""))    </pre>

	<pre> RFIDReader._Config.GetReaderBaseBandSoftVersion(ConnID) == null ) {     System.out.println("The current connection is abnormal"); } else {     System.out.println("The current connection is working."); } } else {     System.out.println("Connection created failed"); } </pre>
--	---

## 2.2 Device configuration

### 2.2.1 Set IP

Package	RFIDReader._Config
Function	static <b>int</b> SetReaderNetworkPortParam( <b>String</b> ConnID, <b>String</b> iP, <b>String</b> mask, <b>String</b> 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.
Example code	<pre> String ConnID = "192.168.1.116:9090"; IAsynchronousMessage log = new SampleCode(); if(RFIDReader.CreateTcpConn(ConnID, log)) {     RFIDReader._Config.Stop(ConnID); //Make sure the reader is idle before configuring it.      if(RFIDReader._Config.SetReaderNetworkPortParam("192.168.1.1 16", "192.168.2.1", "255.255.255.0", "192.168.2.1") != 0) {         System.out.println("Set IP OK");     } else {         System.out.println("Set IP failed");     } } else {     System.out.println("Connection created failed"); } </pre>

### 2.2.2 Get Device IP

Namespace	RFIDReader._Config
Function	static <b>String</b> GetReaderNetworkPortParam( <b>String</b> ConnID)
Parameter	// ConnID: connection identification
Return	<b>IP address</b>   <b>Subnet Mask</b>   <b>gateway</b>
Remark	
Example code	<pre> String ConnID = "192.168.1.116:9090"; IAsynchronousMessage log = new SampleCode(); if(RFIDReader.CreateTcpConn(ConnID, log)) {     String Result = </pre>

	<pre> RFIDReader._Config.GetReaderNetworkPortParam(ConnID);     System.out.println(Result); }else{     System.out.println("Connection created failed"); } </pre>
--	--

### 2.2.3 Stop Instruction

<b>Package</b>	RFIDReader._Config
<b>Function</b>	static int Stop(String ConnID)
<b>Parameter</b>	// ConnID: connection identification
<b>Return</b>	0: succeeded, other: failed.
<b>Remark</b>	1. This method will make reader stop current working. 2. Stop cycle reading tags could use this method
<b>Example code</b>	<pre> 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 OK ");     } }else{     System.out.println("Connection created failed"); } </pre>

### 2.2.4 Set Reader Time

<b>Namespace</b>	RFIDReader._Config
<b>Function</b>	static Int32 SetReaderUTC(String ConnID, String param)
<b>Parameter</b>	// ConnID: connection identification // param: time parameter "yyyy.MM.dd HH:mm:ss", eg: "1970.01.01 00:00:00"
<b>Return</b>	0 means succeed, other means failed.
<b>Remard</b>	None
<b>Example code</b>	<pre> 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("Set reader time OK");     }else{         System.out.println("Set reader time failed");     } }else{     System.out.println("Connection created failed"); } </pre>



## 2.2.5 Get reader time

<b>Namespace</b>	RFIDReader._Config
<b>Function</b>	static <b>String</b> GetReaderUTC( <b>String</b> ConnID)
<b>Parameter</b>	// ConnID: connection Identification
<b>Return</b>	Time parameter "yyyy.MM.dd HH:mm:ss", eg: "1970.01.01 00:00:00"
<b>Remark</b>	NONE
<b>Example code</b>	<pre>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("Connection created failed"); }</pre>

## 2.2.6 Set Serial Port

<b>Namespace</b>	RFIDReader._Config
<b>Function</b>	static <b>int</b> SetReaderSerialPortParam( <b>String</b> ConnID, <b>eBaudrate</b> baudRate)
<b>Parameter</b>	// ConnID: connection identification // baudRate:eBaudrate._9600bps, eBaudrate._19200bps, eBaudrate._115200bps, eBaudrate._230400bps, eBaudrate._460800bps.
<b>Return</b>	0 means succeeded, other means failed.
<b>Remark</b>	NONE
<b>Example code</b>	<pre>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("Set serial baudrate failed ");     }else{         System.out.println("Set serial baudrate OK");     } }else{     System.out.println("Connection created failed"); }</pre>

## 2.2.7 Get Serial Port Setting

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

<b>Name space</b>	RFIDReader._Config
<b>Function</b>	static String GetReaderSerialPortParam2(String ConnID)
<b>Parameter</b>	// ConnID: connection identification
<b>Return</b>	9600 bps,19200 bps,115200 bps,230400 bps,460800 bps
<b>Remark</b>	None
<b>Example code</b>	<pre>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("Connection created failed"); }</pre>

## 2.2.8 Set MAC Address

<b>Namespace</b>	RFIDReader._Config
<b>Function</b>	static int SetReaderMacParam(String ConnID, String param)
<b>Parameter</b>	// ConnID: connection identification // param: MAC address format "00-00-00-00-00-00"
<b>Return</b>	0 means succeed, other value means failed.
<b>Remark</b>	None
<b>Example code</b>	<pre>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("Set failed");     }else{</pre>

	<pre>         System.out.println("Set OK");     } } else {     System.out.println("Connection created failed"); } </pre>
--	--

## 2.2.9 Get MAC Address

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"
Example code	<pre> 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("Connection created failed"); } </pre>

## 2.2.10 Set RS485 Address

Namespace	RFIDReader._Config
Function	static int SetReader485(String ConnID, String param)
Parameter	// ConnID: connection identification // param: 0~255
Return	0 means succeed, other value means failed.
Remark	None
Example code	<pre> 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("Set failed");     } else {         System.out.println("Set OK");     } } else {     System.out.println("Connection created failed"); } </pre>

## 2.2.11 Get RS485 Address

Namespace	RFIDReader._Config
Function	static <b>String</b> GetReader485( <b>String</b> ConnID)
Parameter	// ConnID: connection identification
Return	RS485 address
Remark	None
Example code	<pre>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("Connection created failed"); }</pre>

## 2.2.12 Set Server/Client Mode

Namespace	RFIDReader._Config
Function	static <b>int</b> SetReaderServerOrClient( <b>String</b> ConnID, <b>eWorkMode</b> workMode, <b>String</b> ip, <b>String</b> 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 value means failed.
Remark	When the reader set the server mode, ip parameter is invalid, you can enter any string.
Example code	<pre>String ConnID = "192.168.1.116:9090"; IAsynchronousMessage log = new SampleCode(); if(RFIDReader.CreateTcpConn(ConnID, log)) {     //TCP client: eWorkMode.Client    TCP server: eWorkMode.Server     if(RFIDReader._Config.SetReaderServerOrClient(ConnID, eWorkMode.Client, "192.168.1.12", "8081") != 0){         System.out.println("Set failed");     }else{         System.out.println("Set OK");     } }else{     System.out.println("Connection created failed"); }</pre>

## 2.2.13 Get Server/Client Mode

Namespace	RFIDReader._Config
Function	static <b>String</b> GetReaderServerOrClient( <b>String</b> ConnID)
Parameter	// ConnID: connection identification
Return	Server "Server port" or Client "Client IP" "Client port"
Remark	None
Example code	<pre>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("Connection created failed"); }</pre>

## 2.2.14 Get Reader Information

Namespace	RFIDReader._Config
Function	static <b>String</b> GetReaderInformation( <b>String</b> ConnID)
Parameter	// ConnID: connection identification
Return	Embedded application software version  reader name  reader power-up time
Remark	The reader time is in seconds
Example code	<pre>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("Connection created failed"); }</pre>

## 2.2.15 Get Baseband Software Version

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

<b>Example code</b>	<pre>String ConnID = "192.168.1.116:9090"; IAsynchronousMessage log = new SampleCode(); if(RFIDReader.CreateTcpConn(ConnID, log)) {  System.out.println(RFIDReader._Config.GetReaderBaseBandSoftV ersion(ConnID)); }else{     System.out.println("Connection created failed"); }</pre>
---------------------	--

## 2.2.16 Get antenna standing wave ratio

<b>Namespace</b>	RFIDReader._Config
<b>Function</b>	static String GetAntennaStandingWaveRatio(String ConnID, eAntennaNo antNo)
<b>Parameter</b>	// ConnID: connection identification, antNo: Antenna number enumeration
<b>Return</b>	Forward power detection   backward power detection
<b>Remark</b>	None
<b>Example code</b>	<pre>String ConnID = "192.168.1.116:9090"; IAsynchronousMessage log = new SampleCode(); if(RFIDReader.CreateTcpConn(ConnID, log)) {  System.out.println(RFIDReader._Config.GetAntennaStandingWave Ratio(ConnID,1)); }else{     System.out.println("Connection created failed"); }</pre>

