Hopeland RFID Reader Device- PC

Development Guide --JAVA

Editor：Paul

Shenzhen Hopeland Technologies Co., Ltd

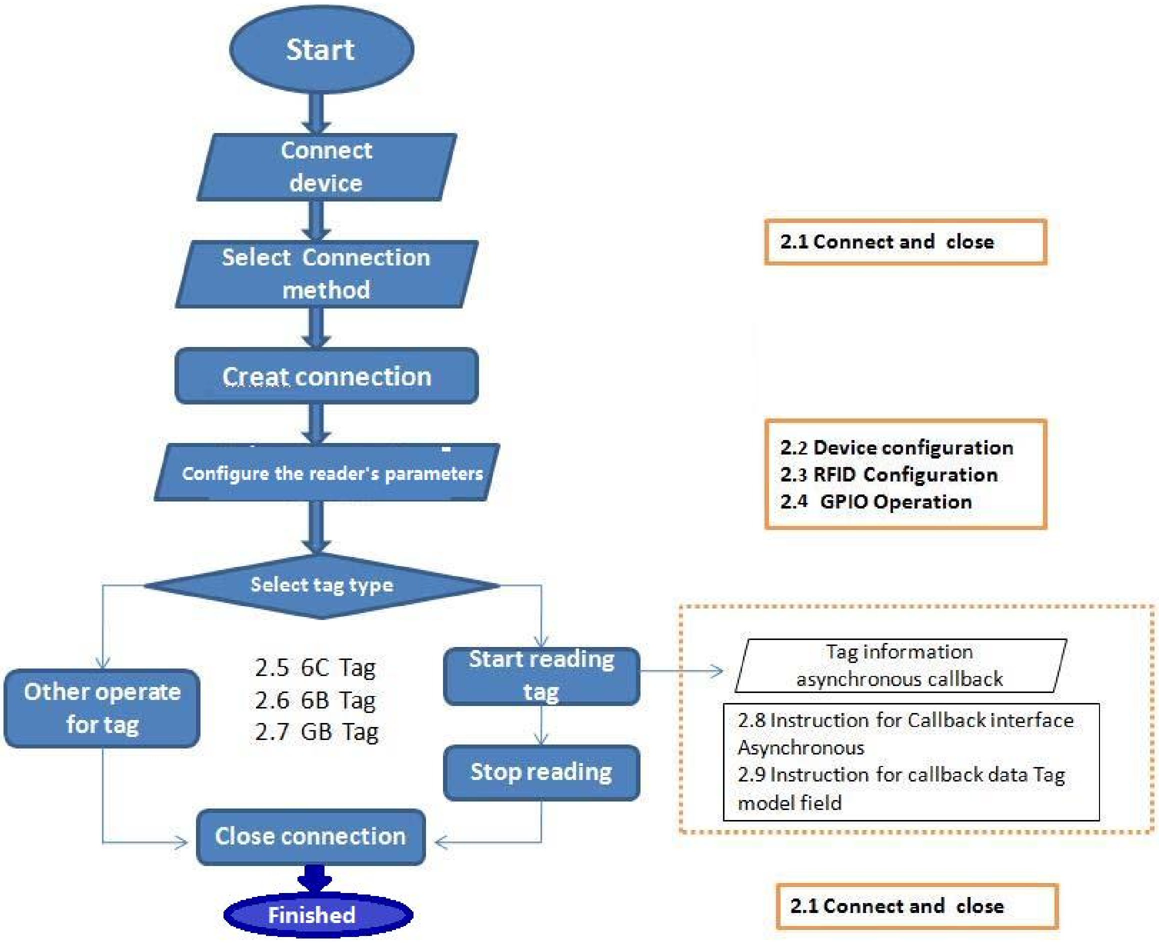
V 2.12

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.

Return	True : succeeded, error: failed.
Remark	1. The connection established by this method, “**tcpParam**” The ID that is the connection channel is distinguished from the other link channel。 2. **log:** Data callback interface, Please refer to 2.8 **callback interface Remark** for more information

Example code

```
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");
}
```

2.1.2 Close single connection

Package	RFIDReader
Function	**static void CloseConn(string connectID)**
Parameter	**connectID:** connection ID, eg:”192.168.1.116:9090”
Return	None
Remark	“**connectID**” is the connection parameter when creates connection

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    RFIDReader.CloseConn(ConnID);

if(RFIDReader._Config.GetReaderBaseBandSoftVersion(ConnID).e
quals("") ||

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");
}
```

2.1.3 Close all connections

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

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    RFIDReader.CloseAllConnect();

if(RFIDReader._Config.GetReaderBaseBandSoftVersion(ConnID).e
quals("") ||
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");
}
```

2.2 Device configuration

2.2.1 Set IP

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

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

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

```
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");
}
```

