



Focus on RFID core technology

# Hopeland RFID READER - PC Development Guide .net

Editor: Paul

Shenzhen Hopeland Technologies Co., Ltd  
V 4.40



# CONTENTS

1. Overview.....	- 4 -
1.1 Content Overview.....	- 4 -
1.2 Development Process.....	- 4 -
1.3 Applicable Models.....	- 5 -
1.4 Copyright Statement.....	- 5 -
2. API Modules Function Description.....	- 5 -
2.1 Connect & Disconnect.....	- 6 -
2.1.1 Create Serial Port Connection.....	- 6 -
2.1.2 Create TCP connection.....	- 6 -
2.1.3 Create 485 Connection.....	- 7 -
2.1.4 Create 485 Connection (multiple 485 connections).....	- 7 -
2.1.5 Create USB Connection.....	- 8 -
2.1.6 Check Connection Status.....	- 8 -
2.1.7 Close Single Connection.....	- 9 -
2.1.8 Close All Connections.....	- 10 -
2.1.9 Open TCP Server Listening.....	- 10 -
2.1.10 Close TCP Server Listening.....	- 11 -
2.1.11 Get Server Listening Status.....	- 11 -
2.1.12 Set API Language Type.....	- 12 -
2.1.13 Get API Language Type.....	- 12 -
2.1.14 Start Search Devices.....	- 13 -
2.1.15 Stop Search Devices.....	- 13 -
2.1.16 Set Command Response Timeout.....	- 13 -
2.2 Device Configuration.....	- 13 -
2.2.1 Set IP Parameters.....	- 13 -
2.2.2 Get IP Parameters.....	- 14 -
2.2.3 Stop Command.....	- 14 -
2.2.4 Set Device Time.....	- 15 -
2.2.5 Get Device Time.....	- 15 -
2.2.6 Set Serial Port Parameter.....	- 15 -
2.2.7 Get Serial Port Parameter.....	- 16 -
2.2.8 Set MAC address.....	- 17 -
2.2.9 Get MAC address.....	- 17 -
2.2.10 Set RS485 Address.....	- 17 -
2.2.11 Get RS485 Address.....	- 18 -
2.2.12 Set DHCP Switch.....	- 18 -
2.2.13 Get DHCP Switch Status.....	- 19 -
2.2.14 Set Server/client Mode.....	- 19 -
2.2.15 Get Server/client Mode.....	- 19 -
2.2.16 Get Device Information.....	- 20 -
2.2.17 Get Baseband version.....	- 20 -
2.2.18 Get Antenna Standing Wave Ratio.....	- 20 -
2.2.19 Get Tag Data Output Format.....	- 21 -
2.2.20 Set Tag Data Output Format.....	- 21 -
2.2.21 Restart.....	- 22 -
2.2.22 Get Reader Temperature.....	- 22 -
2.2.23 Set Custom Code.....	- 23 -
2.2.24 Get Custom Code.....	- 23 -
2.2.25 Set Buzzer Control.....	- 24 -
2.2.26 Set Buzzer Switch.....	- 24 -
2.2.27 Get Reader Serial Number.....	- 24 -
2.3 RFID configuration.....	- 25 -
2.3.1 Reset Factory.....	- 25 -
2.3.2 Set Baseband Parameters.....	- 25 -
2.3.3 Get Baseband Parameters.....	- 26 -
2.3.4 Set Antenna Port Output Power.....	- 26 -
2.3.5 Get Antenna Port Output Power.....	- 27 -
2.3.6 Set Tag Data Upload Parameters.....	- 27 -
2.3.7 Get Tag Data Upload Parameters.....	- 27 -
2.3.8 Get Device Property.....	- 28 -

2.3.9 Set RF Frequency Range.....	- 28 -
2.3.10 Get RF Frequency Range.....	- 29 -
2.3.11 Set RF Working Frequency Points.....	- 29 -
2.3.12 Get RF Working Frequency Points.....	- 30 -
2.3.13 Set Device Auto-sleep Mode.....	- 31 -
2.3.14 Get Device Auto-sleep Mode.....	- 31 -
2.3.15 Set Reader Network Self-check.....	- 32 -
2.3.16 Get Reader Network Self-check Status.....	- 32 -
2.3.17 Set Antenna Enable.....	- 32 -
2.3.18 Get Antenna Enable Status.....	- 33 -
2.3.19 Get RFID Temperature.....	- 33 -
2.3.20 Baseband Expansion Settings.....	- 34 -
2.4 GPIO operation.....	- 34 -
2.4.1 Set GPI Trigger Parameters.....	- 34 -
2.4.2 Get GPI Trigger Parameters.....	- 35 -
2.4.3 Get GPI State.....	- 36 -
2.4.4 Set GPO Level Status.....	- 37 -
2.4.5 Set Wiegand Parameters.....	- 37 -
2.4.6 Get Wiegand Parameters.....	- 38 -
2.5 6C Tag Operation.....	- 38 -
2.5.1 Read Tag.....	- 38 -
2.5.2 Write Tag.....	- 41 -
2.5.3 Lock Tag.....	- 44 -
2.5.4 Kill Tag (destroy).....	- 44 -
2.5.5 Get QT Parameters.....	- 45 -
2.5.6 Set QT Parameters.....	- 46 -
2.5.6 G2V2 Untraceable Operation.....	- 46 -
2.6 6B Tag Operation (omitted).....	- 48 -
2.7 Callback Interface IAsynchronousMessage.....	- 48 -
2.8 Callback Data Tag_Model Field Introductions.....	- 49 -
2.9 Callback Data GPI_Model Field Introductions.....	- 49 -
2.10 Breakpoint Continuous Transferring.....	- 50 -
2.10.1 Set.....	- 50 -
2.10.2 Get.....	- 50 -
2.10.3 Retrieve Breakpoint Cache Data.....	- 50 -
2.10.4 Clear Breakpoint Cache Data.....	- 51 -
2.11 WiFi Operation.....	- 51 -
2.11.1 Get WiFi Switch Status.....	- 51 -
2.11.2 Set Wi-Fi Switch Status.....	- 52 -
2.11.3 Get Wi-Fi Adaptor IP.....	- 52 -
2.11.4 Set Wi-Fi Adaptor IP.....	- 53 -
2.11.5 Search Wi-Fi Hotspot.....	- 53 -
2.11.6 Save Wi-Fi Searching Result.....	- 53 -
2.11.7 Connect Wi-Fi Hotspot.....	- 54 -
2.11.8 Get Current Connected Wi-Fi Name.....	- 55 -
2.12 Whitelist Function.....	- 55 -
2.12.1 Set Whitelist Parameters.....	- 55 -
2.12.2 Get Whitelist Parameters.....	- 56 -
2.12.3 Upload Whitelist Database File To Reader.....	- 56 -
2.13 PSAM card operation.....	- 58 -
2.13.1 Reset PSAM card.....	- 58 -
2.13.2 Send APDU command.....	- 58 -
2.14 Antenna Number Parameter Description.....	- 59 -
2.15 Callback Interface ISearchDevice Field Instructions.....	- 59 -
2.16 Callback Data Device_Model Filed Instructions.....	- 59 -
3. Programming Example.....	- 60 -
4. FAQ and Solutions.....	- 61 -
5. Appendix A: 6C Tag Operation Returned Error Code.....	- 61 -

# 1. Overview

## 1.1 Content Overview

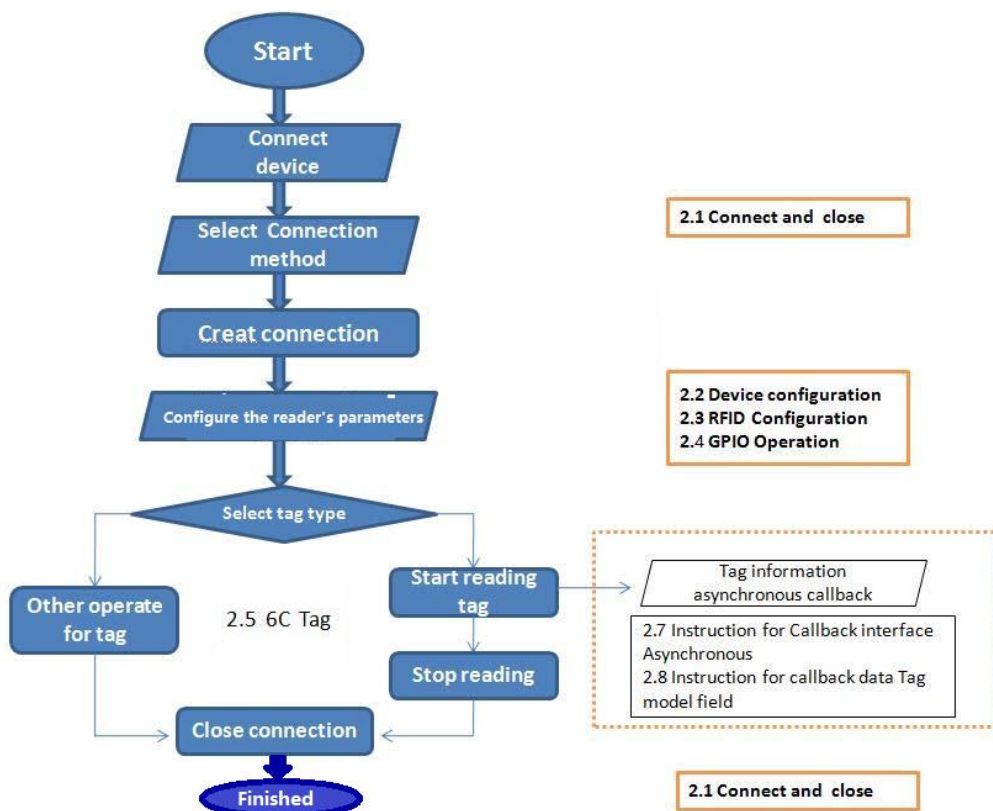
In order to facilitate the secondary development of users, we provide a function library that can run on the .net platform. The function library is written in C# language and packaged into a standard dynamic link library, and the development environment is .Net Framework 2.0.

This guide introduces the corresponding technical indicators, application development notes and precautions, application interface function description and so on.

Statement:

1. For the operation of RFID management software (DEMO), please refer to the demo software user manual and the related product user manual.
2. If this document still couldn't meet your development requirements, please kindly contact us for support

## 1.2 Development Process



## 1.3 Applicable Models

This document lists all RFID devices' API, the following table lists supportable function of different models (please refer to specific notes of function module details)

Function module	Applicable models
Connection operation	CL7206A/CL7206B/CL720C series/Shine/Sharp/Smart
Device configuration	All products
RFID configuration	All products
GPO operation	CL7206B/CL7206C series/Shine/Sharp/Smart
6C tag operation	All products
6B tag operation	All products

## 1.4 Copyright Statement

All contents of this document, including text and pictures, are original. The company reserves the right to pursue legal liabilities for those who use it for commercial purposes without permission.

Without authorization, users are not allowed to add, modify, or delete the contents of this document, or disseminate it through the Internet or CD-ROM. If you violate it, you will be responsible for the consequences.