## 2.2.17 Restart the reader

<b>Namespace</b>	RFIDReader._Config
<b>Function</b>	public String ReSetReader(String ConnID)
<b>Parameter</b>	// ConnID: connection identification
<b>Return</b>	None
<b>Remark</b>	None
<b>Sample code</b>	<pre>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{</pre>

	<pre> System.out.println("Failed to create connection!"); } </pre>
--	--

## 2.2.18 Get reader temperature

Namespace	<b>RFIDReader._Config</b>
Function	<b>public String GetReaderTemperature (String ConnID)</b>
Parameter	// ConnID: connection identification
Return	reader temperature
Remark	None
Sample code	<pre> 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); } </pre>

## 2.2.19 Get reader SN

Namespace	<b>RFIDReader._Config</b>
Function	<b>public String GetSN(String ConnID)</b>
Parameter	// ConnID: connection identification
Return	Reader ID
Remark	None
Sample code	<pre> 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); } </pre>

## 2.2.20 Set hidden light strip switch

Namespace	<b>Param_Option</b>
Function	<b>public String SetReaderStateED (String ConnID,String param)</b>
Parameter	<pre> // ConnID: connection identification // param: 0 or 0 1,20 (1 means turn on the LED control, 0 means turn off the LED control, the next value represents the time the LED lights up after the reader reads the </pre>

	tag, in ms)
Return	0 means succeed, other value means failed.
Remark	None
Sample code	<pre> /// &lt;summary&gt; ///     set status of the LED /// &lt;/summary&gt; /// &lt;param name="connID"&gt; conn id &lt;/param&gt; /// &lt;param name="param"&gt;0 or 0 1,20&lt;/param&gt; public static String SetReaderStateLED(String connID, String param) </pre>

## 2.2.21 Set Buzzer Control

Namespace	<b>RFIDReader._Config</b>
Function	<b>public String SetBuzzerControl(String ConnID, String param)</b>
Parameter	// ConnID: connection identification // param: Refer to the description of the sample code below.
Return	0 means succeed, other value means failed.
Remark	None
Example code	<pre> /// /// Set buzzer control, only applicable when the buzzer switch is 1, that is, when /// controlled by the upper computer. /// /// eg:0001 ==&gt; byte0 (whether to turn on the buzzer) 00: Off, 01: on ///           byte1 (whether it sounds all the time) 00: The buzzer sounds ///           once, 01: The buzzer sounds all the time RFIDReader._Config.SetBuzzerControl(ConnID, "0100");//Set the buzzer to sound once </pre>

## 2.2.22 Set Buzzer Switch

Namespace	<b>RFIDReader._Config</b>
Function	<b>public String SetBuzzerSwitch (String ConnID,String param)</b>
Parameter	// ConnID: connection identification // param: 0, reader control, which means that the buzzer will sound once when the reader reads a tag. 1, Upper computer control, which is controlled through the SetBuzzerControl interface.
Return	0 means succeed, other value means failed.
Remark	None
Example code	<pre> String ConnID = "192.168.1.116:9090"; IAsynchronousMessage log = new SampleCode(); if (RFIDReader.CreateTcpConn(tcp, log)) {     String rt = RFIDReader._Config.SetBuzzerSwitch(ConnID, "0");     System.out.println(rt); } </pre>



	}
--	---

## 2.3 RFID Configuration

### 2.3.1 Restore Factory Settings

<b>Package</b>	RFIDReader._Config
<b>Function</b>	static <b>int</b> SetReaderRestoreFactory( <b>String</b> ConnID)
<b>Parameter</b>	// ConnID: connection identification
<b>Return</b>	0 means succeed, other value means failed.
<b>Remark</b>	Restore all settings to factory state.
<b>Example code</b>	<pre>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("Restore factory settings failed");     }else{         System.out.println("Restore factory settings OK");     } }else{     System.out.println("Connection created failed"); }</pre>

### 2.3.2 Set Baseband Parameters

<b>Package</b>	RFIDReader._Config
<b>Function</b>	static <b>int</b> SetEPCBaseBandParam( <b>String</b> ConnID, <b>int</b> basebandMode, <b>int</b> qValue, <b>int</b> session, <b>int</b> searchType)
<b>Parameter</b>	// 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).
<b>Return</b>	0 means succeed, other value means failed.

<b>Remark</b>	
<b>Example code</b>	<pre>String ConnID = "192.168.1.116:9090"; IAynchronousMessage log = new SampleCode(); if(RFIDReader.CreateTcpConn(ConnID, log)) {     if(RFIDReader._Config.SetEPCBaseBandParam(ConnID, 255, 4, 1, 2) != 0) {         System.out.println("Set failed");     }else{         System.out.println("Set OK");     } }else{     System.out.println("Connection created failed"); }</pre>

### 2.3.3 Get Baseband Parameter

<b>Namespace</b>	RFIDReader._Config
<b>Function</b>	static <b>String</b> GetEPCBaseBandParam( <b>String</b> ConnID)
<b>Parameter</b>	// ConnID: connection identification
<b>Return</b>	<b>basebandMode</b>   <b>qValue</b>   <b>session</b>   <b>searchType</b>
<b>Remark</b>	<p>// 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)</p> <p>// qValue: 0~15, the initial Q value used by the reader.</p> <p>// session: 0~3</p> <p>// 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).</p>
<b>Example code</b>	<pre>String ConnID = "192.168.1.116:9090"; IAynchronousMessage log = new SampleCode(); if(RFIDReader.CreateTcpConn(ConnID, log)) {  System.out.println(RFIDReader._Config.GetEPCBaseBandParam(ConnID)); }else{     System.out.println("Connection created failed"); }</pre>

### 2.3.4 Set Antenna Power

<b>Package</b>	RFIDReader._Config
<b>Function</b>	static <b>int</b> SetANTPowerParam( <b>String</b> ConnID, <b>HashMap</b> < <b>Integer</b> , <b>Integer</b> > dicPower)
<b>Parameter</b>	// ConnID: connection identification // dicPower: Antenna number and power level key value pair
<b>Return</b>	0 means succeed, other value means failed.
<b>Remark</b>	Set the power of each antenna of reader.
<b>Example code</b>	<pre>//Set the output power of both ANT1 and ANT2 to 33dBm String ConnID = "192.168.1.116:9090"; IAynchronousMessage log = new SampleCode(); if (RFIDReader.CreateTcpConn(ConnID, log)) {     HashMap&lt;Integer, Integer&gt; dicPower = new HashMap&lt;Integer, Integer&gt;();     dicPower.put(1, 33);     dicPower.put(2, 33);     if (RFIDReader._Config. SetANTPowerParam (ConnID, dicPower) != 0){         System.out.println("Set failed");     }else{         System.out.println("Set OK");     } } else{     System.out.println("Connection created failed"); }</pre>

### 2.3.5 Get Antenna Power

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

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

<b>Example code</b>	<pre>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("Connection created failed"); }</pre>
---------------------	---

### 2.3.6 Set Tag Upload Parameters

<b>Package</b>	RFIDReader._Config
<b>Function</b>	static <b>int</b> SetTagUpdateParam( <b>String</b> ConnID, <b>int</b> repeatTimeFilter, <b>int</b> RSSIFilter)
<b>Parameter</b>	// ConnID: connection identification // repeatTimeFilter: Duplicate tag upload filter time (Unit: 10ms) // RSSIFilter: RSSI filter
<b>Return</b>	0 means succeed, other value means failed.
<b>Remark</b>	repeatTimeFilter value range 0 ~ 65535 RSSIFilter value range 0 ~ 255
<b>Example code</b>	<pre>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("Set failed");     }else{         System.out.println("Set OK");     } }else{     System.out.println("Connection created failed"); }</pre>

### 2.3.7 Get Tag upload parameters

<b>Namespace</b>	RFIDReader._Config
<b>Function</b>	static <b>String</b> GetTagUpdateParam( <b>String</b> ConnID)
<b>Parameter</b>	// ConnID: Connection identification
<b>Return value</b>	repeatTimeFilter RSSIFilter
<b>Specification</b>	// repeatTimeFilter: Duplicate tag upload filter time (Unit: 10ms) // RSSIFilter: RSSI Filter repeatTimeFilter Value range 0 ~ 65535 RSSIFilter Value range 0 ~ 255

