Invengo

Techincal reference manual

Fixed or handheld device

Version 1.2.3

Invengo Information Technology Co., Ltd.

Foreword

The version of the manual is: V1.2.3, with revision record as follows:

27th July 2012	**Initial draft V1.0**
29th October 2014	**V1.2.2**
5th February 2015	**V1.2.3**

Content

1 Preface.....7

1.1 Purpose.....7

1.2 Range.....7

1.3 Definition.....8

1.4 References.....8

1.5 Abbreviation.....9

1.6 Reading instruction.....9

2 Summary.....10

2.1 Reader basic hardware framework.....10

2.2 Reader basic operating mechanism.....11

2.3 Protocol basic framework.....12

2.4 API component.....13

3 Reader communication method.....15

3.1 Serial port communication.....15

3.2 Ethernet net communcation.....15

4 API Summary.....16

4.1 Reader type.....18

4.2 Reader message.....22

4.3 Host computer message.....24

4.4 6C tag data area enum(MemoryBank).....26

4.5 Configuration fle(Sysit.xml).....26

4.6 Antenna port definition.....27

4.6.1 Tag reading/writing operation.....27

4.6.2 Tag scanning operaiton.....27

5 Reader reference query and configuration.....29

5.1 Reader system parameter query (SysQuery_800 type).....29

5.2 Reader system parameter configuration (SysConfig_800 type) 30
5.3 Reader system parameter explanation 31
5.3.1 Reader version information 31
5.3.2 Commucation parameter configuration 32
5.3.3 RFID parameter configuration 34
5.3.4 GPI trigger parameter configuration 35
5.4 Inquire reader system parameter (SysQuery_500 type) 36
5.5 Configure reader system parameter (SysConfig_500 type) 37
6 Electronic tag identification and access 39
6.1 ISO18000-6C(EPC C1G2) electronic tag storage structure 39
6.2 Tag data reading 41
6.2.1 Tag scanning (ReadTag type) 41
6.2.2 Receive tag data information (RXD_TagData type) 47
6.2.3 6C tag reading user data area (ReadUserData_6C type) 49
6.3 Tag data writing 51
6.3.1 6C tag write EPC (WriteEpc type) 51
6.3.2 6C tag write user data area (WriteUserData_6C type) 53
6.3.3 6B tag write user data area (WriteUserData2_6B type) 54
6.3.4 6C tag configuration access password (AccessPwdConfig_6C type) 55
6.3.5 6C tag kill password configuration (killPwdConfig_6C type) 57
6.3.6 6C tag general writing operation (WriteTag_6C type) 59
6.4 Tag lock operation 62
6.4.1 6C tag lock operation (LockMemoryBank_6C type) 62
6.4.2 6B tag lock user data (LockUserData_6B type) 64
6.4.3 6B tag lock state query (LockStateQuery_6B type) 65
6.5 6C tag kill (KillTag_6C type) 67
6.6 Private function of special tag 68
6.6.1 EAS function in NXP tag 68
6.6.2 6C tag QT command (QT_6C type) 71
6.7 6C tag selection(SelectTag_6C type) 74
6.8 Tag filter by time (FilterByTime type) 76

7 Stop command, power off (PowerOff type)..... 77
8 GPIO function 78
8.1 GPI input status query (GPI_800 type) 78
8.2 GPO output operation (GPO_800 type) 79
8.3 GPI event trigger message upload (RXD_IOTriggerSignal_800 type).... 80
9 Handheld reader special module 81
9.1 Data package 81
9.1.1 Construct 81
9.1.2 Data properties 81
9.1.3 Converting to string 82
9.2 RFID control 83
9.2.1 Switch on RFID 83
9.2.2 Switch off RFID 83
9.2.3 RFID power status 83
9.3 GPRS control 84
9.3.1 Switch on GPRS 84
9.3.2 Switch off GPRS 84
9.3.3 GPRS power status 85
9.3.4 Dial connection 85
9.3.5 Dial disconnection 85
9.4 Wi-Fi control 86
9.4.1 Switch on Wi-Fi 86
9.4.2 Switch off Wi-Fi 86
9.4.3 Wi-Fi power status 86
9.5 Capture system power message 87
9.5.1 Power status request 87
9.5.2 Initiate capture power status message 87
9.5.3 Stop capture power message 87
9.5.4 Power status message value 88
9.6 Barcode identification/camera switch 89
9.6.1 Query current mode 89
9.6.2 Choose barcode reading 89

9.6.3 Choose camera 89
9.7 Barcode identification 90
9.7.1 BarcodeScannerSymbol type 90
9.7.2 BarcodeScannerHoneywell1D type 91
9.7.3 BarcodeScannerHoneywell2D type 92
9.7.4 SDLScanner type 93
9.7.5 DecodeData type 95
9.8 Function key message 96
9.8.1 Capture message 96
9.9 Sound volume 96
9.9.1 Configure volume 96
9.9.2 Get volume 97
9.10 Play sound file 98
9.10.1 Play WAV 98
9.11 Date and time 99
9.11.1 Date and time structure 99
9.11.2 Get date and time 99
9.11.3 Configure date and time 99
9.12 Standby/reboot/shutdown 100
9.12.1 Standby 100
9.12.2 Reboot 100
9.12.3 Shutdown 100
9.13 Data conversion 101
9.13.1 Conversion of byte data to hexadecimal string 101
9.13.2 Conversion of hexadecimal string to byte data 102
10 Appendices 103
10.2 Reader error code chart 104

1 Preface

1.1 Purpose

This article describes the basic operating principles of Invengo reader, with detailed description on the Application Programming Interface (API) between Invengo readers and host computer (control). The host computer (control) can control the data communication between the reader and the electronic tag through API.

1.2 Range

The application Invengo reader model:

- Fixed: XC-RF807, XC-RF850, XC-RF861, XC-RM825, XC-RM829, XC-RF812, XC-RF811(V3.0)
- Handheld: XC2900, XC2903

The application range for fixed reader API is as shown below:

- C# API: Written in Microsoft .NET Framework v2.0. Supports adjustment and secondary development in .NET family language.
- JAVA API: Written in JDK1.5, only supports adjustment and secondary development in JDK environment version 1.5 and above.
- C++ API: Written in Microsoft vc 6.0, supports adjustment and secondary development in c++,dephi,vb and other languages under windows environment.

The application range for handheld reader API is as shown below:

- C# API: Written in Microsoft .NET Compact Framework v2.0.

1.3 Definition

- √ IRP: Invengo Reader Protocol, Invengo reader data transmittion control protol. This article only touches on IRP version 1.0.
- √ API: Application programming interface. Allow secondary development. A package for reader communication protocol.
- √ Host computer: Refers to the personal computer (PC) or other control terminal that performs data exchange with the readers.
- √ Client machine: Refers to the readers
- √ COM: cluster communication port
- √ TCP: Transfer Control Protocol
- √ USB: Universal Serial BUS
- √ Q value: Refers to a parameter used by the reader in the adjustment of tag response probability. In an inventory cycle, the reader can assign the tag to insert a Q-digit random (false-random) value into the slot counter, the reader can also commands the tag to conduct minus 1 operation to their slot counter. The tags will response to the reader's command when the slot counter value turns zero. Q is a positive integer within the (0, 15) range, the corresponding tag response probability range is between 20=1 to 2–15 =0.000031.
- √ RSSI: Received Signal Strength Indication.
- √ UTC: Universal Time Coordinated. The default start point is 1970-1-1 00:00:00.
- √ BCD: Use 4 binary digits to represent a decimal number from 0 to 9 this 10 numbers.
- √ Word: Two bytes as one word.

1.4 References

1. EPC™ Radio-Frequency Identity Protocols Type-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz–960 MHz Version 1.2.0
2. IMPINJ Monza4 Datasheet
3. NXP SL3ICS1002 G2XM Datasheet
4. Invengo reader data transfer protocol
5. Sample code

1.5 Abbreviation

- √ 6C: ISO 18000-6C
- √ 6B: ISO 18000-6B
- √ 807: XC-RF807 reader
- √ 850: XC-RF850 reader
- √ 861: XC-RF861 reader
- √ 825: XC-RM825 reader model
- √ 829: XC-RM829 reader model
- √ 812: XC-RF812 card issuance
- √ 2903: XC2903-F6C portable reader
- √ Host computer message: Automatic message sent by the host computer. This message must implement IMessage interface, refers to 4.3 for details.
- √ Reader message: Automatic message sent by reader. This message must implement IMessageNotification interface, refers to 4.2 for details.

1.6 Reading instruction

This API is written in three languages, UML language is partially used to facilitate general explanation.

If not specified, this API is applicable to both fixed and handheld readers.

2 Summary

2.1 Reader basic hardware framework

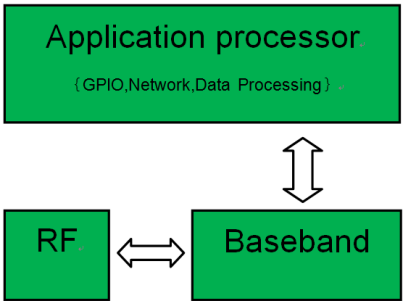


Figure 2-1

The basic hardware of reader is consisted of application processing unit, RFID baseband processing unit, RF radio hardware circuit unit.

Application processing unit: Mainly responsible for network communication, data processing, GPIO and other application related processors between reader and the host computer.

RFID baseband processing unit: Mainly responsible for data exchange and flow protocol control between reader and the host computer.

RF circuit: Responsible for the physical signal transmission between the reader and tag.

XC-RM825 XC-RM829 only provides basic communication protocol (R232) and RFID function. On the hardware, the application processing unit and baseband processing is merged and use single CPU architecture.

XC-RF807 XC-RF850 and XC-RF861 have complete communication interface (RS232) and network communication ability, with stronger data processing power to support large-scale network applications and complex application business logic processing, application processing unit and a baseband processing unit RFID hardware using dual CPU architecture.

2.2 Reader basic operating mechanism

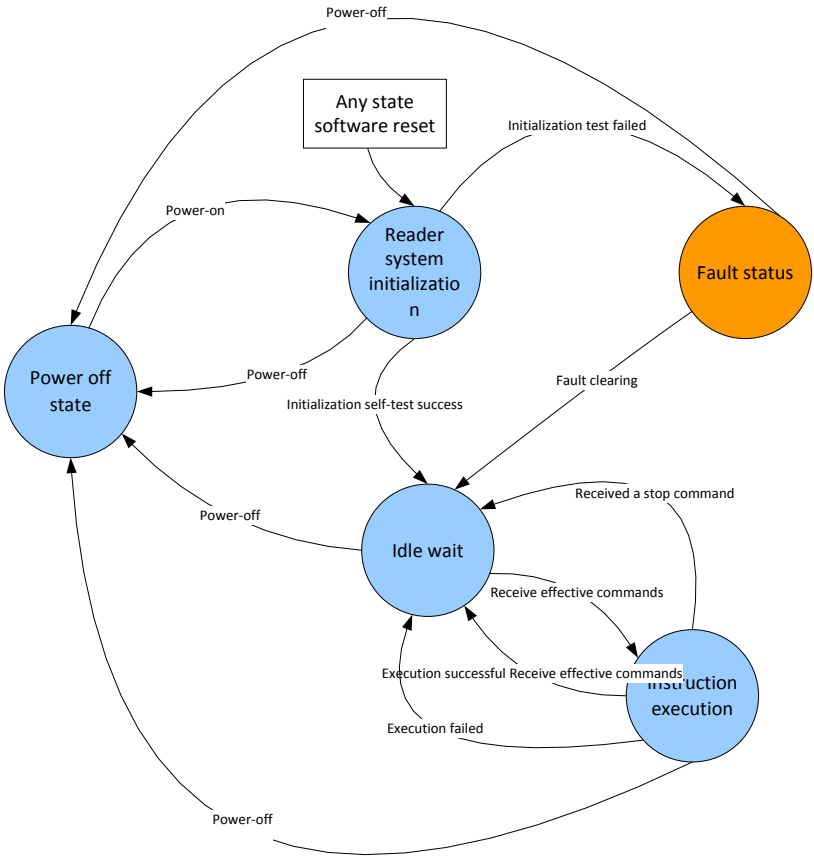


Figure 2-2

Reader system initialization: The operating system starts, each functional module hardware status self-testing, the reader system parameters initialization, the reader can not receive and execute any commands from the host in this condition.

Idle waiting: The condition where the reader has completed the initialization and is waiting for the instruction from host computer, the reader can receive any instructions and executed immediately under this condition.

Command execution: Once the reader receives a complete lawful instruction from the host computer it will immediately switch to the instruction execution condition, reader will respond to operation termination, GPIO input and output operations and parameter query, during the implementation of cyclic operation of tag reading and writing instruction.

Fault status: When the system experiences failure during system initialization and self-test, this status is mainly for fault alarms and system debugging interface to facilitate quick fault identification in the the development debugging and production process.

2.3 Protocol basic framework

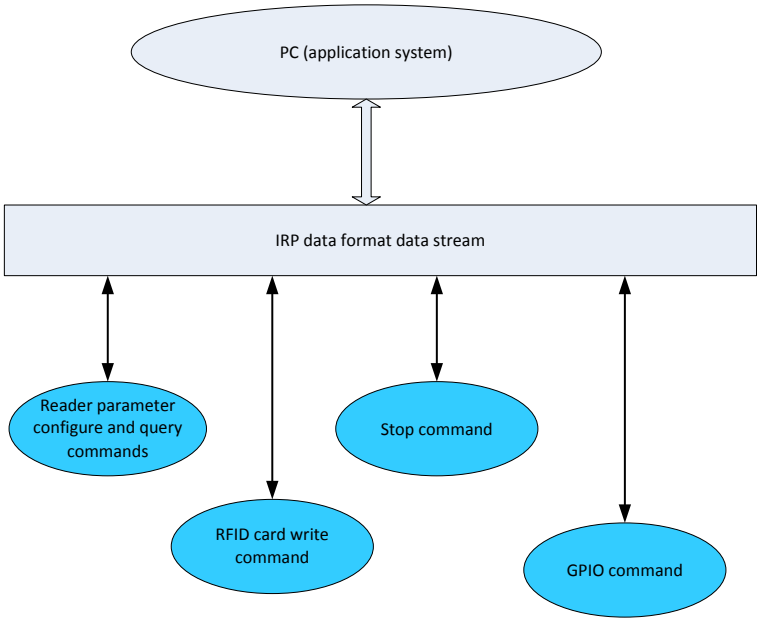


Figure 2-3

Refer to the figure above for Invengo basic framework protocol reader, reader instruction includes the following categories: the reader parameters and configuration, RFID reader card, stop command, GPIO instruction.