## 2. API Modules Function Description

### Instance code variable declaration

```
List<Tag_Model> dataList = new List<Tag_Model>();
static RFIDReaderAPI.Param_Option PARAM_SET = new RFIDReaderAPI.Param_Option();
static RFIDReaderAPI.RFID_Option RFID_OPTION = new RFIDReaderAPI.RFID_Option();
static string tid = "E280110C20007719E0B20973";
static string epc = "ADB5F1DC518721D7710158EF";
static string ConnID = "192.168.1.116:9090";
string usbConnID =
@"\\?\hid#vid_2121&pid_8633#6&297867e4&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}";
static string SerialConnId = "COM3:115200";
```

## 2.1 Connect & Disconnect

### 2.1.1 Create Serial Port Connection

Namespace	<b>RFIDReaderAPI.RFIDReader</b>
Function	<b>static bool CreateSerialConn(string serialParam, IAsynchronousMessage log)</b>
Parameter	<b>serialParam</b> : serial port connection Parameter, eg: "COM1:115200" <b>log</b> : Data callback interface, all tags data will be called back from this interface.
Return	True: successful; false: failed
Remark	1. Connection created by this method, "serialParam" will be this connection channel's ID, which is different with other connection channel. 2. <b>log</b> : Data callback interface, please refer to <a href="#">2.7callback interface introductions</a> for more information.
Example code	<pre> RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateSerialConn(SerialConnId, log)) {     Console.WriteLine("Create connection successfully!"); } else {     Console.WriteLine("Create connection failed!"); } </pre>

### 2.1.2 Create TCP connection

Namespace	<b>RFIDReaderAPI.RFIDReader</b>
Function	<b>static bool CreatTcpConn(string tcpParam, IAsynchronousMessage log)</b>
Parameter	<b>tcpParam</b> : TCP connection Parameter, eg: "192.168.1.116:9090" <b>log</b> : Data callback interface, all tags data will be called back from this interface.
Return	True: successful; false: failed
Remark	1. Connection created by this method, "tcpParam" will be this connection channel's ID, which is different with other connection channel. 2. <b>log</b> : Data callback interface, Please refer to <a href="#">2.7callback interface introductions</a> for more information
Example code	<pre> RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreatTcpConn("192.168.1.116:9090", log)) // TCP Connect {     Console.WriteLine("Create connection successfully!"); } else {     Console.WriteLine("Create connection failed!"); } </pre>

## 2.1.3 Create 485 Connection

Namespace	<b>RFIDReaderAPI.RFIDReader</b>
Function	<b>static bool Create485Conn(string _485Param, IAsynchronousMessage log)</b>
Parameter	<b>_485Param</b> :RS485 connection Parameter, eg:"1:COM1:115200", "1" as 485 connection address. <b>log</b> : Data callback interface, all tags data will be called back from this interface.
Return	True: successful; false: failed
Remark	1. Connection created by this method, " <b>485Param</b> " will be this connection channel's ID which is different with other connection channel. 2. <b>log: Data callback interface</b> , Please refer to <a href="#">2.7callback interface introductions</a> for more information.
Example code	<pre> RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.Create485Conn("1:COM1:115200", log)) {     Console.WriteLine("Create connection successfully!"); } else {     Console.WriteLine("Create connection failed!"); } </pre>

## 2.1.4 Create 485 Connection (multiple 485 connections)

Namespace	<b>RFIDReaderAPI.RFIDReader</b>
Function	<b>static bool Create485sConn(string _485Param, IAsynchronousMessage log)</b>
Parameter	<b>_485Param</b> :RS485 connection Parameter, eg:"1:COM1:115200", "1" as 485 connection address. <b>log</b> :Data callback interface, all tags data will be called back from this interface.
Return	True: successful; false: failed
Remark	1. Connection created by this method, " <b>485Param</b> " will be this connection channel's ID which is different with other connection channel. 2. <b>log: Data callback interface</b> , Please refer to <a href="#">2.7callback interface introductions</a> for more information. 3. This function supports one com port to create multiple 485 connections. eg: 1:COM1:115200 2:COM1:115200 Note: For this funtion, if the current connection is reading tags, we need to stop reading first, then can switch to other 485 connection.
Example code	<pre> RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.Create485sConn("1:COM1:115200", log)) {     Console.WriteLine("485 address 1 connection created successfully!"); } else {     Console.WriteLine("485 address 1 connection created failed!"); } if (RFIDReaderAPI.RFIDReader.Create485sConn("2:COM1:115200", log)) { </pre>

	<pre> Console.WriteLine("485 address 2 connection created successfully!"); } else {     Console.WriteLine("485 address 2 connection created failed!"); } </pre>
--	---

## 2.1.5 Create USB Connection

Namespace	<b>RFIDReaderAPI.RFIDReader</b>
Function	static <b>bool</b> CreateUsbConn( <b>string</b> usbParam, <b>IntPtr</b> Handle, <b>IAynchronousMessage</b> log)
Parameter	<b>usbParam</b> : USB connect parameter <b>Handle</b> : window handle or service handle <b>log</b> : Data callback interface, all the tag data will be callback from the interface object
Return	True: successful; false: failed
Remark	1. parameter " <b>usbParam</b> ", through same namespace static <b>List&lt;string&gt;</b> <b>GetUsbHidDeviceList()</b> to get connection parameter of USB device list 2. <b>log</b> data callback interface, refer to <a href="#">2.7callback interface introductions</a> for detailed information. 3. Default name is "UHF READER" after USB connected
Example code	<pre> RFIDReaderAPI.Interface.IAynchronousMessage log = new Program(); //Gets the console handle IntPtr Handle = User32API.GetCurrentWindowHandle(); List&lt;String&gt; driverlist = RFIDReader.GetUsbHidDeviceList(); ConnID = driverlist[0]; if (RFIDReader.CreateUsbConn(ConnID , Handle, log)) {     Console.WriteLine("Create connection successfully! "); } else {     Console.WriteLine("Create connection failed!"); } </pre>

## 2.1.6 Check Connection Status

Namespace	<b>RFIDReaderAPI.RFIDReader</b>
Function	static <b>bool</b> CheckConnect( <b>string</b> connectID)
Parameter	<b>connectID</b> : connection ID, eg:"192.168.1.116:9090"
Return	True: successful; false: failed
Remark	" <b>connectID</b> " is the connection parameter when creates connection
Example code	<pre> RFIDReaderAPI.Interface.IAynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTCPConn("192.168.1.116:9090", log)) </pre>



	<pre> {     Console.WriteLine("Create connection successfully!");     if (RFIDReader.CheckConnect("192.168.1.116:9090"))     {         Console.WriteLine("The current connection is normal.");     }     else     {         Console.WriteLine("The current connection is abnormal.");     } } else {     Console.WriteLine("Create connection failed!"); } </pre>
--	---

## 2.1.7 Close Single Connection

Namespace	<b>RFIDReaderAPI.RFIDReader</b>
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	<pre> RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn("192.168.1.116:9090", log)) {     Console.WriteLine("Create connection successfully!");     RFIDReader.CloseConn("192.168.1.116:9090");     if (RFIDReader.CheckConnect("192.168.1.116:9090"))     {         Console.WriteLine("The current connection is normal.");     }     else     {         Console.WriteLine("The current connection is abnormal.");     } } else {     Console.WriteLine("Create connection failed!"); } </pre>

## 2.1.8 Close All Connections

Namespace	<b>RFIDReaderAPI.RFIDReader</b>
Function	<b>static void CloseAllConnect()</b>
Parameter	None
Return	None
Remark	This method will close all created connections.
Example code	<pre> RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn("192.168.1.116:9090", log)) {     Console.WriteLine("Create connection successfully!");     RFIDReader.CloseAllConnect();     Console.WriteLine("Close all connections");     if (RFIDReader.CheckConnect("192.168.1.116:9090"))     {         Console.WriteLine("The current connection is normal.");     }     else     {         Console.WriteLine("The current connection is abnormal.");     } } else {     Console.WriteLine("Create connection failed!"); } </pre>

## 2.1.9 Open TCP Server Listening

Namespace	<b>RFIDReaderAPI.RFIDReader</b>
Function	<b>static Boolean OpenTcpServer(string serverIP,string serverPort,IAsynchronousMessage log)</b>
Parameter	<b>serverIP</b> : the IP been listening. <b>serverPort</b> : the port been listening <b>Log</b> : all data callback interface after connected , details refer to: <a href="#">2.7callback interface introductions</a>
Return	Listen successful or not
Remark	1. Open server listening 2. When device in client mode, will connect related listening port automatically eg.: testing IP and port under client mode of reader as"IP:192.168.1.75 Port:9090" Code example: <b>OpenTcpServer</b> ("192.168.1.75","9090",log); 3. The default connecting IP (client mode) is "192.168.1.1", port is "9090"
Example code	<pre> RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReader.OpenTcpServer("192.168.1.116", "9090", log)) { </pre>

	<pre>Console.WriteLine("Start TCP server listening successfully!"); }</pre>
--	---

## 2.1.10 Close TCP Server Listening

Namespace	<b>RFIDReaderAPI.RFIDReader</b>
Function	<b>static void CloseTcpServer()</b>
Parameter	None
Return	None
Remark	Close TCP server listening
Example code	<pre>RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReader.OpenTcpServer("192.168.1.110", "9090", log)) {     Console.WriteLine("Start TCP server listening successfully!");     RFIDReader.CloseTcpServer();     Console.WriteLine("TCP server listening closed.");     if (RFIDReader.GetServerStartUp())     {         Console.WriteLine("listening started");     }     else     {         Console.WriteLine("listening closed");     } }</pre>

## 2.1.11 Get Server Listening Status

Namespace	<b>RFIDReaderAPI.RFIDReader</b>
Function	<b>static bool GetServerStartUp()</b>
Parameter	None
Return	Server listening or not
Remark	True: listening, false: not in listening
Example code	<pre>RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReader.OpenTcpServer("192.168.1.110", "9090", log)) {     Console.WriteLine("Start TCP server listening successfully!");     if (RFIDReader.GetServerStartUp())     {         Console.WriteLine("listening started");     }     else     { </pre>

	<pre> Console.WriteLine("listening closed");     } } </pre>
--	---

## 2.1.12 Set API Language Type

Namespace	<b>RFIDReaderAPI.RFIDReader</b>
Function	static void SetAPILanguageType(int type)
Parameter	Type: eAPILanguage.Chinese is Chinese, eAPILanguage.English is English
Return	None
Remark	Change the API information prompt language type, non-persistent, default is English
Example code	<pre> RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn("192.168.1.116:9090", log)) {     RFIDReader.SetAPILanguageType(eAPILanguage.Chinese);     Console.WriteLine("Set the API prompt to Chinese ");     string language = RFIDReader.GetAPILanguageType();     Console.WriteLine("Current language is:"+language); } else {     Console.WriteLine("Create connection failed!"); } </pre>

## 2.1.13 Get API Language Type

Namespace	<b>RFIDReaderAPI.RFIDReader</b>
Function	static int GetAPILanguageType()
Parameter	None
Return	"0" is English, "1" is Chinese
Remark	The default language is English
Example code	<pre> RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn("192.168.1.116:9090", log)) {     RFIDReader.SetAPILanguageType(eAPILanguage.Chinese);     Console.WriteLine("Set the API prompt to Chinese ");     string language = RFIDReader.GetAPILanguageType();     Console.WriteLine("Current language is:"+language); } else {     Console.WriteLine("Create connection failed!"); } </pre>

## 2.1.14 Start Search Devices

Namespace	<b>RFIDReaderAPI.RFIDReader</b>
Function	<b>static bool StartSearchDevice (ISearchDevice msg)</b>
Parameter	msg: Device data information callback interface For details of the interface, please refer to 2.14 IsearchDevice interface description Only applicable to devices with TCP connection
Return	Ture or false
Remark	None
Example code	<pre>// this: Has implemented the interface ISearchDevice if (!RFIDReaderAPI.RFIDReader.StartSearchDevice(this)) {     ShowMessage("Start Search Failure!"); }</pre>

## 2.1.15 Stop Search Devices

Namespace	<b>RFIDReaderAPI.RFIDReader</b>
Function	<b>static int StopSearchDevice ()</b>
Parameter	None
Return	None
Remark	Turn off search
Example code	<pre>RFIDReaderAPI.RFIDReader.StopSearchDevice();</pre>

## 2.1.16 Set Command Response Timeout

Namespace	<b>RFIDReaderAPI.RFIDReader</b>
Function	<b>static string SetTimeOut (int time)</b>
Parameter	Time: Timeout duration, unit ms
Return	Result 0 OK
Remark	Set API response timeout, default timeout is 2000ms
Example code	<pre>RFIDReader.SetTimeOut(1000)</pre>

## 2.2 Device Configuration

### 2.2.1 Set IP Parameters

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static int32 Set ReaderNetworkPortParam(String ConnID, String iP, String mask, String gateway)</b>
Parameter	// ConnID: connection identifier // 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"
<b>Return</b>	0: successful; others: failed
<b>Remark</b>	1. This method will close all created connection.
<b>Example code</b>	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.SetReaderNetworkPortParam         (tcp, "192.168.1.115", "255.255.255.0", "192.168.1.1");     if (rt == 0) Console.WriteLine("SET OK");     else Console.WriteLine("SET FAILED"); } else {     Console.WriteLine("Create connection failed!"); } </pre>

## 2.2.2 Get IP Parameters

<b>Namespace</b>	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
<b>Function</b>	<b>static String GetReaderNetworkPortParam(String ConnID)</b>
<b>Parameter</b>	// ConnID: connection identifier
<b>Return</b>	<b>IP add   Subnet mask   Gateway</b> , e.g. : "192.168.1.116 255.255.255.0 192.168.1.1"
<b>Remark</b>	
<b>Example code</b>	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     string nowParam = RFIDReader._ReaderConfig.GetReaderNetworkPortParam(tcp);     Console.WriteLine(nowParam); } else {     Console.WriteLine("Create connection failed!"); } </pre>