2.2.2 Get Device IP

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

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

2.2.3 Stop Instruction

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

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    if(RFIDReader._Config.Stop(ConnID) != 0){
        System.out.println("Stop failed");
    }else{
        System.out.println("Stop OK ");
    }
}else{
    System.out.println("Connection created failed");
}
```

2.2.4 Set Reader Time

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

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    if(RFIDReader._Config.SetReaderUTC(ConnID,"2020.4.19:12:02") != 0){
        System.out.println("Set reader time OK");
    }else{
        System.out.println("Set reader time failed");
    }
}else{
    System.out.println("Connection created failed");
}
```

2.2.5 Get reader time

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

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {

System.out.println(RFIDReader._Config.GetReaderUTC(ConnID));
}else{
    System.out.println("Connection created failed");
}
```

2.2.6 Set Serial Port

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

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

Return	0 means succeeded, other means failed.
Remark	NONE

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    if(RFIDReader._Config.SetReaderSerialPortParam(ConnID,
eBaudrate._115200bps) != 0){
        System.out.println("Set serial baudrate failed ");
    }else{
        System.out.println("Set serial baudrate OK");
    }
}else{
    System.out.println("Connection created failed");
}
```

2.2.7 Get Serial Port Setting

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

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

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {

System.out.println(RFIDReader._Config.GetReaderSerialPortPar
am2(ConnID));
}else{
    System.out.println("Connection created failed");
}
```

2.2.8 Set MAC Address

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

Example code

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

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	

```
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");
}
```

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	

```
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");
}
```

2.2.11 Get RS485 Address

Namespace	RFIDReader._Config
Function	**static String GetReader485(String ConnID)**
Parameter	// ConnID: connection identification
Return	RS485 address
Remark	None

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {

System.out.println(RFIDReader._Config.GetReader485(ConnID));
}else{
   System.out.println("Connection created failed");
}
```

2.2.12 Set Server/Client Mode

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

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

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

Example code

```
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");
}
```

2.2.13 Get Server/Client Mode

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

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {

System.out.println(RFIDReader._Config.GetReaderServerOrClient(ConnID));
}else{
    System.out.println("Connection created failed");
}
```

2.2.14 Get Reader Information

Namespace	RFIDReader._Config
Function	**static String GetReaderInformation(String ConnID)**
Parameter	// ConnID: connection identification
Return	Embedded application software version| reader name |reader power-up time
Remark	The reader time is in seconds

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {

System.out.println(RFIDReader._Config.GetReaderInformation(ConnID));
}else{
    System.out.println("Connection created failed");
}
```

2.2.15 Get Baseband Software Version

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

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    System.out.println(RFIDReader._Config.GetReaderBaseBandSoftVersion(ConnID));
}else{
    System.out.println("Connection created failed");
}
```

2.2.16 Get antenna standing wave ratio

Namespace	RFIDReader._Config
Function	**static String GetAntennaStandingWaveRatio(String ConnID, eAntennaNo antNo)**
Parameter	// ConnID: connection identification, antNo: Antenna number enumeration
Return	Forward power detection | backward power detection
Remark	None

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    System.out.println(RFIDReader._Config.GetAntennaStandingWaveRatio(ConnID,1));
}else{
    System.out.println("Connection created failed");
}
```

2.2.17 Restart the reader

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

Sample code

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

2.2.18 Get reader temperature

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

Sample code

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

2.2.19 Get reader SN

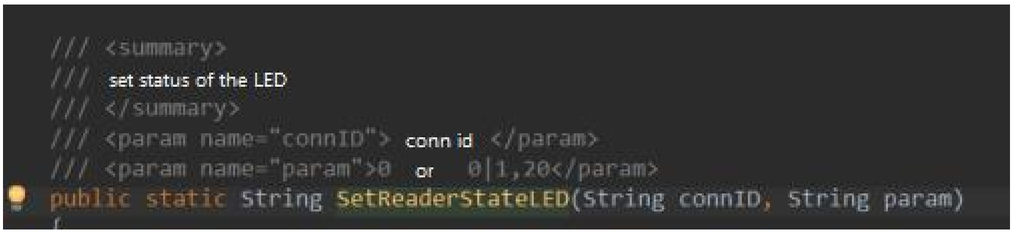
Namespace	RFIDReader._Config
Function	**public String GetSN(String ConnID)**
Parameter	// ConnID: connection identification
Return	Reader ID
Remark	None

Sample code

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

2.2.20 Set hidden light strip switch

Namespace	Param_Option
Function	**public String SetReaderStateED (String ConnID,String param)**
Parameter	// 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

	tag, in ms)
Return	0 means succeed, other value means failed.
Remark	None
Sample code	

```
/// <summary>
///  set status of the LED
/// </summary>
/// <param name="connID"> conn id </param>
/// <param name="param">0  or  0|1,20</param>
public static String SetReaderStateLED(String connID, String param)
```