<b>Example code</b>	<pre>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("Connection created failed"); }</pre>
---------------------	--

### 2.3.8 Get Reader Property

<b>Namespace</b>	RFIDReader._Config
<b>Function</b>	static <b>String</b> GetReaderProperty( <b>String</b> ConnID)
<b>Parameter</b>	// ConnID: Connection identification
<b>Returned value</b>	Minimum transmit power   maximum transmit power   number of antennas   band list   list of RFID protocols
<b>Explain</b>	Output unit is dB
<b>Example code</b>	<pre>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("Connection created failed"); }</pre>

### 2.3.9 Set RF Band

<b>Namespace</b>	RFIDReader._Config
<b>Function</b>	static <b>int</b> SetReaderRF( <b>String</b> ConnID, <b>eRF_Range</b> eRF_Range)
<b>Parameter</b>	// 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
<b>Returned value</b>	0 means succeed, other value means failed.
<b>Explain</b>	None

<b>Example code</b>	<pre>String ConnID = "192.168.1.116:9090"; IAsynchronousMessage log = new SampleCode(); if(RFIDReader.CreateTcpConn(ConnID, log)) {     if(RFIDReader._Config.SetReaderRF(ConnID, eRF_Range.ETSI_866_to_868MHz) != 0){         System.out.println("Set failed");     }else{         System.out.println("Set OK");     } } } else{     System.out.println("Connection created failed"); } }</pre>
---------------------	--

### 2.3.10 Get RF Band

<b>Namespace</b>	RFIDReader._Config
<b>Function</b>	static eRF_Range GetReaderRF(String ConnID)
<b>Parameter</b>	// ConnID: Connection identification
<b>Returned value</b>	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
<b>Explain</b>	None
<b>Example code</b>	<pre>String ConnID = "192.168.1.116:9090"; IAsynchronousMessage log = new SampleCode(); if(RFIDReader.CreateTcpConn(ConnID, log)) {  System.out.println(RFIDReader._Config.GetReaderRF(ConnID)); } else{     System.out.println("Connection created failed"); } }</pre>

### 2.3.11 Set RF Frequency points

<b>Namespace</b>	RFIDReader._Config
<b>Function 1</b>	static Int32 SetReaderWorkFrequency_GB920_to_925MHz(String ConnID, eWF_Mode wfMode, List<eGB920_to_925MHz> ListGB920_to_925MHz)

Function 2	static <b>Int32</b> SetReaderWorkFrequency_GB_840_to_845MHz( <b>String</b> ConnID, <b>eWF_Mode</b> wfMode, <b>List&lt;eGB_840_to_845MHz&gt;</b> ListGB_840_to_845MHz)
Function 3	static <b>Int32</b> SetReaderWorkFrequency_GB_920_to_925MHz_and_GB_840_to_845MHz( <b>String</b> ConnID, <b>eWF_Mode</b> wfMode, <b>List&lt;eGB_920_to_925MHz_and_GB_840_to_845MHz&gt;</b> ListGB_920_to_925MHz_and_GB_840_to_845MHz)
Function 4	static <b>Int32</b> SetReaderWorkFrequency_FCC_902_to_928MHz( <b>String</b> ConnID, <b>eWF_Mode</b> wfMode, <b>List&lt;eFCC_902_to_928MHz&gt;</b> ListFCC_902_to_928MHz)
Function 5	static <b>Int32</b> SetReaderWorkFrequency_ETSI_866_to_868MHz( <b>String</b> ConnID, <b>eWF_Mode</b> wfMode, <b>List&lt;eETSI_866_to_868MHz&gt;</b> ListETSI_866_to_868MHz)
Function 6	static <b>Int32</b> SetReaderWorkFrequency_JP_916_to_921MHz( <b>String</b> ConnID, <b>eWF_Mode</b> wfMode, <b>List&lt;eJP_916_to_921MHz&gt;</b> ListJP_916_to_921MHz)
Function 7	static <b>Int32</b> SetReaderWorkFrequency_TW_922_to_927MHz( <b>String</b> ConnID, <b>eWF_Mode</b> wfMode, <b>List&lt;eTW_922_to_927MHz&gt;</b> ListTW_922_to_927MHz)
Function 8	static <b>Int32</b> SetReaderWorkFrequency_ID_923_to_925MHz( <b>String</b> ConnID, <b>eWF_Mode</b> wfMode, <b>List&lt;eID_923_to_925MHz&gt;</b> ListID_923_to_925MHz)
Function 9	static <b>Int32</b> SetReaderWorkFrequency_RUS_866_to_867MHz( <b>String</b> ConnID, <b>eWF_Mode</b> wfMode, <b>List&lt;eRUS_866_to_867MHz&gt;</b> ListRUS_866_to_867MHz)
Parameter	// ConnID: Connection identification
Returned value	0 means succeed, other value means failed.
Explain	none

### 2.3.12 Get RF Frequency points

Namespace	<b>RFIDReader._Config</b>
Function	static <b>String</b> GetReaderWorkFrequency( <b>String</b> ConnID)
Parameter	// ConnID: Connection identification
Returned value	<b>Mode frequency frequency point value</b> Like: Auto GB_920_to_925MHz 920.625,920.875
Explain	None
Example code	<pre>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("Connection created failed"); }</pre>

### 2.3.13 Set Reader Automatically Idle Mode

Namespace	RFIDReader._Config
Function	static <b>int</b> SetReaderAutoSleepParam( <b>String</b> ConnID, <b>BooleanSwitch</b> , <b>String</b> time)
Parameter	// ConnID: Connection identification, Switch: ON/OFF, time: idle time
Returned value	0 means succeed, other value means failed.
Explain	Switch: true is ON, false is OFF idle time unit is <b>10ms</b>
Example code	<pre>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("Set failed");     }else{         System.out.println("Set OK");     } }else{     System.out.println("Connection created failed"); }</pre>

### 2.3.14 Get Reader Automatically Idle Mode

Namespace	RFIDReader._Config
Function	static <b>String</b> GetReaderAutoSleepParam( <b>String</b> ConnID)
Parameter	// ConnID: Connection identification
Returned value	Close or Open "idle time"
Explain	idle time unit is <b>10ms</b>
Example code	<pre>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("Connection created failed"); }</pre>

### 2.3.15 Set Antenna Enable

Namespace	RFIDReader._Config
-----------	--------------------



<b>Function</b>	<b>static int SetReaderANT(String ConnID, int antNum)</b>
<b>Parameter</b>	// ConnID: Connection identification, antNum: Antenna number enumeration
<b>Returned value</b>	0 means succeed, other value means failed.
<b>Explain</b>	Specify antenna 1 and antenna 2 together, example eAntennaNo._1 eAntennaNo._2
<b>Example code</b>	<pre>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("Set failed");     }else{         System.out.println("Set OK");     } }else{     System.out.println("Connection created failed"); }</pre>

## 2.3.16 Get Antenna Enable Status

<b>Namespace</b>	RFIDReader._Config
<b>Function</b>	<b>static int GetReaderANT(String ConnID)</b>
<b>Parameter</b>	// ConnID: Connection identification
<b>Returned value</b>	The sum of the antenna values, refer to 2.12
<b>Explain</b>	None

<b>Namespace</b>	RFIDReader._Config
<b>Function</b>	<b>static String GetReaderANT2(String ConnID)</b>
<b>Parameter</b>	// ConnID: Connection identification
<b>Returned value</b>	The antenna port number that has enabled, multiple antenna numbers separate with "," like 1,6,8
<b>Explain</b>	None
<b>Example code</b>	<pre>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("Connection created failed"); }</pre>

## 2.3.17 Get RFID Temperature

Name space	<b>RFIDReader._Config</b>
Function	<b>static String GetRFIDTemperature (String ConnID)</b>
Parameter	// ConnID: connection identification
Return	Temperature
Remark	None
Sample code	<pre>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); }</pre>