## 2.2.3 Stop Command

<b>Namespace</b>	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
<b>Function</b>	<b>static Int32 Stop(String ConnID)</b>
<b>Parameter</b>	// ConnID: connection identifier
<b>Return</b>	0: successful; others: failed
<b>Remark</b>	1. This method will make reader stop current workings. 2. Stop inventory reading using this method
<b>Example code</b>	<b>RFIDReader._RFIDConfig.Stop("192.168.1.116:9090");</b>

## 2.2.4 Set Device Time

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	static <b>Int32</b> SetReaderUTC( <b>String</b> ConnID, <b>String</b> param)
Parameter	// ConnID: connection identifier // param: time parameter"yyyy.MM.dd HH:mm:ss", e.g.: "1970.01.01 00:00:00"
Return	0: successful; others: failed
Remark	None
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.SetReaderUTC(tcp, "2019.11.15 15:15:02");     if (rt == 0) Console.WriteLine("SET OK"); } else {     Console.WriteLine("Create connection failed!"); } </pre>

## 2.2.5 Get Device Time

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	static <b>String</b> GetReaderUTC( <b>String</b> ConnID)
Parameter	// ConnID: connection identifier
Return	Time Parameter"yyyy.MM.dd HH:mm:ss", e.g.: "1970.01.01 00:00:00"
Remark	None
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     string time = RFIDReader._ReaderConfig.GetReaderUTC(tcp);     Console.WriteLine(time); } else {     Console.WriteLine("Create connection failed!"); } </pre>

## 2.2.6 Set Serial Port Parameter

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	static <b>Int32</b> SetReaderSerialPortParam( <b>String</b> ConnID, <b>eBaudrate</b> baudRate)

Parameter	// ConnID: connection identifier // baudRate: eBaudrate._9600bps, eBaudrate._19200bps, eBaudrate._115200bps, eBaudrate._230400bps, eBaudrate._460800bps.
Return	0: successful; others: failed
Remark	None
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.SetReaderSerialPortParam(tcp,eBaudrate._115200bps);     if (rt == 0) Console.WriteLine("SET OK"); } else {     Console.WriteLine("Create connection failed!"); }</pre>

## 2.2.7 Get Serial Port Parameter

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static eBaudrate GetReaderSerialPortParam(String ConnID)</b>
Parameter	// ConnID: connection identifier
Return	eBaudrate._9600bps, eBaudrate._19200bps, eBaudrate._115200bps, eBaudrate._230400bps, eBaudrate._460800bps.
Remark	None
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     eBaudrate rt = RFIDReader._ReaderConfig.GetReaderSerialPortParam(tcp);     Console.WriteLine(rt.ToString()); } else {     Console.WriteLine("Create connection failed!"); }</pre>



## 2.2.8 Set MAC address

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static Int32 SetReaderMacParam(String ConnID, String param)</b>
Parameter	// ConnID: connection identifier // param: MAC address format"00-00-00-00-00-00"
Return	0: successful; others: failed
Remark	None
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.SetReaderMacParam(tcp,"6E-7A-1C-AA-FF-0B");     if (rt == 0) Console.WriteLine("SET OK");     else Console.WriteLine("SET FAILED"); } </pre>

## 2.2.9 Get MAC address

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static String GetReaderMacParam(String ConnID)</b>
Parameter	// ConnID: connection identifier
Return	MAC address
Remark	MAC address format"00-00-00-00-00-00"
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     string rt = RFIDReader._ReaderConfig.GetReaderMacParam(tcp);     Console.WriteLine(rt); } else {     Console.WriteLine("Create connection failed!"); } </pre>

## 2.2.10 Set RS485 Address

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static Int32 SetReader485(String ConnID, String param)</b>
Parameter	// ConnID: connection identifier // param: 0~255
Return	0: successful; others: failed

Remark	None
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.SetReader485(tcp,"2");     if (rt == 0) Console.WriteLine("SET OK");     else Console.WriteLine("SET FAILED"); }</pre>

## 2.2.11 Get RS485 Address

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static String GetReader485(String ConnID)</b>
Parameter	// ConnID: connection identifier
Return	RS485 address
Remark	None
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.GetReader485(tcp);     Console.WriteLine(rt); }</pre>

## 2.2.12 Set DHCP Switch

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static String SetDHCP(String ConnID, Boolean param)</b>
Parameter	// ConnID: connection identifier // param: true:open, false:close
Return	0: successful; others: failed
Remark	None
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.SetDHCP(tcp,true);     if (rt == 0) Console.WriteLine("SET OK ");     else Console.WriteLine("SET FAILED "); }</pre>

## 2.2.13 Get DHCP Switch Status

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static Boolean GetDHCP(String ConnID)</b>
Parameter	// ConnID: connection identifier
Return	False:close, true:open
Remark	None
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.GetDHCP(tcp);     Console.WriteLine(rt); }</pre>

## 2.2.14 Set Server/client Mode

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static Int32 SetReaderServerOrClient(String ConnID, eWorkMode workMode,String ip,String port)</b>
Parameter	// ConnID: connection identifier // workMode: eWorkMode.Server, eWorkMode.Client ip: such as "192.168.1.75" port: such as "9090"
Return	0: successful; others: failed
Remark	When reader at the server mode, Parameter ip is invalid, can input any character string
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.SetReaderServerOrClient(tcp,eWorkMode.Server,"","9090");     if (rt == 0) Console.WriteLine("SET OK ");     else Console.WriteLine("SET FAILED "); }</pre>

## 2.2.15 Get Server/client Mode

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static String GetReaderServerOrClient(String ConnID)</b>
Parameter	// ConnID: connection identifier
Return	Server  "server port" or Client   "client port IP"   "client port"
Remark	None

Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.GetReaderServerOrClient(tcp);     Console.WriteLine(rt); }</pre>
--------------	--

## 2.2.16 Get Device Information

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	static <b>String</b> GetReaderInformation( <b>String</b> ConnID)
Parameter	// ConnID: connection identifier
Return	Application version   reader name   reader power on time
Remark	Reader power on time unit is "second"
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.GetReaderInformation(tcp);     Console.WriteLine(rt); }</pre>

## 2.2.17 Get Baseband version

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	static <b>String</b> GetReaderBaseBandSoftVersion( <b>String</b> ConnID)
Parameter	// ConnID: connection identifier
Return	Baseband version
Remark	None
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.GetReaderBaseBandSoftVersion(tcp);     Console.WriteLine(rt); }</pre>

## 2.2.18 Get Antenna Standing Wave Ratio

Namespace	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
Function	static <b>String</b> GetAntennaStandingWaveRatio( <b>String</b> ConnID, <b>eAntennaNo</b> antNo)
Parameter	// ConnID: connection identification, antNo: Antenna number enumeration

<b>Return</b>	Forward power detection   backward power detection
<b>Remark</b>	If the difference is greater than 25, the antenna is connected, otherwise, the antenna is not connected
<b>Example code</b>	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._RFIDConfig.GetAntennaStandingWaveRatio(tcp, 1);     Console.WriteLine(rt); }</pre>

## 2.2.19 Get Tag Data Output Format

<b>Namespace</b>	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
<b>Function</b>	<b>static String GetDataOutPutFormat (String ConnID)</b>
<b>Parameter</b>	// ConnID: Connection identifier
<b>Return</b>	Switch Data format  Data content  STX  ETX
<b>Remark</b>	None
<b>Example code</b>	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.GetDataOutPutFormat(tcp);     // Switch Data format  Data content  STX  ETX     Console.WriteLine(rt); }</pre>

## 2.2.20 Set Tag Data Output Format

<b>Namespace</b>	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
<b>Function</b>	<b>static String SetDataOutPutFormat (string connID,string switch,string outPutFormat,string outDataType,string startData,string endData)</b>
<b>Parameter</b>	// ConnID: Connection identifier // switch: 0-close, 1-open, 2-UDP output // outPutFormat: 0-hex, 1-ASCII //outDataType: 0-EPC, 1-TID //startData: Start of Text //endData: End of Text
<b>Return</b>	0: successful; others: failed

Remark	<p>Output format configuration:</p> <div> <div>Switch: <input type="text" value="Close"/></div> <div>Start of Text: <input type="text" value="40"/></div> <div>Data format: <input type="text" value="ASCII"/></div> <div>STX length: <input type="text" value="1"/></div> <div>Data content: <input type="text" value="EPC"/></div> <div>End of Text: <input type="text" value="24"/></div> <div>ETX length: <input type="text" value="1"/></div> <div> <input type="button" value="Get"/> <input type="button" value="Set"/> </div> </div>
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     string rt =     RFIDReader._ReaderConfig.SetDataOutPutFormat(tcp,"1","1","0","E200","7206");     // Switch Data format  Data content  STX  ETX     Console.WriteLine(rt); }</pre>

## 2.2.21 Restart

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static String ReSetReader(String ConnID)</b>
Parameter	// ConnID: Connection identifier
Return	None
Remark	None
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     RFIDReader._ReaderConfig.ReSetReader(tcp);     Console.WriteLine("Restarting"); }</pre>

## 2.2.22 Get Reader Temperature

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static String GetReaderTemperature (String ConnID)</b>
Parameter	// ConnID: Connection identifier
Return	Reader temperature

Remark	None
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     string rt = RFIDReader._ReaderConfig.GetReaderTemperature (tcp);     Console.WriteLine("当前读写器温度: " + rt); }</pre>

## 2.2.23 Set Custom Code

Namespace	<b>RFIDReaderAPI.RFIDReader._ ReaderConfig</b>
Function	<b>static eAntennaNo SetCustomCode (String ConnID:)</b>
Parameter	// ConnID: Connection identifier // param: The string to be set
Return	Result
Remark	None
Example code	<pre>string tcp = "192.168.1.116:9090"; string param = "write Code"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.SetCustomCode(tcp,param);     Console.WriteLine(rt); }</pre>

## 2.2.24 Get Custom Code

Namespace	<b>RFIDReaderAPI.RFIDReader._ ReaderConfig</b>
Function	<b>static eAntennaNo GetCustomCode (String ConnID)</b>
Parameter	// ConnID: Connection identifier
Return	Custom code
Remark	None
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.GetCustomCode(tcp);     Console.WriteLine(rt); }</pre>

## 2.2.25 Set Buzzer Control

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static String SetBuzzerControl(String ConnID, bool isOpen, bool isOftenRing)</b>
Parameter	// ConnID: Connection identifier // isOpen: Whether to turn on the buzzer // isOftenRing: Set whether to keep beeping, false -- Ring once, true -- Keep ringing
Return	Result
Remark	None
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.SetBuzzerControl(tcp, true, true);     Console.WriteLine(rt); } </pre>

## 2.2.26 Set Buzzer Switch

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static String SetBuzzerSwitch (String ConnID,String param)</b>
Parameter	// ConnID: Connection identifier // param: 0 -- set the buzzer sounds when the tag is read, 1-- set the buzzer does not sound when the tag is read
Return	Result
Remark	None
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.SetBuzzerSwitch(tcp, "1");     Console.WriteLine(rt); } </pre>

## 2.2.27 Get Reader Serial Number

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static String GetSN(String ConnID)</b>
Parameter	// ConnID: Connection identifier
Return	Reader Serial Number
Remark	None



Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.GetSN(tcp);     Console.WriteLine(rt); } </pre>
--------------	--

## 2.3 RFID configuration

### 2.3.1 Reset Factory

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static Int32 SetReaderRestoreFactory(String ConnID)</b>
Parameter	// ConnID: connection identifier
Return	0: successful, other: failed
Remark	Restore all setting back to factory status. Please use this interface with caution (MAC address and reader time will not change)
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.SetReaderRestoreFactory(tcp);     if (rt == 0) Console.WriteLine("Restore factory ok");     else Console.WriteLine("Restore factory failed"); } </pre>

### 2.3.2 Set Baseband Parameters

Namespace	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
Function	<b>static Int32 SetEPCBaseBandParam(String ConnID,Int32 basebandMode,Int32 qValue,Int32 session,Int32 searchType)</b>
Parameter	// ConnID: connection identifier // basebandMode: EPC Base band speed(0~255, 255 is AUTO) (0-Tari=25us, FM0, LF=40KHz) (1-TTari=25us, Miller4, LF=250KHz) (2-Tari=25us, Miller4, LF=300KHz) (3-Tari=6.25us, FM0, LF=400KHz) (255-Auto) // qValue: 0~15, reader's initial Q value. // session: 0~3 // searchType: inventory Parameter (0 only use Flag A to inventory, 1 only use Flag B to inventory, 2 use both Flag A and Flag B in turn to inventory).
Return	0: successful; others: failed
Remark	

Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._RFIDConfig.SetEPCBaseBandParam(tcp,1,0,0,2);     if (rt == 0) Console.WriteLine("SET OK ");     else Console.WriteLine("SET FAILED "); }</pre>
--------------	---

### 2.3.3 Get Baseband Parameters

Namespace	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
Function	<b>static String GetEPCBaseBandParam(String ConnID)</b>
Parameter	// ConnID: connection identifier
Return	<b>basebandMode qValue session searchType</b>
Remark	// basebandMode: EPC Base band speed(0~255, 255 is AUTO) (0-Tari=25us, FM0, LHF=40KHz) (1-TTari=25us, Miller4, LHF=250KHz) (2-Tari=25us, Miller4, LHF=300KHz) (3-Tari=6.25us, FM0, LHF=400KHz) (255-Auto) // qValue: 0~15, reader's initial Q value. // session: 0~3 // searchType: inventory Parameter (0 only use Flag A to inventory, 1 only use Flag B to inventory, 2 use both Flag A and Flag B in turn to inventory).
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._RFIDConfig.GetEPCBaseBandParam(tcp);     Console.WriteLine(rt); }</pre>

### 2.3.4 Set Antenna Port Output Power

Namespace	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
Function	<b>static Int32 SetANTPowerParam(String ConnID, Dictionary&lt;Int32, Int32&gt; dicPower)</b>
Parameter	// ConnID: connection identifier //dicPower: antenna number and power level key-value pair e.g.: SetANTPowerParam("192.168.1.116:9090",new Dictionary<Int32, Int32>(){1,30},{2,30}); This method will set Ant1 and Ant2 's power to 30
Return	0: successful; others: failed
Remark	Set each antenna's output power
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {</pre>

```

int rt = RFIDReader._RFIDConfig.SetANTPowerParam(tcp, new Dictionary<int, int>()
{ { 1, 15 } });
if (rt == 0) Console.WriteLine("SET OK ");
else Console.WriteLine("SET FAILED ");
}

```

### 2.3.5 Get Antenna Port Output Power

Namespace	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
Function	static <b>Dictionary&lt;Int32, Int32&gt;</b> GetANTPowerParam( <b>String</b> ConnID)
Parameter	// ConnID: connection identifier
Return	Can refer to the Dictionary<Int32, Int32> of 2.3.4 antenna power setting
Remark	
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._RFIDConfig.GetANTPowerParam(tcp);     Console.WriteLine(rt); } </pre>

### 2.3.6 Set Tag Data Upload Parameters

Namespace	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
Function	static <b>Int32</b> SetTagUpdateParam( <b>String</b> ConnID, <b>Int32</b> repeatTimeFilter, <b>Int32</b> RSSIFilter)
Parameter	// ConnID: connection identifier // repeatTimeFilter: repeat tag uploading filtration time // RSSIFilter: RSSI filter
Return	0: successful, others: failed
Remark	repeatTimeFilter value range: 0 ~ 65535. RSSIFilter value range: 0 ~ 255
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._RFIDConfig.SetTagUpdateParam(tcp, 10, 0);     if (rt == 0) Console.WriteLine("SET OK");     else Console.WriteLine("SET FAILED"); } </pre>

### 2.3.7 Get Tag Data Upload Parameters

Namespace	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
Function	static <b>String</b> GetTagUpdateParam( <b>String</b> ConnID)
Parameter	// ConnID: connection identifier

<b>Return</b>	<a href="#">repeatTimeFilter</a>   <a href="#">RSSIFilter</a> .
<b>Remark</b>	// repeatTimeFilter: repeat 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 tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._RFIDConfig.GetTagUpdateParam(tcp);     Console.WriteLine(rt); } </pre>

## 2.3.8 Get Device Property

<b>Namespace</b>	<a href="#">RFIDReaderAPI.RFIDReader._RFIDConfig</a>
<b>Function</b>	static <a href="#">String</a> GetReaderProperty( <a href="#">String</a> ConnID)
<b>Parameter</b>	// ConnID: connection identifier
<b>Return</b>	Minimum output power   maximum output power   number of antennas   band list   list of RFID protocols.
<b>Remark</b>	Power unit is dB
<b>Example code</b>	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._RFIDConfig.GetReaderProperty(tcp);     Console.WriteLine(rt); } </pre>

## 2.3.9 Set RF Frequency Range

<b>Namespace</b>	<a href="#">RFIDReaderAPI.RFIDReader._RFIDConfig</a>
<b>Function</b>	static <a href="#">Int32</a> SetReaderRF( <a href="#">String</a> ConnID, <a href="#">eRF_Range</a> eRF_Range)
<b>Parameter</b>	// ConnID: connection identifier // eRF_Range : <div style="margin-left: 40px;"> 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 </div>

	eRF_Range.RUS_866_to_867MHz
<b>Return</b>	0: successful; others: failed
<b>Remark</b>	<b>Note: the power of all antennas will be reset to 30 after the RF frequency is modified</b>
<b>Example code</b>	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._RFIDConfig.SetReaderRF(tcp,eRF_Range.ETSI_866_to_868MHz);     if (rt == 0) Console.WriteLine("SET OK");     else Console.WriteLine("SET FAILED"); }</pre>

### 2.3.10 Get RF Frequency Range

<b>Namespace</b>	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
<b>Function</b>	<b>static String GetReaderRF(String ConnID)</b>
<b>Parameter</b>	// ConnID: connection identifier
<b>Return</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>Remark</b>	None
<b>Example code</b>	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._RFIDConfig.GetReaderRF(tcp);     Console.WriteLine(rt); }</pre>

### 2.3.11 Set RF Working Frequency Points

<b>Namespace</b>	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
<b>Function</b>	<b>static Int32 SetReaderWorkFrequency(String ConnID, eWF_Mode wfMode, List&lt;eGB_840_to_845MHz&gt; ListGB_840_to_845MHz)</b>
<b>Parameter</b>	// ConnID: connection identifier //wfMode: eWF_Mode.Specified, eWF_Mode.Auto. //eGB_840_to_845MHz: eGB_840_to_845MHz._840_625f,

	eGB_840_to_845MHz._840_875f, eGB_840_to_845MHz._841_125f, eGB_840_to_845MHz._841_375f, eGB_840_to_845MHz._841_625f, eGB_840_to_845MHz._841_875f, eGB_840_to_845MHz._842_125f, eGB_840_to_845MHz._842_375f, eGB_840_to_845MHz._842_625f, eGB_840_to_845MHz._842_875f, eGB_840_to_845MHz._843_125f, eGB_840_to_845MHz._843_375f, eGB_840_to_845MHz._843_625f, eGB_840_to_845MHz._843_875f, eGB_840_to_845MHz._844_125f, eGB_840_to_845MHz._844_375f.
<b>Return</b>	0: successful; others: failed
<b>Remark</b>	<p><b>This function has many overload, other frequency setting only need change function eGB_840_to_845MHz to</b></p> <p><b>GB_920_to_925MHz,</b>  <b>GB_840_to_845MHz,</b>  <b>GB_920_to_925MHz_and_GB_840_to_845MHz,</b>  <b>FCC_902_to_928MHz,</b>  <b>ETSI_866_to_868MHz,</b>  <b>JP_916_to_921MHz,</b>  <b>TW_922_to_927MHz,</b>  <b>ID_923_to_925MHz,</b>  <b>RUS_866_to_867MHz</b></p> <p><b>then you could set related frequency points</b></p>
<b>Example code</b>	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     //After setting Auto, the List no need to be passed, but if it is not Auto, the frequency     point enumeration set needs to be instantiated     int rt = RFIDReader._RFIDConfig.SetReaderWorkFrequency(tcp,eWF_Mode.Auto);     if (rt == 0) Console.WriteLine("SET OK");     else Console.WriteLine("SET FAILED"); }</pre>

## 2.3.12 Get RF Working Frequency Points

<b>Namespace</b>	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
<b>Function</b>	<b>static String GetReaderWorkFrequency(String ConnID)</b>
<b>Parameter</b>	// ConnID: connection identifier
<b>Return</b>	<b>Mode   Frequency range   Frequency points</b> e.g.: Auto GB_920_to_925MHz 920.625,920.875

Remark	None
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._RFIDConfig.GetReaderWorkFrequency(tcp);     Console.WriteLine(rt); }</pre>

### 2.3.13 Set Device Auto-sleep Mode

Namespace	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
Function	static <b>Int32</b> SetReaderAutoSleepParam( <b>String</b> ConnID, <b>bool</b> Switch, <b>String</b> time)
Parameter	// ConnID: connection identifier, Switch: on / off, time: sleep time
Return	0: successful; others: failed
Remark	Switch: true-open, false-close Sleep time unit: <b>10ms</b>
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._RFIDConfig.SetReaderAutoSleepParam(tcp, true, "100");     if (rt == 0) Console.WriteLine("SET OK");     else Console.WriteLine("SET FAILED"); }</pre>

### 2.3.14 Get Device Auto-sleep Mode

Namespace	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
Function	static <b>String</b> GetReaderAutoSleepParam( <b>String</b> ConnID)
Parameter	// ConnID: connection identifier
Return	Close or Open  "sleep time"
Remark	Unit is <b>10ms</b>
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._RFIDConfig.GetReaderAutoSleepParam(tcp);     Console.WriteLine(rt); }</pre>

### 2.3.15 Set Reader Network Self-check

Namespace	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
Function	static <b>Int32</b> SetReaderSelfCheck( <b>String</b> ConnID, <b>bool</b> Switch, <b>String</b> ip)
Parameter	// ConnID: connection identifier, Switch: on / off, ip: ip address
Return	0: successful; others: failed
Remark	Switch: true-open, false-close
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.SetReaderSelfCheck(tcp, true, " IP address that need to be self-checked");     if (rt == 0) Console.WriteLine("SET OK ");     else Console.WriteLine("SET FAILED "); } </pre>

### 2.3.16 Get Reader Network Self-check Status

Namespace	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
Function	static <b>String</b> GetReaderSelfCheck( <b>String</b> ConnID)
Parameter	// ConnID: connection identifier
Return	Close or Open IP address"
Remark	
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.GetReaderSelfCheck(tcp);     Console.WriteLine(rt); } </pre>

### 2.3.17 Set Antenna Enable

Namespace	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
Function	static <b>Int32</b> SetReaderANT( <b>String</b> ConnID, <b>eAntennaNo</b> antNum)
Parameter	// ConnID: connection identifier, antNum: antenna number enumeration
Return	0: successful; others: failed
Remark	Enable ant1 and ant2, e.g.: <b>eAntennaNo._1 eAntennaNo._2</b>
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) { </pre>



	<pre> int rt = RFIDReader._RFIDConfig.SetReaderANT(tcp,eAntennaNo._1 eAntennaNo._2); if (rt == 0) Console.WriteLine("SET OK"); else Console.WriteLine("SET FAILED"); } </pre>
--	---

### 2.3.18 Get Antenna Enable Status

Namespace	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
Function	<b>static eAntennaNo GetReaderANT(String ConnID)</b>
Parameter	// ConnID: connection identifier
Return	Refer to 2.3.17 - eAntennaNo
Remark	None
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._RFIDConfig.GetReaderANT(tcp);     Console.WriteLine(rt); } </pre>

Namespace	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
Function	<b>static String GetReaderANT2(String ConnID)</b>
Parameter	// ConnID: connection identifier
Return	Antenna numbers already enabled, numbers apart by “,” such as 1,6,8
Remark	None
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._RFIDConfig.GetReaderANT2(tcp);     Console.WriteLine(rt); } </pre>

### 2.3.19 Get RFID Temperature

Namespace	<b>RFIDReaderAPI.RFIDReader._RFIDConfig</b>
Function	<b>static String GetRFIDTemperature (String ConnID)</b>
Parameter	// ConnID: connection identifier
Return	Temperature value
Remark	None
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); </pre>

```

if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log))
{
    var rt = RFIDReader._RFIDConfig.GetRFIDTemperature (tcp);
    Console.WriteLine(rt);
}

```

## 2.3.20 Baseband Expansion Settings

Namespace	RFIDReaderAPI.RFIDReader._RFIDConfig
Function	public Int32 SetEPCBaseExpandBandParam(string ConnID, Int32 tagFocus, Int32 fastId, Int32 maxQ, Int32 mixQ, Int32 tmult, Int32 antModel, Int32 rCount, Int32 residenceTime, Int32 antSleepTime)
Parameter	// ConnID: connection identifier // Impinj expansion parameters // tagFocus: 0 No, 1 yes // fastId: 0 No, 1 yes // DQN expansion parameters // maxQ: Maximum Q value // mixQ: Minimum Q value // tmult: tmult // AST expansion parameters // antModel: 0 Switch immediately without tags, 1 running out of residence time // rCount: retry times // residenceTime: antenna residence time // AST2 expansion parameters // antSleepTime: Antenna switching waiting time
Return	0: successful; others: failed
Remark	None
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._RFIDConfig.SetEPCBaseExpandBandParam (tcp,1,1,4,1,0,0,10,10,20);     Console.WriteLine(rt); } </pre>