2.4 API component

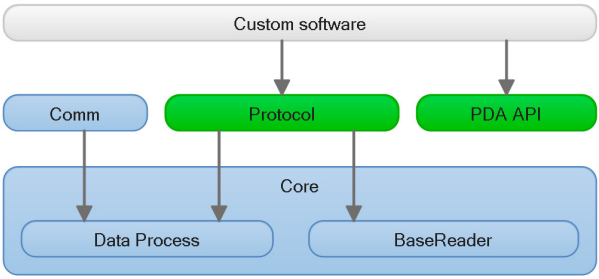


Figure 2-4 API Hierarchy Figure

As shown in the figure above, apart from client software, the rest are API. The green module in the figure above is the application module for secondary development by the client; client does not need to be concered about the blue part.

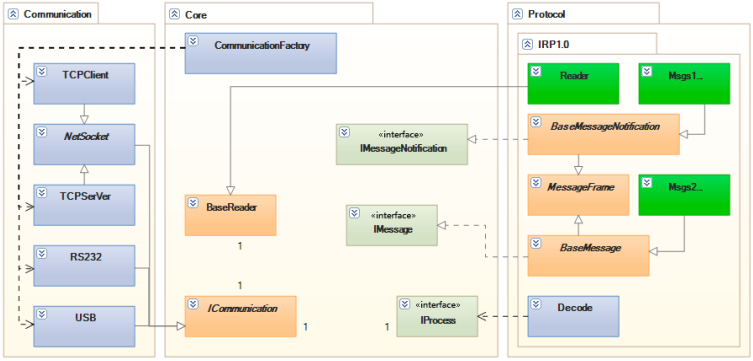


Figure2-5 API Type Figure (Apart from handheld reader specific module)

API contails four large modules:

1. Communication module: The communication module in the corresponding relationship diagram. The “Communication” package in the figure refers to communication module, listed in the figure are the four communication modes.
2. Protocol module: The instruction module in the corresponding relationship diagram. The “Protocol” package in the figure above refers to protocol module, currently only supports IRP1.0 protocol.

3. Core module: The data exchange center in the corresponding exchange center. The module uses factory mode to connect with communication module and uses adapter mode to exchange information with communication module.
4. Handheld reader specific module: Invengo handheld reader specific module controls Invengo handheld reader RFID module, GRPS, Wi-FI and other hardware equipment. Refers to chapter 9 for more details.

Fixed reader offers three sets of programming languages for API, namely C#, JAVA andC++. Handheld reader uses C# language API.

The main components of fixed reader API is as follows:

- C# API includes:
  - RFIDInterface.dll: Data exchange center
  - Communication.dll: Communication module
  - IRP1.dll, IRP1Message.xml: Invengo protocol instruction module
  - language document file: Includes XML file that contains multiple languages package, ErrCode(****).xml
- JAVA API includes:
  - IRP1.jar: Invengo protocol module
- C++ API includes:
  - XCRFAPI.DLL reader delivery api
  - Header files of XCRFHEAD dll application

The main component of handheld API is as follows:

- C# API includes:
  - RFIDInterface.dll: Data exchange center
  - Communication.dll: Communication module
  - IRP1.dll, IRP1Message.xml: Invengo protocol instruction module
  - language document file: Includes XML file that contains multiple languages package, ErrCode(****).xml
  - Invengo.dll: Provides common operation
  - Invengo.Devices.ModuleControl.dll: Modular power supply control
  - Invengo.Barcode.dll: Honeywell or Symbol barcode identification
  - MotoSDL.dll: Motorola ½ dimensional identification

3 Reader communication method

3.1 Serial port communication

All readers support RS232 interface. For communication to take place between serial port and reader, the communication parameters of the reader must be established properly, such as baud rate. The default reader serial port (RS232) communication parameters are: baud rate 115 200, 8 data bits, 1 stop bit, no parity.

3.2 Ethernet net communcation

The eternet communication (based on TCP/IP protocol, only suppprts IPV4) provided by the reader includes reader as a server (also known as passive mode) and reader as client (also known as active mode). At any point of time, the reader can only be one of the two modes. Reader as server mode refers to the external device (such as PC) actively initiates network connection with the reader. Reader as client mode refers to the reader actively initiates network communication with the external device. The factory default IP of the reader is 192.168.0.210. Please ensure the reader's IP and PC's IP is at the same network segment during network connection.

Reader as client is a working mode where the reader actively connects with the remote server under configured parameters. The active mode and backup server are used together, once the primary server fails, the reader can automatically connect to the backup server.

Active mode is controlled by the active mode switch. When the active mode switch is switched on, passive mode is disabled. When the active mode switch is switched off, the active mode is automatically switched off.

4 API Summary

API mainly refers to the interface between the client software and reader communication, mainly includes reader type and a series of message types, and the realization of various functions through these types, such as establish and disable connection with the reader and send and receive message.

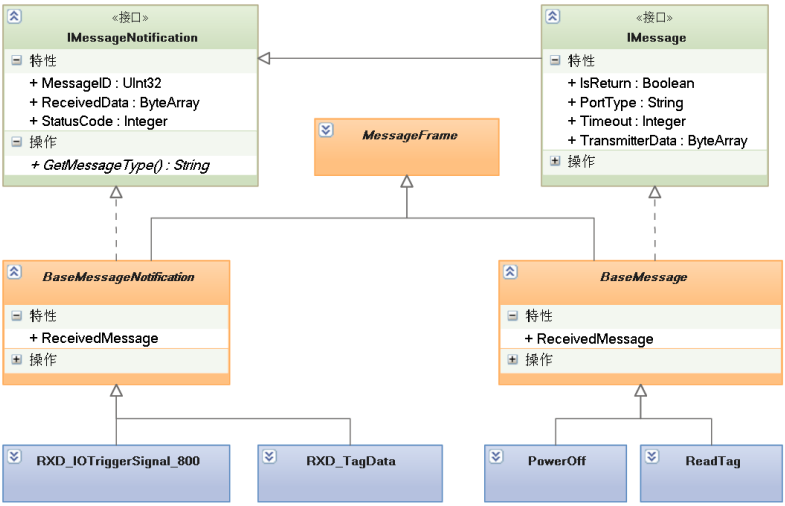


Figure 4-1 Message type figure

Message type packages Invengo protocol command information, for example RXD_IOTriggerSingnal_800 type reader message, RXD_TagData type and PowerOff type and ReadTag type host computer message in the figure above.

The reader type packages the reader control of the message type and reader self status, as shown in the figure below.

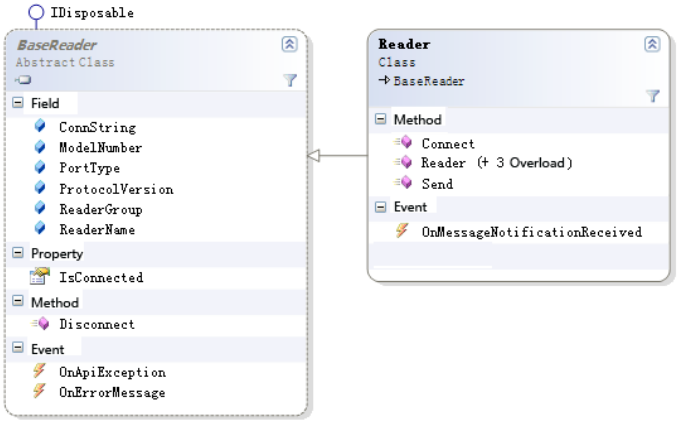


Figure 4-2 Reader type figure

These types exist in the following files. Note: The actual development process should include all the files mentioned in the “API component” chapter.

C#:

Namespace: Invengo.NetAPI.Protocol.IRP1

Assemby: IRP1 (in IRP1.dll)

JAVA:

Package name: invengo.javaapi.protocol.IRP1 (in IRP1.jar)

C++:

Header file under IRP1APIHEAD//IRP1_H directory

4.1 Reader type

This type of reader carries out operation on one communication channel, capable of performing four main core operations in reader connection, disconnection, data sending and data receiving.

1. Constructor

Purpose: Instantiation of Reader subtype

Syntax:

+ Reader (readerName: String)
+ Reader (readerName: String,
  portType: String,
  connStr: String)
+ Reader (server : Socket)

Parameters:

readerName: Reader name, string.

portType: Communication port type, string.

connStr: Connection string.

Server: Reader connection socket, used in active mode, refers to 3.2.

Parameter description:

Customizable reader name, but unable to replicate.

The communication port definition is as followed:

portType	Explanation
RS232	RS-232 serial port
TCPIP_Client	TCP/IP protol client

Connection string rules are as follows:

The example of serial port string are: “COM1,115200”, with “COM1” as the name of the serial port, “115200” as baud rate, separated by commas in the middle. The common baud rates are “115200”, “19200” and “9600”.

The example of TCP/IP protocol client string are: “192.168.0.210:7086”, with “192.168.0.210” as IP, “7086” as serial port, separated by commas in the middle.

Constructor explanation:

Constructor 1: This constructor must be used together with configuration files (refers to 4.5).

Constructor 2: This constructor should not be used with configuration file.

Constructor 3: When host computer as client, use the connected socket transmission as backup.

2. Establish connection (Connect method)

Purpose: Establish communication connection with reader through designated port.

Syntax: + Connect() : Boolean

Returns:

Check if connection is successful, Boolean.

Remark: During client software as server, despite the transmission is done through connected socket, adjustment is still needed to use Reader type normally.

3. Disconnect connection (Disconnect method)

Purpose: Disconnect ports that have established connection.

Syntax: + Disconnect()

4. Send message (Send method)

Purpose: Send message.

Syntax: + Send(msg : IMessage) : Boolean

+ Send(msg : IMessage, timeout : Integer) : Boolean

Parameter:

msg: Pending sending message type, IMessage type, note IMessage as message interface, all types that can realize the connection can be used as parameters.

timeout: timeout, integer, unit as milliseconds

Returns:

Check if connection is successful, Boolean.

Note:

- For message that requires response (see IMessage interface IsReturn member), messages have to be sent and successful execution message has to be received to be considered successful, otherwise the process is a failure.
- For message that does nto requires respond (see IMessage interface IsReturn member), the byte number sent has to be equal to the original byte number to be returned successful.
- Method with no timeout parameter, the default timeout is 1000 miliseconds.

5. API Expectation Notification (OnApiException event)

Purpose: During the abnormal operation of API, notification is done uniformly through this event.

Syntax: + OnApiException: ApiExceptionHandle

Remark:

This event call backs abnormal message through the realization of ApiExceptionHandle delegate. Most of the API abnormality will be notified through here.

Figure 4-3 ApiExceptionHandle delegate

Parameter e is ErrInfo type. ErrInfo type includes wrong reader name (ReaderName) and wrong message information (ErrMsg).

6. Message notification (OnMessageNotificationReceived event)

Purpose: Receive reader message

Syntax: + OnMessageNotificationReceived

:MessageNotificationReceivedHandle

Remark: This event call backs reader message through the realization of MessageNotificationReceivedHandle delegate.

Figure 4-4 MessageNotificationReceivedHandle delegate

Parameter msg is IMessageNotification type, for details refer to the explanation in 4.2.

7. Connection status icon properties (IsConnected member)

Purpose: Get the connection status label.

Syntax: + IsConnected : Boolean

Remark: This member is Boolean. Note that the abnormal disconnection of the reader will not be immediately detected by API during usage.

8. Reader type properties (ModelNumber member)

Purpose: Get reader model.

Syntax: + ModelNumber : String

Remark: String type. Initial value is “unknown”, the member only has specific value upon successful connection.

9. Reader name properties (ReaderName member)

Purpose: Get reader name.

Syntax: + ReaderName : String

Remark: String type. readerName parameter in the same constructor.

10. Reader group properties (ReaderGroup member)

Purpose: Get the group name of reader group.

Syntax: + ReaderGroup : String

Remark: String type. Customizable string. The grouping information in the configuration file is used to allow user’s reader categorization.

11. ProtocolVersion properties (ProtocolVersion member)

Purpose: Get communication protocol and version information.

Syntax: + ProtocolVersion : String

Remark: String type. “IRP1” refers to Invengo protocol version 1.0.

12. Communication port type properties (PortType member)

Purpose: Get communication port name.

Syntax: + PortType : String

Remark: String type. portType parameter in the same constructor.

4.2 Reader message

Reader message: The automatic message sent by the reader. The message type must realize IMessageNotification interface.

The message includes:

1) The automatic, non-responsive message sent by the reader, such as during the input message change in the reader I/O port, the reader automatically sends message to inform the host computer.

2) Response from the reader, but not one-to-one response, such as returning tag scanning information.

The message utilization flow of the reader is shown in the figure below: (Assuming the reader message msg has passed through OnMessageNotificationReceived evnent callback)

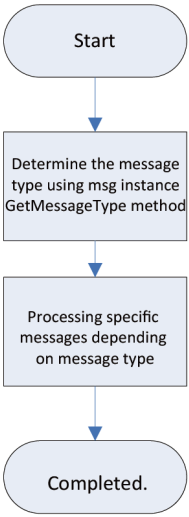


Figure 4-5 Reader message utilization flow

Note:

1) Message status can be confirmed using StatusCode of the msg, before further processing; if the StatusCode is not 0, ErrInfo of the msg can be used to display error information.

2) Message callback and message sending are on different flow, please pay attention during the synchoronization processing of of messages.

Reader message member and members are as follows:

1. Get current type's type name

Purpose: Get the current type's type name. Note that if the current type is connected through connector, the name geted is derived type.

Syntax: + GetMessageType() : String

2. Return value: Type name string.

Remark: During actual application, use GetMessageType method to get the type name of the event or callback parameters in a OnMessageNotificationReceived event or callback, to determine the returned reader message type.

3. Get message return status code (StatusCode member)

Purpose: Defined interface, this interface is used to realize the return of message received to status code, integer.

Syntax: + StatusCode : Integer

Remark: Under IRP1.0 protocol, status code of 0 indicates success, non 0 indicates correspindig error code. Please refer to the appendices for error clode.

4. Get error information in the message (ErrInfo member)

Purpose: Defined interface, this interface is used to realize the error information in the message, string type.

Syntax: + ErrInfo : String

4.3 Host computer message

Host computer message: The automatic message sent by the host computer. This type of message must realize IMessage interface (IMessage succeeds IMessageNotification).