## 2.3.18 Set Expand EPC Baseband Params

Name space	<b>RFIDReader._Config</b>
Function	<b>public String SetEPCBaseExpandBandParam(String ConnID, String param)</b>
Parameter	<pre>// ConnID: connection identification Example of parameter data: "1,00000000&amp;2,00000000" // The param is a set of optional configuration parameters, separated by '&amp;', each optional parameter includes id and parameter, separated by ',', and the parameter is 4 bytes of data (returned in hexadecimal string format) // Parameter 1: Big-endian format composes U32 bit3-bit0: rfu bit4: tag focus enable bit5: fast id enable bit15-bit6: rfu bit16: NXP fast id enable bit31-bit17: rfu // Parameter 2: Byte 1: maxQ Byte 2: minQ Byte 3: tmult Byte 4: bit 0 Dynamic start Q enable bit1: Force Loop enable // Parameter 3: Byte 1: Antenna switching mode. 0--Switch immediately without tags, 1--Running out of resistance time Byte 2: Number of retries (Switch immediately without tags mode) Byte 3-4: Big-endian format composes U16,max antenna resistance time (x10ms) // Parameter 4: Byte 1: Waiting time between antenna switching(x10ms) Byte 2: antenna switching sequence Byte 3: Antenna protection threshold (unit is dBm), set to 0 to disable protection.</pre>

	<p>Byte 4: reserved</p> <p>// Parameter 5:</p> <p>Byte 1: LBT working mode</p> <p>0: disable</p> <p>1: listening only</p> <p>2: read tag after listening</p> <p>3: read tag after meeting RSSI</p> <p>Byte 2: RSSI maximum value</p> <p>Byte 3-4: reserved</p>
Return	0 success, non 0 failure
Remark	None
Sample code	<pre>String ConnID = "192.168.1.116:9090"; IAsynchronousMessage log = new SampleCode(); if (RFIDReader.CreateTcpConn(tcp, log)) {     String rt = RFIDReader._Config.SetEPCBaseExpandBandParam (ConnID,"1,00000000&amp;2,00000000");     System.out.println(rt); }</pre>

## 2.3.19 Get Expand EPC Baseband Params

Name space	<b>RFIDReader._Config</b>
Function	<b>public String GetEPCBaseExpandBandParam(String ConnID)</b>
Parameter	// ConnID: connection identification
Return	<p>Return data example:</p> <p>"1,00000000&amp;2,00000000&amp;3,00000000&amp;4,00000000&amp;5,00000000", return null ("), indicating acquisition failure.</p> <p>// The returned data is a set of optional configuration parameters, separated by '&amp;', each optional parameter includes id and parameter, separated by ',', and the parameter is 4 bytes of data (returned in hexadecimal string format)</p> <p>// Parameter 1: Big-endian format composes U32</p> <p>bit3-bit0: rfu</p> <p>bit4: tag focus enable</p> <p>bit5: fast id enable</p> <p>bit15-bit6: rfu</p> <p>bit16: NXP fast id enable</p> <p>bit31-bit17: rfu</p> <p>// Parameter 2:</p> <p>Byte 1: maxQ</p> <p>Byte 2: minQ</p> <p>Byte 3: tmult</p> <p>Byte 4: bit 0 Dynamic start Q enable</p> <p>bit1: Force Loop enable</p> <p>// Parameter 3:</p> <p>Byte 1: Antenna switching mode. 0--Switch immediately without tags, 1--Running out of resistance time</p> <p>Byte 2: Number of retries (Switch immediately without tags mode)</p> <p>Byte 3-4: Big-endian format composes U16,max antenna resistance time (x10ms)</p>

	// 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); }

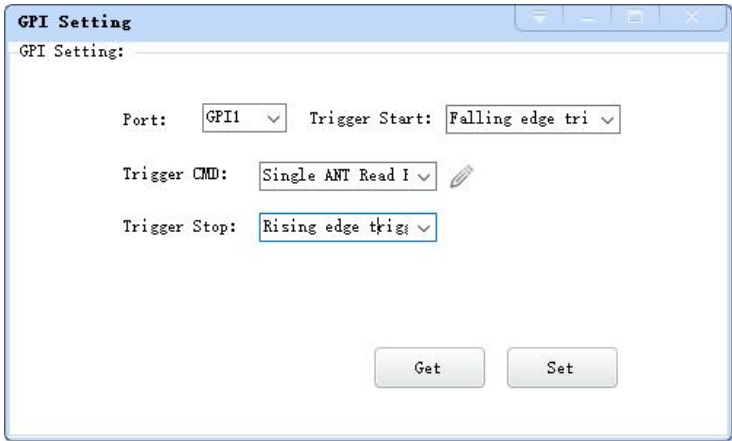
## 2.3.20 Get Reader Working Status

Name space	<b>RFIDReader</b>
Function	<b>static String GetRFIDWorkStatus(String ConnID)</b>
Parameter	// ConnID: connection identification
Return	0: Idle state !0: Working state
Remark	Check whether the reader is in working state.
Sample code	String ConnID = "192.168.1.116:9090"; IAsynchronousMessage log = new SampleCode(); if (RFIDReader.CreateTcpConn(ConnID , log)) { String rt = RFIDReader.GetRFIDWorkStatus (ConnID); System.out.println(rt); }

## 2.4 GPIO Operation

### 2.4.1 Set GPI Trigger Parameters

Namespace	RFIDReader._Config
Function1	public <b>Int32</b> SetReaderGPIParam( <b>String</b> ConnID, <b>eGPI</b> GPINum, <b>eTriggerStart</b> triggerStart, <b>eTriggerCode</b> triggerCode, <b>eTriggerStop</b> triggerStop, <b>String</b> DelayTime, <b>Boolean</b> isUpload, <b>String</b> customCMD)
Function2	static <b>Int32</b> SetReaderGPIParam( <b>String</b> ConnID, <b>eGPI</b> GPINum, <b>eTriggerStart</b> triggerStart, <b>eTriggerCode</b> triggerCode, <b>eTriggerStop</b> triggerStop, <b>String</b> DelayTime, <b>Boolean</b> isUpload)
Parameter	<p>// ConnID: connection identification ;</p> <p>// GPINum: eGPI._1,eGPI._2,eGPI._3,eGPI._4 ;</p> <p>//triggerStart: Trigger start condition eTriggerStart.OFF,eTriggerStart.Low_level,eTriggerStart.High_level, eTriggerStart.Rising_edge,eTriggerStart.Falling_edge,eTriggerStart.Any_edge ;</p> <p>//triggerCode: Command executed after triggering 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, <b>triggerCode.Custom_Read_0</b></p> <p>//triggerStop: Trigger stop condition eTriggerStop.OFF,eTriggerStop.Low_level,eTriggerStop.High_level, eTriggerStop.Rising_edge,eTriggerStop.Falling_edge,eTriggerStop.Any_edge, eTriggerStop.Delay ;</p> <p>// isUpload: Trigger state callback or not When triggerStop is eTriggerStop.OFF, set whether to upload the infrared trigger state change, true is to upload, false is not to upload.</p> <p>//customCMD: Custom trigger code, which is valid when triggerCode is set</p>

	<p>to Custom_Read_0.</p> <p>Eg: SetReaderGPIParam("192.168.1.116:9090",eGPI._2,eTriggerStart.Low_level,triggerCode.Double_Antenna_read_EPC_and_TID,eTriggerStop.Dealy,"100",true);</p> <p>The current method configures the GPI port 2 low level to trigger and execute the trigger code, and ends after a delay of 1000ms. The trigger stop condition is not OFF, isUpload is invalid, you can fill in at will.</p>
<b>Return value</b>	0 means succeed, other value means failed.
<b>Description</b>	<ul style="list-style-type: none"> <li><b>Delay time:</b>unit is 10ms (Valid when the trigger stop condition is "delay")</li> </ul> <p>For details, please refer to "RFID Reader Demo Software"</p> <p>For specific GPI trigger parameter callback , pls refer to <a href="#">2.8 Callback interface</a> GPIControlMsg();</p> <p><b>The setting of the trigger button of Bluetooth handheld reader is realized by setting the parameters of GPI1:</b></p> <p><b>The default setting:</b></p> <p><b>Port:GPI1</b></p> <p><b>Trigger Start: Falling edge (Press the trigger button)</b></p> <p><b>Trigger Command: Single ANT Read EPC / Custom_Read_0</b></p> <p><b>Trigger Stop: Rising edge (Loosen the trigger button)</b></p> <p><b>CustomCMD:</b></p> <p style="color: red;">02100006010109002001 //Single ANT GetEPC_EpcData</p> <p style="color: red;">02100009010109002001020006 //Single ANT GetEPC_TID_EpcData</p> 
<b>Example code</b>	<pre>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.OF</pre>

	<pre> F, "", true) != 0) {     System.out.println("Set failed"); } else {     System.out.println("Set OK"); } } else {     System.out.println("Connection created failed"); } </pre>
--	--