2.3 RFID Configuration

2.3.1 Restore Factory Settings

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

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    if(RFIDReader._Config.SetReaderRestoreFactory(ConnID) !=
0){
        System.out.println("Restore factory settings failed");
    }else{
        System.out.println("Restore factory settings OK");
    }
}else{
    System.out.println("Connection created failed");
}
```

2.3.2 Set Baseband Parameters

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

Parameter	// ConnID: connection identification
	// basebandMode: EPC Baseband rate (0~255, 255 means AUTO)
	(0-Tari=25us, FM0, LHF=40KHz)
	(1-Tari=25us, Miller4, LHF=250KHz)

	(2-Tari=25us, Miller4, LHF=300KHz)
	(3-Tari=6.25us, FM0, LHF=400KHz)
	(255-Auto)
	// qValue: 0~15, the initial Q value used by the reader.
	// session: 0~3
	// searchType: Inventory flag parameter (0 only with Flag A inventory, 1 only Flag B inventory, 2 turns using with Flag A and Flag B double-sided inventory).

Return	0 means succeed, other value means failed.
Remark	

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    if(RFIDReader._Config.SetEPCBaseBandParam(ConnID, 255, 4, 1, 2) !=
0){
        System.out.println("Set failed");
    }else{
        System.out.println("Set OK");
    }
}else{
    System.out.println("Connection created failed");
}
```

2.3.3 Get Baseband Parameter

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

Remark	// basebandMode: EPCBaseband rate(0~255, 255 means AUTO)
	(0-Tari=25us, FM0, LHF=40KHz)
	(1-Tari=25us, Miller4, LHF=250KHz)
	(2-Tari=25us, Miller4, LHF=300KHz)
	(3-Tari=6.25us, FM0, LHF=400KHz)
	(255-Auto)
	// qValue: 0~15, the initial Q value used by the reader。
	// session: 0~3
	// searchType: Inventory flag parameters(0 only with Flag A inventory, 1 only Flag B inventory, 2 turns using with Flag A and Flag B double-sided inventory).

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
```

```
System.out.println(RFIDReader._Config.GetEPCBaseBandParam(ConnID));
}else{
    System.out.println("Connection created failed");
}
```

2.3.4 Set Antenna Power

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

Example code

```
//Set the output power of both ANT1 and ANT2 to 33dBm
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    HashMap<Integer, Integer> dicPower = new HashMap<Integer, Integer>();
    dicPower.put(1, 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");
}
```

2.3.5 Get Antenna Power

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

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

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

2.3.6 Set Tag Upload Parameters

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

Parameter	// ConnID: connection identification
	// repeatTimeFilter: Duplicate tag upload filter time (Unit: 10ms)
	// RSSIFilter: RSSI filter

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

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    if(RFIDReader._Config.SetTagUpdateParam(ConnID, 10, 0) !=
0){
        System.out.println("Set failed");
    }else{
        System.out.println("Set OK");
    }
}else{
    System.out.println("Connection created failed");
}
```

2.3.7 Get Tag upload parameters

Namespace	RFIDReader._Config
Function	**static String GetTagUpdateParam(String ConnID)**

Parameter	// ConnID: Connection identification
Return value	repeatTimeFilter|RSSIFilter

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

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {

System.out.println(RFIDReader._Config.GetTagUpdateParam(ConnID));
}else{
    System.out.println("Connection created failed");
}
```

2.3.8 Get Reader Property

Namespace	RFIDReader._Config
Function	**static String GetReaderProperty(String ConnID)**
Parameter	// ConnID: Connection identification
Returned value	Minimum transmit power | maximum transmit power | number of antennas | band list | list of RFID protocols
Explain	Output unit is dB

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {

System.out.println(RFIDReader._Config.GetTagUpdateParam(ConnID));
}else{
    System.out.println("Connection created failed");
}
```

2.3.9 Set RF Band

Namespace	RFIDReader._Config
Function	**static int SetReaderRF(String ConnID, eRF_Range eRF_Range)**

Parameter	// ConnID: Connection identification
	// eRF_Range:
	eRF_Range.GB_920_to_925MHz

	eRF_Range.GB_840_to_845MHz eRF_Range.GB_920_to_925MHz_and_GB_840_to_845MHz eRF_Range.FCC_902_to_928MHz eRF_Range.ETSI_866_to_868MHz
Returned value	0 means succeed, other value means failed.
Explain	None
Example code	

```
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");
}
```

2.3.10 Get RF Band

Namespace	RFIDReader._Config
Function	**static eRF_Range GetReaderRF(String ConnID)**
Parameter	// ConnID: Connection identification
Returned value	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
Explain	None
Example code	

```
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");
}
```