The messages include:

1) Message sent by host computer that requires reader's response;

2) Message sent by host computer that does not require reader's response;

3) Host computer responding to reader's message;

The message utilization flow by the host computer is shown in the figure below: (Assuming the reader type hass been instantized, and connection has been successfully established)

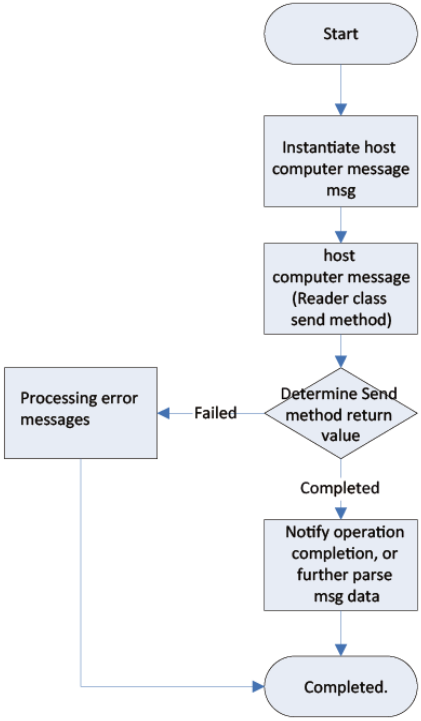


Figure 4-6 Host computer message utilization flow

Note:

1) For failed message sending, error information can be retrieved through ErrInfo instantized by msg;

2) For successful message sending, data frame can be geted through ReceivedData instantized by msg, and analysis can be performed based on specific condition. Some message portions have no message for further analysis (the returned data frame does not contain actual data content), here the concern is focused on the Send method return value.

3) Not recommend to use instantized message multiple time as the status of the message type will change after responding once.

The host computer message method and members are as follows:

1. Get or configure whether the message needs to wait for respond (Is Return member)

   Purpose: Defined interface, this interface is used to realize the geting or configuration of whether the message needs to wait for respond, Boolean.

   Syntax: + IsReturn : Boolean

   Remark: Message that does not require returning can be seen as reader information.

2. Configure message timeout time (Timeout member) Purpose: Defined interface, this interface is used to realize the configuration of timeout time, integer, unit is milliseconds.

   Syntax: + Timeout: Integer

   Remark:

   Only applicable when the IsReturn time has exceeded the timeout time. When the timeout time is working, host computer will wait for the response to the message within the time frame after sending off the message. If there is no response, it will be considered as timeout.

4.4 6C tag data area enum(MemoryBank)

Designate 6C tag various data area.

Member name	Enum value	Explanation
ReservedMemory	0	Reserve
EPCMemory	1	EPC data area
TIDMemory	2	TID data area
UserMemory	3	User data area

4.5 Configuration fle(Sysit.xml)

Configuration file is optional, but it is compulsory for multi-reader management. The file name is “Sysit.cfg”, xml document and same directory as API.

Configuration file exists as reader node, the basic content includes:

1. Reader name, group (Reader name is not replicable)
2. Enable/disable. Enable means the reader is ready for connection.
3. Connection port type.
4. Connection protocol. Designte the protocol used by the reader.
5. Connection string.

The configuration file format is shown in the figure below:

```
<?xml version="1.0" encoding="utf-8" ?>
<Readers>
  <Reader Name="Reader1" Group="Group1" Enable="true">
    <Port Type="RS232" Protocol="IRP2">COM1,115200</Port>
  </Reader>
</Readers>
```

Figure4-7 Configuration file format

Among them, “Readers” is the root node.

“Reader” node is the reader configuration node. “Name” porperties is the reader name, “Group” properties is the reader grouyp, the name and group are both customizable string, and the name should be unique; “Enable” means enable/disable reader, the value is “true/false”, when configured as disable, API will not establish connection.

“Port” node is the communication port node, this content of the node is string. “Type” properties refer to the name of the port, “Protocol” properties refer to the communication protocol, currently only supports “IRP1”.

4.6 Antenna port definition

4.6.1 Tag reading/writing operation

When the tag is writing, it uses the actual antenna port no. (i.e. 01H, 02H, 03H, 04H).

4.6.2 Tag scanning operaiton

Scan operation means tag idenfificaiton, similar to the ReadTag type operation below.

During returning response to the host computer the reader will indicates the actual antenna port used to complete the operation (i.e. 01H, 02H, 03H, 04H). However during the scanning process, the antenna number used will be different, as shown below:

1. Use the following domain

ID_6B,

ID_UserData_6B,

EPC_6C_ID_6B,

TID_6C_ID_6B,

EPC_TID_UserData_6C_ID_UserData_6B during scanning

Configure antenna port before scanning (see SysConfig_500 type).

During ReadTag type scanning, designate the antenna port used by the reader, with the following definition:

00H: Multi-antenna (All the antennas after configuration)

01H: Antenna 1

02H: Antenna 2

03H: Antenna 3

04H: Antenna 4

2. During the scanning of other domains,

Designate the antenna port used by the reader, with the bit definition as follows:

Bit	7	6	5	4	3	2	1	0
Definication	1	0	0	0	Antenna 4	Antenna 3	Antenna 2	Antenna 1

Bit3: Enable flag of antenna 4, 1: Allow; 0: Forbidden

Bit2: Enable flag of antenna 3, 1: Allow; 0: Forbidden

Bit1: Enable flag of antenna 2, 1: Allow; 0: Forbidden

Bit0: Enable flag of antenna 1, 1: Allow; 0: Forbidden

5 Reader reference query and configuration

5.1 Reader system parameter query (SysQuery_800 type)

Purpose: Inquire reader system parameters.

1. Constructor

C#: public SysQuery_800(Byte parameter)
public SysQuery_800(Byte parameter, Byte data)

JAVA: public SysQuery_800(byte parameter)
public SysQuery_800(byte parameter,byte data)

C++: SysQuery_800(unsigned char parameter)
SysQuery_800(unsigned char parameter, unsigned char data)

Parameter:

parameter: Inquire parameter.

data: Currently only use in the query of I/O trigger parameter operation, indicates I/O port number (01H – 04H).

2. Command response (ReceivedMessage)

Inquire data

Purpose: Get data queried.

C#: public Byte[] QueryData { get; }

JAVA: public byte[] getQueryData()

C++: bool GetQueryData(unsigned char *data,unsigned char &dataLen)

3. Command description

Please refer to 5.3 for details.

5.2 Reader system parameter configuration (SysConfig_800 type)

Purpose: Configure reader system parameters.

1. Constructor:

C#: public SysConfig_800(Byte parameter, Byte[] pData)

JAVA: public SysConfig_800(byte parameter, byte[] pData)

C++: SysConfig_800(
unsigned char parameter,
unsigned char *pData,
unsigned char pDataLen)

Parameter:

parameter: Configuration parameter, please refer to 5.3 for details.
pData: Configuration parameter, please refer to 5.3 for details.
pData Len: *pData parameter length

2. Command response(ReceivedMessage)

There is no data that can be further analyzed, ReceivedMessage is empty.

3. Command description

Please refer to 5.3 for details.

5.3 Reader system parameter explanation

Function query and configuration comparison chart as shown below:

Code	Parameter type	Remark
03H	Antenna port power	Supports query, configuration
05H	MAC address	Supports query
06H	Ethernet configuration	Supports query, configuration
07H	SOCKET port number	Supports query, configuration
0DH	Serial port baud rate	Supports query, configuration
21H	Reader product model	Supports query
23H	Reader processing software version number	Supports query
29H	Application processor hardware version	Supports query
50H	Reader client mode switch	Supports query, configuration
51H	Server address	Supports query, configuration
52H	Largest connection failure number	Supports query, configuration
6DH	Read tag mode	Supports query, configuration
E2H	Query I/O trigger parameter	Supports query, configuration

5.3.1 Reader version information

The reader version information contains three contents: the reader model, the baseband processor hardware version, application processor hardware version.

1. Reader version

Parameter: 21H, this parameter can only be queried and is not configurable.

Data query: String that is supposed to be changed to ASII format, the basic format is model numerical code+area code, the last alphabet in the string is the area code, indicating the local regulation code that the reader is in compliance with, “A” refers to compliance with FCC related requirement; “C” refers to compliance with State Radio Regulation of China related requirement; “E” refers to compliance with ETSI related requirement. The code for XC-RF807 reader that can be sold in China is “807C”. The string mentioned in this article does not contain terminator \0, this applies to the rest of the article).

2. Baseband processor hardware version

Parameter: 23H, this parameter can only be queried and is not configurable.

Data query: String that is supposed to be changed to ASII format, the basic format is“*.**”, version V1.00 indicates “1.00”.

3. Application processor hardware version

Parameter: 29H, this parameter can only be queried and is not configurable.

Data query: String that is supposed to be changed to ASII format, the basic format is “Reader model number SVN version number –RELEASE”. For example the current version number for XC-RF807 application processor hardware will be “XC-RF807svn236-RELEASE”.

5.3.2 Commucation parameter configuration

1. Serial port parameter

Parameter: 0DH, baud rate query or configuration.

Data: 0, 9600bps;1, 19200bps; 2, 115200bps.Default baud rate 115200bps

2. Eterhent communication parameter

Ethernet communication parameter includes MAC address, IP address, reader TPC connection mode configuration and related parameters.

The eternet communication (based on TCP/IP protocol, only suppprts IPV4) provided by the reader includes reader as a server (also known as passive mode) and reader as client (also known as active mode). At any point of time, the reader can only be one of the two modes. Reader as server mode refers to the external device (such as PC) actively initiates network connection with the reader. Reader as client mode refers to the reader actively initiates network communication with the external device. The factory default IP of the reader is 192.168.0.210. Please ensure the reader’s IP and PC’s IP is at the same network segment during network connection.

Reader as client is a working mode where the reader actively connects with the remote server under configured parameters. The active mode and backup server are used together, once the primary server fails, the reader can automatically connect to the backup server.

1) MAC address

Parameter: 05H, this parameter can only be queried and is not configurable.

Data: 6 byte length, hexadecimal format MAC address, high priority, for example MAC address for {0x00, 0x1D, 0x78, 0x12, 0x34, 0x56} will be 00-1D-78-12-34-46.

2) IP address

Parameter: 06H

Data: 12 byte length, format: 4 byte+4 byte subnet mask+4 byte gateway, high priority. The reader default IP: 192.168.0.210, subnet mask: 255.255.255.0, gateway: 192.168.0.1. To configure the reader IP, the configuration IP command data is: C0 A8 00 D2 FF FF FF 00 C0 A8 00 01.

3) Reader TCP connect mode configuration

Parameter: 50H

Data: 1 byte length, 0, indicates the reader as TCP server, 1, indicates the reader as TCP client. Reader default connection method is reader as TCP server.

4) Reader TCP service port number

Parameter: 07H

Data: 2 byte length, high priority, server port number for the configuration of reader as TCP server. The default for this parameter is 7086. This parameter is only meaningful if the reader configuration is in server mode.

5) Backup server configuration

Parameter: 51H

Data: 18 byte length. High priority. Configurations of three backup servers are allowed. The first backup server in the parameter refers to the primary working server, the other two are the backup servers. The IP and the port number of the first server in the backup server parameter should not be configured as 0. The parameter of the second server (or the third server) in the backup server parameter should be configured as 0, meaning the server is not enabled as backup server. This parameter is only meaningful when the reader is configured as client mode. Data format: Server IP1 (4 byte)+Port 1 (2 byte)+Server IP2 (4 byte)+Port 2 (2 byte)+Server IP3 (4 byte_+Port 3 (2 byte)

6) Number of connection retry

Parameter: 52H

Data: 2 byte length, high priorty, largest number of connection failure indicates the largest allowable number of connection failure when the reader connects with remote server. Number of failure starts from zero, with every connection failure, 1 is added. After successful connection, the number of failure returns to zero. When the number of failure accumulates to the largest number of connection failure, the reader will automatically connect to the next server. When the remote server actively disconnects, no failure is registered. For every

connection failure, a new connection is started at the interval of 2 second. When the number of connection failure to the final server is larger than the largest number of connection failure, the reader begins to reconnect with primary server (based on the sequence of backup server 1 – backup server 2 – backup server 3 – backup server 1…). When the largest number of connection failure is configured as 0, the reader will not connect to the subsequent backup server. The default configuration of this parameter is 0 and is only meaningful when the reader configuration is under client mode.

5.3.3 RFID parameter configuration

1. Reader antenna port power configuration

Parameter: 03H

Data (configuration): 2 byte length, antenna port number (0-3 or FFH)+power level. The first byte indicates the antenna port number that is to be configured, the antenna number range is 0-3, indicating antenna 1, 2, 3 and 4, When the antenna port number is FFH, this means all antenna port will be configured to the same power level parameter; the second byte refers to the level parameters of the power that is to be configured. The corresponding transmit power dBm of the power level is based on the hardware design of the reader model. The specific corresponding relationship is shown below:

Reader mode	Power level range	Actual transmit power and stepping
XC-RF807	0~25	11～36dBm, stepping 1dB
XC-RF850	0~25	9～34dBm, stepping 1dB
XC-RF861	0~25	9～34dBm, stepping 1dB
XC-RM825	0~36	0～36dBm, stepping 1dB
XC-RM829	0~36	0～36dBm, stepping 1dB
XC-RF812	0~36	0～36dBm, stepping 1dB
XC2903	0~36	0～36dBm, stepping 1dB

Data (Query): When the host computer inquires the antenna port power configuration, the reader will return all the antenna port power. 4 byte length, each representing the power level of antenna 1-4.

2. Read tag mode

Parameter: 6DH

Data: 1 byte length, 00H indicates multitag mode, 02H indicates singletag mode.

5.3.4 GPI trigger parameter configuration

Parameter: E2H

Data: Relevant configuration used in the GP event trigger mechanism. Invengo reader GPIO realizes GPI event trigger mechanism on the basis of basic input/output function, the adoption of GPI event trigger mechanism allows easy linkage with other equipment. The basic principle of GPI event trigger mechanism: Tie up GPI input status with relevant operation of relevant equipment, different GPI input status can tringger reader operation. Each GPU port can only be tied up with one pair of reader command, and each GPI port can be tied up individually. This function is generally used in the GPI triggered RFID tag reading operation. The reader will automatically stop tag read when the termination condition is fulfilled.

