Hopeland

Focus on RFID core technology

Hopeland RFID READER - PC

Development Guide

.net

Editor: Paul

Shenzhen Hopeland Technologies Co., Ltd

V 3.31

Catalogue

1. Overview

1.1 Content Overview

For facilitating the user software development, we provide library running in the .Net platform. The library is written in C # language and encapsulated into a standard dynamic link library, the development environment for the .Net Framework 2.0.

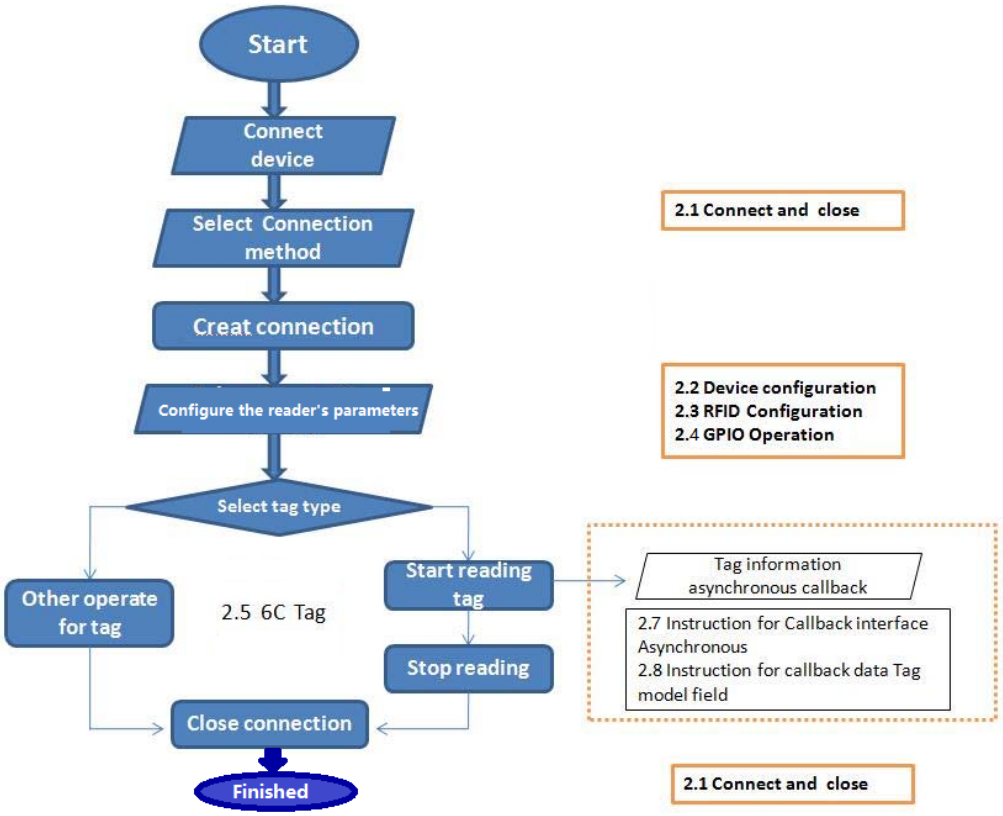
This guide introduces the corresponding technical indicators, application development notes and precautions, application interface function description and so on.

Statement:

1. For the operation of RFID management software (DEMO), please refer to the user manual of related products.

2. If this document still couldn't meet your development requirements, please kindly contact with manufacturer 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
Device configuration	All products
RFID configuration	All products
GPO operation	CL7206B/CL7206C series
6C tag operation	All products
6B tag operation	All products

1.4 Copyright Statement

2. API modules function description

2.1 connect & disconnect

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";

2.1.1 Create serial port connect

Namespace	**RFIDReaderAPI.RFIDReader**
Function	static bool CreateSerialConn(string seriaParam, IAsynchronousMessage log)
Parameter	seriaParam: serial port connection Parameter, eg:"COM1:115200" log: 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, "seriaParam" will be this connection channel's ID, which is different with other connection channel. 2. log: Data callback interface, please refer to 2.7callback interface introductions for more information.

Example code	FIDReaderAPI.Interface.IAsynchronousMessage log = new Program();
	if (RFIDReaderAPI.RFIDReader.CreateSerialConn(SerialConnId, log))
	{
	Console.WriteLine("Create connection successfully!");
	}
	else
	{
	Console.WriteLine("Create connection failed!");
	}

2.1.2 Create TCP connection

Namespace	**RFIDReaderAPI.RFIDReader**
Function	static bool CreatTcpConn(string tcpParam, IAsynchronousMessage log)
Parameter	tcpParam: TCP connection Parameter, eg:"192.168.1.116:9090" log: Data callback interface, all tags data will be called back from this interface.
Return	True: 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. log: Data callback interface, Please refer to 2.7callback interface introductions for more information

Example code	RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program();
	if (RFIDReaderAPI.RFIDReader.CreateTcpConn("192.168.1.116:9090", log))
	// TCP Connect
	{
	Console.WriteLine("Create connection successfully!");
	}

	else
	{
	Console.WriteLine("Create connection failed!");
	}

2.1.3 Create 485 connection

Namespace	**RFIDReaderAPI.RFIDReader**
Function	static bool Create485Conn(string _485Param, IAsynchronousMessage log)
Parameter	_485Param: RS485 connection Parameter, eg:"1:COM1:115200", "1" as 485 connection address. log: 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, "485Param" will be this connection channel's ID which is different with other connection channel. 2. log: Data callback interface, Please refer to 2.7callback interface introductions for more information.