2.3.11 Set RF Frequency points

Namespace	RFIDReader._Config
Function 1	**static Int32 SetReaderWorkFrequency_GB920_to_925MHz(String ConnID, eWF_Mode wfMode, List<eGB920_to_925MHz> ListGB920_to_925MHz)**
Function 2	**static Int32 SetReaderWorkFrequency_GB_840_to_845MHz(String ConnID, eWF_Mode wfMode, List<eGB_840_to_845MHz> ListGB_840_to_845MHz)**
Function 3	**static Int32 SetReaderWorkFrequency_GB_920_to_925MHz_and_GB_840_to_845MHz(String ConnID, eWF_Mode wfMode, List<eGB_920_to_925MHz_and_GB_840_to_845MHz> ListGB_920_to_925MHz_and_GB_840_to_845MHz)**
Function 4	**static Int32 SetReaderWorkFrequency_FCC_902_to_928MHz(String ConnID, eWF_Mode wfMode, List<eFCC_902_to_928MHz> ListFCC_902_to_928MHz)**
Function 5	**static Int32 SetReaderWorkFrequency_ETSI_866_to_868MHz(String ConnID, eWF_Mode wfMode, List<eETSI_866_to_868MHz> ListETSI_866_to_868MHz)**
Function 6	**static Int32 SetReaderWorkFrequency_JP_916_to_921MHz(String ConnID, eWF_Mode wfMode, List<eJP_916_to_921MHzz> ListJP_916_to_921MHz)**
Function 7	**static Int32 SetReaderWorkFrequency_TW_922_to_927MHz(String ConnID, eWF_Mode wfMode, List<eTW_922_to_927MHz> ListTW_922_to_927MHz)**
Function 8	**static Int32 SetReaderWorkFrequency_ID_923_to_925MHz(String ConnID, eWF_Mode wfMode, List<eID_923_to_925MHz> ListID_923_to_925MHz)**
Function 9	**static Int32 SetReaderWorkFrequency_RUS_866_to_867MHz(String ConnID, eWF_Mode wfMode, List<eRUS_866_to_867MHz> ListRUS_866_to_867MHz)**
Parameter	// ConnID: Connection identification
Returned value	0 means succeed, other value means failed.
Explain	none

2.3.12 Get RF Frequency points

Namespace	RFIDReader._Config
Function	**static String GetReaderWorkFrequency(String ConnID)**
Parameter	// ConnID: Connection identification
Returned value	**Mode|frequency|frequency point value** Like: Auto|GB_920_to_925MHz|920.625,920.875
Explain	None

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {

System.out.println(RFIDReader._Config.GetReaderWorkFrequency
(ConnID));
}else{
    System.out.println("Connection created failed");
}
```

2.3.13 Set Reader Automatically Idle Mode

Namespace	RFIDReader._Config
Function	**static int SetReaderAutoSleepParam(String ConnID, BooleanSwitch, String 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 **10ms**

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    if(RFIDReader._Config.SetReaderAutoSleepParam(ConnID,
true,"100") != 0){
        System.out.println("Set failed");
    }else{
        System.out.println("Set OK");
    }
}else{
    System.out.println("Connection created failed");
}
```

2.3.14 Get Reader Automatically Idle Mode

Namespace	RFIDReader._Config
Function	**static String GetReaderAutoSleepParam(String ConnID)**
Parameter	// ConnID: Connection identification
Returned value	Close or Open|”idle time”
Explain	idle time unit is **10ms**

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {

System.out.println(RFIDReader._Config.GetReaderAutoSleepPara
m(ConnID));
}else{
    System.out.println("Connection created failed");
}
```

2.3.15 Set Antenna Enable

Namespace	RFIDReader._Config
Function	**static int SetReaderANT(String ConnID, int antNum)**
Parameter	// ConnID: Connection identification, antNum: Antenna number enumeration
Returned value	0 means succeed, other value means failed.
Explain	Specify antenna 1 and antenna 2 together, example eAntennaNo._1|eAntennaNo._2

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    if(RFIDReader._Config.SetReaderANT(ConnID, 3) != 0){
        System.out.println("Set failed");
    }else{
        System.out.println("Set OK");
    }
}else{
    System.out.println("Connection created failed");
}
```

2.3.16 Get Antenna Enable Status

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

Namespace	RFIDReader._Config
Function	**static String GetReaderANT2(String ConnID)**
Parameter	// ConnID: Connection identification
Returned value	The antenna port number that has enabled, multiple antenna numbers separate with "," like 1,6,8
Explain	None