IO trigger data structure is as follows:

GPI port no.	Trigger condition	Termination condition	Delay time	Command tied with trigger condition	Command tied with termination condition
1～4H	0~4H	0 or 1	(2 byte)	(N byte)	(N byte)

GPI port number: Designate the GPI port number that needs to be configured

Trigger condition: GPI input status for designated trigger event. indicates the switch off of current port GPI tigger function, 1 indicates the adoption of GPI ascending edge trigger, 2 indicates the adoption of GPI descending edge trigger, 3 indicates GPI high level trigger, 4 indicates GPI low level trigger. The difference between level trigger and edge trigger: During power up after a power down, level trigger will automatically execute trigger action based on the current IO status in comparison to the trigger condition, while edge trigger requires level transition to execute trigger.

Termination condition: 0 indicates the reserve adoption of GP status as termination status (opposite of trigger condition, ascending edge corresponds to descending edge, high level corresponds with low level), 1 indicates the adoption of delay time as termination condition.

Delay time: 16 bits unsigned integer, high priority, use tod designate the delay time as the termination condition

Command tied with trigger condition and termination condition; designated corresponding reader operation command, complete IRP1 command, with length not exceeding 128 byte. Empty command indicates by {0x0, 0x0}. Note: During the tied up of switch off command, PowerOff_800 type should be used instead of PowerOff type.

Steps tp get IRP1 command data:

1) Instantize one IPR command type (such as ReadTag, PowerOff_800 and etc.)

2) Configure the PortType member properties of the command type as “”, with empty string.

3) Get command data through the application of TransmitterData of the command type.

The designated GPI port number in the reserve area must be filled in during the GPI trigger parameter query.

5.4 Inquire reader system parameter (SysQuery_500 type)

Purpose: Inquire reader system parameter

1. Constructor

C#: public SysQuery_500(Byte infoType, Byte infoLength)

JAVA: public SysQuery_500(byte infoType, byte infoLength)

C++: SysQuery_500(
unsigned char infoType,
unsigned char infoLength)

Parameter:
infoType: Information type.
infoLength: Return parameter length, in bytes.

2. Command response (ReceivedMessage)

1) Data query

Purpose: Get inquired data

C#: public Byte[] QueryData { get; }

JAVA: public byte[] getQueryData()

C++: bool GetQueryData(unsigned char *data,
unsigned char &dataLen)

3. Command description

Information type:

Code	Parameter type	Information legth
02H	Antenna port	1 byte

Definition of antenna port is as follows:

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
Reserved	Reserved	Reserved	Reserved	Port 4	Port 3	Port 2	Port 1

Bit3: Enable flag of antenna 4, 1: Allow; 0: Forbidden

Bit2: Enable flag of antenna 3, 1: Allow; 0: Forbidden

Bit1: Enable flag of antenna 2, 1: Allow; 0: Forbidden

Bit0: Enable flag of antenna 1, 1: Allow; 0: Forbidden

Note: Only in the scan region (see 6.2.1) where ID_6B, ID_UserData_6B, EPC_6C_ID_6B, EPC_TID_UserData_6C_ID_UserData_6B, TID_6C_ID_6B, configuration is required before tag reading. The information will be retained for subsequent alteration or when the reader is power down.

5.5 Configure reader system parameter (SysConfig_500 type)

Purpose: Configure reader system parameter; Constructor

C#: public SysConfig_500(
Byte infoType,
Byte infoLength,
Byte[] pData)

JAVA: public SysConfig_500(
byte infoType,
byte infoLength,
byte[] pData)

C++: SysConfig_500(
unsigned char infoType,
unsigned char infoLength,
unsigned char *pData,
unsigned char pDataLen)

Parameter:

infoType: Information type.

infoLength: Information length, in bytes.

pData: Configure data.

pDataLen: *pData parameter length.

1. Command response(ReceivedMessage)

This message has no data for further analysis, ReceivedMessage is empty.

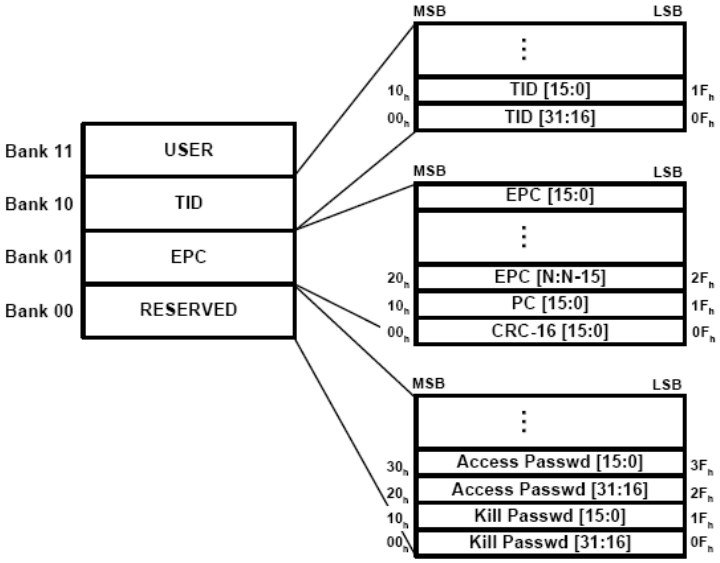
2. Command description

Information type and information length is similar with aobove, data configuration is similar with data query in the above.

6 Electronic tag identification and access

6.1 ISO18000-6C(EPC C1G2) electronic tag storage structure

Theoreotically, the storage area of ISO18000-6C(EPC C1G2) electronic tag (hereinafter known as 6C tag) is divided into four different storage area, each storage area is consisted of 1 storage word (16 bits), the smallest unit of bitwidth in 6C tag storage reading and writing access is 16 bitss words, with the logical storage mappin g shown in the figure below.



The specific definitions of these storage areas are:

1. **RESERVED** area, storage code is 00’b. Use in the storage of kill and access function password. Kill password is stored in sorage aaddress 00h to 1Fh, access password is stored at 20h to 3Fh.

2. **EPC** storage area. Store CRC-16 in address 00h to 0Fh, store protocol control bit (PC) in address 10h to 1Fh, address 20h and larger is used to store identification tag code that has image attached (i.e. EPC code, this code will be known as EPC code from here onwards). Subdivision is performed on PC, address 10h to 14h is for the storage of EPC code lenth, 15h to 17h is

for the storage of RFU, 18h to 1FH is for the storage of numbering system identifier (NSI). 16, PC and EPC is stored with high priority (EPC code high priority storage is at 20h).

3. **TID** storage area. 8 bits ISO/IEC 15693 assigned identification code is stored in 00h to 07h (EPCglobal code is 11100010'b). TID storage in address 07h and above has sufficient identification message, allowing the reader to uniquely identify custom commands and (or) selected commands supported by the tag. For tag with ISO/IEC 15693 assigned identification code of 11100010'b, these identification information will constitute a 12 bits long tag mask designer identifier (EPCglobal organization assigned vendor number), the tag mask designer identifier is stored in 08h to 13h, model code is stored in 14h to 1Fh. Tag can also store special data of the tag and vendor in TID area address larger than 1Fh (typically is the sequence number of the tag).

4. **USER** storage area. Allow the storage of user special data. The area's storage organizational structure is defined by the user.

For relevant detailed information, please refer to information [1].

6.2 Tag data reading

6.2.1 Tag scanning (ReadTag type)

Purpose: Construct tag scanning command.

1. Constructor

C#: public ReadTag(ReadMemoryBank rmb)

```
public ReadTag(
        ReadMemoryBank rmb,
        Int32 readTime,
        Int32 stopTime)
public ReadTag(
        ReadMemoryBank rmb,
        Boolean isGetOneTag)
```

JAVA: public ReadTag(ReadMemoryBank rmb)

```
public ReadTag(
        ReadMemoryBank rmb,
        int readTime,
        Int stopTime)
public ReadTag(
        ReadMemoryBank rmb,
        boolean isGetOneTag)
```

C++: ReadTag(ReadMemoryBank rmb)

```
ReadTag(ReadMemoryBank rmb,
        int readTime,
        int stopTime)
ReadTag(ReadMemoryBank rmb,
        BOOL isGetOneTag)
```

parameter:

rmb: Scan area, refers to third point of this section for details.

readTime: Scan time, integer, in milliseconds.

stopTime: Stop time, integer, in milliseconds.

isGetOneTag: Whether to return with 1 tag, Boorlean. True, scan 1 tag and

return; false, return all scanned tag data within timeout time.

Explanation:

1) The scan time of the second constructor indicates switch on time, when the designated scan time is reached (readTime), it will automatically adjust to switch off command, stopping the designated time (stopTime) before continue scanning and the cycle continues until corresponding stop operation is sent (refers to PowerOff). This processing mechanism is specifically used for long operation conditions.

2) Third constructor forces data information return or waits for timeout, greatly simplifying the operational flow of certain scenario..

2. Member/Properties

Purpose of the member/properties is to alter the various configurations of tag scanning.

1) Antenna port

Purpose: Configurre antenna port. Default value:01H.

C#: public Byte Antenna { set; }

JAVA: public void setAntenna(byte antenna)

C++: void Antenna(unsigned char ucAntenna)

Supported scan area: All value of ReadMemoryBank

Note: Refer to 4.6 for antenna port usage details.

2) Q value

Purpose: Configure Q value. Value rane: 0-15, Default value: 00H.

C#: public Byte Q { set; }

JAVA: public void setQ(byte q)

C++: void SetQ (unsigned char ucQ)

Supported scan area: Apart from ID_6B and ID_UserData_6B, all value of ReadMemoryBank.

3) Reading method

Purpose: Configure loop reading. True for loop reading, false for single reading. Default value: True.

C#: public Boolean IsLoop { set; }

JAVA: public void setLoop(boolean isLoop)

C++: void SetIsLoop (unsigned char uc IsLoop)

Supported scan area: All value of ReadMemoryBank.

4) TID length

Purpose: Configure TID length, in bytes. Default value: 04H.

C#: public Byte TidLen { set; }

JAVA: public void setTidLen(byte tidLen)

C++: public void SetTidLen (unsigned char ucTidLen)

Supported scan area:

- EPC_TID_UserData_6C_2
- EPC_TID_UserData_Reserved_6C_ID_UserData_6B

5) 6C tag user start address

Purpose: Configure 6C label user start address, in words. Default value: 0.

C#: public UInt32 UserDataPtr_6C { set; }

JAVA: public void setUserDataPtr_6C(byte userDataPtr)

C++: void SetUserdataPtr_6C(unsigned char ucUserdataPtr_6C)

Supported scan area:

- EPC_TID_UserData_6C_2
- EPC_TID_UserData_Reserved_6C_ID_UserData_6B

6) 6C tag user area length

Purpose: Configure 6C tag user area length, in word. Default value: 0EH.

C#: public Byte UserDataLen_6C { set; }

JAVA: public void setUserDataLen_6C(byte userDataLen)

C++: void SetUserDataLen_6C (unsigned char ucUserDataLen_6C)

Supported scan area:

- EPC_TID_UserData_6C_2
- EPC_TID_UserData_Reserved_6C_ID_UserData_6B

7) 6B tag user area start address

Purpose: Configure 6B tag user area start address, in bytes. Default value: 00H.

C#: public Byte UserDataPtr_6B { set; }

JAVA: public void setUserDataPtr_6B(byte userDataPtr)

C++: void SetUserdataPtr_6B (unsigned char ucUserDataLen_6C)

Supported scan area:

- EPC_TID_UserData_Reserved_6C_ID_UserData_6B

8) 6B tag user area length

Purpose: Configure 6B tag user area length, in bytes. Default value: 08H.

C#: public Byte UserDataLen_6B { set; }

JAVA: public void setUserDataLen_6B(byte userDatLen)

C++: void SetUserDataLen_6B (unsigned char ucUserDataLen_6B)

Supported scan area:

- EPC_TID_UserData_Resered_6C_ID_UserData_6B

9) Reserve area length

Purpose: Configure reserve area length, in word. Default value: 00H.

C#: public Byte ReservedLen { set; }

JAVA: public void setReservedLen(byte reservedLen)

C++: void SetReservedLen (unsigned char ucReservedLen)

Supported scan area:

- EPC_TID_UserData_Reserved_6C_ID_UserData_6B

10) Access password

Purpose: Configure access password, 4 bytes. Default value: all 0.

C#: public Byte[] AccessPwd { set; }

JAVA: public void setAccessPwd(byte[] accessPwd)

C++: void SetAccessPwd (unsigned char* ucAccessPwd)

Supported scan area:

- EPC_TID_UserData_Reserved_6C_ID_UserData_6B

11) 6C tag periodic reading requency

Purpose: Configure 6C tag reading frequency in one reading cycle. Default value: 01H.

C#: public Byte ReadTimes_6C { set; }

JAVA: public void setReadTimes_6C(byte readTimes)

C++: void SetReadTimes_6C(unsigned char ucReadTimes_6C)

Supported scan area:

- EPC_TID_UserData_Reserved_6C_ID_UserData_6B

12) 6B tag periodic reading frequency

Purpose: Configure 6B tag reading frequency in one reading cycle. Default value: 01H.

C#: public Byte ReadTimes_6B { set; }

JAVA: public void setReadTimes_6B(byte readTimes)

C++: public void SetReadTimes_6B(unsigned char ucReadTimes_6B)

Supported scan area:

- EPC_TID_UserData_Reserved_6C_ID_UserData_6B

3. Enumerate(ReadMemoryBank)

1) EPC_6C

Purpose: Scan 6C tag EPC data. This function can be used to read tag EPC code, also known as inventory. EPC code can be used to identify tracking object during actual application, its function is similar to the barcode in commercial products, and EPC code is the easiest tag data that can be geted.

2) TID_6C

Purpose: Scan 6C tag TID data. This function is used to identify tag TID code. Currently the tag TID codes used by mainstream vendors can't be rewritten and the codes are unique, therefore TID codes are mostly used for unique identification during application. The TID code length of different tag chip model may varies; this function is applicable to adaptive TID code with the length between 32 to 96 bits.

3) EPC_TID_UserData_6C

Purpose: Scan 6C tag EPC, TID and user data.

4) EPC_TID_UserData_6C_2

Purpose: Scan 6C tag EPC, TID and user data. This function can be used to get

the EPC code, TID code and user area data of a tag in one reading. When using this unfciton, the TID code length should not have adaptive features.

5) ID_6B

Purpose: Scan 6B tag ID data.

6) ID_UserData_6B

Purpose: Scan 6B tag ID and user data.

7) EPC_6C_ID_6B

Purpose: Scan 6C tag EPC data and 6B tag ID data.

8) TID_6C_ID_6B

Purpose: Scan 6C tag TID data and 6B tag ID data.

9) EPC_TID_UserData_6C_ID_UserData_6B

Purpose: Scan 6C tag EPC, TID, user data and 6B tag ID and user data.