## 2.4 GPIO operation

### 2.4.1 Set GPI Trigger Parameters

Namespace	RFIDReaderAPI.RFIDReader._ReaderConfig
Function	static Int32 SetReaderGPIParam(String ConnID, eGPI GPINum,

	<b>eTriggerStart</b> triggerStart, <b>eTriggerCode</b> triggerCode, <b>eTriggerStop</b> triggerStop, <b>String</b> DelayTime, <b>Boolean</b> isUpload)
<b>Parameter</b>	<p>// ConnID: connection identifier</p> <p>// GPINum: eGPI._1,eGPI._2,eGPI._3,eGPI._4;</p> <p>//triggerStart: eTriggerStart.OFF,eTriggerStart.Low_level,eTriggerStart.High_level, eTriggerStart.Rising_edge,eTriggerStart.Falling_edge,eTriggerStart.Any_edge;</p> <p>//triggerCode: triggerCode.Single_Antenna_read_EPC, triggerCode.Single_Antenna_read_EPC_and_TID, triggerCode.Double_Antenna_read_EPC, triggerCode.Double_Antenna_read_EPC_and_TID, triggerCode.Four_Antenna_read_EPC, triggerCode.Four_Antenna_read_EPC_and_TID;</p> <p>//triggerStop: eTriggerStop.OFF,eTriggerStop.Low_level,eTriggerStop.High_level, eTriggerStop.Rising_edge,eTriggerStop.Falling_edge,eTriggerStop.Any_edge, eTriggerStop.Delay;</p> <p>//DelayTime: unit is 10ms</p> <p>//isUpload: when triggerStop is eTriggerStop.OFF, whether to upload the GPI trigger status, true – upload, false – Don't upload</p> <p>e.g.: SetReaderGPIParam("192.168.1.116:9090",eGPI._2,eTriggerStart.Low_level, triggerCode.Double_Antenna_read_EPC_and_TID,eTriggerStop.Dealy,"100",false);</p> <p>This sample code is to set a low level trigger start on GPI2 port and execute trigger code when the GPI2 be triggered by low level, end after 1000ms delay.</p>
<b>Return</b>	0: successful; others: failed
<b>Remark</b>	<p>Stop delay time: unit is 10ms (when Trigger stop condition is "Delay stop" effective)</p> <p>Related operation performance refers to the RFIDReader demo software</p> <p>Detailed GPI trigger parameter callback, please refer to <a href="#">2.7 callback interface instruction</a> - GPIControlMsg();</p>
<b>Example code</b>	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.SetReaderGPIParam         (tcp, eGPI._1, eTriggerStart.High_level,         eTriggerCode.Double_Antenna_read_EPC, eTriggerStop.Low_level, "100", true);     if (rt == 0) Console.WriteLine("SET OK");     else Console.WriteLine("SET FAILED"); }</pre>

## 2.4.2 Get GPI Trigger Parameters

<b>Namespace</b>	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
<b>Function</b>	<b>static String</b> GetReaderGPIParam( <b>String</b> ConnID, <b>eGPI</b> GPINum)
<b>Parameter</b>	<p>// ConnID: connection identifier</p> <p>// GPONum: GPI._1,GPI._2,GPI._3,GPI._4.</p> <p>e.g. <b>GetReaderGPIParam</b>("192.168.1.116:9090",GPI._1);</p>
<b>Return</b>	"trigger GPI number Trigger start condition Trigger code Trigger stop condition stop delay time isUpload" e.g. "GPI1 Low level Double Antenna read EPC Delay 100 ON"
<b>Remark</b>	Stop delay time: unit is 10ms (when Trigger stop condition is "Delay stop" effective)

	<b>isUpload:</b> true – upload, false – Don't upload Related operation performance refers to the RFIDReader demo software
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.GetReaderGPIParam(tcp, eGPI._1);     Console.WriteLine(rt); }</pre>

## 2.4.3 Get GPI State

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static String GetReaderGPIState(String ConnID)</b>
Parameter	// ConnID: connection identifier e.g. <b>GetReaderGPIState</b> ("192.168.1.116:9090");
Return	e.g. "1,Low & 2,High", this return: 1# GPI port is in low level, 2# GPI port in high level
Remark	This method is to get GPI present level state, no business with trigger event Detailed GPI trigger parameter callback, please refer to <a href="#">2.7 callback interface instruction</a> - GPIControlMsg();
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.GetReaderGPIState(tcp);     Console.WriteLine(rt); }</pre>

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static Int32 SetReaderGPOState(String ConnID, Dictionary&lt;eGPO, eGPOState&gt; dicState)</b>
Parameter	// ConnID: connection identifier // dicState: GPO serial number and level related key-value pair e.g. <b>SetReaderGPOState</b> ("192.168.1.116:9090",new Dictionary<eGPO, eGPOState> {{1,0},{2,1}}); this method is to set #1 GPO port into low level (0), and set #2 GPO port into High level (1)
Return	0: successful; others: failed
Remark	None
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.SetReaderGPOState</pre>

```

        (tcp, new Dictionary<eGPO, eGPOState>() { {eGPO._1,eGPOState.High } });
        if (rt == 0) Console.WriteLine("SET OK");
        else Console.WriteLine("SET FAILED");
    }

```

## 2.4.4 Set GPO Level Status

## 2.4.5 Set Wiegand Parameters

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static Int32 SetReaderWG(String ConnID, eWiegandSwitch wiegandSwitch, eWiegandFormat wiegandFormat, eWiegandDetails param, int OutputAddress=0)</b>
Parameter	<p>// ConnID: connection identifier</p> <p>// eWiegandSwitch:</p> <p style="padding-left: 40px;">eWiegandSwitch.Close, eWiegandSwitch.Open.</p> <p>// eWiegandFormat:</p> <p style="padding-left: 40px;">eWiegandFormat.Wiegand26, eWiegandFormat.Wiegand34, eWiegandFormat.Wiegand66</p> <p>// eWiegandDetails:</p> <p style="padding-left: 40px;">eWiegandDetails.end_of_the_EPC_data, eWiegandDetails.end_of_the_TID_data.</p> <p>// OutputAddress: Output wiegand data corresponding to the starting address e.g. SetReaderWG("192.168.1.116:9090",eWiegandSwitch.Open, eWiegandFormat.Wiegand26,eWiegandDetails.end_of_the_TID_data); this method is to open Wiegand, data format is Wiegand 26, appointed transferred data is TID end data</p>
Return	0: successful; others: failed
Remark	None
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.SetReaderWG  (tcp ,eWiegandSwitch.Open,eWiegandFormat.Wiegand34,eWiegandDetails.end_of_the_ EPC_data,2);     if (rt == 0) Console.WriteLine("SET OK");     else Console.WriteLine("SET FAILED"); } </pre>

## 2.4.6 Get Wiegand Parameters

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	static <b>String</b> GetReaderWG( <b>String</b> ConnID)
Parameter	// ConnID: connection identifier
Return	// Return: <b>Wiegand switch Wiegand format Wiegand transferred data</b> e.g. Return "Open Wiegand66 end_of_the_EPC_data" this return means switch is open, communication format is Wiegand 66, appointed transferred data is EPC end data
Remark	None
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.GetReaderWG(tcp);     Console.WriteLine(rt); } </pre>

## 2.5 6C Tag Operation

### 2.5.1 Read Tag

Namespace	<b>RFIDReaderAPI.RFIDReader._Tag6C</b>
Function1	<p>static <b>int</b> GetEPC(<b>string</b> ConnID, <b>eAntennaNo</b> antNum, <b>eReadType</b> readType, <b>eMatchCode</b> matchType = <b>eMatchCode.None</b>, <b>string</b> matchCode = "", <b>int</b> matchWordStartIndex = -1, <b>string</b> accessPassword = "")</p> <p>// only read EPC  // ConnID: connection identifier  // antNum: Antenna number enumeration.  Appoint Antenna 1 and 2 working at same time; e.g.: <b>eAntennaNo._1 eAntennaNo._2</b>  // ReadType: read type enumeration, Single or Inventory (one-time or cyclically reading)</p> <p><b>Optional parameters :(all Tag6C functions will contain optional parameters, which will not be specified in the future)</b></p> <p>// matchType: matching area EPC TID  // matchCode: tag data need to be matched  // matchWordStartIndex: matching Data starting index  // accessPassword: Tag access password</p> <p><b>Note: matchType, matchCode and matchWordStartIndex need to be used together.</b></p>
Function2	<p><b>int</b> GetEPC_TID(<b>string</b> ConnID, <b>eAntennaNo</b> antNum, <b>eReadType</b> readType, <b>int</b> length = -1, <b>eMatchCode</b> matchType = <b>eMatchCode.None</b>, <b>string</b> matchCode = "", <b>int</b> matchWordStartIndex = -1, <b>string</b> accessPassword = "")</p> <p>// read EPC and TID  // length: optional parameter, reading length of TID</p>
Function3	<b>int</b> GetEPC_UserData( <b>string</b> ConnID, <b>eAntennaNo</b> antNum, <b>eReadType</b>

	<pre> readType, int readStart, int readLen, eMatchCode matchType = eMatchCode.None, string matchCode = "", int matchWordStartIndex = -1, string accessPassword = "")  // Read EPC and UserData // readStart  starting index of reading user area // readLen    reading length of user area (unit: Word) </pre>
Function4	<pre> int GetEPC_ReservedData(string ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, eMatchCode matchType = eMatchCode.None, string matchCode = "", int matchWordStartIndex = -1, string accessPassword = "") //read EPC and password area // readStart  starting index of reading password area // readLen    reading length of password area (unit: Word) </pre>
Function5	<pre> int GetEPC_TID_UserData(string ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, eMatchCode matchType = eMatchCode.None, string matchCode = "", int matchWordStartIndex = -1, string accessPassword = "") // Read EPC,TID and UserData // readStart  starting index of reading user area // readLen    reading length of user area (unit: Word) </pre>
Function6	<pre> int GetEPC_TID_ReservedData(string ConnID, eAntennaNo antNum, eReadType readType, int ReservedReadStart, int ReservedReadLen, eMatchCode matchType = eMatchCode.None, string matchCode = "", int matchWordStartIndex = -1, string accessPassword = "") // Read EPC,TID and password area // ReservedReadStart  starting index of reading password area // ReservedReadLen    reading length of password area (unit: Word) </pre>
Function7	<pre> int GetEPC_TID_UserData_ReservedData(string ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, int ReservedReadStart, int ReservedReadLen, eMatchCode matchType = eMatchCode.None, string matchCode = "", int matchWordStartIndex = -1, string accessPassword = "") // Read EPC,TID,UserData and password // readStart  starting index of reading user area // readLen    reading length of user area (unit: Word) // ReservedReadStart  starting index of reading password area // ReservedReadLen    reading length of password area (unit: Word) </pre>
Function8	<pre> int GetEPC_With_G2V2Authenticate(string ConnID, eAntennaNo antNum, eReadType readType, G2V2AuthenticateModel g2v2 , eMatchCode matchType = eMatchCode.None, string matchCode = "", int matchWordStartIndex = -1, string accessPassword = "")  // read EPC and G2V2 data <b>G2V2AuthenticateModel:</b> Set the C2G2 authentication parameters </pre>