## 2.4.2 Get GPI Trigger parameters

Namespace	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 value	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"
Description	<ul style="list-style-type: none"> <li>Delay time:unit is 10ms (Valid when the trigger stop condition is "delay")</li> <li>Upload flag: ON means upload, OFF means no upload</li> </ul> For details, please refer to "RFID Reader Demo Software"
Example code	<pre> String ConnID = "192.168.1.116:9090"; IAsynchronousMessage log = new SampleCode(); if(RFIDReader.CreateTcpConn(ConnID, log)) {  System.out.println(RFIDReader._Config.GetReaderGPIParam(Conn ID,eGPI._1)); } else {     System.out.println("Connection created failed"); } </pre>

## 2.4.3 Get GPI Status

Namespace	RFIDReader._Config
Function	static String GetReaderGPIState(String ConnID)
Parameter	// ConnID: connection identification Eg: GetReaderGPIState("192.168.1.116:9090");
Return value	Eg: "1,Low & 2,High", This return value means :

	<p>No. 1 GPI port is currently at a low level,</p> <p>No. 2 GPI port is currently at a high level.</p>
<b>Description</b>	<p>The method is to actively query the current GPI level status, unrelated with trigger event!</p> <p>For specific GPI trigger parameter callback , pls refer to <a href="#">2.8 Callback interface</a> GPIControlMsg();</p>
<b>Example code</b>	<pre>String ConnID = "192.168.1.116:9090"; IAynchronousMessage log = new SampleCode(); if(RFIDReader.CreateTcpConn(ConnID, log)) {  System.out.println(RFIDReader._Config.GetReaderGPIState(ConnID)); }else{     System.out.println("Connection created failed"); }</pre>

## 2.4.4 Set GPO Level

<b>Namespace</b>	<b>RFIDReader._Config</b>
<b>Function</b>	<b>static Int32 SetReaderGPOState(String ConnID, HashMap&lt;eGPO, eGPOState&gt; dicState)</b>
<b>Parameter</b>	<p>// ConnID: connection identification</p> <p>// dicState: GPO Number and level corresponding key value pairs</p>
<b>Return value</b>	0 means succeed, other value means failed.
<b>Description</b>	None
<b>Example code</b>	<pre>//Set the #1 and #2 GPO ports to high level. String ConnID = "192.168.1.116:9090"; IAynchronousMessage log = new SampleCode(); if(RFIDReader.CreateTcpConn(ConnID, log)) {     HashMap&lt;eGPO, eGPOState&gt; dicState = new HashMap&lt;eGPO, eGPOState&gt;();     dicState.put(eGPO._1, eGPOState._High);     dicState.put(eGPO._2, eGPOState._High);     if (RFIDReader._Config.SetReaderGPOState(ConnID, dicState) != 0) {         System.out.println("Set failed");     } else {         System.out.println("Set OK");     } } }else{</pre>



	<pre>System.out.println("Connection created failed"); }</pre>
--	---

## 2.4.5 Set Wiegand Communication Parameters

<b>Namespace</b>	<b>RFIDReader._Config</b>
<b>Function</b>	<b>static int SetReaderWG(String ConnID, eWiegandSwitch wiegandSwitch, eWiegandFormat wiegandFormat, eWiegandDetails param)</b>
<b>Parameter</b>	<pre>// 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.</pre> <p>Eg: SetReaderWG("192.168.1.116:9090",eWiegandSwitch.Open, eWiegandFormat.Wiegand26,eWiegandDetails.end_of_the_TID_data);</p> <p>The method is to turn the Wiegand switch on, communication format is Wiegand26 ,Specifies to transmit TID end data</p>
<b>Return value</b>	0 means succeed, other value means failed.
<b>Description</b>	None
<b>Example code</b>	<pre>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("Set failed");     }else{         System.out.println("Set OK");     } }else{     System.out.println("Connection created failed"); }</pre>

## 2.4.6 Get Wiegand Communication Parameters

<b>Namespace</b>	RFIDReader._Config
<b>Function</b>	static <b>String</b> GetReaderWG( <b>String</b> ConnID)
<b>Parameter</b>	// ConnID: Connection identification
<b>Return value</b>	// Return value: Wiegand switch Wiegand format Wiegand data content Eg: return value "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.
<b>Description</b>	No
<b>Example code</b>	<pre>String ConnID = "192.168.1.116:9090"; IAynchronousMessage log = new SampleCode(); if(RFIDReader.CreateTcpConn(ConnID, log)) {  System.out.println(RFIDReader._Config.GetReaderWG(ConnID)); }else{     System.out.println("Connection created failed"); }</pre>

## 2.5 6C Tag operation

### 2.5.1 read tag

<b>Package</b>	RFIDReader._Tag6C
<b>Function 1</b>	<b>static int GetEPC(String ConnID, int antNum, eReadType readType)</b> // Read EPC // antNum: Antenna number enumeration. Appoint Antenna 1 and 2 working at same time: 3, Please refer to the description of antenna number parameters // readType: 0 single, 1 inventory(single or cyclical reading)
<b>Function2</b>	<b>static int GetEPC(String ConnID, int antNum, int readType, String accessPassword)</b> // accessPassword: Tag access password(8 Hexadecimal string)
<b>Function3</b>	<b>static int GetEPC_MatchEPC(String ConnID, int antNum, int readType, String sEPC)</b> // Match EPC, Read EPC // sEPC : EPC value to be matched(Hexadecimal string)
<b>Function4</b>	<b>static int GetEPC_MatchEPC(String ConnID, int antNum, int readType, String sEPC, int matchWordStartIndex)</b> // 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 string)
Function6	static int GetEPC_MatchTID(String ConnID, int antNum, int readType, String sTID) // match TID, Read EPC // sTID: TID value to be matched(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 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 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 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 string)
Function17	static int GetEPC_TID_UserData(String ConnID, int antNum, int readType, int readStart, int readLen) // Read EPC, TID and UserData // readStart user area's starting index // readLen user block's length (unit: Word)
Function18	static int GetEPC_TID_UserData(String ConnID, int antNum, int readType, int

	<b>readStart, int readLen, String accessPassword)</b> // accessPassword: Tag access password(8 Hexadecimal string)
<b>Function19</b>	<b>static int GetEPC_TID_UserData_MatchEPC(String ConnID, int antNum, int readType, int readStart, int readLen, String sEPC)</b> // match EPC, Read EPC,TID and UserData
<b>Function20</b>	<b>static int GetEPC_TID_UserData_MatchEPC(String ConnID, int antNum, int readType, int readStart, int readLen, String sEPC, int matchWordStartIndex)</b> // match EPC, Read EPC, TID and UserData
<b>Function21</b>	<b>static int GetEPC_TID_UserData_MatchEPC(String ConnID, int antNum, int readType, int readStart, int readLen, String sEPC, int matchWordStartIndex, String accessPassword)</b> // accessPassword: Tag access password(8 Hexadecimal string)
<b>Function22</b>	<b>static int GetEPC_TID_UserData_MatchTID(String ConnID, int antNum, int readType, int readStart, int readLen, String sTID)</b> // match TID, Read EPC, TID and UserData
<b>Function23</b>	<b>static int GetEPC_TID_UserData_MatchTID(String ConnID, int antNum, int readType, int readStart, int readLen, String sTID, int matchWordStartIndex)</b> // match TID, Read EPC,TID and UserData
<b>Function24</b>	<b>static int GetEPC_TID_UserData_MatchTID(String ConnID, int antNum, int readType, int readStart, int readLen, String sTID, int matchWordStartIndex, String accessPassword)</b> // accessPassword: Tag access password(8 Hexadecimal string)
<b>Function25</b>	<b>static int GetEPC_ReservedData(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen)</b> //read EPC and password // ConnID: connection identifier // readStart: The starting index for reading reserved bank(0 or 2) // readLen: Data length for reading reserved bank(unit: Word, max is 4)
<b>Function26</b>	<b>static int GetEPC_ReservedData(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String accessPassword)</b> //accessPassword: tag access password
<b>Function27</b>	<b>static int GetEPC_ReservedData_MacthEPC(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String sEPC )</b> // match EPC to read reserved bank // sEPC: matched EPC value (Hexadecimal string)
<b>Function28</b>	<b>static int GetEPC_ReservedData_MacthEPC(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String sEPC, String accessPassword )</b> // accessPassword: tag access password
<b>Function29</b>	<b>static int GetEPC_ReservedData_MacthTID(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String sTID, int matchWordStartIndex)</b> // match TID to read reserved bank // sTID: matched TID value (Hexadecimal string)

	// matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)
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
Function31	static int GetEPC_ReservedData_TID(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen) // read EPC, reserved bank and TID
Function32	static int GetEPC_ReservedData_TID(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String accessPassword) // accessPassword: tag access password
Function33	static int GetEPC_ReservedData_TID_MacThEPC(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String sEPC ) // match EPC to read EPC, reserved bank and TID // sEPC: matched EPC value (Hexadecimal string)
Function34	static int GetEPC_ReservedData_TID_MacThEPC(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String sEPC , String accessPassword) // accessPassword: tag access password
Function35	static 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 bank and TID // sTID: matched TID value (Hexadecimal string) // matchWordStartIndex: Starting index of the tag memory bank to be matched
Function36	static 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
Function37	int GetEPC_EpcData(String ConnID,int antNum, int readType, int readStart, int readLen) //read EPC and EPCData //readStart: The starting index for reading EPCData //readLen: Data length for reading EPCData (unit: Word)
Function38	int GetEPC_TID_EpcData(String ConnID,int antNum, int readType, int readStart, int readLen) // simultaneously read EPC, TID and EPCData // readStart: The starting index for reading EPCData // readLen: Data length for reading EPCData (unit: Word)
Function39	public int GetRFMicron_Temperature(String ConnID, int antNo, int readType) //Read the RFMicron S3 temperature tag. //After reading the tag data, convert it into a temperature value through the following public float RFMicron_ConvertTemperature(EPCModel model)

<b>Function40</b>	<b>public int GetCAB_Temperature(String ConnID, int antNo, int readType)</b> //Read the CAB temperature tag. //After reading the tag data, convert it into a temperature value through the following public float CAB_ConvertTemperature(EPCModel model)
<b>Function41</b>	<b>public int GetEM_Temperature(String ConnID, int antNo, int readType)</b> //Read the EM temperature tag. //After reading the tag data, convert it into a temperature value through the following public float EM_ConvertTemperature(EPCModel model)
<b>Function42</b>	<b>public int GetLTU_Temperature(String ConnID, int antNo, int readType)</b> //Read the 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)
<b>Function43</b>	<b>public int GetEPC_LightKX2005X(String ConnID, eAntennaNo antNum, eReadType readType)</b> // Light the KX2005X tag. // antNum: Antenna number. Appoint Antenna 1 and 2 working at same time: 3, Please refer to the description of antenna number parameters // readType: 0 single, 1 inventory(single or cyclical reading)
<b>Function44</b>	<b>public int GetEPC_LightKX2005X(String ConnID, eAntennaNo antNum, eReadType readType, String accessPassword)</b> // Light the KX2005X tag. // antNum: Antenna number. Appoint Antenna 1 and 2 working at same time: 3, Please refer to the description of antenna number parameters // readType: 0 single, 1 inventory(single or cyclical reading) // accessPassword: access password
<b>Function45</b>	<b>public int GetEPC_LightKX2005X_MatchEPC(String ConnID, eAntennaNo antNum, eReadType readType, String sEPC, int matchWordStartIndex)</b> // Light the KX2005X tag. // antNum: Antenna number. Appoint Antenna 1 and 2 working at same time: 3, Please refer to the description of antenna number parameters // readType: 0 single, 1 inventory(single or cyclical reading) // sEPC: matched EPC value (Hexadecimal string) // matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)
<b>Function46</b>	<b>public int GetEPC_LightKX2005X_MatchEPC(String ConnID, eAntennaNo antNum, eReadType readType, String sEPC, int matchWordStartIndex, String accessPassword)</b> // Light the KX2005X tag.

	<p>// antNum: Antenna number.  Appoint Antenna 1 and 2 working at same time: 3, Please refer to the description of antenna number parameters  // readType: 0 single, 1 inventory(single or cyclical reading)  // sEPC: matched EPC value (Hexadecimal string)  // matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)  // accessPassword: access password</p>
<b>Function47</b>	<p><b>public int GetEPC_LightKX2005X_MatchTID(String ConnID, eAntennaNo antNum, eReadType readType, String sTID, int matchWordStartIndex)</b>  // Light the KX2005X tag.  // antNum: Antenna number.  Appoint Antenna 1 and 2 working at same time: 3, Please refer to the description of antenna number parameters  // readType: 0 single, 1 inventory(single or cyclical reading)  // sTID: matched TID value (Hexadecimal string)  // matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)</p>
<b>Function48</b>	<p><b>public int GetEPC_LightKX2005X_MatchTID(String ConnID, eAntennaNo antNum, eReadType readType, String sTID, int matchWordStartIndex, String accessPassword)</b>  // Light the KX2005X tag.  // antNum: Antenna number.  Appoint Antenna 1 and 2 working at same time: 3, Please refer to the description of antenna number parameters  // readType: 0 single, 1 inventory(single or cyclical reading)  // sTID: matched TID value (Hexadecimal string)  // matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word)  // accessPassword: access password</p>
<b>Function49</b>	<p><b>public int GetEPC_LightQStar6T(String ConnID, eAntennaNo antNum, eReadType readType)</b>  // Light the QStar6T tag.  // antNum: Antenna number.  Appoint Antenna 1 and 2 working at same time: 3, Please refer to the description of antenna number parameters  // readType: 0 single, 1 inventory(single or cyclical reading)</p>
<b>Function50</b>	<p><b>public int GetEPC_LightQStar6T(String ConnID, eAntennaNo antNum, eReadType readType, String accessPassword)</b>  // Light the QStar6T tag.  // antNum: Antenna number.  Appoint Antenna 1 and 2 working at same time: 3, Please refer to the description of antenna number parameters  // readType: 0 single, 1 inventory(single or cyclical reading)  // accessPassword: access password</p>



<b>Function51</b>	<pre> <b>public int GetEPC_LightQStar6T_MatchEPC(String ConnID, eAntennaNo antNum, eReadType readType, String sEPC, int matchWordStartIndex)</b> // Light the QStar6T tag. // antNum: Antenna number. Appoint Antenna 1 and 2 working at same time: 3, Please refer to the description of antenna number parameters // readType: 0 single, 1 inventory(single or cyclical reading) // sEPC: matched EPC value (Hexadecimal string) // matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word) </pre>
<b>Function52</b>	<pre> <b>public int GetEPC_LightQStar6T_MatchEPC(String ConnID, eAntennaNo antNum, eReadType readType, String sEPC, int matchWordStartIndex, String accessPassword)</b> // Light the QStar6T tag. // antNum: Antenna number. Appoint Antenna 1 and 2 working at same time: 3, Please refer to the description of antenna number parameters // readType: 0 single, 1 inventory(single or cyclical reading) // sEPC: matched EPC value (Hexadecimal string) // matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word) // accessPassword: access password </pre>
<b>Function53</b>	<pre> <b>public int GetEPC_LightQStar6T_MatchTID(String ConnID, eAntennaNo antNum, eReadType readType, String sTID, int matchWordStartIndex)</b> // Light the QStar6T tag. // antNum: Antenna number. Appoint Antenna 1 and 2 working at same time: 3, Please refer to the description of antenna number parameters // readType: 0 single, 1 inventory(single or cyclical reading) // sTID: matched TID value (Hexadecimal string) // matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word) </pre>
<b>Function54</b>	<pre> <b>public int GetEPC_LightQStar6T_MatchTID(String ConnID, eAntennaNo antNum, eReadType readType, String sTID, int matchWordStartIndex, String accessPassword)</b> // Light the QStar6T tag. // antNum: Antenna number. Appoint Antenna 1 and 2 working at same time: 3, Please refer to the description of antenna number parameters // readType: 0 single, 1 inventory(single or cyclical reading) // sTID: matched TID value (Hexadecimal string) // matchWordStartIndex: Starting index of the tag memory bank to be matched (unit: Word) // accessPassword: access password </pre>
<b>Remark</b>	<ol style="list-style-type: none"> <li>1. For detailed Return , please kindly follow <a href="#">Appendix A</a></li> <li>2. Stop inventory (cycle) reading using "stop" instruction.</li> </ol>



	<p>3. Difference between inventory and single reading is that single read automatically stops reading after one time reading, but inventory read requires a stop function to stop reading.</p> <p>4. The same part for inventory and single reading is that after the last tag data is uploaded, they will notify PC side through asynchronous callback that tag upload finished, please refer to <a href="#">callback interface instruction</a> - OutPutTagsOver();</p>
<b>Example code</b>	<pre>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("Connection created failed"); }  //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); }</pre>