10) EPC_TID_UserData_Reserved_6C_ID_UserData_6B

Purpose: Scan 6C tag EPC, TID, user area, and reserved area data and 6B tag ID and user data.

11) EPC_PC_6C

Purpose: Scan 6C tag EPC and PC data.

4. Command response(ReceivedMessage)

Only the third constructor can respond to command.

1) Tag data

Purpose: Get tag data information.

C#: public RXD_TagData[] List_RXD_ TagData { get; }

JAVA: public RXD_TagData [] getList_RXD_TagData ()

C++: RXD_TagData * List_RXD_TagData (unsigned int &len);

Remark: C++ response lens start value 0, cycle call this function until the return List_ RXD_TagData target pointer is NULL

6.2.2 Receive tag data information (RXD_TagData type)

Purpose: Receive tag data information. Succeed from BaseMessageNotification abstract type.

1. Command response(ReceivedMessage)

1) Reader name

Purpose: Get reader name.

C#: public String ReaderName { get; }

JAVA: public string getReaderName (){ }

C++: CString GetReaderName ()

2) Tag type.

Purpose: Get tag type, the value could be “6C”, “6B”.

C#: public String TagType { get; }

JAVA: public string getTagType (){ }

C++: CString GetTagType ()

3) Antenna port

Purpose: Get antenna port

C#: public Byte Antenna { get; }

JAVA: public byte getAntenna(){ }

C++: unsigned char GetAntenna()

4) Tag EPC data

Purpose: Get tag EPC data. Empty for no data.

C#: public Byte[] EPC { get; }

JAVA: public byte[] getEPC(){ }

C++: bool GetEPC(unsigned char *Epc,unsigned char &epcLen)

5) Tag TID/ID data Purpose: Get 6C tag TID data or 6B tag ID data. Empty for no data.

C#: public Byte[] TID { get; }

JAVA: public byte[] getTID(){ }

C++: bool GetTID(unsigned char *tid,unsigned char &tidLen)

6) Tag user area data

Purpose: Get 6C or 6B tag user area data. Empty for no data.

C#: public Byte[] UserData { get; }

JAVA: public byte[] getUserData(){ }

C++: bool GetUserData(unsigned char *userdata,unsigned char &userdataLen)

7) Tag reserved area data

Purpose: Get tag reserved area data.

C#: public Byte[] Reserved { get; }

JAVA: public byte[] getReserved { }

C++: bool GetReserved (unsigned char *pReserved,unsigned char &ReservedLen)

Explaantion: Reserved area includes access password and kill password. For details refer to 6.1.

8) Tag received signal strength indication

Purpose: Get tag received signal strength indication. Empty for no data.

C#: public Byte[] RSSI { get; }

JAVA: public byte[] getRSSI(){ }

C++: bool GetRSSI(unsigned char *Epc,unsigned char &epcLen)

9) Tag reading time

Purpose: Get tag reading time. Empyt for no data.

C#: public Byte[] RXDTime{ get; }

JAVA: public byte[] getRXDTime (){ }

C++: bool GetRXDTime (unsigned char *time,unsigned char &timeLen)

Remark: If the reading time is in 8 bytes, it indicates UTC time, first 4 bytes are second, last 4 bytes are milisecnonds. If the reading time is in 6 bytes, it indicates BCD code time, the 6 bytes refer to year, month, date, hour, minute and second respectively.

6.2.3 6C tag reading user data area (ReadUserData_6C type)

Purpose: Read tag user data area. This function is used to get tag USER area data, usually select the function combination with the tag to read designated tag user data area content. This function can only be used for single tag single operation.

1. Constructor

C#: public ReadUserData_6C(
Byte antenna,
UInt32 ptr,
Byte length)
public ReadUserData_6C(
Byte antenna,
UInt32 ptr,
Byte length,
Byte[] tagID,
MemoryBank tagIDType)

JAVA: public ReadUserData_6C(
byte antenna,
int ptr,
byte length)
public ReadUserData_6C(
byte antenna,
int ptr,
byte length,
byte[] tagID,
MemoryBank tagIDType)

C++: ReadUserData_6C(
unsigned char antenna,
unsigned char ptr,
unsigned char length)
ReadUserData_6C(
unsigned char antenna,
unsigned char ptr,

unsigned char length,
unsigned char *tagID,
unsigned char tagIDLen,
unsigned char tagIDType)

parameter:

antenna: Antenna port

ptr: Start address, in word.

length: Read length, in word.

tagID: Designate operational tag EPC or TID.

tagIDType: tagID corresponding data area enumeration.

tagIDLen: *tagIDparameter length.

2. Command response(ReceivedMessage)

1) Antenna port

Purpose: Get antenna port.

C#: public Byte Antenna { get; }

JAVA: public byte getAntenna(){ }

C++: unsigned char GetAntenna()

2) Tag user area data

Purpose: Get tag user area data.

C#: public Byte[] UserData { get; }

JAVA: public byte[] getUserData() { }

C++: bool GetUserData(unsigned char *usderData,unsigned char &userDataLen)

6.3 Tag data writing

Tag writing function includes: EPC code writing, USER area data writing, tag access password writing, tag kill password writing, common writing function, general tag writing function and tag selection function. All these tag writing function can only be used in single tag single operation.

6.3.1 6C tag write EPC (WriteEpc type)

Purpose: Write tag EPC data area. This function can be used to alter the EPC code data in the tag. As the tag EPC code length is controlled by the PC (Protocol Control) of the EPC data area, the reader will automatically calculate the PC value based on the EPC code content length that is intended to be written and perform rewriting.

1. Constructor

C#: public WriteEpc(
Byte antenna,
Byte[] accessPwd,
Byte[] epcData)
public WriteEpc(
Byte antenna,
Byte[] accessPwd,
Byte[] epcData,
Byte[] tagID,
MemoryBank tagIDType)

JAVA: public WriteEpc(
byte antenna,
byte[] accessPwd,
byte[] epcData)
public WriteEpc(
byte antenna,
byte[] accessPwd,
byte[] epcData,
byte[] tagID,
MemoryBank tagIDType)

C++: WriteEpc(
unsigned char antenna,
unsigned char *accessPwd,
unsigned char accessPwdLen,
unsigned char *epcData,
unsigned char epcDataLen)

WriteEpc(
unsigned char antenna,
unsigned char *accessPwd,
unsigned char accessPwdLen,
unsigned char *epcData,
unsigned char epcDataLen,
unsigned char *tagID,
unsigned char tagIDLen,
unsigned char tagIDType)

parameter:

antenna: Antenna port.

accessPwd: Access password, 4 bytes.

epcData: Designate the EPC data to be written.

tagID: Designate the operational tag EPC or TID

tagIDType: tagID corresponding data area enumeration.

accessPwdLen: *accessPwdparameter length

epcDataLen: *epcDataparameter length

tagIDLen: *tagIDparameter length

2. Command response(ReceivedMessage)

This message has no data for further analysis. ReceivedMessage is empty.

3. Command description

Tag access password: 32 bits unsigned integer, high priority, used for password verification before write operation.

6.3.2 6C tag write user data area (WriteUserData_6C type)

Purpose: Write tag user data area.

1. Constructor

C#: public WriteUserData_6C(
Byte antenna,
Byte[] accessPwd,
UInt32 ptr,
Byte[] userData)
public WriteUserData_6C(
Byte antenna,
Byte[] accessPwd,
UInt32 ptr,
Byte[] userData,
Byte[] tagID,
MemoryBank tagIDType)

JAVA: public WriteUserData_6C(
byte antenna,
byte[] accessPwd,
int ptr,
byte[] userData)
public WriteUserData_6C(
byte antenna,
byte[] accessPwd,
int ptr,
byte[] userData,
byte[] tagID,
MemoryBank tagIDType)

C++: WriteUserData_6C(unsigned char antenna,
unsigned char *accessPwd,
unsigned char accessPwdLen,
unsigned char ptr,
unsigned char *userData,

unsigned char userDataLen)
WriteUserData_6C(unsigned char antenna,
unsigned char *accessPwd,
unsigned char accessPwdLen,
unsigned char ptr,
unsigned char *userData,
unsigned char userDataLen,
unsigned char* tagID,
unsigned char tagIDLen,
unsigned char tagIDType)

parameter:

antenna: Antenna port.

accessPwd: Access password, 4 bytes.

ptr: Start address, in word. userData: Designate the user data to be written.

tagID: Designate the operational tag EPC or TID.

tagIDType: tagID corresponding data area enumeration.

accessPwdLen: * accessPwdparameter length.

userDataLen: *userDataparameter length.

tagIDLen: * tagIDparameter length.

2. Command response(ReceivedMessage)

This message has no data for further analysis. ReceivedMessage is empty.

6.3.3 6B tag write user data area (WriteUserData2_6B type)

Purpose: Write tag user area data.

1. Constructor

C#: public WriteUserData2_6B(
Byte antenna,
Byte[] tagID,
Byte ptr,
Byte[] userdata)

JAVA: public WriteUserData2_6B(

byte antenna,
byte[] tagID,
byte ptr,
byte[] userdata)

C++: WriteUserData2_6B(
unsigned char antenna,
unsigned char *tagID,
unsigned char tagIDLen,
unsigned char ptr,
unsigned char *userData,
unsigned char userDataLen)

parameter:

antenna: Antenna port.

tagID: Designate the operational tag ID.

ptr: Start address (0-215), in bytes.

userdata: Data to be written.

tagIDLen: * tagIDparameter length.

userDataLen: *userDataparameter length.

2. Command response(ReceivedMessage)

This message has no data for further analysis. ReceivedMessage is empty.

6.3.4 6C tag configuration access password (AccessPwdConfig_6C type)

Purpose: This function is used to alter the tag access password. The 6C tag default access password is 0.

1. Constructor

C#: public AccessPwdConfig_6C(
Byte antenna,
Byte[] oldPwd,
Byte[] newPwd)

public AccessPwdConfig_6C(
Byte antenna,
Byte[] oldPwd,
Byte[] newPwd,
Byte[] tagID,
MemoryBank tagIDType)

JAVA: public AccessPwdConfig_6C(
byte antenna,
byte[] oldPwd,
byte[] newPwd)
public AccessPwdConfig_6C(
byte antenna,
byte[] oldPwd,
byte[] newPwd,
byte[] tagID,
MemoryBank tagIDType)

C++: AccessPwdConfig_6C(
unsigned char antenna,
unsigned char *oldPwd,
unsigned char oldPwdLen,
unsigned char *newPwd,
unsigned char newPwdLen)
AccessPwdConfig_6C(
unsigned char antenna,
unsigned char *oldPwd,
unsigned char oldPwdLen,
unsigned char *newPwd,
unsigned char newPwdLen,
unsigned char* tagID,
unsigned char tagIDLen,
unsigned char tagIDType)

parameter:

antenna: Antenna port.

oldPwd: Original access password, 4 bytes.

newPwd: New access password, 4 bytes.

tagID: Designate the operational tag EPC or TID.

tagIDType: tagIDcorresponding data area enumeration.

oldPwdLen: *oldPwdparameter length.

newPwdLen: *newPwdparameter length.

tagIDLen: * tagIDparameter length.

2. Command response(ReceivedMessage)

This message has no data for further analysis. ReceivedMessage is empty.

3. Command description

Tag factory default access password is 8 zeros.

If the password area has yet been locked, the all zero password can be used to execute write operation.

6.3.5 6C tag kill password configuration (killPwdConfig_6C type)

Purpose: This function is used to alter tag kill password, the default password for 6C tag is 0, this operation is used only when the tag kill password is non zero.

1. Constructor

C#: public KillPwdConfig_6C(
Byte antenna,
Byte[] accessPwd,
Byte[] killPwd)
public KillPwdConfig_6C(
Byte antenna,
Byte[] accessPwd,
Byte[] killPwd,
Byte[] tagID,
MemoryBank tagIDType)

JAVA: public KillPwdConfig_6C(
byte antenna,
byte[] accessPwd,
byte[] killPwd)

public KillPwdConfig_6C(
byte antenna,
byte[] accessPwd,
byte[] killPwd,
byte[] tagID,
MemoryBank tagIDType)

C++: KillPwdConfig_6C(
unsigned char antenna,
unsigned char *accessPwd,
unsigned char accessPwdLen,
unsigned char *killPwd
unsigned char killPwdLen)
KillPwdConfig_6C(
unsigned char antenna,
unsigned char *accessPwd,
unsigned char accessPwdLen,
unsigned char *killPwd,
unsigned char killPwdLen,
unsigned char* tagID,
unsigned char tagIDLen,
unsigned char tagIDType)

parameter:

antenna: Antenna port.

accessPwd: Access password, 4 bytes.

killPwd: Kill password, 4 bytes.

tagID: Designate operational tag EPC or ID.

tagIDType: tagIDcorresponding data area enumeration.

accessPwdLen: *accessPwdparameter length.

killPwdLen: *killPwdparameter length.

tagIDLen: * tagIDparameter length.

2. Command response(ReceivedMessage)

This message has no data for further analysis. ReceivedMessage is empty.

3. Command description

EPC protocol requires that the kill password for a tag must be non zero during the execution of kill operation.

6.3.6 6C tag general writing operation (WriteTag_6C type)

Purpose: Can be used to write reserved area, EPC, TID or user data area. Note: Reserved area, EPC and TID do not recommend the writing of such command, unless the user is very familiar with 6C tag.

1. Constructor

C#: public WriteTag_6C(
Byte antenna,
Byte[] accessPwd,
MemoryBank memoryBank,
UInt32 ptr,
Byte length,
Byte[] data)
public WriteTag_6C(
Byte antenna,
Byte[] accessPwd,
MemoryBank memoryBank,
UInt32 ptr,
Byte length,
Byte[] data
Byte[] tagID,
MemoryBank tagIDType)

JAVA: public WriteTag_6C(
byte antenna,
byte[] accessPwd,
MemoryBank memoryBank,
int ptr,
byte length,
byte[] data)
public WriteTag_6C(

byte antenna,
byte[] accessPwd,
MemoryBank memoryBank,
int ptr,
byte length,
byte[] data
byte[] tagID,
MemoryBank tagIDType)

C++: WriteTag_6C(unsigned char antenna,
unsigned char *accessPwd,
unsigned char accessPwdLen,
MemoryBank memoryBank,
unsigned char ptr,
unsigned char length,
unsigned char *data,
unsigned char dataLen)

WriteTag_6C(
unsigned char antenna,
unsigned char *accessPwd,
unsigned char accessPwdLen,
MemoryBank memoryBank,
unsigned char ptr,
unsigned char length,
unsigned char *data,
unsigned char dataLen,
unsigned char *tagID,
unsigned char tagIDLen,
MemoryBank tagIDType)

parameter:

antenna: Antenna port.

accessPwd: Access password, 4 bytes.

memoryBank: Designate data area to be written, corresponding MemoryBank enumeration.

ptr: Write start address, in byte.

length: Write data length, in byte.

data: Designate data to be written.

tagID: Designate operational tag's EPC or TID.

tagIDType: tagIDcorresponding data area enumeration.

accessPwdLen: *accessPwdparameter length.

dataLen: *dataparameter length.

tagIDLen: *tagIDparameter length.