	<pre>//_AuthMethod: TAM Authentication Method //_CustomData: TAM Custom Data //_KeyID: TAM KeyID //_Profile: memory parameter. 0,EPC area; 1, TID area; 2, user area //_Offset: Define the first address of the custom data block. //_BlockCount: Block Count //_ProtMode: Specifies the mode of operation that shall be used for the encipherment and/or protection of the custom data //G2V2Authenticate(): Converts the decimal parameters to a hexadecimal String, returning a String // accessPassword: Tag access password</pre>
Function9	<pre>int GetEPC_TID_UserData_With_G2V2Authenticate (string ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, G2V2AuthenticateModel g2v2, eMatchCode matchType = eMatchCode.None, string matchCode = "", int matchWordStartIndex = -1, string accessPassword = "") // read EPC,TID,UserData and G2V2 // readStart: starting index of reading user area // readLen: reading length of user area (unit: Word)</pre>
Function10	<pre>int GetEPC_TID_With_G2V2Authenticate(string ConnID, eAntennaNo antNum, eReadType readType, G2V2AuthenticateModel g2v2, eMatchCode matchType = eMatchCode.None, string matchCode = "", int matchWordStartIndex = -1, string accessPassword = "") // read TID, EPC and G2V2 data</pre>
Function11	<pre>int GetEPC_RFMicron_Temperature(string ConnID, eAntennaNo antNum, eReadType readType) // read EPC and RFMicron temperature tag</pre>
Function12	<pre>int GetEPC_EM4325_Temperature(string ConnID, eAntennaNo antNum, eReadType readType) //read EPC and EM4325 temperature tag</pre>
Function13	<pre>int GetEPC_EpcData(string ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, eMatchCode matchType = eMatchCode.None, string matchCode = "", int matchWordStartIndex = -1, string accessPassword = "") // read EPC and EpcData // readStart: starting index of reading EPCData area // readLen: data length of reading EPCData area (unit: Word)</pre>
Function14	<pre>int GetEPC_TID_EpcData(string ConnID, eAntennaNo antNum, eReadType readType, int readStart, int readLen, eMatchCode matchType = eMatchCode.None, string matchCode = "", int matchWordStartIndex = -1, string accessPassword = "") // read EPC,TID and EpcData // readStart: starting index of reading EPCData area // readLen: data length of reading EPCData area (unit: Word)</pre>
Function15	<pre>int GetEPC_TID_UserData_JoharTemp(string ConnID, eAntennaNo antNum, eReadType readType, eMatchCode matchType = eMatchCode.None, string</pre>



	<pre>matchCode = "", int matchWordStartIndex = -1, string accessPassword = "", bool IsJoharTemp = false) // This function will read the data of multiple data areas of Johar LTU32 temperature tag at the same time, including EPC, TID, UserData and JoharTemp temperature // RFIDReader._Tag6C.GetEPC_TID_UserData_JoharTemp(ConnId, eAntennaNo._1, eReadType.Inventory, IsJoharTemp: true);</pre>
<b>Parameter</b>	<a href="#">refers to each function instruction</a>
<b>Return</b>	True: successful; false: failed
<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> <li>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.</li> <li>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="#">2.7 callback interface instruction</a> - OutPutTagsOver();</li> </ol>
<b>Example code</b>	<pre>Program log = new Program(); if (RFIDReader.CreateSerialConn(SerialConnId, log)) {     int rt = RFIDReader._Tag6C.GetEPC(SerialConnId, eAntennaNo._1, eReadType.Inventory, matchType: eMatchCode.TID, matchCode: tid, matchWordStartIndex: 0, accessPassword: "00000000");     RFIDReader._Tag6C.Stop(SerialConnId);     rt = RFIDReader._Tag6C.GetEPC_TID_UserData_ReservedData(SerialConnId, eAntennaNo._1, eReadType.Inventory, 0,4,0, 4, matchType: eMatchCode.TID, matchCode: tid, matchWordStartIndex: 0, accessPassword: "00000000");     // If you use only partial optional parameters, it is recommended to fill in the parameters with the format parameter name: value     //eg: accessPassword: "00000000" If you need to fill in all the parameters, just fill in the order     RFIDReader._Tag6C.Stop(SerialConnId); } public void OutPutTags(RFIDReaderAPI.Models.Tag_Model tag) {     Console.WriteLine("EPC:" + tag.EPC + " - TID:" + tag.TID + " - UserData:" + tag.UserData + " - TagetData:" + tag.TagetData); } // read operation finished callback function public void OutPutTagsOver() {     Console.WriteLine("read operation finished"); }</pre>

## 2.5.2 Write Tag

### 2.5.2.1 Write EPC

Namespace	<b>RFIDReaderAPI.RFIDReader._Tag6C</b>
Function1	<p><b>string</b> WriteEPC(<b>string</b> ConnID, <b>eAntennaNo</b> antNum, <b>string</b> sWriteData, <b>string</b> dataStartIndex = "", <b>eMatchCode</b> matchType = eMatchCode.None, <b>string</b> matchCode = "", <b>int</b> matchWordStartIndex = -1, <b>string</b> accessPassword = "")</p> <p>// ConnID: connection identifier  // antNum: Antenna number enumeration.  Appoint Antenna 1 and 2 working at same time; e.g.:  eAntennaNo._1 eAntennaNo._2  // sWriteData: data need to be written(hexadecimal String)</p> <p><b>Optional parameters :</b></p> <p>// matchType: matching area EPC TID  // matchCode: tag data need to be matched  // matchWordStartIndex: matching Data starting index  // accessPassword: Tag access password  // dataStartIndex: starting index to be written</p> <p><b>Note: matchType, matchCode and matchWordStartIndex need to be used together.</b></p>
Parameter	Please refer to Function Remark.
Return	True: successful; others: failed
Remark	<p>1. Suggest using matched TID to write  2. For detailed Return, <a href="#">refers to Appendix A</a>  3. If the length of EPC data to be written not a multiple of 4, then 0 will be added up, e.g. if we want to write 201807 to EPC area, actually the data 20180700 will be written to tag.</p>
Example code	<pre> Program log = new Program(); if (RFIDReader.CreateSerialConn(SerialConnId, log)) {     string srt = RFIDReader._Tag6C.WriteEPC(SerialConnId, eAntennaNo._1, "88888888", matchType: eMatchCode.TID, matchCode: tid, matchWordStartIndex: 0, accessPassword: "11111111");     Console.WriteLine("Write result: " + srt); } </pre>

### 2.5.2.2 Write Userdata

Namespace	<b>RFIDReaderAPI.RFIDReader._Tag6C</b>
Function1	<p><b>static Int32</b> WriteUserData(<b>String</b> ConnID, <b>eAntennaNo</b> antNum, <b>String</b> sWriteData, <b>Int32</b> offset, <b>eMatchCode</b> matchType = eMatchCode.None, <b>string</b> matchCode = "", <b>int</b> matchWordStartIndex = -1, <b>string</b> accessPassword = "")</p> <p>// ConnID: connection identifier  // antNum: Antenna number enumeration.  Appoint Antenna 1 and 2 working at same time; e.g.:  eAntennaNo._1 eAntennaNo._2  // sWriteData: data to be written (Hexadecimal string)  // offset: the offset of user area, that is, the number of 0 before writing data</p> <p><b>Optional parameters:</b></p>

	// matchType: matching area EPC TID // matchCode: tag data need to be matched // matchWordStartIndex: matching Data starting index // accessPassword: Tag access password // dataStartIndex: starting index to be written <b>Note: matchType, matchCode and matchWordStartIndex need to be used together.</b>
Parameter	Please refer to FunctionRemark.
Return	True: successful; others: failed
Remark	1. Suggest using matched TID to write 2. For detailed Return <a href="#">_refers to Appendix A</a>
Example code	<pre> Program log = new Program(); if (RFIDReader.CreateSerialConn(SerialConnId, log)) {     string srt = RFIDReader._Tag6C.WriteUserData(SerialConnId, eAntennaNo._1, "88888888",0, matchType: eMatchCode.TID, matchCode: tid, matchWordStartIndex: 0, accessPassword: "11111111");     Console.WriteLine("Write result:" + srt); } 3. } </pre>

### 2.5.2.3 Write Password

Namespace	<b>RFIDReaderAPI.RFIDReader._Tag6C</b>
Function1	<b>static Int32 WriteAccessPassWord(String ConnID, eAntennaNo antNum, String sWriteData, eMatchCode matchType = eMatchCode.None, string matchCode = "", int matchWordStartIndex = -1, string accessPassword = "")</b> // sWriteData: password content (8 Hexadecimal string data) <b>Optional parameters:</b> // matchType: matching area EPC TID // matchCode: tag data need to be matched // matchWordStartIndex: matching Data starting index // accessPassword: Tag access password // dataStartIndex: starting index to be written <b>Note: matchType, matchCode and matchWordStartIndex need to be used together.</b>
Parameter	Please refer to FunctionRemark.
Return	True: successful; others: failed. <a href="#">_refers to Appendix A</a>
Remark	
Example code	<pre> Program log = new Program(); if (RFIDReader.CreateSerialConn(SerialConnId, log)) {     string srt = RFIDReader._Tag6C.WriteAccessPassWord(SerialConnId, eAntennaNo._1, "88888888", matchType: eMatchCode.TID, matchCode: tid, matchWordStartIndex: 0, accessPassword: "11111111");     Console.WriteLine("Write result:" + srt); } </pre>

## 2.5.3 Lock Tag

Namespace	<b>RFIDReaderAPI.RFIDReader._Tag6C</b>
Function1	<b>static Int32 Lock(String ConnID, eAntennaNo antNum, eLockArea lockArea, eLockType lockType)</b> // ConnID: connection identifier // antNum: antenna number // lockArea: lock area enumeration // lockType: lock type enumeration
Function2	<b>static Int32 Lock_MatchEPC(String ConnID, eAntennaNo antNum, eLockArea lockArea, eLockType lockType, String sMatchData, Int32 matchWordStartIndex)</b> // sMatchData: EPC data to be matched (Hexadecimal string) // matchWordStartIndex: match Data starting address, unit is word
Function3	<b>static Int32 Lock_MatchEPC(String ConnID, eAntennaNo antNum, eLockArea lockArea, eLockType lockType, String sMatchData, Int32 matchWordStartIndex, String accessPassword)</b> // accessPassword: Tag access password
Function4	<b>static Int32 Lock_MatchTID(String ConnID, eAntennaNo antNum, eLockArea lockArea, eLockType lockType, String sMatchData, Int32 matchWordStartIndex)</b> // sMatchData: TID data to be matched (Hexadecimal string) // matchWordStartIndex: match Data starting address, unit is word
Function5	<b>static Int32 Lock_MatchTID(String ConnID, eAntennaNo antNum, eLockArea lockArea, eLockType lockType, String sMatchData, Int32 matchWordStartIndex, String accessPassword)</b> // accessPassword: Tag access password
Parameter	// refer to above method Remark
Return	True: successful; false: failed <a href="#">refers to Appendix A</a>
Remark	
Example code	<pre> Program log = new Program(); if (RFIDReader.CreateSerialConn(SerialConnId, log)) {     int rt = RFIDReader._Tag6C.Lock_MatchEPC(SerialConnId, eAntennaNo._12   eAntennaNo._4, eLockArea.epc, eLockType.Unlock, "4321", 0, "00000001");     Console.WriteLine("Unlock result:" + rt);     rt = RFIDReader._Tag6C.Lock_MatchEPC(SerialConnId, eAntennaNo._4, eLockArea.epc, eLockType.Lock, "4321", 0, "00000001");     Console.WriteLine("Lock result:" + rt); } </pre>

## 2.5.4 Kill Tag (destroy)

Namespace	<b>RFIDReaderAPI.RFIDReader._Tag6C</b>
Function1	<b>static Int32 Destroy(String ConnID, eAntennaNo antNum, String destroyPassword)</b> // ConnID: connection identifier // antNum: antenna number // destroyPassword: kill password (Hexadecimal string)
Function2	<b>static Int32 Destroy_MatchEPC(String ConnID, eAntennaNo antNum, String destroyPassword, String sMatchData, Int32 matchWordStartIndex)</b> // sMatchData: EPC data to be matched (Hexadecimal string) // matchWordStartIndex: match Data starting address, unit is word
Function3	<b>static Int32 Destroy_MatchTID(String ConnID, eAntennaNo antNum, String</b>

	<b>destroyPassword, String sMatchData, Int32 matchWordStartIndex)</b> // sMatchData: TID data to be matched (Hexadecimal string)
<b>Parameter</b>	// refer to above method Remark
<b>Return</b>	True: successful; false: failed <a href="#">refers to Appendix A</a>
<b>Remark</b>	
<b>Example code</b>	<pre> Program log = new Program(); if (RFIDReader.CreateSerialConn(SerialConnId, log)) {     int rt = RFIDReader._Tag6C.Destroy_MatchEPC(SerialConnId, eAntennaNo._4   eAntennaNo._23, "00000001", "4321", 0);     Console.WriteLine("Excute result:" + rt);     Thread.Sleep(300); } </pre>