## 2.5.2 Write tag

### 2.5.2.1 Write EPC

<b>Package</b>	RFIDReader._Tag6C
<b>Function1</b>	<p><b>static int WriteEPC(String ConnID, int antNum, String sWriteData)</b></p> <p>// ConnID: connection identification</p> <p>// antNum: Antenna number enumeration.</p> <p>Appoint antenna 1 and 2 working at same time; e.g.:</p>

	eAntennaNo._1 eAntennaNo._2 3, Please refer to the description of antenna number parameters // sWriteData: data to be written (Hexadecimal string)
Function2	<b>static int WriteEPC_MatchEPC(String ConnID, int antNum, String sWriteData, String sMatchData, int matchWordStartIndex)</b> // match EPC, Write EPC // sMatchData, EPC data to be matched // matchWordStartIndex, match Data starting index
Function3	<b>static int WriteEPC_MatchEPC(String ConnID, int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)</b> // match EPC, Write EPC // accessPassword: Tag access password(8 Hexadecimal string)
Function4	<b>static int WriteEPC_MatchTID(String ConnID, int antNum, String sWriteData, String sMatchData, int matchWordStartIndex)</b> // match TID, Write EPC // sMatchData, TID data to be matched // matchWordStartIndex, match Data starting index
Function5	<b>static int WriteEPC_MatchTID(String ConnID, int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)</b> // match TID, Write EPC
Parameter	Please refer to Function Remark.
Return	0 means succeed, other value means failed. <a href="#">Appendix A</a>
Remark	1. Suggest to use matched ID to write, means to use "Function4" and "Function5". 2. For detailed Return Remark, please see the appendix.
Example code	<pre>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",     "E28011052000308565F90157",0,"00000001") != 0){         System.out.println("Write EPC failed");     }else{         System.out.println("Write EPC OK");     } }else{     System.out.println("Connection created failed"); }</pre>

### 2.5.2.2 Write Userdata

Package	RFIDReader._Tag6C
Function1	<b>static int WriteUserData(String ConnID, int antNum, String sWriteData,int offset)</b>

	// ConnID: connection identification // antNum: Antenna number enumeration.  // sWriteData: data to be written (Hexadecimal string)  //offset: the offset of user area, that is, the number of 0 before writing data Appoint antenna 1and 2 working at same time e.g.: 3
Function2	<b>static int WriteUserData_MatchEPC(String ConnID, int antNum, String sWriteData, int offset, String sMatchData, int matchWordStartIndex)</b> // match EPC, Write user data // sMatchData, EPC data to be matched (Hexadecimal string) // matchWordStartIndex, match Data starting index
Function3	<b>static int WriteUserData_MatchEPC(String ConnID, int antNum, String sWriteData, int offset, String sMatchData, int matchWordStartIndex, String accessPassword)</b> // match EPC, Write user data // accessPassword, Tag access password
Function4	<b>static int WriteUserData_MatchTID(String ConnID, int antNum, String sWriteData, int offset, String sMatchData, int matchWordStartIndex)</b> // match TID, Write user data // sMatchData, TID data to be matched (Hexadecimal string) // matchWordStartIndex, match Data starting index
Function5	<b>static int WriteUserData_MatchTID(String ConnID, int antNum, String sWriteData, int offset, String sMatchData, int matchWordStartIndex, String accessPassword)</b> // match TID, Write user data
Parameter	Please refer to Function Remark.
Return	0 means succeed, other value means failed. <a href="#">Appendix A</a>
Remark	1. Suggest to use matched ID to write, means to use "Function4" and "Function5". 2. For detailed Return Remark, please see the appendix.
Example code	<pre>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, "E00210 0SF001", 0, "E28011052000308565F90157", 0, "00000001") != 0) {         System.out.println("Write failed");     }else{         System.out.println("Write OK");     } }else{     System.out.println("Connection created failed"); }</pre>

### 2.5.2.3 Write password

<b>Package</b>	RFIDReader._Tag6C
<b>Function1</b>	<b>static int WriteAccessPassWord(String ConnID, int antNum, String sWriteData)</b> // Write Tag access password // sWriteData: password content (8 Hexadecimal string data)
<b>Function2</b>	<b>static int WriteAccessPassWord(String ConnID, int antNum, String sWriteData, String accessPassword)</b> // Write Tag access password // accessPassword: Original Tag access password (8 Hexadecimal string data)
<b>Function3</b>	<b>static int WriteAccessPassWord_MatchTID(String ConnID, int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)</b> // Write Tag access password // sMatchData: TID data to be matched // matchWordStartIndex: match Data starting index
<b>Function4</b>	<b>static int WriteDestroyPassWord(String ConnID, int antNum, String sWriteData)</b> // Write the kill password
<b>Function5</b>	<b>static int WriteDestroyPassWord(String ConnID, int antNum, String sWriteData, String accessPassword)</b> // Write the kill password // accessPassword: Tag access password (8 Hexadecimal string data)
<b>Function6</b>	<b>static int WriteDestroyPassWord_MatchTID(String ConnID, int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)</b> // Write the kill password // sMatchData: TID data to be matched // matchWordStartIndex: match Data starting index
<b>Parameter</b>	Please refer to Function Remark.
<b>Return</b>	0 means succeed, other value means failed. <a href="#">Appendix A</a>
<b>Remark</b>	
<b>Example code</b>	<pre>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 OK");     } }else{     System.out.println("Connection created failed"); }</pre>

## 2.5.3 Lock tag

<b>Package</b>	RFIDReader._Tag6C
<b>Function1</b>	<b>static int Lock(String ConnID, int antNum, eLockArea lockArea, eLockType lockType)</b> // ConnID: connection identification // antNum: antenna number // lockArea: lock area enumeration // lockType: lock type enumeration
<b>Function2</b>	<b>static int Lock_MatchEPC(String ConnID, int antNum, eLockArea lockArea, eLockType lockType, String sMatchData, Int32 matchWordStartIndex)</b> // sMatchData: EPC data to be matched (Hexadecimal string) // matchWordStartIndex: match data's word initial address
<b>Function3</b>	<b>static int Lock_MatchEPC(String ConnID, int antNum, eLockArea lockArea, eLockType lockType, String sMatchData, Int32 matchWordStartIndex, String accessPassword)</b> // accessPassword: Tag access password
<b>Function4</b>	<b>static int Lock_MatchTID(String ConnID, int antNum, eLockArea lockArea, eLockType lockType, String sMatchData, Int32 matchWordStartIndex)</b> // sMatchData: TID data to be matched(Hexadecimal string) // matchWordStartIndex: match Data starting index
<b>Function5</b>	<b>static int Lock_MatchTID(String ConnID, int antNum, eLockArea lockArea, eLockType lockType, String sMatchData, Int32 matchWordStartIndex, String accessPassword)</b> // accessPassword: Tag access password
<b>Parameter</b>	// refer to above method Remark
<b>Return</b>	0 means succeed, other value means failed. <a href="#">Appendix A</a>
<b>Remark</b>	
<b>Example code</b>	<pre>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("Lock failed");     }else{         System.out.println("Lock OK");     } }else{     System.out.println("Connection created failed"); }</pre>

## 2.5.4 Kill tag

Package	RFIDReader._Tag6C
Function1	<b>static int Destroy(String ConnID, int antNum, String destroyPassword)</b> // ConnID: connection identification // antNum: antenna number // destroyPassword: kill password (Hexadecimal string)
Function2	<b>static int Destroy_MatchEPC(String ConnID, int antNum, String destroyPassword, String sMatchData, int matchWordStartIndex)</b> // sMatchData: EPC data to be matched(Hexadecimal string) // matchWordStartIndex: match Data starting index
Function3	<b>static int Destroy_MatchTID(String ConnID, int antNum, String destroyPassword, String sMatchData, int matchWordStartIndex)</b> // 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. <a href="#">Appendix A</a>
Remark	
Example code	<pre>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","E 28011052000308565F90157",0) != 0){         System.out.println("Kill failed");     }else{         System.out.println("Kill OK");     } }else{     System.out.println("Connection created failed"); }</pre>