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {

System.out.println(RFIDReader._Config.GetReaderANT2(ConnID))
;
}else{
    System.out.println("Connection created failed");
```

	}

2.3.17 Get RFID Temperature

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

Sample code

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

2.3.18 Set Expand EPC Baseband Params

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

Parameter	// ConnID: connection identification
	Example of parameter data: “1,00000000&2,00000000”
	// The param is a set of optional configuration parameters, separated by ‘&’, each optional parameter includes id and parameter, separated by ‘,’, and the parameter is 4 bytes of data (returned in hexadecimal string format)
	// Parameter 1: Big-endian format composes U32
	bit3-bit0: rfu
	bit4: tag focus enable
	bit5: fast id enable
	bit31-bit6: rfu
	// Parameter 2:
	Byte 1: maxQ
	Byte 2: minQ
	Byte 3: tmult
	Byte 4: bit 0 Dynamic start Q enable
	// Parameter 3:
	Byte 1: Antenna switching mode. 0--Switch immediately without tags, 1--Running out of resistance time
	Byte 2: Number of retries (Switch immediately without tags mode)
	Byte 3-4: Big-endian format composes U16,max antenna resistance time (x10ms)
	// Parameter 4：
	Byte 1: Waiting time between antenna switching(x10ms)
	Byte 2: antenna switching sequence
	Byte 3: Antenna protection threshold (unit is dBm), set to 0 to disable protection.
	Byte 4: reserved

	// Parameter 5： Byte 1:LBT working mode 0: disable 1: listening only 2: read tag after listening 3: read tag after meeting RSSI Byte 2: RSSI maximum value Byte3-4: reserved
Return	0 success, non 0 failure
Remark	None
Sample code	

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

2.3.19 Get Expand EPC Baseband Params

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

Return	Return data example: "1,00000000&2,00000000&3,00000000&4,00000000&5,00000000", return null ("), indicating acquisition failure.
	// The returned data is a set of optional configuration parameters, separated by ‘&’, each optional parameter includes id and parameter, separated by ‘,’, and the parameter is 4 bytes of data (returned in hexadecimal string format)
	// Parameter 1: Big-endian format composes U32
	bit3-bit0: rfu
	bit4: tag focus enable
	bit5: fast id enable
	bit31-bit6: rfu
	// Parameter 2:
	Byte 1: maxQ
	Byte 2: minQ
	Byte 3: tmult
	Byte 4: bit 0 Dynamic start Q enable
	// Parameter 3:
	Byte 1: Antenna switching mode. 0--Switch immediately without tags, 1--Running out of resistance time
	Byte 2: Number of retries (Switch immediately without tags mode)
	Byte 3-4: Big-endian format composes U16,max antenna resistance time (x10ms)
	// Parameter 4：
	Byte 1: Waiting time between antenna switching(x10ms)
	Byte 2: antenna switching sequence
	Byte 3: Antenna protection threshold (unit is dBm), set to 0 to disable protection.

	Byte 4: reserved // Parameter 5： Byte 1:LBT working mode 0: disable 1: listening only 2: read tag after listening 3: read tag after meeting RSSI Byte 2: RSSI maximum value Byte3-4: reserved
Remark	None
Sample code	

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

2.4 GPIO Operation

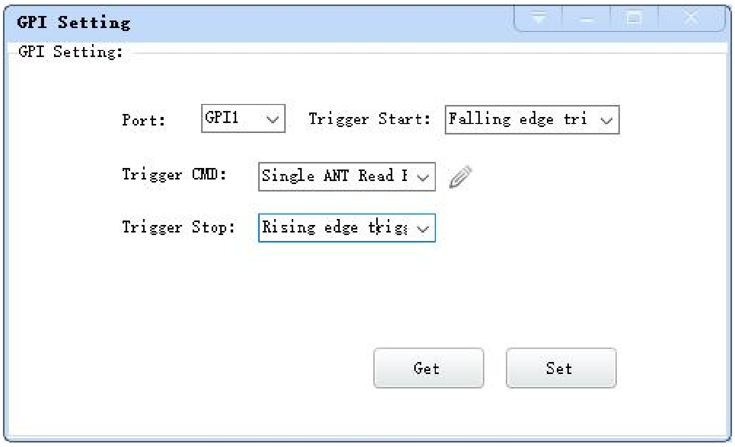
2.4.1 Set GPI Trigger Parameters

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

Parameter	// ConnID: connection identification；
	// GPINum: eGPI._1,eGPI._2,eGPI._3,eGPI._4；
	//triggerStart: Trigger start condition
	eTriggerStart.OFF,eTriggerStart.Low_level,eTriggerStart.High_level,
	eTriggerStart.Rising_edge,eTriggerStart.Falling_edge,eTriggerStart.Any_edge；
	//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；
	
	//triggerStop: Trigger stop condition
	eTriggerStop.OFF,eTriggerStop.Low_level,eTriggerStop.High_level,
	eTriggerStop.Rising_edge,eTriggerStop.Falling_edge,eTriggerStop.Any_edge,
	eTriggerStop.Delay；
	
	// 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.
	
	Eg: SetReaderGPIParam(“192.168.1.116:9090”,eGPI._2,eTriggerStart.Low_level, triggerCode.Double_Antenna_read_EPC_and_TID,eTriggerStop.Dealy,”100”,true);
	The current method configures the 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.

Return value	0 means succeed, other value means failed.
Description	● Delay time:unit is 10ms (Valid when the trigger stop condition is “delay”) For details, please refer to "RFID Reader Demo Software" For specific GPI trigger parameter callback , pls refer to 2.8 Callback interface GPIControlMsg(); **The setting of the trigger button of Bluetooth handheld reader is realized by setting the parameters of GPI1:** **The default setting:** **Port：GPI1** **Trigger Start：Falling edge (Press the trigger button)** **Trigger Command：Single ANT Read EPC** **Trigger Stop：Rising edge (Loosen the trigger button)**

	 GPI Setting GPI Setting: Port: GPI1 Trigger Start: Falling edge tri Trigger CMD: Single ANT Read I Trigger Stop: Rising edge trig

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {

if(RFIDReader._Config.SetReaderGPIParam(ConnID,eGPI._1,eTrig
gerStart.High_level,
eTriggerCode.Double_Antenna_read_EPC_and_TID,eTriggerStop.OF
F,"",true) != 0){
        System.out.println("Set failed");
    }else{
        System.out.println("Set OK");
    }
}else{
    System.out.println("Connection created failed");
}
```