Example code	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!");
	}

2.1.4 Create 485 connection (one serial com port creating multiple 485 connections)

Namespace	**RFIDReaderAPI.RFIDReader**
Function	static bool Create485sConn(string _485Param, IAsynchronousMessage log)
Parameter	_485Param: RS485 connection Parameter, eg:"1:COM1:115200", "1" as 485 connection address. log: 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, "485Param" will be this connection channel's ID which is different with other connection channel. 2. log: Data callback interface, Please refer to 2.7callback interface introductions 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	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))
	{
	Console.WriteLine("485 address 2 connection created successfully!");
	}
	else
	{
	Console.WriteLine("485 address 2 connection created failed!");
	}

2.1.5 Create USB connection

Namespace	**RFIDReaderAPI.RFIDReader**
Function	static bool CreateUsbConn(string usbParam, IntPtr Handle, IAsynchronousMessage log)
Parameter	usbParam: usb connect parameter。 Handle: window handle or service handle log: Data callback interface, all the tag data will be callback from the interface object
Return	True: successful; false: failed
Remark	1. parameter "usbParam", through same namespace static List<string> GetUsbHidDeviceList() to get connection parameter of USB device list 2. log data callback interface, refer to 2.7callback interface introductions for detailed information. 3. Default name is "UHF READER" after USB connected

Example code	RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program();
	//Gets the console handle
	IntPtr Handle = User32API.GetCurrentWindowHandle();
	if (RFIDReader.CreateUsbConn(usbConnID, Handle, log))
	{
	Console.WriteLine("Create connection successfully! ");
	}
	else
	{

	Console.WriteLine("Create connection failed!"); }

2.1.6 Check connection status

Namespace	**RFIDReaderAPI.RFIDReader**
Function	static bool CheckConnect(string connectID)
Parameter	connectID: connection ID, eg:"192.168.1.116:9090"
Return	True: successful; false: failed
Remark	"connectID" is the connection parameter when creates connection

Example code	RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program();
	if (RFIDReaderAPI.RFIDReader.CreateTCPConn("192.168.1.116:9090", log))
	{
	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!");
	}

2.1.7 Close single connection

Namespace	**RFIDReaderAPI.RFIDReader**
Function	static void CloseConn(string connectID)
Parameter	connectID: connection ID, eg:"192.168.1.116:9090"
Return	None
Remark	"connectID" is the connection parameter when creates connection

Example code	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!");
	}

2.1.8 Close all connections

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

Example code	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!");
	}

2.1.9 Open TCP server listening

Namespace	**RFIDReaderAPI.RFIDReader**

Function	static Boolean OpenTcpServer(string serverIP,string serverPort,IAsynchronousMessage log)
Parameter	serverIP: the IP been listening。 serverPort: the port been listening Log: all data callback interface after connected , details refer to: 2.7callback interface introductions
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: OpenTcpServer("192.168.1.75","9090",log); 3. The default connecting IP (client mode) is "192.168.1.1", port is "9090"

Example code	RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program();
	if (RFIDReader.OpenTcpServer("192.168.1.116", "9090", log))
	{
	Console.WriteLine("Start TCP server listening successfully!");
	}

2.1.10 Close TCP server listening

Namespace	RFIDReaderAPI.RFIDReader
Function	static void CloseTcpServer()
Parameter	None
Return	None
Remark	Close TCP server listening

Example code	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");
	}
	}

2.1.11 Get server listening status

Namespace	**RFIDReaderAPI.RFIDReader**
Function	static bool GetServerStartUp()
Parameter	None
Return	Server listening or not
Remark	True: listening，false: not in listening

Example code	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
	{
	Console.WriteLine("listening closed");
	}
	}

2.1.12 API language type set

Namespace	**RFIDReaderAPI.RFIDReader**
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	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!");
	}

2.1.13 API language type query

Namespace	**RFIDReaderAPI.RFIDReader**
Function	static Int GetAPILanguageType()
Parameter	None
Return	"0" is English, "1" is Chinese
Remark	The default language is English

Example code	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!");
	}

2.2 Device configuration

2.2.1 IP configuration

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static Int32 Set ReaderNetworkPortParam(String ConnID, String iP, String mask, String gateway)

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"

Return	0: successful; others: failed
Remark	1. This method will close all created connection.

Example code	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!");
	}

2.2.2 Query IP configuration

Namespace	**RFIDReaderAPI.RFIDReader**._ReaderConfig
Function	static String GetReaderNetworkPortParam(String ConnID)
Parameter	// ConnID: connection identifier
Return	IP add | Subnet mask | Gateway, e.g. :"192.168.1.116|255.255.255.0|192.168.1.1"
Remark	

Example code	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!");
	}

2.2.3 Stop instruction

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

2.2.4 Set device time

Namespace	**RFIDReaderAPI.RFIDReader**._ReaderConfig
Function	static Int32 SetReaderUTC(String ConnID, String 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	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!");
	}

2.2.5 Query device time

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static String GetReaderUTC(String 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	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!");
	}

2.2.6 Set serial port parameter

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static Int32 SetReaderSerialPortParam(String ConnID, eBaudrate 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	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!");
	}

2.2.7 Get serial port parameter

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static eBaudrate GetReaderSerialPortParam(String ConnID)
Parameter	// ConnID: connection identifier
Return	eBaudrate._9600bps, eBaudrate._19200bps, eBaudrate._115200bps, eBaudrate._230400bps, eBaudrate._460800bps.
Remark	None

Example code	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!");
	}

2.2.8 MAC address setting

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static Int32 SetReaderMacParam(String ConnID, String param)

Parameter	// ConnID: connection identifier // param: MAC address format"00-00-00-00-00-00"
Return	0: successful; others: failed
Remark	None

Example code	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");
	}

2.2.9 Get MAC address

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static String GetReaderMacParam(String ConnID)
Parameter	// ConnID: connection identifier
Return	MAC address
Remark	MAC address format"00-00-00-00-00-00"

Example code	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!");
	}

2.2.10 RS485 address setting

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static Int32 SetReader485(String ConnID, String param)
Parameter	// ConnID: connection identifier // param: 0~255
Return	0: successful; others: failed
Remark	None
Example code	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");
	}