## 2.6 6B tag operation

### 2.6.1 Read tag

Package	RFIDReader._Tag6B
Function 1	<b>static int Get6B(String ConnID, int antNum, int readType, e6BReaderContent readerContent)</b> // 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) // readerContent: read content enumeration
<b>Function 2</b>	<b>static int Get6B_UserData(String ConnID, int antNum, int readType, e6BReaderContent readerContent, int readStart, int readLen)</b> // readStart: User data area starting address // readLen: Read the byte length of the user area
<b>Function 3</b>	<b>static int Get6B_UserData_MatchTID(String ConnID, int antNum, int readType, e6BReaderContent readerContent, int readStart, int readLen, String sMatchData)</b> // sMatchData: Matching tag TID data
<b>Parameter</b>	
<b>Return</b>	0 means succeed, other value means failed.
<b>Remark</b>	Error code please see the appendix Remark

## 2.6.2 Write tag

<b>Package</b>	RFIDReader._Tag6B
<b>Function</b>	<b>static int Write6B(String ConnID, int antNum, String sTID, int startIndex, String sWriteData)</b>
<b>Parameter</b>	// 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: Matched TID data (Hex.) // sWriteData: User data to be written(Hex.)
<b>Return</b>	0 means succeed, other value means failed.
<b>Remark</b>	Error code please see the appendix Remark

## 2.6.3 Lock tag

<b>Package</b>	RFIDReader._Tag6B
<b>Function</b>	<b>static int Lock6B(String ConnID, int antNum, String sTID, int lockIndex)</b>
<b>Parameter</b>	// 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: Matched TID data (Hex.)  // lockIndex: The index of the lock area
<b>Return</b>	0 means succeed, other value means failed.
<b>Remark</b>	Error code please see the appendix Remark

## 2.6.4 Lock query

Package	RFIDReader._Tag6B
Function	static <b>int</b> GetLock6B_State( <b>String</b> ConnID, <b>int</b> antNum, <b>String</b> sTID, <b>int</b> 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: Matched TID data (Hex.)  // lockIndex: The index of the lock area
Return	Query result (0 success, 1 failure)   query status (0 unlocked, 1 locked)
Remark	No

## 2.7 GB tag operation

### 2.7.1 Read tag

Package	RFIDReader._TagGB
Function	static <b>int</b> GetGB( <b>String</b> ConnID, <b>int</b> antNum, <b>int</b> 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	No
Return	0 means succeed, other value means failed.
Remark	Error code please see the appendix Remark

### 2.7.2 Write tag

Package	RFIDReader._TagGB
Function	static <b>int</b> WriteGB( <b>String</b> ConnID, <b>int</b> antNum, <b>String</b> sWriteData)
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 // sTID: Matched TID Data ( Hexadecimal string) // sWriteData: The content of the user data area to be written (Hexadecimal string)
Return Value	0 means succeed, other value means failed.



<b>Description</b>	Please refer to the appendix for the error code
--------------------	---

### 2.7.3 Lock tag

<b>Package</b>	RFIDReader._TagGB
<b>Function</b>	static <b>int</b> LockGB( <b>String</b> ConnID, <b>int</b> antNum, <b>eLockAreaGB</b> LockAreaGB, <b>eLockTypeGB</b> LockTypeGB)
<b>Parameter</b>	// 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
<b>Return Value</b>	0 means succeed, other value means failed.
<b>Description</b>	Please refer to the appendix for the error code

### 2.7.4 Destroy tag

<b>Package</b>	RFIDReader._TagGB
<b>Function</b>	static <b>int</b> DestoryGB( <b>String</b> ConnID, <b>int</b> antNum, <b>String</b> destroyPassword)
<b>Parameter</b>	// 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
<b>Return Value</b>	0 means succeed, other value means failed.
<b>Description</b>	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(int gpiIndex, int gpiState,int startOrStop);
}
```

Callback method	Description
<b>WriteDebugMsg</b>	Print API internal process debugging information.
<b>WriteLog</b>	API record log callback (not yet open) .
<b>PortConneting</b>	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.
<b>PortClosing</b>	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
<b>OutPutTags</b>	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. <b>Note:</b> 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.
<b>OutPutTagsOver</b>	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.
<b>GPIControlMsg</b>	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. gpiIndex: GPI port subscript, 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.

## 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"
_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
-------	-------------

<b>ReaderName</b>	Reader connection identifier, means which reader are reading data , e.g.: "192.168.1.116:9090"
<b>GpiIndex</b>	GPI port index, starting at 1, 1 represents GPI1, and so on.
<b>GpiState</b>	0 means low level, 1 means high level
<b>StartOrStop</b>	0 means trigger start, 1 means trigger end
<b>UTC</b>	Sensor trigger UTC time, byte[] type, length is 8, The first 4 bytes is UTC seconds and the last 4 bytes is microseconds
<b>Utc_Time</b>	Sensor trigger UTC time, string type, format is: "yyyy.MM.dd HH:mm:ss.fff"

## 2.11 Breakpoint resume

### 2.11.1 Set Breakpoint Resume

<b>Package</b>	<b>RFIDReader._Config</b>
<b>Function</b>	<b>static int SetBreakPointUpload(String ConnID, bool Switch)</b>
<b>Parameter</b>	// ConnID: connection identification // Switch: false:close, true:open e.g.: <b>SetBreakPointUpload</b> ("192.168.1.116:9090",false); This method is to close the breakpoint resume function
<b>Return</b>	0 means succeed, other value means failed.
<b>Remark</b>	Enable this function, the <b>ReadTime</b> field in the <b>Tag Model</b> will be effective
<b>Example code</b>	<pre>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("Set failed");     }else{         System.out.println("Set OK");     } }else{     System.out.println("Connection created failed"); }</pre>

### 2.11.2 Get Breakpoint Resume Setting

<b>Package</b>	<b>RFIDReader._Config</b>
<b>Function</b>	<b>static String GetBreakPointUpload(String ConnID)</b>
<b>Parameter</b>	// ConnID: connection identification

	e.g.: <b>GetBreakPointUpload</b> ("192.168.1.116:9090");
<b>Return</b>	Open or Close
<b>Remark</b>	None
<b>Example code</b>	<pre>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("Connection created failed"); }</pre>

### 2.11.3 Retrieve breakpoint cache

<b>Package</b>	RFIDReader._Config
<b>Function</b>	static <b>String</b> GetBreakPointCacheTag( <b>String</b> ConnID)
<b>Parameter</b>	// ConnID: connection identification
<b>Return</b>	Success: exist cache info, Null: no cache info, Receive Over: data returned completely
<b>Remark</b>	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.
<b>Example code</b>	<pre>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("Connection created failed"); }</pre>

### 2.11.4 Clear breakpoint cache

<b>Package</b>	RFIDReader._Config
<b>Function</b>	static <b>int</b> ClearBreakPointCache( <b>String</b> ConnID)

<b>Parameter</b>	// ConnID: connection identification
<b>Return</b>	0 means succeed, other value means failed.
<b>Remark</b>	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.
<b>Example code</b>	<pre>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("Clear cache data failed");     }else{         System.out.println("Clear cache data OK");     } } else{     System.out.println("Connection created failed"); }</pre>

## 2.12 Antenna number parameter description

- Regarding the tag read, write, lock, and 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 Example implements IAsynchronousMessage {

    // example
    public static void main(String[] args) {
        try {
```

```

Scanner sc = new Scanner(System.in);
String connID = "COM6:115200";
sc.next();
Example mc = new Example();
// RFIDReader.CreateTcpConn("192.168.1.116:9090", mc); // Create TCP connection
if(RFIDReader.CreateSerialConn(connID, mc)){ // Create Serial connection
    System.out.println("connection success...");
    RFIDReader.Stop(connID); // Stop

}else{
    System.out.println("connection failure!");
}

RFIDReader._Tag6C.GetEPC_TID(connID, eAntennaNo._1.GetNum() +
eAntennaNo._2.GetNum() + eAntennaNo._3.GetNum() + eAntennaNo._4.GetNum(),
eReadType.Inventory); // using ant1 + ant2 + ant3 + ant4 to read EPC and TID
System.out.println("Reading...");
String readKey = sc.next();
RFIDReader.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	<ol style="list-style-type: none"> <li>1. Check power light normal or not.</li> <li>2. If normal, there should be some sound when power on.</li> </ol>
Serial port couldn't work normally	<ol style="list-style-type: none"> <li>1. Check connection cable connecting normal or not .</li> <li>2. If conditional, use another device to check this cable normal or not.</li> <li>3. Try to use RJ45 to communicate.</li> <li>4. Default baud rate: 115200.</li> </ol>
RJ45 couldn't work normally	<ol style="list-style-type: none"> <li>1. Check LED working normal or not.</li> <li>2. To use Ping instruction to check cable working normal or not</li> </ol>

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