2.4.2 Get GPI Trigger parameters

Nacespace	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	● Delay time:unit is 10ms (Valid when the trigger stop condition is “delay”) Upload flag: ON means upload, OFF means no upload For details, please refer to "RFID Reader Demo Software"

Example code

```
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");
}
```

2.4.3 Get GPI Status

Nacespace	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: No. 1 GPI port is currently at a low level, No. 2 GPI port is currently at a high level.
Description	The method is to actively query the current GPI level status, unrelated with trigger event! For specific GPI trigger parameter callback , pls refer to 2.8 Callback interface GPIControlMsg();

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {

System.out.println(RFIDReader._Config.GetReaderGPIState(Conn
ID));
}else{
    System.out.println("Connection created failed");
}
```

2.4.4 Set GPO Level

Nacespace	RFIDReader._Config
Function	**static Int32 SetReaderGPOState(String ConnID, HashMap<eGPO, eGPOState> dicState)**
Parameter	// ConnID: connection identification // dicState: GPO Number and level corresponding key value pairs
Return value	0 means succeed, other value means failed.

Description	None
Example code	

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

}else{
  System.out.println("Connection created failed");
}
```

2.4.5 Set Wiegand Communication Parameters

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

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

Return value	0 means succeed, other value means failed.

Description	None

Example code

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

2.4.6 Get Wiegand Communication Parameters

Namespace	RFIDReader._Config
Function	**static String GetReaderWG(String ConnID)**
Parameter	// ConnID: Connection identification
Return value	// 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.
Description	No

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {

System.out.println(RFIDReader._Config.GetReaderWG(ConnID));
}else{
    System.out.println("Connection created failed");
}
```

2.5 6C Tag operation

2.5.1 read tag

Package	RFIDReader._Tag6C
Function 1	**static int GetEPC(String ConnID, int antNum, eReadType readType)** // 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)

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

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

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

Function5	**static int GetEPC_MatchEPC(String ConnID, int antNum, int readType, String sEPC, int matchWordStartIndex, String accessPassword)** // accessPassword： Tag access password(8 Hexadecimal 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 readStart, int readLen, String accessPassword)** // accessPassword: Tag access password(8 Hexadecimal string)
Function19	**static int GetEPC_TID_UserData_MatchEPC(String ConnID, int antNum, int readType,int readStart, int readLen, String sEPC)** // match EPC, Read EPC,TID and UserData
Function20	**static int GetEPC_TID_UserData_MatchEPC(String ConnID, int antNum, int readType,int readStart, int readLen, String sEPC, int matchWordStartIndex)** // match EPC, Read EPC, TID and UserData
Function21	**static int GetEPC_TID_UserData_MatchEPC(String ConnID, int antNum, int readType,int readStart, int readLen, String sEPC, int matchWordStartIndex, String accessPassword)** // accessPassword: Tag access password(8 Hexadecimal string)
Function22	**static int GetEPC_TID_UserData_MatchTID(String ConnID, int antNum, int readType,int readStart, int readLen, String sTID)** // match TID, Read EPC, TID and UserData
Function23	**static int GetEPC_TID_UserData_MatchTID(String ConnID, int antNum, int readType,int readStart, int readLen, String sTID, int matchWordStartIndex)** // match TID, Read EPC,TID and UserData
Function24	**static int GetEPC_TID_UserData_MatchTID(String ConnID, int antNum, int readType,int readStart, int readLen, String sTID, int matchWordStartIndex, String accessPassword)** // accessPassword: Tag access password(8 Hexadecimal string)
Function25	**static int GetEPC_ReservedData(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen)** //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)

Function26	**static int GetEPC_ReservedData(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String accessPassword)** //accessPassword: tag access password

Function27	**static int GetEPC_ReservedData_MacthEPC(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String sEPC)**
	// match EPC to read reserved bank
	// sEPC: matched EPC value (Hexadecimal string)

Function28	**static int GetEPC_ReservedData_MacthEPC(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String sEPC, String accessPassword)** // accessPassword: tag access password

Function29	**static int GetEPC_ReservedData_MacthTID(String ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, String sTID, int matchWordStartIndex)**
	// 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
Parameter	Please refer to Function Remark.
Return	0 means succeed, other value means failed. Appendix A
Remark	1. For detailed Return , please kindly follow Appendix A 2. Stop inventory (cycle) reading using ”stop” instruction. 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. 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 callback interface instrunction - OutPutTagsOver();
Example code	

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    RFIDReader._Config.Stop(ConnID);

RFIDReader._Tag6C.GetEPC_MatchTID(ConnID,1,eReadType.Inventory,"E28011052000308565F90157");
    RFIDReader._Config.Stop(ConnID);

RFIDReader._Tag6C.GetEPC_MatchTID(ConnID,1,eReadType.Inventory,"E28011052000308565F90157",0);

    RFIDReader._Config.Stop(ConnID);

RFIDReader._Tag6C.GetEPC_MatchTID(ConnID,1,eReadType.Inventory,"E28011052000308565F90157",0,"00000001");
    RFIDReader._Config.Stop(ConnID);
}else{
    System.out.println("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);
}
```