2.2.11 RS485 address Query

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

Example code	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);
	}

2.2.12 DHCP configuration

Namespace	**RFIDReaderAPI.RFIDReader**._ReaderConfig
Function	static String SetDHCP(String ConnID, Boolean param)
Parameter	// ConnID: connection identifier // param： true:open, false:close
Return	0: successful; others: failed
Remark	None

Example code	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 ");
	}

2.2.13 Query DHCP

Namespace	**RFIDReaderAPI.RFIDReader**._ReaderConfig

Function	static Boolean GetDHCP(String ConnID)
Parameter	// ConnID: connection identifier
Return	False:close，true:open
Remark	None

Example code	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);
	}

2.2.14 Server/client mode configuration

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static Int32 SetReaderServerOrClient(String ConnID, eWorkMode workMode,String ip,String port)

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	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 ");
	}

2.2.15 Query Server/client mode

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static String GetReaderServerOrClient(String ConnID)
Parameter	// ConnID: connection identifier
Return	Server| "server port"　or　Client | "client port IP" | "client port"
Remark	None
Example code	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);
	}

2.2.16 Query device information

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static String GetReaderInformation(String ConnID)
Parameter	// ConnID: connection identifier
Return	Application version | reader name | reader power on time
Remark	Reader power on time unit is "second"

Example code	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);
	}

2.2.17 Query Baseband version

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static String GetReaderBaseBandSoftVersion(String ConnID)
Parameter	// ConnID: connection identifier
Return	Baseband version
Remark	None

Example code	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);
	}

2.2.18 Query antenna standing wave ratio

Namespace	**RFIDReaderAPI.RFIDReader._RFIDConfig**
Function	static String GetAntennaStandingWaveRatio(String ConnID, eAntennaNo antNo)
Parameter	// ConnID: connection identification, antNo: Antenna number enumeration
Return	Forward power detection | backward power detection

Remark	If the difference is greater than 25, the antenna is connected, otherwise, the antenna is not connected

Example code	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);
	}

2.2.19 Query tag data output format

Namespace	**RFIDReaderAPI.RFIDReader**._ReaderConfig
Function	**static String GetDataOutPutFormat (String ConnID)**
Parameter	// ConnID: Connection identifier
Return	Switch|Data format| Data content| STX| ETX
Remark	None

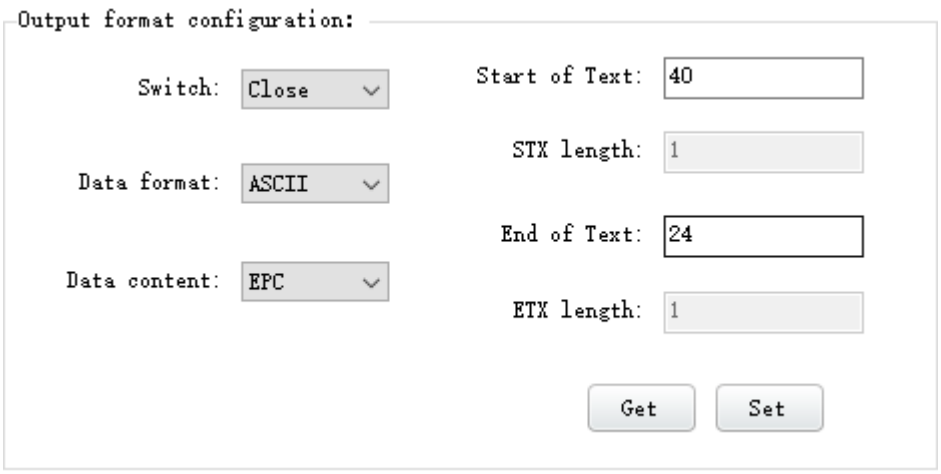
Example code	string tcp = "192.168.1.116:9090";
	RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program();
	if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log))
	{
	var rt = RFIDReader._ReaderConfig.GetDataOutPutFormat(tcp);
	// Switch|Data format| Data content| STX| ETX
	Console.WriteLine(rt);
	}

2.2.20 Set tag data output format

Namespace	**RFIDReaderAPI.RFIDReader**._ReaderConfig
Function	**static String SetDataOutPutFormat (string connID,string switch,string outPutFormat,string outDataType,string startData,string endData)**

Parameter	// 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

Return	0: successful; others: failed

Remark	

Example code	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);
	}

2.2.21 Restart

Namespace	**RFIDReaderAPI.RFIDReader**._ReaderConfig
Function	**static String ReSetReader(String ConnID)**
Parameter	// ConnID: Connection identifier
Return	None
Remark	None

Example code	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");
	}

2.3 RFID configuration

2.3.1 Factory reset

Namespace	**RFIDReaderAPI.RFIDReader**._ReaderConfig

Function	static Int32 SetReaderRestoreFactory(String ConnID)
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	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");
	}

2.3.2 Base band parameter setting

Namespace	**RFIDReaderAPI.RFIDReader._RFIDConfig**
Function	static Int32 SetEPCBaseBandParam(String ConnID,Int32 basebandMode,Int32 qValue,Int32 session,Int32 searchType)

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	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 ");
	}

2.3.3 Query Base band parameter setting

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

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	`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);`
	`}`

2.3.4 Power setting

Namespace	**RFIDReaderAPI.RFIDReader._RFIDConfig**
Function	static Int32 SetANTPowerParam(String ConnID, Dictionary<Int32, Int32> dicPower)

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	`string tcp = "192.168.1.116:9090";`
	`RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program();`
	`if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log))`
	`{`
	`  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 Query antenna power set

Namespace	**RFIDReaderAPI.RFIDReader._RFIDConfig**
Function	static Dictionary<Int32, Int32> GetANTPowerParam(String ConnID)
Parameter	// ConnID: connection identifier
Return	Can refer to the Dictionary<Int32, Int32> of 2.3.4 antenna power setting
Remark	

Example code	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);
	}

2.3.6 Tag data uploading

Namespace	**RFIDReaderAPI.RFIDReader._RFIDConfig**
Function	static Int32 SetTagUpdateParam(String ConnID, Int32 repeatTimeFilter, Int32 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	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");
	}