2. Command response(ReceivedMessage)

This message has no data for further analysis. ReceivedMessage is empty.

6.4 Tag lock operation

6.4.1 6C tag lock operation (LockMemoryBank_6C type)

Purpose: This function can realize the lock and unlock function for tag data area. The different lock statuses of data area lock operation correspond to the different tag data access authority limitation. The lock status of each data area can be independently controlled. There are four lock statuses for 6C tag data area: unlock, lock, permanent unlock, permanent lock.

1. Constructor

C#: public LockMemoryBank_6C(
Byte antenna,
Byte[] accessPwd,
Byte lockType,
Byte bank)
public LockMemoryBank_6C(
Byte antenna,
Byte[] accessPwd,
Byte lockType,
Byte bank,
Byte[] tagID,
MemoryBank tagIDType)

JAVA: public LockMemoryBank_6C(
byte antenna,
byte[] accessPwd,
byte lockType,
byte bank)
public LockMemoryBank_6C(
byte antenna,
byte[] accessPwd,
byte lockType,
byte bank,
byte[] tagID,
MemoryBank tagIDType)

C++: LockMemoryBank_6C(
unsigned char antenna,
unsigned char *accessPwd,
unsigned char accessPwdLen,
unsigned char lockType,
unsigned char memoryBank)

LockMemoryBank_6C(
unsigned char antenna,
unsigned char *accessPwd,
unsigned char accessPwdLen,
unsigned char lockType,
unsigned char memoryBank,
unsigned char* tagID,
unsigned char tagIDLen,
unsigned char tagIDType)

parameter:

antenna: Antenna port.

accessPwd: Access password, 4 bytes..

lockType: Lock operation type. 00, lock, 01, unlock, 02, permanent lock, 03, permanent unlock.

bank: Lock operation area. 00, all data area; 01, TID data area (typically tag chip vendor will permanently lock TID area and disable unlock); 02, EPC data area; 03 USER area; 04, access password; 05, kill password.

tagID: Designate operational tag's EPC or TID.

tagIDType: tagIDcorresponding data area enumeration.

accessPwdLen: *accessPwdparameter length.

tagIDLen: * tagIDparameter length.

2. Command response(ReceivedMessage)

This message has no data for further analysis. ReceivedMessage is empty.

3. Command description

The TID area in EPC tag has been written into TID code during manufacturing and is configured as permanent locked. Therefore the execution of "All Bank" unlock operation will not execute the unlock of TID area.

Unlock status: No verification password needed before read or write.

Lock status: Readable (Access password and kill password must be verified before reading), if the tag access password is non zero, verification of access password is needed before writing.

Permanent lock: Able to read and write, unable to change lock status after permanent unlock.

Permanent lock: Able to read only (Unable to read after the permanent lock of access password and kill password), unable to change lock status after permanent unlock.

6.4.2 6B tag lock user data (LockUserData_6B type)

Purpose: Lock user area data.

1. Constructor

C#: public LockUserData_6B(
Byte antenna,
Byte[] tagID,
Byte[] lockInfo)

JAVA: public LockUserData_6B(
byte antenna,
byte[] tagID,
byte[] lockInfo)

C++: LockUserData_6B(
unsigned char antenna,
unsigned char *tagID,
unsigned char tagIDLen,
unsigned char *lockInfo,
unsigned char lockInfoLen)

parameter:

antenna: Antenna port.

tagID: Designate operational tag ID.

lockInfo: Lock information, the length must be identical to the length of the tag user data area. Information in every byte indicates whether lock operation must be performed on that byte, non zero refers to the locking of user data in that

position.

tagIDLen: * tagIDparameter length.

lockInfoLen: *lockInfoparameter length.

2. Command response(ReceivedMessage)

This message has no data for further analysis. ReceivedMessage is empty.

3. Command description

Note: No unlock after locking.

6.4.3 6B tag lock state query (LockStateQuery_6B type)

Purpose: Inquire tag user area data lock state.

1. Constructor

C#: public LockStateQuery_6B(
Byte antenna,
Byte[] tagID,
Byte[] queryInfo)

JAVA: public LockStateQuery_6B(
byte antenna,
byte[] tagID,
byte[] queryInfo)

C++: LockStateQuery_6B(
unsigned char antenna,
unsigned char *tagID,
unsigned char tagIDLen,
unsigned char *queryInfo,
unsigned char queryInfoLen)

parameter:

antenna: Antenna port.

tagID: Designate the operational tag ID.

queryInfo: Inquire information, the length must be identical length of the tag user data area. Information in every byte indicates whether lock operation must be performed on that byte, non zero refers to the locking of user data in that

position.

tagIDLen: * tagIDparameter length.

queryInfoLen: *queryInfoparameter length.

2. Command response(ReceivedMessage)

1) Lock state

Purpose: Get tag data area lock state. Corresponding byte 00H indicates data not locked, 01H indicates data locked.

C#: public Byte[] LockResult { get; }

JAVA: public byte[] getLockResult() { }

C++: bool GetLockResult(unsigned char *LockResult,unsigned char &LockResultLen)

3. Command description

queryInfo corresponds to LockResult, and both correspond to the user data area. For example, when inquiring the lock status of the 10th byte of the user area, the 10th byte of the queryinfo must be configured as 1 and the rest configured as 0. The 10th byte of the responding LockResult represents the query result.

6.5 6C tag kill (KillTag_6C type)

Purpose: This function is used to kill tag, so that the tag will not response to any reader command and losses its value. This process is irreversible. The reader can only perform kill operation on tag with non-zero kill password.

1. Constructor

C#: public KillTag_6C(
Byte antenna,
Byte[] killPwd,
Byte[] epcData)

JAVA: public KillTag_6C(
byte antenna,
byte[] killPwd,
byte[] epcData)

C++: KillTag_6C(
unsigned char antenna,
unsigned char *killPwd,
unsigned char killPwdLen,
unsigned char *epcData,
unsigned char epcDataLen)

parameter:

antenna: Antenna port.

killPwd: Kill password, 4 bytes. The non zero password Used in the tag kill operation.

epcData: Designate the operational tag's EPC code.

killPwdLen: *killPwdparameter length.

epcDataLen: *epcDataparameter length.

2. Command response(ReceivedMessage)

This message has no data for further analysis. ReceivedMessage is empty.

3. Command description

As tag kill command is an irreversible operation, in order to prevent misoperation on other tags, this command must have an additional EOC serial code domain, the reader will only execute kill operation when the EPC code matches with the tag's EPC code.

6.6 Private function of special tag

As ISO18000-6C allows tag chip vendors to add special functions, Invego readers support private tag functions that have more practical applications. Mainly supports the EAS function in NXP tag and QT function in Impinj MONZA tag.

6.6.1 EAS function in NXP tag

The chip in NXP tag contains EAS storage label. When the EAS label location is set, the tag will return the reader with a string of warning message once it receives the EAS warning command from the reader, to indicate the EAS location has been set up, otherwise the tag will not response to the EAS warning command. This function is generally used in the access control system. NXP EAS function has the following known limitations: 1. All tag send back the content of warning message, so the warning messages do not have tag function; 2. As the mechanism lacks anti crush design, multiple tags may be sending warning message to the reader at the same time and the reader will not be able to accurately identify the warning message due to signal crush and this will lead to misreporting. For more details please refer to reference information [3].

The use of NXP EAS function requires EAS bit configuration and EAS command broadcast.

1. 6C tag EAS label configuration (EasConfig_6C type)

Purpose: Configure or cancel EAS label

1) Constructor

C#: public EasConfig_6C(
Byte antenna,
Byte[] accessPwd,
Byte flag)
public EasConfig_6C(
Byte antenna,
Byte[] accessPwd,
Byte flag,
Byte[] tagID,
MemoryBank tagIDType)

JAVA: public EasConfig_6C(

byte antenna,
byte[] accessPwd,
byte flag)
public EasConfig_6C(
byte antenna,
byte[] accessPwd,
byte flag,
byte[] tagID,
MemoryBank tagIDType)

C++: EasConfig_6C(
unsigned char antenna,
unsigned char *accessPwd,
unsigned char accessPwdLen,
unsigned char flag)
EasConfig_6C(
unsigned char antenna,
unsigned char *accessPwd,
unsigned char accessPwdLen,
unsigned char flag,
unsigned char *tagID,
unsigned char tagIDLen,
MemoryBank tagIDType)

Parameter:

antenna: Antenna port.

accessPwd: Access password, 4 bytes.

flag: EAS label configuration. 00H, reset EAS label, tag will not response to EAS broadcast command; 01H, set up EAS label, tag will respond to EAS broadcast command.

tagID: Designate the operational tag's EPC or TID.

tagIDType: tagIDcorresponding data area enumeration.

accessPwdLen: *accessPwdparameter length.

2) Command response(ReceivedMessage)

This message has no data for further analysis. ReceivedMessage is empty.

2. 6C tag EAS command broadcast (EasAlarm_6C type)

Purpose: EAS surveillance function configuration (only applicable to tag with NXP chip).

1) constructor

C#: public EasAlarm_6C(Byte antenna, Byte easCfg)

JAVA: public EasAlarm_6C(byte antenna, byte easCfg)

C++: public EasAlarm_6C(unsigned char antenna, unsigned char easCfg)

parameter:

antenna: Antenna port.

easCfg: EAS surveillance configuration. 01H is initiate EAS surveillance. Other values have no meaning. The reader will only terminate EAS command broadcast when the host computer sends termination command.

2) Command response(ReceivedMessage)

- Antenna port

Purpose: Get antenna port.

C#: public Byte Antenna { get; }

JAVA: public byte getAntenna(){ }

C++: unsigned char GetAntenna()

- EAS response type

Purpose: Get EAS response type

C#: public Byte AnswerType { get; }

JAVA: public byte getAnswerType() { }

C++: unsigned char GetAnswerType()

Description: Response type

00H EAS surveillance launch successfully

A0H Discover EAS location tag

6.6.2 6C tag QT command (QT_6C type)

Purpose: Configure or inquire tag's QT properties. The QT function of Impinj MONZA mainly provides the tag with two storage work modes and limit tag data remote access. For more detailed information on QT function, please refer to reference information [2].

1. Constructor

C#: public QT_6C(
Byte antenna,
Byte[] accessPwd,
Byte opType,
Byte persistent,
Byte[] payload)
public QT_6C(
Byte antenna,
Byte[] accessPwd,
Byte opType,
Byte persistent,
Byte[] payload,
Byte[] tagID,
MemoryBank tagIDType)

JAVA: public QT_6C(
byte antenna,
byte[] accessPwd,
byte opType,
byte persistent,
byte[] payload)
public QT_6C(
byte antenna,
byte[] accessPwd,
byte opType,
byte persistent,
byte[] payload,
byte[] tagID,
MemoryBank tagIDType)

C++: QT_6C(unsigned char antenna,
unsigned char *accessPwd,
unsigned char accessPwdLen,
unsigned char opType,
unsigned char persistent,
unsigned char *payload,
unsigned char payloadLen)
QT_6C(unsigned char antenna,
unsigned char *accessPwd,
unsigned char accessPwdLen,
unsigned char opType,
unsigned char persistent,
unsigned char *payload,
unsigned char payloadLen,
unsigned char tagIDLen,
MemoryBank tagIDType)

parameter:

antenna: Antenna port.

accessPwd: Access password, 4 bytes.

opType: Operation type(R/W).00H, inquire the tag original QT properties; 01H, configure the tag QT properties.

persistent: Configuration durability. 00H, configuration is only valid in the current power on of the tag; 01H, configuration can be kept forever.

payload: Configure parameter, 2 bytes.

accessPwdLen: *accessPwdparameter length.

payloadLen: * payloadparameter length.

tagID: Designate the operational tag's EPC or TID.

tagIDType: tagIDcorresponding data area enumeration.

Configure parameter description:

The meaning of the first byte common position control mode is in the chart below. The second byte is reserved.

Bit	7	6	5	4	3	2	1	0
Definition	QT_SR	QT_MEM	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

QT_SR: 1, Tag reduces response distance (or only responses in close distance) during power on and secure state; 0, tag does not reduce response distance.

QT_MEM: 1, Tag storage mapping is on public mode; 0, Tag storage mapping is on private mode.

2. Command response(ReceivedMessage)

1) Antenna port

Purpose: Get antenna port.

C#: public Byte Antenna { get; }

JAVA: public byte getAntenna(){ }

C++: unsigned char GetAntenna()

2) Configure parameter

Purpose: Get parameter configuration

C#: public Byte[] Payload { get; }

JAVA: public byte[] getPayload() { }

C++: bool GetPayload(unsigned char *data, unsigned char &dataLen)

6.7 6C tag selection(SelectTag_6C type)

Purpose: Select tag.

1. Constructor

C#: public SelectTag_6C(
MemoryBank memoryBank,
UInt32 ptr,
Byte matchBitLength,
Byte[] tagData)

JAVA: public SelectTag_6C(
MemoryBank memoryBank,
int ptr,
byte matchBitLength,
byte[] tagData)

C++: SelectTag_6C(
MemoryBank memoryBank,
unsigned char ptr,
unsigned char matchBitLength,
unsigned char *tagData,
unsigned char tagDataLen)

Parameter:

memoryBank: Designate the data area to match, MemoryBank type. Note: Can't be configured as reserved area.

ptr: Match data start address, in bit.

matchBitLength: Match data length, in bit.

tagData: Match data.

tagDataLen: *tagDataparameter length.

2. Command response(ReceivedMessage)

This message has no data for further analysis. ReceivedMessage is empty.

3. Command description

- Selection command is only valid in the next operation.

- Match data length refers to the bit number for matching, bit data refers to the matching data. As the data transfer is unable to transfer individual bit, byte transfer is adopted, 0 is added when the number is less than 8 digits.