2.5.2 Write tag

2.5.2.1 Write EPC

Package	RFIDReader._Tag6C

Function1	**static int WriteEPC(String ConnID, int antNum, String sWriteData)**
	// ConnID: connection identification
	// antNum: Antenna number enumeration.
	Appoint antenna 1 and 2 working at same time; e.g.: eAntennaNo._1|eAntennaNo._2 3, Please refer to the description of antenna number parameters
	// sWriteData: data to be written (Hexadecimal string)

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

Function3	**static int WriteEPC_MatchEPC(String ConnID, int antNum, String sWriteData, String sMatchData, int matchWordStartIndex, String accessPassword)**
	// match EPC, Write EPC
	// accessPassword: Tag access password(8 Hexadecimal string)

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

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

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    RFIDReader._Config.Stop(ConnID);

if(RFIDReader._Tag6C.WriteEPC_MatchTID(ConnID,1,"E002100SF00
1","E28011052000308565F90157",0,"00000001") != 0){
        System.out.println("Write EPC failed");
    }else{
        System.out.println("Write EPC OK");
    }
}else{
    System.out.println("Connection created failed");
}
```

2.5.2.2 Write Userdata

Package	RFIDReader._Tag6C

Function1	**static int WriteUserData(String ConnID, int antNum, String sWriteData,int offset)**
	// ConnID: connection identification
	// antNum: Antenna number enumeration.
	// sWriteData: data to be written（Hexadecimal string）
	//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	**static int WriteUserData_MatchEPC(String ConnID, int antNum, String sWriteData, int offset,String sMatchData, int matchWordStartIndex)**
	// match EPC, Write user data
	// sMatchData, EPC data to be matched (Hexadecimal string)
	// matchWordStartIndex, match Data starting index

Function3	**static int WriteUserData_MatchEPC(String ConnID, int antNum, String sWriteData, int offset,String sMatchData, int matchWordStartIndex, String accessPassword)**
	// match EPC, Write user data
	// accessPassword, Tag access password

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

Function5	**static int WriteUserData_MatchTID(String ConnID, int antNum, String sWriteData, int offset,String sMatchData, int matchWordStartIndex, String accessPassword)** // match TID, Write user data
Parameter	Please refer to Function Remark.
Return	0 means succeed, other value means failed.Appendix A
Remark	1. Suggest to use matched ID to write, means to use “Function4” and “Function5”. 2. For detailed Return Remark, please see the appendix.

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    RFIDReader._Config.Stop(ConnID);
if(RFIDReader._Tag6C.WriteUserData_MatchTID(ConnID,1,"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");
}
```

2.5.2.3 Write password

Package	RFIDReader._Tag6C

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

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

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

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

Function5	**static int WriteDestroyPassWord(String ConnID, int antNum, String sWriteData, String accessPassword)**
	// Write the kill password
	// accessPassword: Tag access password (8 Hexadecimal string data)

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

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

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    RFIDReader._Config.Stop(ConnID);

if(RFIDReader._Tag6C.WriteAccessPassWord_MatchTID(ConnID,1,"
00000002","E28011052000308565F90157",0,"00000001") != 0){
        System.out.println("Write failed");
    }else{
        System.out.println("Write OK");
    }
}else{
    System.out.println("Connection created failed");
}
```

2.5.3 Lock tag

Package	RFIDReader._Tag6C

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

Function2	**static int Lock_MatchEPC(String ConnID, int antNum, eLockArea lockArea, eLockType lockType, String sMatchData, Int32 matchWordStartIndex)**
	// sMatchData: EPC data to be matched (Hexadecimal string)
	// matchWordStartIndex: match data' s word initial address

Function3	**static int Lock_MatchEPC(String ConnID, int antNum, eLockArea lockArea, eLockType lockType, String sMatchData, Int32 matchWordStartIndex, String accessPassword)** // accessPassword: Tag access password

Function4	**static int Lock_MatchTID(String ConnID, int antNum, eLockArea lockArea, eLockType lockType, String sMatchData, Int32 matchWordStartIndex)**
	// sMatchData: TID data to be matched(Hexadecimal string)
	// matchWordStartIndex: match Data starting index

Function5	**static int Lock_MatchTID(String ConnID, int antNum, eLockArea lockArea, eLockType lockType, String sMatchData, Int32 matchWordStartIndex, String accessPassword)** // accessPassword: Tag access password
Parameter	// refer to above method Remark
Return	0 means succeed, other value means failed. Appendix A
Remark	

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    RFIDReader._Config.Stop(ConnID);

if(RFIDReader._Tag6C.Lock_MatchTID(ConnID,1,eLockArea.epc,eLockType.Lock,"E280110520003O8565F90157",0,"00000001") != 0){
        System.out.println("Lock failed");
    }else{
        System.out.println("Lock OK");
    }
}else{
    System.out.println("Connection created failed");
}
```