2.3.7 Tag date upload Parameter Query

Namespace	**RFIDReaderAPI.RFIDReader._RFIDConfig**
Function	static String GetTagUpdateParam(String ConnID)
Parameter	// ConnID: connection identifier
Return	repeatTimeFilter|RSSIFilter。
Remark	// repeatTimeFilter: repeat tag upload filter time (unit: 10ms) // RSSIFilter: RSSI filter

	repeatTimeFilter value range: 0 ~ 65535 RSSIFilter value range: 0 ~ 255

Example code	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);
	}

2.3.8 Get Device property

Namespace	**RFIDReaderAPI.RFIDReader._RFIDConfig**
Function	static String GetReaderProperty(String ConnID)
Parameter	// ConnID: connection identifier
Return	Minimum output power | maximum output power | number of antennas | band list | list of RFID protocols.
Remark	Power unit is dB

Example code	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);
	}

2.3.9 RF frequency range Set

Namespace	**RFIDReaderAPI.RFIDReader._RFIDConfig**
Function	static Int32 SetReaderRF(String ConnID, eRF_Range eRF_Range)

Parameter	// ConnID: connection identifier
	// eRF_Range:
	eRF_Range.GB_920_to_925MHz
	eRF_Range.GB_840_to_845MHz
	eRF_Range.GB_920_to_925MHz_and_GB_840_to_845MHz
	eRF_Range.FCC_902_to_928MHz
	eRF_Range.ETSI_866_to_868MHz
	eRF_Range.JP_916_to_921MHz
	eRF_Range.TW_922_to_927MHz
	eRF_Range.ID_923_to_925MHz
	eRF_Range.RUS_866_to_867MHz

Return	0: successful; others: failed

Remark	Note: the power of all antennas will be reset to 30 after the RF frequency is modified

Example code	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");
	}

2.3.10 Get RF frequency range

Namespace	**RFIDReaderAPI.RFIDReader._RFIDConfig**
Function	static String GetReaderRF(String ConnID)
Parameter	// ConnID: connection identifier
Return	eRF_Range.GB_920_to_925MHz eRF_Range.GB_840_to_845MHz eRF_Range.GB_920_to_925MHz_and_GB_840_to_845MHz eRF_Range.FCC_902_to_928MHz eRF_Range.ETSI_866_to_868MHz eRF_Range.JP_916_to_921MHz eRF_Range.TW_922_to_927MHz eRF_Range.ID_923_to_925MHz eRF_Range.RUS_866_to_867MHz
Remark	None

Example code	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);
	}

2.3.11 Set RF working frequency

Namespace	**RFIDReaderAPI.RFIDReader._RFIDConfig**
Function	static Int32 SetReaderWorkFrequency(String ConnID, eWF_Mode wfMode, List<eGB_840_to_845MHz> ListGB_840_to_845MHz)

Parameter	// 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.
Return	0: successful; others: failed
Remark	This function has many overload，other frequency setting only need change function eGB_840_to_845MHz to GB_920_to_925MHz, GB_840_to_845MHz, GB_920_to_925MHz_and_GB_840_to_845MHz, FCC_902_to_928MHz, ETSI_866_to_868MHz, JP_916_to_921MHz, TW_922_to_927MHz, ID_923_to_925MHz, RUS_866_to_867MHz then you could set related frequency points

Example code	`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");`
	`}`

2.3.12 Get RF working frequency

Namespace	**RFIDReaderAPI.RFIDReader._RFIDConfig**
Function	static String GetReaderWorkFrequency(String ConnID)
Parameter	// ConnID: connection identifier
Return	Mode | Frequency range | Frequency points e.g.: Auto|GB_920_to_925MHz|920.625,920.875
Remark	None

Example code	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);
	}

2.3.13 Set device Auto-sleep mode

Namespace	**RFIDReaderAPI.RFIDReader._RFIDConfig**
Function	static Int32 SetReaderAutoSleepParam(String ConnID, bool Switch, String 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:10ms

Example code	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");
	}

2.3.14 Get device Auto-sleep mode

Namespace	**RFIDReaderAPI.RFIDReader._RFIDConfig**
Function	static String GetReaderAutoSleepParam(String ConnID)
Parameter	// ConnID: connection identifier
Return	Close or Open| "sleep time"
Remark	Unit is 10ms

Example code	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);
	}

2.3.15 Network self-check Setting

Namespace	**RFIDReaderAPI.RFIDReader._RFIDConfig**
Function	static Int32 SetReaderSelfCheck(String ConnID, bool Switch, String ip)
Parameter	// ConnID: connection identifier, Switch: on / off, ip: ip address
Return	0: successful; others: failed
Remark	Switch: true-open, false-close

Example code	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 ");
	}

2.3.16 Query Network self-check status

Namespace	**RFIDReaderAPI.RFIDReader._RFIDConfig**
Function	static String GetReaderSelfCheck(String ConnID)
Parameter	// ConnID: connection identifier
Return	Close or Open|"IP address"
Remark	

Example code	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);
	}

2.3.17 Set antenna enabled

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

	if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log))
	{
	int rt = RFIDReader._RFIDConfig.SetReaderANT(tcp,eAntennaNo._1|eAntennaNo._2);
	if (rt == 0) Console.WriteLine("SET OK");
	else Console.WriteLine("SET FAILED");
	}

2.3.18 Get antenna enabled

Namespace	**RFIDReaderAPI.RFIDReader**._RFIDConfig
Function	static eAntennaNo GetReaderANT(String ConnID)
Parameter	// ConnID: connection identifier
Return	Refer to 2.3.17 - eAntennaNo
Remark	None

Example code	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);
	}

Namespace	**RFIDReaderAPI.RFIDReader**._RFIDConfig
Function	static String GetReaderANT2(String ConnID)
Parameter	// ConnID: connection identifier
Return	Antenna numbers already enabled, numbers apart by "," such as 1,6,8
Remark	None