  For example: Match bit number = 6, match data is

B7	B6	B5	B4	B3	B2	B1	B0
D	D	D	D	D	D	0	0

  D: Bit data 0: Supplement bit

- Tag selection command can't be used independently, and must be used together with the tag reading and writing command, to realize the selection of specific tag to perform read and write function. When using this function, the host computer will first send tag selection command to the reader before sending response tag read and write command, tag slection command is only valid once for tag reading and writing command. For example reader must first read the TID code in tag A before writing the tag A's USER area data. The entire process has to be realized with the following procedures: Host computer sends tag selection command and selects A -> Host computer sends TID code command -> Host computer sends tag selection command and selects A -> Host computer send write tag USER area data command.

6.8 Tag filter by time (FilterByTime type)

Purpose: Data from the same tag will not be uploaded again within the designated timing.This function is used to filter repeated tag data received from cyclic tag reader by the reader in designated time and reduce the host computer data processing overhead by reducing the uploading of redundant data.

1. Constructor

C#: public FilterByTime(Byte opType, Byte[] time)

JAVA: public FilterByTime(byte opType, byte[] time)

C++: FilterByTime(
unsigned char opType,
unsigned char *time,
unsigned char timeLen)

parameter:

opType: Operation type. 00H: Configure, 01H: Cancel (Ignore the latter item)

time: Filter time, 2 bytes, in 100 miliseconds.

timeLen: *timeparameter length.

2. Command response(ReceivedMessage)

This message has no data for further analysis. ReceivedMessage is empty.

7 Stop command, power off (PowerOff type)

Purpose: Stop command us used to terminate reader tag read and write operation, and close the reader carrier frequency, switching the reader to idle waiting state.

1. Constructor

C#: public PowerOff ()

JAVA: public PowerOff ()

C++: PowerOff ()

2. Command response(ReceivedMessage)

This message has no data for further analysis. ReceivedMessage is empty.

3. Command description

Note: This command will stop the functioning of all reader's antennas.

8 GPIO function

8.1 GPI input status query (GPI_800 type)

Purpose: This function is used to inquire the reader GPI port input level status.

1. Constructor

C#: public Gpi_800()

JAVA: public Gpi_800()

C++: Gpi_800(unsigned char checkMode)

2. Command response(ReceivedMessage)

Data query

Purpose: Get queried data, 2 bytes, I/O status and port status respectively.

C#: public Byte[] QueryData { get; }

JAVA: public byte[] getQueryData()

C++: bool GetQueryData(unsigned char *data,unsigned char &dataLen)

3. Command description

I/O Status

Port	7	6	5	4	3	2	1	0
Definition	Reserved	Reserved	Reserved	Reserved	Port 4	Port 3	Port 2	Port 1

Port status: 0: Low level; 1: High level

8.2 GPO output operation (GPO_800 type)

Purpose: This function controls the designated reader GPO port output status. It has to be specially pointed out that some reader has relay output port. We also define them as GPO port. We logically compare the close staus and open status to high and low level.

1. Constructor

C#: public Gpo_800(Byte ioPort, Byte level)

JAVA: public Gpo_800(byte ioPort, byte level)

C++: Gpo_800(unsigned char ioPort,unsigned char level)

Parameter:

ioPort: Output port. Used to designate the port number that requires output control, identical to its hardware label.

level: Output status. 00 refers to low level output or open relay, 01 refers to high level output or close relay.

2. Command response(ReceivedMessage)

This message has no data for further analysis. ReceivedMessage is empty.

3. Command description

Output port: 01H Port 1
02H Port 2
03H Port 3
04H Port 4

Output level: 00H Low level output
01H High level output
03H Positive pulse output
04H Negative pulse output

8.3 GPI event trigger message upload (RXD_IOTriggerSignal_800 type)

Purpose: GPI trigger signal; succeeded from BaseMessageNotification abstract type. As the GPI port of our reader has realized event trigger mechanism, when the GPI status fulfills the trigger or terminate condition in 5.3.4, the reader or the host computer will upload a notification message.

1. Command response(ReceivedMessage)

1) Input port

Purpose: Get input port number.

C#: public Byte GPIPort { get; }

JAVA: public byte getGPIPort()

C++: unsigned char GetGPIPort ()

2) Trigger sigla

Purpose: Get trigger start or stop signal. True for start, false for stop.

C#: public Boolean IsStart{ get; }

JAVA: public boolean isStart ()

C++: bool GetIsStart ()

3) Trigger time

Purpose: Get trigger time, 8 byte UTC time. First 4 bytes refers to the seconds in UTC time, last 4 bytes refers to milliseconds in UTC time.

C#: public Byte[] UTCTime{ get; }

JAVA: public byte[] getUTCTime()

C++: bool GetUTCTime (unsigned char *pUTCTime, unsigned char UTCTimeLen)

2. Command description

Input port

Bit	Bit7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Definition	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Port 2	Port 1

9 Handheld reader special module

Invengo handheld reader special module, controls Invengo handheld reader RFID module, GRPS, Wi-Fi and other hardware equipment.

9.1 Data package

Data packet that encapsulates byte array.

Assembly: Invengo.dll

Namespace: Invengo

Type Name: Datagram

Supported reader: XC2900, XC2903

9.1.1 Construct

public Datagram()

Construct an empty instance;

public Datagram(int length)

Designate the length of the instantized data packet;

public Datagram(byte[] data)

Use valid byte data instantized data packet;

public Datagram(string hexstring)

Hexadecimal string instantized data packet, the string length must be in even number and must be in hexadecimal, for example “19A0B1C2D3F4”;

9.1.2 Data properties

public byte[] Data Data packet byte data;

public int Length Data length;

9.1.3 Converting to string

Purpose: Converting the data in data packet to designated string;

Prototype: public string ToString(string formatter) or

public override string ToString()

parameter:

string formatter

ASCII, UTF-8, UTF-16, UNICODE, other that are switched to hexadecimal characters by default.

Return value:

String after conversion.

9.2 RFID control

Switch on the power of RFID module, the RFID power needs to be switched on to performed related RFID operation.

Assembly: Invengo.Devices.ModuleControl.dll

Namespace: Invengo.Devices.ModuleControl

Type Name: RfidControl

Supported reader: XC2900, XC2903

9.2.1 Switch on RFID

Function: Switch on the power of RFID module to allow it to function normally;

Prototype: public static int Active()

Parameter: None

Return value:

int 0 Module power on failure
1 Module power on successful (from power off to power on)
2 Module has been powered on

9.2.2 Switch off RFID

Function: Switch off the power of the RFID module;

Prototype: public static bool Deactive()

Parameter: None

Return value:

true: Successful; false: Failed;

9.2.3 RFID power status

Purpose: Get the current power source of RFID module, whether it is switched on or switched off.

Prototype: public static bool IsActived()

Parameter: None

Return value:

true: Switch on; false: Switch off;

9.3 GPRS control

Realize the activation, termination, dial connection, dial disconnection of the GPRS module; activate the GPRS module to perform dial operation using the GPRS dial program in the operating system. Apart from the provided dial connection method, users can also refer to the RAS API data in Windowns CE operating system to realize dial Function.

Assembly: Invengo.Devices.ModuleControl.dll

Namespace: Invengo.Devices.ModuleControl

Type Name: GprsControl

Supported reader: XC2900, XC2903

9.3.1 Switch on GPRS

Function: Switch on GPRS module electric supply, to allow GPRS module in working condition;

Prototype: public static bool Active()

Parameter: None

Return value:

true: Activation successful; false: Failed;

Note: When the GPRS module is switched on for the first time, the GRPS module needs to do network scanning, so users need to wait for 20 seconds before dialing.

9.3.2 Switch off GPRS

Purpose: Switch off GPRS module;

Prototype: public static bool Deactive()

Parameter: None

Return value:

true: Successful; false: Failed;

9.3.3 GPRS power status

Purpose: Get the current power status of the module, whether it is switched on or switched off.

Prototype: public static bool IsActived()

Parameter: None

Return value:

true: Switched on; false: Switched off;

9.3.4 Dial connection

Purpose: Switch on the GPRS module and strat dial connection; before dialing, alter the connection parameter using the operating system GRPS dial program;

Prototype: public static bool Connect()

Parameter: none

Return value:

true: Successful; false: Failed;

9.3.5 Dial disconnection

Purpose: Disconnect dial connection and switch off GPRS module.

Prototype: public static bool Disconnect()

Parameter: None

Return value:

true: Successful; false: Failed;

9.4 Wi-Fi control

Realize the power on and power off of the Wi-Fi module. Once the Wi-Fi module has been switched on, Wi-Fi configuration can be performed using the Wi-Fi connection function in the operating system.

Assembly: Invengo.Devices.ModuleControl.dll

Namespace: Invengo.Devices.ModuleControl

Type Name: WifiControl

Supported reader: XC2900, XC2903

9.4.1 Switch on Wi-Fi

Purpose: Switch on the power of Wi-Fi module, to allow Wi-Fi to function normally; once the Wi-Fi module has been activated, related operation can be performed using the Wi-Fi configuration program installed in the operating system;

Prototype: public static bool Active()

Parameter: None;

Return value:

true: Successful; false: Failed;

9.4.2 Switch off Wi-Fi

Purpose: Switch off Wi-Fi module.

Prototype: public static bool Deactive()

Parameter: None

Return value:

true: Successful; false: Failed;

9.4.3 Wi-Fi power status

Purpose: Get the current power status of the Wi-Fi module, whether it is switched on or switched off.

Prototype: public static bool IsActived()

Parameter: None

Return value:

true: Switch on; false: Switch off;

9.5 Capture system power message

When a handheld reader (PDA) switches to power saving mode (standby), the power for related module (such as RFID) might automatically disconnect, work can only be continued by waking up the PDA and turning on the power for the module; the handheld reader operating system provides messages that captures this power source change, to allow application development, we also provide the processing method in caputirng power status change.

Assembly: Invengo.dll

Namespace: Invengo.Platform

Type Name: PowerStateNotify

Supported reader: XC2900, XC2903

9.5.1 Power status request

Purpose: Capture power status message request, used to receive message.

Prototype: public delegate void FnPowerNotifyCallback(uint stateFlags)

Parameter:

uint stateFlags Power status message value;

9.5.2 Initiate capture power status message

Purpose: Initiate capture power status message.

Prototype: public static void PowerNotifyStart(

PowerStateNotify.FnPowerNotifyCallback func)

parameter:

FnPowerNotifyCallback func FnPowerNotifyCallback request, used to receive message;

9.5.3 Stop capture power message

Purpose: Sop capture power message, adjust once the program exits.

Prototype: public static extern void PowerNotifyStop()

9.5.4 Power status message value

Awake:

public const uint POWER_STATE_ON = 0x10000

Hibernate:

public const uint POWER_STATE_OFF = 0x20000

Start:

public const uint POWER_STATE_BOOT = 0x80000

Idle:

public const uint POWER_STATE_IDLE = 0x100000

Pending:

public const uint POWER_STATE_SUSPEND = 0x200000

Reset:

public const uint POWER_STATE_RESET = 0x800000

9.6 Barcode identification/camera switch

For XC2903 reader, barcode identification and camera function can't be used at the same time, only one function can be selected everytime.

Assembly: Invengo.Devices.ModuleControl.dll

Namespace: Invengo.Devices.ModuleControl

Type Name: CameraBarcodeSwitch

Supported reader: XC2903

9.6.1 Query current mode

Purpose: Query if the current mode is barcode reading or camera mode;

Prototype: public static int BarCodeOrCameraQuery()

Parameter: None

Return value:

int 0: Camera; 1: Barcode reading;

9.6.2 Choose barcode reading

Purpose: Switch to barcode reading mode;

Prototype: public static void SelBarCode()

9.6.3 Choose camera

Purpose: Switch to camera mode;

Prototype: public static void SelCamera()

9.7 Barcode identification

Handheld reader may be equipped with different barcode head, (please refer to user manual for the specific model of the barcode head), different barcode head provide corresponding barcode type.

9.7.1 BarcodeScannerSymbol type

Purpose: Control Symbol ½ D barcode head operation;

Assembly: Invengo.Barcode.dll

Namespace: Invengo.Barcode

Supported reader: XC2900, XC2903

1. Instantiation

Function: Instantize BarcodeScannerSymbol type.

Prototype: public static BarcodeScannerSymbol CreateInstance()

Parameter: None

Return value:

Return BarcodeScannerSymbol instance.

2. Receive barcode data event

Purpose: Receive barcode data event;

Prototype: public event BarcodeRecvEventHandler BarcodeRecvEvent

Event parameter:

BarcodeRecvEventArgs e Code member is barcode data;

3. Initialization

Purpose: Initialize barcode reading function;

Prototype: public void Initialize()

4. Trigger barcode reading

Function: Trigger barcode head to scan barcode, applicable to Symbol ½ D adaptive barcode head;

Prototype: public bool BarcodeRead()

Return value:

true: Successful; false: Failed;

5. Switch off brcode reading

Purpose: Switch off barcode reading function;

Prototype: public void Close()

6. Release resources

Function: Release barcode reading resources, at the same time stop barcode operation;

Prototype: public void Dispose()

9.7.2 BarcodeScannerHoneywell1D type

Function: Control Honeywell 1D adaptive barcode head operation;

Assembly: Invengo.Barcode.dll

Namespace: Invengo.Barcode

Supported reader: XC2900, XC2903

1. Instantiation

Function: Instantize BarcodeScannerHoneywell1D type.

Prototype: public static BarcodeScannerHoneywell1D CreateInstance()

Parameter: None

Return value:

Return BarcodeScannerHoneywell1D instance.

2. Receive barcode data event

Purpose: Receive barcode data event;

Prototype: public event BarcodeRecvEventHandler BarcodeRecvEvent

Event parameter:

BarcodeRecvEventArgs e Code member is barcode data;

3. Initialization

Purpose: Initialization barcode reading function;

Prototype: public void Initialize()

4. Trigger barcode reading

Function: Trigger barcode head to perform barcode scanning, applicable to Symbol ½ D adaptive barcode head;

Prototype: public bool BarcodeRead()

Return value:

true: Successful; false: Failed;

5. Switch off brcode reading

Purpose: Switch off brcode readingFunction;

Prototype: public void Close()

6. Release resources

Function: Release barcode reading resources, at the same time stop barcode operation;

Prototype: public void Dispose()

9.7.3 BarcodeScannerHoneywell2D type

Function: Control Honeywell 1D adaptive barcode head operation;

Assembly: Invengo.Barcode.dll

Namespace: Invengo.Barcode

Supported reader: XC2900, XC2903

1. Instantiation

Function: InstantiationBarcodeScannerHoneywell2D type

Prototype: public static BarcodeScannerHoneywell2D CreateInstance()

Parameter: None

Return value:

Return BarcodeScannerHoneywell2D instance

2. Receive barcode data event

Purpose: Receive barcode data event;

Prototype: public event BarcodeRecvEventHandler BarcodeRecvEvent

Event parameter:

BarcodeRecvEventArgs e Code member is barcode data;

3. Initialization

Purpose: Initialization barcode reading function;

Prototype: public bool BarcodeCE5_SInitBig()

Return value:

true: Successful; false: Failed;

4. Trigger barcode reading

Function: Trigger barcode head to perform barcode scanning, applicable to Symbol ½ D adaptive barcode head;

Prototype: public bool BarcodeRead()

Return value:

true: Successful; false: Failed;

5. Release resources

Function: Release barcode reading resources, at the same time stop barcode operation;

Prototype: public void Dispose()

9.7.4 SDLScanner type

Function: Control Motorola ½ D barcode operation;

Assembly: MotoSDL.dll

Namespace: MotoSDL

Type Name: SDLScanner

Supported reader: XC2900, XC2903

1. Instantiation

Function: InstantiationSDLScanner type.