## 2.5.5 Get QT Parameters

<b>Namespace</b>	<b>RFIDReaderAPI.RFIDReader._Tag6C</b>
<b>Function1</b>	<b>static String GetQtTagOption (String ConnID, eAntennaNo antNum)</b> // ConnID: connection identifier // antNum: antenna number
<b>Function2</b>	<b>static String GetQtTagOption (String ConnID, eAntennaNo antNum, String accessPassword)</b> // accessPassword: Access Password (Hexadecimal string)
<b>Function3</b>	<b>static String GetQtTagOption_MatchEPC (String ConnID, eAntennaNo antNum, String sMatchData, Int32 matchWordStartIndex, String accessPassword)</b> // sMatchData: matched TID data(Hexadecimal string) // matchWordStartIndex: match Data starting address, unit is word
<b>Function4</b>	<b>static String GetQtTagOption_MatchTID (String ConnID, eAntennaNo antNum, String sMatchData, Int32 matchWordStartIndex, String accessPassword)</b> // sMatchData: matched TID data(Hexadecimal string) // matchWordStartIndex: match Data starting address, unit is word
<b>Parameter</b>	// refers to above method
<b>Return</b>	successfully: qt1 + " " + qt2 failed: ""
<b>Remark</b>	qt1 value: 1, when OPEN and SECURED state, lower response distance; 0, when OPEN and SECURED state, not lower response distance qt2 value: 1, tag into open mode; 0, tag into private mode
<b>Example code</b>	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._Tag6C.GetQtTagOption_MatchEPC (tcp,eAntennaNo._2,"EPCCode",0,"000000");     Console.WriteLine(rt); } </pre>

## 2.5.6 Set QT Parameters

Namespace	<b>RFIDReaderAPI.RFIDReader._Tag6C</b>
Function1	<b>static String SetQtTagOption (String ConnID, eAntennaNo antNum, bool IsNotReduceResponseDistance, bool IsPrivateMode)</b> // ConnID: connection identifier // antNum: antenna number // IsNotReduceResponseDistance: in open and secured state, if lower response distance or not // IsPrivateMode: tag into private mode or not
Function2	<b>static String SetQtTagOption (String ConnID, eAntennaNo antNum, bool IsNotReduceResponseDistance, bool IsPrivateMode, String accessPassword)</b> // accessPassword: Access Password (Hexadecimal string)
Function3	<b>static String SetQtTagOption_MatchEPC (String ConnID, eAntennaNo antNum, String sMatchData, Int32 matchWordStartIndex, bool IsNotReduceResponseDistance, bool IsPrivateMode, String accessPassword)</b> // sMatchData: EPC data need to be matched (Hexadecimal string) // matchWordStartIndex: match Data starting address, unit is word
Function4	<b>static String SetQtTagOption_MatchTID (String ConnID, eAntennaNo antNum, String sMatchData, Int32 matchWordStartIndex, bool IsNotReduceResponseDistance, bool IsPrivateMode, String accessPassword)</b> // sMatchData: TID data need to be matched (Hexadecimal string) // matchWordStartIndex: match Data starting address, unit is word
Parameter	// refers to above method
Return	Successful: successful information; Failed: "" or failed information
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     string rt = RFIDReader._Tag6C.SetQtTagOption_MatchEPC         (tcp,eAntennaNo._2,"EPCCode",0,true,true,"000000");     if (rt.StartsWith("0")) Console.WriteLine("SET OK");     else Console.WriteLine("SET FAILED"); }</pre>

## 2.5.6 G2V2 Untraceable Operation

**Specific firmware support is required**

Namespace	<b>RFIDReaderAPI.RFIDReader._Tag6C</b>
Function1	<b>static String SetG2V2UntraceableOption(String ConnID, eAntennaNo antNum, G2V2UntraceableModel g2v2)</b> // ConnID: connection identifier // antNum: antenna number // g2v2: G2V2 Untraceable parameters

Function2	<b>static String SetG2V2UntraceableOption(String ConnID, eAntennaNo antNum, G2V2UntraceableModel g2v2, String accessPassword)</b> // accessPassword: Tag access password
Function3	<b>static String SetG2V2UntraceableOption_MatchEPC (String ConnID, eAntennaNo antNum, G2V2UntraceableModel g2v2, String sMatchData, Int32 matchWordStartIndex, String accessPassword)</b> // sMatchData: the EPC data to be matched // matchWordStartIndex: match Data starting index
Function4	<b>static String SetG2V2UntraceableOption_MatchTID (String ConnID, eAntennaNo antNum, G2V2UntraceableModel g2v2, String sMatchData, Int32 matchWordStartIndex, String accessPassword)</b> // sMatchData: the TID data to be matched // matchWordStartIndex: match Data starting index
Parameter	<b>G2V2UntraceableModel</b> : G2V2 Untraceable parameters // <b>int _Untraceable_flag_bit</b> : Untraceable flag bit 0/1 // <b>int _EPC</b> : EPC show/hide flag bit. 0,show; 1, a Tag untraceably hides EPC memory above that set by its EPC length field // <b>int _EPC_length</b> : new EPC length. // <b>int _TID</b> : TID hide parameter. 0, show; 1, the Tag untraceably hides TID memory above 20h; 2, the Tag untraceably hides all TID memory. // <b>int _User</b> : USER hide parameter. 0,show; 1, hide // <b>int _Range</b> : Tag response range. 0,normal; 1, toggle temporarily; 2, reduced // <b>UntraceableParam()</b> : Converts the decimal parameters to a hexadecimal String, returning a String
Return	0 success 1 antenna port parameter error 2 select read area error 3 Untraceable parameters error 4 CRC checksum error 5 power is not enough 6 access password error 7 not supported command 8 other error 9 tag missed 10 send command failed
Remark	
Example code	<pre> Program log = new Program(); if (RFIDReader.CreateSerialConn(SerialConnId, log)) {     G2V2UntraceableModel G2V2 = new G2V2UntraceableModel(         0, // Untraceable_flag_bit         0, //EPC show/hide flag bit. 0,show; 1, hide         1, //new EPC length         0, //TID hide parameter. 0, show; 1, the Tag untraceably hides TID memory above         20h; 2, the Tag untraceably hides all TID memory.     ); } </pre>

```

1, // USER hide parameter. 0,show; 1, hide
1 //Tag response range. 0,normal; 1, toggle temporarily; 2, reduced
);
var res = RFIDReader._Tag6C.SetG2V2UntraceableOption(SerialConnId, eAntennaNo._1,
G2V2);
Console.WriteLine("G2V2 setting result:" + res);
RFIDReader.CloseConn(SerialConnId);
}

```

## 2.6 6B Tag Operation (omitted)

If your need to use ISO18000-6B RFID tag for your projects, then ask us to offer 6B development support

## 2.7 Callback Interface IAsynchronousMessage

```

// asynchronous callback Message interface
public interface IAsynchronousMessage
{
    void WriteDebugMsg(String msg);
    void WriteLog(String msg);
    void PortConneting(String connID);
    void PortClosing(String connID);
    void OutPutTags(Models.Tag_Model tag);
    void OutPutTagsOver();
    void GPIControlMsg(GPI_Model gpi_model);
}

```

Call back method	Remark
<b>WriteDebugMsg</b>	output debug message
<b>WriteLog</b>	output log message
<b>PortConnecting</b>	Reader at TCP server mode, client-side connection callback When the connection ID is obtained from the callback, the device can be read and written through the connection ID.
<b>PortClosing</b>	When the device is disconnected, the API calls back the connection ID, indicating that the device with the current connection ID is disconnected.
<b>OutPutTags</b>	The output tag information callback, whether it is single read, inventory (circular) read, or getting the tag data in the cache, is the same callback interface. <b>Note: all the tag data is asynchronous callback in API, do not deal with complex logic in the callback to make sure the inside of the API cache cleaning, when complex processing is required, put it into a thread.</b>
<b>OutPutTagsOver</b>	After the last tag is uploaded, a sync end signal is uploaded indicating the end of the current read tag action.
<b>GPIControlMsg</b>	When the GPI trigger parameter is turned on and there is a GPI trigger event, the function will call back the GPI port number where the current event is located, as well as the level status information.



## 2.8 Callback Data Tag\_Model Field Introductions

Field	Remark
<b>ReaderName</b>	Reader connection identifier, means which reader are reading data, e.g.: "192.168.1.116:9090"
<b>TagType</b>	Tag type, "6c","6b","gb" 3 types.
<b>EPC</b>	Tag EPC data, Hexadecimal string.
<b>PC</b>	Tag PC value
<b>ANT_NUM</b>	uploaded antenna number
<b>RSSI</b>	RSSI value
<b>TID</b>	Tag TID, Hexadecimal string.
<b>UserData</b>	Tag user area data, Hexadecimal string.
<b>TagetData</b>	Tag password area data, include access password and kill password, Hexadecimal string.
<b>Frequency</b>	Tag carrier frequency
<b>Phase</b>	Tag phase
<b>ReadTime</b>	Tag reading time
<b>G2V2Challenge</b>	G2V2 Challenge data
<b>G2V2Data</b>	G2V2data
<b>EPCData</b>	Tag EPCdata
<b>Bag</b>	The sub antenna number corresponding to the hub
<b>RSSI_dB</b>	RSSIvalue unit is dBm
<b>CatchCount</b>	Count in cache
<b>JoharTemp</b>	Temperature value of Johar Technology CTESIUS series LTU32 temperature sensor chip
<b>ReaderSN</b>	Reader SN, this field will only exist after calling the GetSN interface

## 2.9 Callback Data GPI\_Model Field Introductions

Filed	Remark
<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"

## 2.10 Breakpoint Continuous Transferring

### 2.10.1 Set

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static Int32 SetBreakPointUpload(String ConnID, bool Switch)</b>
Parameter	// ConnID: connection identifier // Switch: false:close, true:open e.g.: <b>SetBreakPointUpload</b> ("192.168.1.116:9090",false); this method is for close the breakpoint function (cache function)
Return	0: successful; others: failed
Remark	Start up this function, and the <b>ReadTime</b> field in the <b>Tag Model</b> will be effective
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.SetBreakPointUpload(tcp,true);     if (rt == 0) Console.WriteLine("SET OK");     else Console.WriteLine("SET FAILED"); }</pre>

### 2.10.2 Get

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static String GetBreakPointUpload(String ConnID)</b>
Parameter	// ConnID: connection identifier e.g.: <b>GetBreakPointUpload</b> ("192.168.1.116:9090");
Return	Open or Close
Remark	None
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.GetBreakPointUpload(tcp);     Console.WriteLine(rt); }</pre>

### 2.10.3 Retrieve Breakpoint Cache Data

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static String GetBreakPointCacheTag(String ConnID)</b>

<b>Parameter</b>	// ConnID: connection identifier
<b>Return</b>	Success: have cache, Null: no cache, 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 tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.GetBreakPointCacheTag(tcp);     Console.WriteLine(rt); }</pre>

## 2.10.4 Clear Breakpoint Cache Data

<b>Namespace</b>	RFIDReaderAPI.RFIDReader._ReaderConfig
<b>Function</b>	static Int32 ClearBreakPointCache(String ConnID)
<b>Parameter</b>	// ConnID: connection identifier
<b>Return</b>	0: successful; others: 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 tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.ClearBreakPointCache(tcp);     if (rt == 0) Console.WriteLine("Clear OK");     else Console.WriteLine("Clear failed"); }</pre>

## 2.11 WiFi Operation

### 2.11.1 Get WiFi Switch Status

<b>Namespace</b>	RFIDReaderAPI.RFIDReader._ReaderConfig
<b>Function</b>	static Int32 GetWiFiSwitchState(String ConnID)
<b>Parameter</b>	// ConnID: connection identifier
<b>Return</b>	0: open; 1: close
<b>Remark</b>	

Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.GetWiFiSwitchState(tcp);     if (rt == 0) Console.WriteLine("OK");     else Console.WriteLine("Failed"); }</pre>
--------------	--

## 2.11.2 Set Wi-Fi Switch Status

Namespace	RFIDReaderAPI.RFIDReader._ReaderConfig
Function	static <b>Int32</b> SetWiFiSwitchState( <b>String</b> ConnID, <b>bool</b> isOpen)
Parameter	// ConnID: connection identifier // isOpen: Set switch state. True: open; false: close
Return	0: successful; 1: failed
Remark	
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.SetWiFiSwitch(tcp,true);     if (rt == 0) Console.WriteLine("SET OK");     else Console.WriteLine("SET FAILED"); }</pre>

## 2.11.3 Get Wi-Fi Adaptor IP

Namespace	RFIDReaderAPI.RFIDReader._ReaderConfig
Function	static <b>String</b> GetReaderWifilP( <b>String</b> ConnID)
Parameter	// ConnID: connection identifier
Return	IP  Subnet mask   Gateway
Remark	
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.GetReaderWifilP(tcp);     Console.WriteLine(rt); }</pre>