Example code	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);
	}

2.4 GPIO operation

2.4.1 GPI trigger parameter configuration

Namespace	**RFIDReaderAPI.RFIDReader**._ReaderConfig

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

Parameter	// ConnID: connection identifier
	// GPINum: eGPI._1,eGPI._2,eGPI._3,eGPI._4;
	
	//triggerStart: eTriggerStart.OFF,eTriggerStart.Low_level,eTriggerStart.High_level,
	eTriggerStart.Rising_edge,eTriggerStart.Falling_edge,eTriggerStart.Any_edge;
	
	//triggerCode: triggerCode.Single_Antenna_read_EPC,
	triggerCode.Single_Antenna_read_EPC_and_TID,
	triggerCode.Double_Antenna_read_EPC,
	triggerCode.Double_Antenna_read_EPC_and_TID,
	triggerCode.Four_Antenna_read_EPC,
	triggerCode.Four_Antenna_read_EPC_and_TID;
	
	//triggerStop: eTriggerStop.OFF,eTriggerStop.Low_level,eTriggerStop.High_level,
	eTriggerStop.Rising_edge,eTriggerStop.Falling_edge,eTriggerStop.Any_edge,
	eTriggerStop.Delay;
	//DelayTime: unit is 10ms
	//isUpload: when triggerStop is eTriggerStop.OFF, whether to upload the GPI trigger status,
	true – upload, false – Don't upload
	
	e.g.: SetReaderGPIParam("192.168.1.116:9090",eGPI._2,eTriggerStart.Low_level,
	triggerCode.Double_Antenna_read_EPC_and_TID,eTriggerStop.Dealy,"100",false);
	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.

Return	0: successful; others: failed
Remark	Stop delay time: unit is 10ms (when Trigger stop condition is "Delay stop" effective) Related operation performance refers to the RFIDreader demo software Detailed GPI trigger parameter callback, please refer to 2.7 callback interface instrunction - GPIControlMsg();

Example code	`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");`
	`}`

2.4.2 Get GPI trigger Parameter

Namespace	**RFIDReaderAPI.RFIDReader**._ReaderConfig
Function	static String GetReaderGPIParam(String ConnID,eGPI GPINum)

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

Return	"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"
Remark	Stop delay time: unit is 10ms (when Trigger stop condition is" Delay stop" effective) isUpload: true – upload, false – Don't upload Related operation performance refers to the RFIDreader demo software

Example code	`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);`
	`}`

2.4.3 Get GPI state

Namespace	**RFIDReaderAPI.RFIDReader**._ReaderConfig
Function	static String GetReaderGPIState(String ConnID)
Parameter	// ConnID: connection identifier e.g. GetReaderGPIState("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 2.7 callback interface instrunction - GPIControlMsg();

Example code	`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);`
	`}`

2.4.4 GPO level operation

Namespace	**RFIDReaderAPI.RFIDReader**._ReaderConfig
Function	static Int32 SetReaderGPOState(String ConnID, Dictionary<eGPO, eGPOState> dicState)

Parameter	// ConnID: connection identifier
	// dicState: GPO serial number and level related key-value pair
	e.g. SetReaderGPOState("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	string tcp = "192.168.1.116:9090";
	RFIDReaderAPI.Interface.IAsynchronousMessage log = new Program();
	if (RFIDReaderAPI.RFIDReader.CreateTcpConn(tcp, log))
	{
	int rt = RFIDReader._ReaderConfig.SetReaderGPOState
	(tcp, new Dictionary<eGPO, eGPOState>() { {eGPO._1,eGPOState.High }});
	if (rt == 0) Console.WriteLine("SET OK");
	else Console.WriteLine("SET FAILED");
	}

2.4.5 Set Wiegand parameter

Namespace	**RFIDReaderAPI.RFIDReader**._ReaderConfig
Function	static Int32 SetReaderWG(String ConnID, eWiegandSwitch wiegandSwitch, eWiegandFormat wiegandFormat, eWiegandDetails param, int OutputAddress=0)

Parameter	// ConnID: connection identifier
	// eWiegandSwitch:
	eWiegandSwitch.Close,
	eWiegandSwitch.Open.
	// eWiegandFormat:
	eWiegandFormat.Wiegand26,
	eWiegandFormat.Wiegand34,
	eWiegandFormat.Wiegand66
	// eWiegandDetails:
	eWiegandDetails.end_of_the_EPC_data,
	eWiegandDetails.end_of_the_TID_data.
	
	// 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

Return	0: successful; others: failed
Remark	None
Example code	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");
	}

2.4.6 Get Wiegand parameter

Namespace	**RFIDReaderAPI.RFIDReader**._ReaderConfig
Function	static String GetReaderWG(String ConnID)
Parameter	// ConnID: connection identifier
Return	// Return: Wiegand switch|Wiegand format|Wiegand transferred data 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	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);
	}

2.5 6C tag operation

2.5.1 Read tag

Namespace	**RFIDReaderAPI.RFIDReader**._Tag6C

Function1	**static int GetEPC(string ConnID, eAntennaNo antNum, eReadType readType, eMatchCode matchType = eMatchCode.None, string matchCode = "", int matchWordStartIndex = -1, string accessPassword = "")**
	// only read EPC
	// ConnID: connection identifier
	// antNum: Antenna number enumeration.
	Appoint Antenna 1 and 2 working at same time; e.g.: eAntennaNo._1|eAntennaNo._2
	// ReadType: read type enumeration, Single or Inventory (one-time or cyclically reading)
	Optional parameters :(all Tag6C functions will contain optional parameters, which will not be specified in the future)

	// matchType： matching area EPC|TID
	// matchCode: tag data need to be matched
	// matchWordStartIndex: matching Data starting index
	// accessPassword: Tag access password
	Note: matchType, matchCode and matchWordStartIndex need to be used together.