2.5.4 Kill tag

Package	RFIDReader._Tag6C

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

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

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

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

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    RFIDReader._Config.Stop(ConnID);

if(RFIDReader._Tag6C.Destroy_MatchTID(ConnID,1,"00000002","E
28011052000308565F90157",0) != 0){
        System.out.println("Kill failed");
    }else{
        System.out.println("Kill OK");
    }
}else{
    System.out.println("Connection created failed");
}
```

2.6 6B tag operation

2.6.1 Read tag

Package	RFIDReader._Tag6B
Function 1	**static int Get6B(String ConnID, int antNum, int readType, e6BReaderContent readerContent)**

	// ConnID: connection identification
	// antNum: Antenna number enumeration
	Appoint Antenna 1 and 2 working at same time; e.g.: 3 Please refer to the antenna numbering parameter
	// ReadType:read type enumeration（single or cyclical reading）
	// readerContent: read content enumeration

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

2.6.2 Write tag

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

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

Return	0 means succeed, other value means failed.
Remark	Error code please see the appendix Remark

2.6.3 Lock tag

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

Parameter	// ConnID: connection identification
	// antNum: Antenna number enumeration.
	Appoint Antenna 1 and 2 working at same time; e.g.: 3 Please refer to the antenna numbering parameter
	// sTID: Matched TID data (Hex.)
	// lockIndex： The index of the lock area

Return	0 means succeed, other value means failed.
Remark	Error code please see the appendix Remark

2.6.4Lock query

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

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

Parameter	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 int WriteGB(String ConnID, int antNum, String 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.

Description	Please refer to the appendix for the error code

2.7.3 Lock tag

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

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

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

2.7.4 Destroy tag

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

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

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

2.8 Callback interface IAsynchronousMessage description

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

}
```

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

ReaderName	Reader connection identifier, means which reader are reading data, e.g.: ”192.168.1.116:9090”
GpiIndex	GPI port index, starting at 1, 1 represents GPI1, and so on.
GpiState	0 means low level, 1 means high level
StartOrStop	0 means trigger start, 1 means trigger end
UTC	Sensor trigger UTC time, byte[] type, length is 8, The first 4 bytes is UTC seconds and the last 4 bytes is microseconds
Utc_Time	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

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

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

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

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    RFIDReader._Config.Stop(ConnID);
    if(RFIDReader._Config.SetBreakPointUpload(ConnID,true) !=
0){
        System.out.println("Set failed");
    }else{
        System.out.println("Set OK");
    }
}else{
    System.out.println("Connection created failed");
}
```

2.11.2 Get Breakpoint Resume Setting

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

	e.g.: **GetBreakPointUpload**(“192.168.1.116:9090”);
Return	Open or Close
Remark	None

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    RFIDReader._Config.Stop(ConnID);

System.out.println(RFIDReader._Config.GetBreakPointUpload(ConnID));
}else{
    System.out.println("Connection created failed);
}
```

2.11.3 Retrieve breakpoint cache

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

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {

System.out.println(RFIDReader._Config.GetBreakPointCacheTag(ConnID));
}else{
    System.out.println("Connection created failed");
}
```

2.11.4 Clear breakpoint cache

Package	RFIDReader._Config
Function	**static int ClearBreakPointCache(String ConnID)**

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

Example code

```
String ConnID = "192.168.1.116:9090";
IAsynchronousMessage log = new SampleCode();
if(RFIDReader.CreateTcpConn(ConnID, log)) {
    RFIDReader._Config.Stop(ConnID);
    if(RFIDReader._Config.ClearBreakPointCache(ConnID) != 0){
        System.out.println("Clear cache data failed");
    }else{
        System.out.println("Clear cache data OK");
    }
}else{
    System.out.println("Connection created failed");
}
```

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	1. Check power light normal or not. 2. If normal, there should be some sound when power on.
Serial port couldn't work normally	1. Check connection cable connecting normal or not . 2. If conditional, use another device to check this cable normal or not. 3. Try to use RJ45 to communicate. 4. Default baud rate: 115200.
RJ45 couldn't work normally	1. Check LED working normal or not. 2. To use Ping instruction to check cable working normal or not

	3. Try serial port connection, inquiry IP correct by Demo 4. Default IP and port: ”192.168.1.116:9090”

Appendix A: 6C tag operation returns the error code

Read Error code table:

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

Write Error code table:

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

Lock Error code table:

Code	Remark
0	Lock succeeded
1	Antenna port parameter error
2	Choose parameter error
3	Write parameter error
4	CRC correcting error
5	Power no enough
6	data block overflow
7	data block is locked
8	Access password error
9	Other tag error
10	Tag lost
11	Reader instruction error

Kill Error code table:

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

8	Instruction error

Appendix B: 6B tag operation returns the error code

Read Error code table:

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

Write Error code table:

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

Lock Error code table:

Code	Remark
0	Lock succeeded
1	Other error