## 2.11.4 Set Wi-Fi Adaptor IP

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	static <b>Int32</b> SetReaderWifiIP( <b>String</b> ConnID, <b>String</b> ip, <b>String</b> mask, <b>String</b> gateway)
Parameter	// ConnID: connection identifier // ip: IP address //mask: Subnet mask //gateway: network gateway
Return	0: successful; 1: failed
Remark	
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.SetReaderWifiIP(tcp,"192.168.1.114","255.255.255.0","192.1 68.1.1");     if (rt == 0) Console.WriteLine("SET OK");     else Console.WriteLine("SET FAILED"); } </pre>

## 2.11.5 Search Wi-Fi Hotspot

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	static <b>Int32</b> SearchWIFI( <b>String</b> ConnID)
Parameter	// ConnID: connection identifier
Return	0: successful; 1: failed
Remark	
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.SearchWIFI(tcp);     Console.WriteLine(rt); } </pre>

## 2.11.6 Save Wi-Fi Searching Result

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	static <b>Boolean</b> RequestWiFiDataToTxt( <b>String</b> ConnID, <b>String</b> filepath)
Parameter	// ConnID: connection identifier

	//filepath: saved txt file path
<b>Return</b>	Ture: successful; false: failed
<b>Remark</b>	
<b>Example code</b>	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.RequestWiFiDataToTxt(tcp,"D:\\Project\\a.txt");     Console.WriteLine(rt); } </pre>

## 2.11.7 Connect Wi-Fi Hotspot

<b>Namespace</b>	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
<b>Function</b>	<b>static Int32 ConnectWIFI(String ConnID,string name, string pwd, int authType, int encryption)</b>
<b>Parameter</b>	// ConnID: connection identifier // name: wifi name //pwd: wifi password //authType: Authentication type //encryption: Encryption Algorithm
<b>Return</b>	True: successful; false: failed
<b>Remark</b>	e.g.: 1、 when Wi-Fi hotspot need password : <pre> if (WifiAuth=="WPA") {     authType = 1; } else if (WifiAuth == "WPA2") {     authType = 2; } </pre> 2、 when Wi-Fi hotspot without password : <pre> int ret = CLReader._ReaderConfig.ConnectWIFI(ConnID, WifiName, "", 0, 2); </pre>
<b>Example code</b>	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     int rt = RFIDReader._ReaderConfig.ConnectWIFI(tcp, "iPhone", "12345678", 1,2);     if (rt == 0) Console.WriteLine("Connect successfully");     else Console.WriteLine("Connect failed"); } </pre>

```
}
```

## 2.11.8 Get Current Connected Wi-Fi Name

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	static <b>String</b> GetWiFiConnectInfo( <b>String</b> ConnID)
Parameter	// ConnID: connection identifier
Return	Present connected Wi-Fi name
Remark	

## 2.12 Whitelist Function

The process for integrating whitelist function.

A. All the whitelist data stored in the file userdata.db, we can view or edit the record of database file - userdata.db through the Sqlite DataBase browser, you can download DB Browser for SQLite 3.10 from <http://sqlitebrowser.org/>, and then we can get the whitelist information stored in reader indirectly through viewing the database file that had been synchronized to reader.

Develop the similar function as the program SQLite 3.10 to your own application

B. Using the whitelist function API to upload the whitelist database file to the reader.

### 2.12.1 Set Whitelist Parameters

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	static <b>Int32</b> SetWhitelistLabelAction ( <b>String</b> ConnID, <b>string</b> param)
Parameter	// ConnID: connection identifier // param: relay no.(1-4) closing time(unit is 100ms, 4 hex numbers)  function switch(0-turn off /1-turn on whitelist function/2-Open Buzzer)  data type (0 TIID/1 EPC)
Return	0: successful; others: failed
Remark	None
Example code	<pre>string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     string rt = RFIDReader._ReaderConfig.SetWhitelistLabelAction(tcp,"1 0001 0 1");     if (rt.StartsWith("0")) Console.WriteLine("SET OK");     else Console.WriteLine("SET FAILED"); }</pre>

## 2.12.2 Get Whitelist Parameters

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static Int32 GetWhitelistLabelAction (String ConnID)</b>
Parameter	// ConnID: connection identifier
Return	relay no.(1-4) closing time(unit is 100ms)  function switch(0 -turn off /1- turn on)  data type (0 TIID/1 EPC)
Remark	None
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     var rt = RFIDReader._ReaderConfig.GetWhitelistLabelAction(tcp);     Console.WriteLine(rt); } </pre>

## 2.12.3 Upload Whitelist Database File To Reader

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>static Int32 LoadWhiteList (String ConnID, uint dataIndex, byte[] param)</b>
Parameter	// ConnID: connection identifier // dataIndex: upgrade data serial number 0x00000000 start, 0xFFFFFFFF end // param : whitelist data
Return	0: successful; others: failed
Remark	
Example code	<pre> string datasource = Application.StartupPath + "\\userdata.db"; FileStream fs = null; try {     fs = new FileStream(datasource, FileMode.Open);     // byte[] fileBuffer = new byte[fs.Length];     // fs.Read(fileBuffer, 0, fileBuffer.Length);     if (!String.IsNullOrEmpty(datasource))     {         if (fs.Length &gt;= 1024 * 1024 * 100) { throw new Exception("Max File 100M! "); }         Int32 fileSize = (Int32)fs.Length;         byte[] buffer = new byte[256];         UInt32 blockCount = (UInt32)((fileSize % 256) == 0 ? fileSize / 256 : fileSize / 256 + 1);         Int32 maxProcess = (Int32)blockCount;         //SetMaxProcess();         Int32 copyIndex = 0;         String isOK = "";     } } </pre>



```

isOK = Program.PARAM_SET.LoadWhiteList(ConnID, 0, new byte[0]);
if (isOK.EndsWith("0"))
{
    for (UInt32 i = 0; i < blockCount; i++)
    {
        Int32 readCount = 0;
        if (copyIndex + 256 <= fileSize)
        {
            readCount = fs.Read(buffer, 0, 256);
        }
        else
        {
            byte[] lastBuffer = new byte[fileSize - copyIndex];
            readCount = fs.Read(lastBuffer, 0, fileSize - copyIndex);
            buffer = lastBuffer;
        }
        isOK = RFIDReaderAPI.RFIDReader._ReaderConfig.LoadWhiteList(ConnID,
i, buffer);

        if (String.IsNullOrEmpty(isOK)) { throw new Exception("Time Out! "); }
        else if (isOK.EndsWith("1")) { throw new Exception("return false! "); }
        copyIndex += 256;
        //ProcessPerformStep();
    }
}
else
{
    throw new Exception("upgrade error! ");
}
if (Program.PARAM_SET.LoadWhiteList(ConnID, UInt32.MaxValue, new
byte[0]).EndsWith("0"))
{
    if (DialogResult.OK == ShowQuestion(_RC.GetString("WhiteListMsg10")))
    {
        Program.PARAM_SET.ReSetReader(ConnID);
    }
}
else
{
    ShowMessage("CRC ERROR! ");
}
isWorking = false;
}
else
{

```

```

        isWorking = false;
        ShowMessage(_RC.GetString("WhiteListMsg11"));
    }
}
catch (Exception ex)
{
    ShowMessage(ex.Message);
    isWorking = false;
}
finally
{
    if (fs != null)
    {
        fs.Close();
    }
}

```

## 2.13 PSAM card operation

### 2.13.1 Reset PSAM card

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>byte[] PSAMReset(string connID, int index = 0)</b>
Parameter	// ConnID: connection identifier // index:PSAM card index
Return	Return ATR information or null.
Remark	The number of PSAM cards depends on different models, 0 means PSAM1, 1 means PSAM2
Example code	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     byte[] atr = RFIDReader._ReaderConfig.PSAMReset(tcp);     Console.WriteLine(atr); } </pre>

### 2.13.2 Send APDU command

Namespace	<b>RFIDReaderAPI.RFIDReader._ReaderConfig</b>
Function	<b>byte[] PSAMAPDU(string connID, int index, byte[] apdu)</b>
Function 2	<b>byte[] PSAMAPDU(string connID, byte[] apdu)</b>

<b>Parameter</b>	// ConnID: connection identifier // index:PSAM card index // apdu: The apdu command to be sent
<b>Return</b>	adpu response or null
<b>Remark</b>	1. The number of PSAM cards depends on different models, 0 means PSAM1, 1 means PSAM2 2. Function 2 uses PSAM1 by default
<b>Example code</b>	<pre> string tcp = "192.168.1.116:9090"; RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program(); if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log)) {     byte[] apdu = new byte[] {0x00, 0x84, 0x00, 0x00, 0x08}     byte[] rpdu = RFIDReader._ReaderConfig.PSAMAPDU(tcp, apdu);     Console.WriteLine(rt); } </pre>

## 2.14 Antenna Number Parameter Description

- Regarding the tag read, write, lock, kill operation of the antenna number parameter: **antNum**. The function is to specify whether an antenna or multiple antennas for a reader work.
- While specifying multiple antennas to work with antNum for their total value, for instance:  
Specify antenna 1+ antenna 2 to work: **antNum = eAntennaNo.\_1 | eAntennaNo.\_2**  
Specify antenna1+antenna2+antenna 3 to work: **antNum = eAntennaNo.\_1 | eAntennaNo.\_2 | eAntennaNo.\_3**  
Specify antenna 1+ antenna 2+ antenna 3+ antenna 4 to work: **antNum = eAntennaNo.\_1 | eAntennaNo.\_2 | eAntennaNo.\_3 | eAntennaNo.\_4**  
Specify antenna 1+ antenna 24 to work: **antNum = eAntennaNo.\_1 | eAntennaNo.\_24**

## 2.15 Callback Interface ISearchDevice Field Instructions

Field	Instruction
<b>DeviceInfo(Models.Device_Model model)</b>	Device information callback
<b>DebugMsg(string msg)</b>	Debug information callback

## 2.16 Callback Data Device\_Model Filed Instructions

Field	Instruction
<b>MAC</b>	Reader MAC address
<b>IP</b>	Reader IP address
<b>ServerPort</b>	Server port

<b>RemoteIP</b>	Host IP that the reader actively connects to
<b>RemotePort</b>	Host IP port
<b>WorkingMode</b>	TCP working mode of the reader
<b>ConnectMode</b>	Reader connection status
<b>DeviceType</b>	Reader type
<b>DHCP</b>	DHCP
<b>Mask</b>	Reader mask
<b>Gateway</b>	Reader gateway address

## 3.Programming Example

### C# Code: read 6C tag example

```

class Program : RFIDReaderAPI.Interface.IAsynchronousMessage
{
    static void Main(string[] args)
    {
        // RFIDReaderAPI.RFIDReader.CreateSerialConn("COM1:115200",new
Program()); // use serial port to connect RFID reader
        if (RFIDReaderAPI.RFIDReader.CreateTcpConn("192.168.1.116:9090", new
Program())) // use TCP to connect RFID reader
        {

            RFIDReaderAPI.RFIDReader._Tag6C.GetEPC("192.168.1.116:9090",eAntennaNo._1|eA
ntennaNo._2| eAntennaNo._3|eAntennaNo._4, eReadType.Inventory);
            // send reading EPC instruction to the reader using Ant1+ Ant2+Ant3+Ant4
at the same time
        }
        else
        {
        }
        Console. ReadKey();

        RFIDReaderAPI.RFIDReader._RFIDConfig.Stop("192.168.1.116:9090");//send stop
instruction
        RFIDReaderAPI.RFIDReader.CloseConn("192.168.1.116:9090"); // close
connection
    }
    #region interface implement
    // Tag CallBack
    public void OutPutTags(RFIDReaderAPI.Models.Tag_Model tag)
    {
        Console.WriteLine("EPC:" + tag.EPC + " - TID:" + tag.TID);
    }
    public void WriteDebugMsg(string msg)
    {

    }
    public void WriteLog(string msg)
    {

    }
    public void PortConneting(string connID)
    {

    }
}

```

```

public void PortClosing(string connID)
{

}
public void OutPutTagsOver()
{

}
public void GPIControlMsg(GpiModel gpiModel)
{

}
}
#endregion
}

```

## 4.FAQ and Solutions

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 notice 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> <li>3. Try serial port connection, inquiry IP correct by Demo</li> <li>4. Default IP and port: "192.168.1.116:9090"</li> </ol>

## 5.Appendix A: 6C Tag Operation Returned Error Code

Read tag error codes:

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

Write tag error codes:

Code	Remark
0	Write successfully
1	Antenna port parameter error
2	Choose parameter error
3	Write parameter error
4	CRC correct error
5	Power not enough
6	Data area overflow

7	Data area locked
8	Access password error
9	Other tag error
10	Tag lost
11	Instruction sent error

Lock tag error codes:

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

Kill tag error codes:

Code	Remark
0	Kill successfully
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 sent error