Function2	**int GetEPC_TID(string ConnID, eAntennaNo antNum, eReadType readType, int length = -1, eMatchCode matchType = eMatchCode.None, string matchCode = "", int matchWordStartIndex = -1, string accessPassword = "")**
	// read EPC and TID
	// length: optional parameter, reading length of TID

Function3	**int GetEPC_UserData(string ConnID, eAntennaNo antNum, eReadType 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)

Function4	**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)

Function5	**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)

Function6	**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)

Function7	**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)

Function8	**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
	G2V2AuthenticateModel: Set the C2G2 authentication parameters
	//_AuthMethod: TAM Authentication Method
	//_CustomData: TAM Custom Data
	//_KeyID: TAM KeyID
	//_Profile: memory parameter. 0,EPC area; 1, TID area; 2, user area
	//_Offset: Define the first address of the custom data block.
	//_BlockCount: Block Count
	//_ProtMode: Specifies the mode of operation that shall be used for the encipherment and/or protection of the custom data
	//G2V2Authenticate(): Converts the decimal parameters to a hexadecimal String, returning a String
	// accessPassword: Tag access password

Function9	**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)

Function10	**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
Function11	**int GetEPC_RFMicron_Temperature(string ConnID, eAntennaNo antNum, eReadType readType)** // read EPC and RFMicron temperature tag
Function12	**int GetEPC_EM4325_Temperature(string ConnID, eAntennaNo antNum, eReadType readType)** //read EPC and EM4325 temperature tag

Function13	**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)

Function14	**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)
Parameter	refers to each function instruction
Return	True: successful; false: failed
Remark	1. For detailed Return , please kindly follow Appendix A 2. Stop inventory (cycle) reading using"stop" instruction. 3. Difference between inventory and single reading is that single read automatically stops reading after one time reading, but inventory read requires a stop function to stop reading. 4. The same part for inventory and single reading is that after the last tag data is uploaded, they will notify PC side through asynchronous callback that tag upload finished, please refer to 2.7 callback interface instrunction - OutPutTagsOver();

Example code	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");
	}

2.5.2 Write tag

2.5.2.1 Write EPC

Namespace	**RFIDReaderAPI.RFIDReader**._Tag6C
Function1	**string WriteEPC(string ConnID, eAntennaNo antNum, string sWriteData, string**

	dataStartIndex = "", eMatchCode matchType = eMatchCode.None, string matchCode = "", int matchWordStartIndex = -1, string accessPassword = "")
	// 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)
	Optional parameters :
	// 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
	Note: matchType, matchCode and matchWordStartIndex need to be used together.

Parameter	Please refer to Function Remark.
Return	True: successful; others: failed
Remark	1. Suggest using matched TID to write 2. For detailed Return， refers to Appendix 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.

Example code	`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);`
	`}`

2.5.2.2 Write Userdata

Namespace	**RFIDReaderAPI.RFIDReader**._Tag6C

Function1	**static Int32 WriteUserData(String ConnID, eAntennaNo antNum, String sWriteData, Int32 offset, eMatchCode matchType = eMatchCode.None, string matchCode = "", int matchWordStartIndex = -1, string accessPassword = "")**
	// 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
	Optional parameters:
	// 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 **Note: matchType, matchCode and matchWordStartIndex need to be used together.**
Parameter	Please refer to FunctionRemark.
Return	True: successful; others: failed
Remark	1. Suggest using matched TID to write 2. For detailed Return ,refers to Appendix A

Example code	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. }

2.5.2.3 Write password

Namespace	**RFIDReaderAPI.RFIDReader._Tag6C**

Function1	**static Int32 WriteAccessPassWord(String ConnID, eAntennaNo antNum, String sWriteData, eMatchCode matchType = eMatchCode.None, string matchCode = "", int matchWordStartIndex = -1, string accessPassword = "")**
	// sWriteData: password content（8 Hexadecimal string data）
	Optional parameters:
	// 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
	Note: matchType, matchCode and matchWordStartIndex need to be used together.

Parameter	Please refer to FunctionRemark.
Return	True: successful; others: failed. refers to Appendix A
Remark	

Example code	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);
	}

2.5.3 Lock tag

Namespace	**RFIDReaderAPI.RFIDReader._Tag6C**

Function1	static Int32 Lock(String ConnID, eAntennaNo antNum, eLockArea lockArea, eLockType lockType)
	// ConnID: connection identifier
	// antNum: antenna number
	// lockArea: lock area enumeration
	// lockType: lock type enumeration

Function2	static Int32 Lock_MatchEPC(String ConnID, eAntennaNo antNum, eLockArea lockArea, eLockType lockType, String sMatchData, Int32 matchWordStartIndex)
	// sMatchData: EPC data to be matched (Hexadecimal string)
	// matchWordStartIndex: match Data starting address, unit is word

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

Function4	static Int32 Lock_MatchTID(String ConnID, eAntennaNo antNum, eLockArea lockArea, eLockType lockType, String sMatchData, Int32 matchWordStartIndex)
	// sMatchData: TID data to be matched(Hexadecimal string)
	// matchWordStartIndex: match Data starting address, unit is word

Function5	static Int32 Lock_MatchTID(String ConnID, eAntennaNo antNum, eLockArea lockArea, eLockType lockType, String sMatchData, Int32 matchWordStartIndex, String accessPassword) // accessPassword: Tag access password
Parameter	// refer to above method Remark
Return	True: successful; false: failed refers to Appendix A
Remark	

Example code	`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);`
	`}`

2.5.4 Tag killing (destroy)

Namespace	**RFIDReaderAPI.RFIDReader._Tag6C**

Function1	static Int32 Destroy(String ConnID, eAntennaNo antNum, String destroyPassword)
	// ConnID: connection identifier
	// antNum: antenna number
	// destroyPassword: kill password (Hexadecimal string)

Function2	static Int32 Destroy_MatchEPC(String ConnID, eAntennaNo antNum, String destroyPassword, String sMatchData, Int32 matchWordStartIndex)
	// sMatchData: EPC data to be matched(Hexadecimal string)
	// matchWordStartIndex: match Data starting address, unit is word

Function3	static Int32 Destroy_MatchTID(String ConnID, eAntennaNo antNum, String destroyPassword, String sMatchData, Int32 matchWordStartIndex) // sMatchData: TID data to be matched (Hexadecimal string)
Parameter	// refer to above method Remark
Return	True: successful; false: failed refers to Appendix A
Remark	

Example code	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);
	
	}

2.5.5 Get QTParameter

Namespace	**RFIDReaderAPI.RFIDReader._Tag6C**

Function1	static String GetQtTagOption (String ConnID, eAntennaNo antNum)
	// ConnID: connection identifier
	// antNum: antenna number

Function2	static String GetQtTagOption (String ConnID, eAntennaNo antNum, String accessPassword) // accessPassword: Access Password (Hexadecimal string)

Function3	static String GetQtTagOption_MatchEPC (String ConnID, eAntennaNo antNum, String sMatchData, Int32 matchWordStartIndex, String accessPassword)
	// sMatchData: matched TID data(Hexadecimal string)
	// matchWordStartIndex: match Data starting address, unit is word

Function4	static String GetQtTagOption_MatchTID (String ConnID, eAntennaNo antNum, String sMatchData, Int32 matchWordStartIndex, String accessPassword)
	// sMatchData: matched TID data(Hexadecimal string)
	// matchWordStartIndex: match Data starting address, unit is word

Parameter	// refers to above method
Return	successfully: qt1 + "|" + qt2 failed: ""
Remark	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
Example code	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);
	}

2.5.6 Set QTParameter

Namespace	**RFIDReaderAPI.RFIDReader._Tag6C**

Function1	static String SetQtTagOption (String ConnID, eAntennaNo antNum, bool IsNotReduceResponseDistance,bool IsPrivateMode)
	// 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	static String SetQtTagOption (String ConnID, eAntennaNo antNum, bool IsNotReduceResponseDistance, bool IsPrivateMode, String accessPassword) // accessPassword: Access Password (Hexadecimal string)

Function3	static String SetQtTagOption_MatchEPC (String ConnID, eAntennaNo antNum, String sMatchData, Int32 matchWordStartIndex, bool IsNotReduceResponseDistance, bool IsPrivateMode, String accessPassword)
	// sMatchData: EPC data need to be matched (Hexadecimal string)
	// matchWordStartIndex: match Data starting address, unit is word

Function4	static String SetQtTagOption_MatchTID (String ConnID, eAntennaNo antNum, String sMatchData, Int32 matchWordStartIndex, bool IsNotReduceResponseDistance, bool IsPrivateMode, String accessPassword)
	// 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	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");
	}

2.5.7 G2V2 Untraceable operation

Specific firmware support is required

Namespace	**RFIDReaderAPI.RFIDReader._Tag6C**

Function1	static String SetG2V2UntraceableOption(String ConnID, eAntennaNo antNum, G2V2UntraceableModel g2v2)
	// ConnID: connection identifier
	// antNum: antenna number
	// g2v2: G2V2 Untraceable parameters

Function2	static String SetG2V2UntraceableOption(String ConnID, eAntennaNo antNum, G2V2UntraceableModel g2v2, String accessPassword) // accessPassword: Tag access password

Function3	static String SetG2V2UntraceableOption_MatchEPC (String ConnID, eAntennaNo antNum, G2V2UntraceableModel g2v2, String sMatchData, Int32 matchWordStartIndex, String accessPassword)
	// sMatchData: the EPC data to be matched
	// matchWordStartIndex: match Data starting index

Function4	static String SetG2V2UntraceableOption_MatchTID (String ConnID, eAntennaNo antNum, G2V2UntraceableModel g2v2, String sMatchData, Int32 matchWordStartIndex, String accessPassword)
	// sMatchData: the TID data to be matched
	// matchWordStartIndex: match Data starting index

Parameter	G2V2UntraceableModel : G2V2 Untraceable parameters
	// int _Untraceable_flag_bit: Untraceable flag bit 0/1
	// int _EPC: EPC show/hide flag bit. 0,show; 1, a Tag untraceably hides EPC memory above that set by its EPC length field
	// int _EPC_length: new EPC length.
	// int _TID: TID hide parameter. 0, show; 1, the Tag untraceably hides TID memory above 20h; 2, the Tag untraceably hides all TID memory.
	// int _User: USER hide parameter. 0,show; 1, hide
	// int _Range: Tag response range. 0,normal; 1, toggle temporarily; 2, reduced
	// UntraceableParam(): 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	`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.
            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 operations (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 Remark

```
// 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
WriteDebugMsg	output debug message
WriteLog	output log message
PortConnecting	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.

PortClosing	When the device is disconnected, the API calls back the connection ID, indicating that the device with the current connection ID is disconnected.
OutPutTags	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. 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.
OutPutTagsOver	After the last tag is uploaded, a sync end signal is uploaded indicating the end of the current read tag action.
GPIControlMsg	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
ReaderName	Reader connection identifier, means which reader are reading data，e.g.: "192.168.1.116:9090"
TagType	Tag type，"6c","6b","gb" 3 types.
EPC	Tag EPC data，Hexadecimal string.
PC	Tag PC value
ANT_NUM	uploaded antenna number
RSSI	RSSI value
TID	Tag TID，Hexadecimal string.
UserData	Tag user area data，Hexadecimal string.
TagetData	Tag password area data, include access password and kill password, Hexadecimal string.
Frequency	Tag carrier frequency
Phase	Tag phase
ReadTime	Tag reading time
G2V2Challenge	G2V2 Challenge data
G2V2Data	G2V2data
EPCData	Tag EPCdata
Bag	The sub antenna number corresponding to the hub
RSSI_dB	RSSIvalue unit is dBm
CatchCount	Count in cache

2.9 callback data GPI_Model field introductions

Filed	Remark
ReaderName	Reader connection identifier, means which reader are reading data，e.g.: "192.168.1.116:9090"

GpiIndex	GPI port index, starting at 1, 1 represents GPI1, and so on.
GpiState	0 means low level, 1 means high level
StartOrStop	0 means trigger start, 1 means trigger end
UTC	Sensor trigger UTC time, byte[] type, length is 8, The first 4 bytes is UTC seconds and the last 4 bytes is microseconds
Utc_Time	Sensor trigger UTC time, string type, format is: "yyyy.MM.dd HH:mm:ss.fff"

2.10 Breakpoint continuous transferring

2.10.1 Configuration

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static Int32 SetBreakPointUpload(String ConnID, bool Switch)

Parameter	// ConnID: connection identifier
	// Switch: false:close, true:open
	e.g.: SetBreakPointUpload("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 ReadTime field in the Tag Model will be effective

Example code	`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");`
	`}`

2.10.2 Get/Query

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static String GetBreakPointUpload(String ConnID)
Parameter	// ConnID: connection identifier e.g.: GetBreakPointUpload("192.168.1.116:9090");
Return	Open or Close
Remark	None

Example code	`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); }

2.10.3 Retrieve breakpoint cache

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

Example code	string 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);
	}

2.10.4 Clear breakpoint cache

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static Int32 ClearBreakPointCache(String ConnID)
Parameter	// ConnID: connection identifier
Return	0: successful; others: failed
Remark	When the PC and the device link layer is interrupted, the data read will be stored in the cache of the reader, When this method is called, the cache when the reader is interrupted will be cleared.

Example code	string 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");
	}

2.11 WiFi operation

2.11.1 Query WiFi switch state

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

Example code	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");
	}

2.11.2 Set Wi-Fi switch state

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static Int32 SetWiFiSwitchState(String ConnID,bool isOpen)
Parameter	// ConnID: connection identifier // isOpen: Set switch state. True: open； false: close
Return	0: successful; 1: failed
Remark	

Example code	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");
	}

2.11.3 Query Wi-Fi adaptor IP

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static String GetReaderWifiIP(String ConnID)

Parameter	// ConnID: connection identifier
Return	IP| Subnet mask | Gateway
Remark	

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

2.11.4 Set Wi-Fi adaptor IP

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static Int32 SetReaderWifiIP(String ConnID,String ip,String mask,String gateway)

Parameter	// ConnID: connection identifier
	// ip: IP address
	//mask: Subnet mask
	//gateway: network gateway

Return	0: successful; 1: failed
Remark	

Example code	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","192.168.1.1");
	if (rt == 0) Console.WriteLine("SET OK");
	else Console.WriteLine("SET FAILED");
	}

2.11.5 Search Wi-Fi hotspot

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static Int32 SearchWIFI(String ConnID)
Parameter	// ConnID: connection identifier
Return	0: successful; 1: failed
Remark	
Example code	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);
	}

2.11.6 save Wi-Fi searching result

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static Boolean RequestWiFiDataToTxt(String ConnID,String filepath)
Parameter	// ConnID: connection identifier //filepath: saved txt file path
Return	Ture: successful; false: failed
Remark	

Example code	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);
	}

2.11.7connect Wi-Fi hotspot

Namespace	**RFIDReaderAPI.RFIDReader._ReaderConfig**
Function	static Int32 ConnectWIFI(String ConnID,string name, string pwd, int authType, int encryption)

Parameter	// ConnID: connection identifier
	// name: wifi name
	//pwd: wifi password
	//authType: Authentication type
	//encryption: Encryption Algorithm

Return	True: successful; false: failed

Remark	e.g.:
	1、when Wi-Fi hotspot need password:
	if (WifiAuth=="WPA")
	{
	authType = 1;
	}
	else if (WifiAuth == "WPA2")
	{
	authType = 2;

	}
	int isConnect = CLReader._ReaderConfig.ConnectWIFI(ConnID, WifiName, pwd, authType, 2);
	
	2、when Wi-Fi hotspot without password:
	int ret = CLReader._ReaderConfig.ConnectWIFI(ConnID, WifiName, "", 0, 2);

Example code	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");
	}

2.11.8 Query connecting Wi-Fi name

Namespace	**RFIDReaderAPI.RFIDReader**._ReaderConfig
Function	static String GetWiFiConnectInfo(String 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 Sqllite 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 Configure

Namespace	RFIDReaderAPI.RFIDReader._ReaderConfig
Function	static Int32 SetWhitelistLabelAction (String ConnID, string 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)| data type (0 TIID/1 EPC)
Return	0: successful; others: failed

Remark	None

Example code	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");
	}

2.12.2 Query configuration

Namespace	RFIDReaderAPI.RFIDReader._ReaderConfig
Function	static Int32 GetWhitelistLabelAction (String ConnID)
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	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);
	}

2.12.3 Upload whitelist database file to reader

Namespace	RFIDReaderAPI.RFIDReader._ReaderConfig
Function	static Int32 LoadWhiteList (String ConnID, uint dataIndex, byte[] param)

Parameter	// ConnID: connection identifier
	// dataIndex: upgrade data serial number 0x00000000 start, 0xFFFFFFFF end
	// param : whitelist data

Return	0: successful; others: failed
Remark	

Example code	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 >= 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 = "";
        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 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

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|eAntennaNo._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 solution

Question	Solution
Device couldn't work normally	1. Check power light normal or not. 2. If normal, there should be some notice sound when power on.
serial port couldn't work normally	1. Check connection cable connecting normal or not . 2. If conditional, use another device to check this cable normal or not. 3. Try to use RJ45 to communicate. 4. Default baud rate: 115200.
RJ45 couldn't work normally	1. Check LED working normal or not. 2. To use Ping instruction to check cable working normal or not 3. Try serial port connection, inquiry IP correct by Demo 4. Default IP and port: "192.168.1.116:9090"

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