Prototype: public static SDLScanner CreateInstance()

Parameter: None

Return value:

ReturnSDLScanner instance;

2. Receive barcode decode data event

Purpose: Receive barcode decode data event;

Prototype: public event EventHandler Decoded

Event parameter:

EventArgs e

3. Recover trigger mode

Purpose: Recover trigger mode;

Prototype: public void ResetTrigger()

Return value: None

4. Set scanner to decode mode

Function: Set scanner to decode mode

Prototype: public void DecodeMode()

Return value: None

5. Configure barcode reading trigger

Function: Trigger barcode head to perform barcode scanning, applicable to Motorola ½ D adaptive barcode head;

Prototype: public bool Trigger

Configuration:

true: Trigger scan; false: Stop can;

6. Barcode dialogue

Function: Determine whether it is decoding

Prototype: public bool fInSession

Return value:

true: In the processing of decoding; false: Decoding ended;

7. Release resources

Function: Release barcode reading resources, at the same time stop barcode operation;

Prototype: public void Release()

Return value: None

9.7.5 DecodeData type

Function: Motorola ½ D adaptive barcode head decode data

Assembly: MotoSDL.dll

Namespace: MotoSDL

Type Name: DecodeData

Supported reader: XC2900, XC2903

1. Decode data

Function: Decode data.

Prototype: public static string text

9.8 Function key message

9.8.1 Capture message

When clicking on Function key, they will have different message. Configure Key Preview of the Form Properties to “true”, these messages will be captured once Form Event KeyDown and KeyPress event are realized.

1. Handhend reader XC2900’s Function key definition:

SCAN key: 0XF5

Left Function key: 0xF2

Right Function key: 0xF1

2. Handhend reader XC2903’s Function key definition:

Left Function key: 0xF2

Right Function key: 0xF1

9.9 Sound volume

Get or configure sound volume

Assembly: Invengo.dll

Namespace: Invengo.Audio

Type Name: SoundVolume

Supported reader: XC2900, XC2903

9.9.1 Configure volume

Purpose: Configure sound volume

Prototype: public static void SetSoundVolume(ulong dwVolume);

parameter:

ulong dwVolume

Volume value, the range: 0x0000 ~ 0xFFFF, can also be calculated in percentages, such as oriVolume = (ulong)(0xFFFF * 1.0 * 70 / 100.0).

9.9.2 Get volume

Purpose: Get sound volume

Prototype: public static void GetSoundVolume(ref ulong dwVolume);

parameter:

ref ulong dwVolume

Return current sould volume value

9.10 Play sound file

To play a sound file

Assembly: Invengo.dll

Namespace: Invengo.Audio

Type Name: SoundPlayer

Supported reader: XC2900, XC2903

9.10.1 Play WAV

Purpose: Play WAV format sound file

Prototype: public static bool PlayWAV(string strFileName)

parameter:

string strFileName

Complete name of the sound file Return value:

true: Successful; false: Failed

9.11 Date and time

Configure/Get handheld reader system's date and time.

Assembly: Invengo.dll

Namespace: Invengo.Platform

Type Name: SysTime

Supported reader: XC2900, XC2903

9.11.1 Date and time structure

The structural definition of date and time:

```
public struct SYSTEMTIME
{
    public ushort wYear;                //Year
    public ushort wMonth;               //Month
    public ushort wDayOfWeek;           //Week
    public ushort wDay;                 //Day
    public ushort wHour;                //Hour
    public ushort wMinute;  //Minute
    public ushort wSecond;  //Second
    public ushort wMilliseconds;        //Milisecond
}
```

9.11.2 Get date and time

Purpose: Get current system's date and time;

Prototype: public static void GetLocalTime(ref SYSTEMTIME lpSystemTime)

parameter:

SYSTEMTIME lpSystemTime Date and time structure;

9.11.3 Configure date and time

Purpose: Configure current system's date and time;

Prototype: public static bool SetLocalTime(ref SYSTEMTIME lpSystemTime)

parameter:

SYSTEMTIME lpSystemTime Date and time structure;

Return value:

true: Successful; false: Failed;

9.12 Standby/reboot/shutdown

Control the handheld reader’s standby/reboot/shut down operation. Assembly: Invengo.dll

Namespace: Invengo.Platform

Type Name: SystemPower

Supported reader: XC2903

9.12.1 Standby

Purpose: Switch handheld reader to standby mode;

Prototype: public static bool Suspend()

Parameter: None

Return value:

true: Successful; false: Failed;

9.12.2 Reboot

Purpose: Reboot handheld reader;

Prototype: public static bool Reboot()

Parameter: None

Return value:

true: Successful; false: Failed;

9.12.3 Shutdown

Purpose: Shut down handheld reader;

Prototype: public static bool ShutDown()

Parameter: None

Return value:

true: Successful; false: Failed;

9.13 Data conversion

Provide data format conversion, such as conversion from byte data to hexadecimal string or from hexadecimal string to byte array;

Assembly: Invengo.dll

Namespace: Invengo

Type Name: FormatConvert

Supported reader: XC2900, XC2903

9.13.1 Conversion of byte data to hexadecimal string

Purpose: Conversion of byte data to hexadecimal string (hexadecimal string comprised of 0-9 and A-F);

Prototype: public static string BytesToString(byte[] dataBytes, int size) or

public static string BytesToString(byte[] dataBytes, int offset, int size)

parameter:

byte[] dataBytes
byte array;

int offset
start address;

int size
Number of byte to convert;

Return value:
Hexadecimal string (Hexadecimal string comprised of 0-9 and A-F);

9.13.2 Conversion of hexadecimal string to byte data

Purpose: Conversion of hexadecimal string to byte data

Prototype: public static byte[] HexStringToBytes(string hexValue)

parameter:

string hexValue

Hexadecimal string comprised of 0-9 and A-F, with the length must be in even number, such as “19A0B1C2D3F4”;

Return value:

Byte array;

10 Appendices

A flow example of host computer controlling reader tag reading and writing function

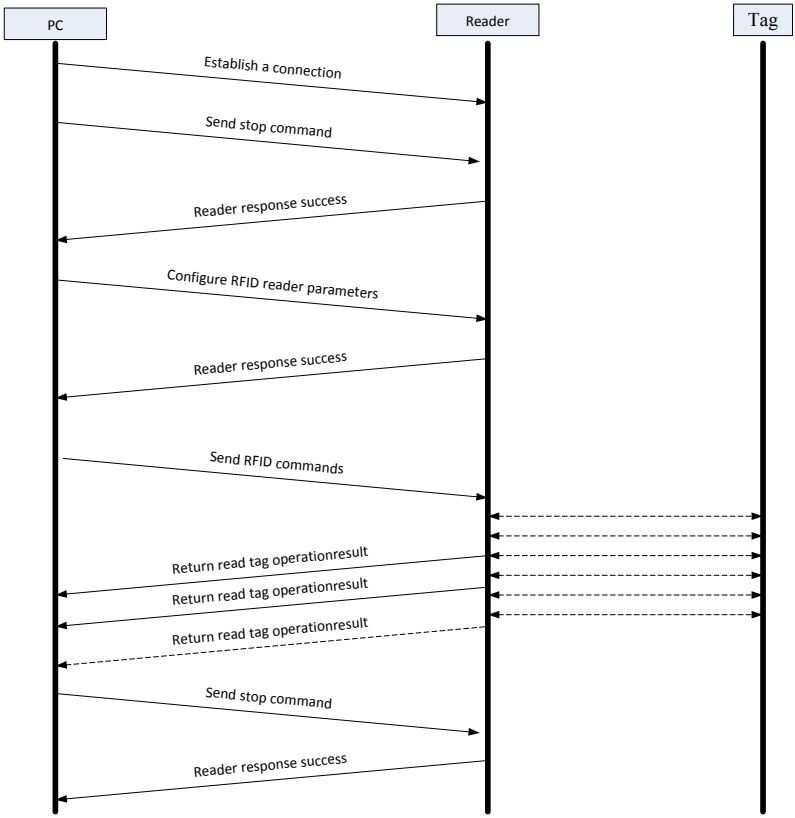


Figure 10-1

The host computer connects with the reader through assigned port, upon successful connection, the host computer will first send a termination command to the reader, the main purpose being: 1. To switch the reader to idle mode to allow the reader to response to the subsequent command normally; 2. To determine if the current statis of the reader is normal and useable based on whether the reader's response s successful.

10.2 Reader error code chart

Error code chart includes error codes that are returned to the host computer when the reader fails to execute its operation.

Chart 1 Reader system error code chart

Error code	Error code definition
01H	Reserved
02H	Reserved
03H	Reserved
04H	Reserved
05H	Reserved
06H	Reader RF hardware error
07H	Reserved
08H	Reserved
09H	Reserved
0AH	Reserved
0BH	Reserved
0CH	Reserved
0DH	Reserved
0EH	Reserved
0FH	Belong to the range byt the reader is unable to determine or unknown type error

Chart 2 Reader system error code chart

Error code	Error code definition
11H	Reserved
12H	Wrong antenna port is used
13H	Reserved
14H	Reserved
15H	This operation is not allowed under current status
16H	Reserved
17H	Reserved
18H	Reserved
19H	The reader internal temperature has exceeded the temperature limit
1AH	The difference between the foreward power and the backward power is too low (high standing wave ratio)
1BH	Parameter can't be changed
1CH	Reserved
1DH	Reserved
1EH	Reserved
1FH	Belong to the range byt the reader is unable to determine or unknown type error

Chart 3 Commmand receive/data transfer error code chart

Error code	Error code definition
20H	Incomplete command or data received
21H	CRC verification error for command or data received
22H	Reserved
23H	Reserved
24H	Error in command parameter
25H	Error in command frame structure (missing domain)
26H	Unsupported command type
27H	Reader receives too many commands, unable to process at the moment
28H	Reserved
29H	Reserved
2AH	Reserved
2BH	Reserved
2CH	Reserved
2DH	Reserved
2EH	Reserved
2FH	Belong to the range byt the reader is unable to determine or unknown type error

Chart 4 EPC tag operation error code chart

Error code	Error code definition
60H	Unresponsive or missing tag
61H	Reserved
62H	Tag Return message：Tag operation address overflow
63H	Tag Return Message：Operation storage area has been locked
64H	Tag storage password error
65H	Tag kill password error
66H	Reserved
67H	Reserved
68H	Reserved
69H	Tag Return Message：Unknown type error
6AH	Tag ReturnMessage：Insufficient power
6BH	Reserved
6CH	Reserved
6DH	Reserved
6EH	Reserved
6FH	Belong to the range byt the reader is unable to determine or unknown type error

10.3 Reader Function chart1

Note: "˜" represents standard function, "™" represents specific product function

Reader Function corresponding item	807	850	861	825	829	811	812	2900	2903
5.1 SysQuery_800									
21H Reader model	●	●	●	●	●	●	●	●	●
23H Baseband processing hardware version	●	●	●	●	●	●	●	●	●
29H Application processing hardware version	●	●	●			●	●	●	●
0DH Serial port baud rate	●	●	●	●	●	●	●		
05H MAC address	●	●	●			●	●	●	●
06H IP address	●	●	●			●	●	●	●
50H Reader TCP connection mode	●	●	●			●	●		
07H Reader TCP service port number	●	●	●			●	●	●	●
51H Backup server	●	●	●			●	●		
52H Number of connection retries	●	●	●			●	●		
03H Reader antenna port power	●	●	●	●	●	●	●	●	●
E2H GPI trigger parameter	●	●	●		●	●	●	●	●
5.2 SysConfig_800									
0DH Serial port baud rate	●	●	●	●	●	●	●		
06H IP address	●	●	●			●	●	●	●
50H Reader TCP connection mode	●	●	●			●	●		
07H Reader TCP service port number	●	●	●			●	●	●	●
51H Backup server	●	●	●			●	●		
52H Number of connection retries	●	●	●			●	●		
03H Reader antenna port power	●	●	●	●	●	●	●	●	●
E2H GPI trigger parameter	●	●	●		●	●	●	●	●
6.2.1 ReadTag									
EPC_6C	●	●	●	●	●	●	●	●	●
TID_6C	●	●	●	●	●	●	●	●	●
EPC_TID_UserData_6C								●	●

EPC_TID_UserData_6C_2	●	●	●	●	●	●	●	●	●
ID_6B	●	●	●	●	●	●	●	●	●
ID_UserData_6B								●	●
EPC_6C_ID_6B								●	●
TID_6C_ID_6B									
EPC_TID_UserData_6C _ID_UserData_6B									
EPC_TID_UserData_ Reserved_6C_ID_UserData_6B	●	●	●	●	●	●	●		
EPC_PC_6C									
6.2.2 RXD_TagData	●	●	●	●	●	●	●	●	●
6.2.3 ReadUserData_6C	●	●	●	●	●	●	●	●	●
6.3.1 WriteEpc	●	●	●	●	●	●	●	●	●
6.3.2 WriteUserData_6C	●	●	●	●	●	●	●	●	●
6.3.3 WriteUserData2_6B	●	●	●	●	●	●	●	●	●
6.3.4 AccessPwdConfig_6C	●	●	●	●	●	●	●	●	●
6.3.5 KillPwdConfig_6C	●	●	●	●	●	●	●	●	●
6.3.6 WriteTag_6C	●	●	●	●	●	●	●	●	●
6.4.1 LockMemoryBank_6C	●	●	●	●	●	●	●	●	●
6.4.2 LockUserData_6B	●	●	●	●	●	●	●	●	●
6.4.3 LockStateQuery_6B	●	●	●	●	●	●	●	●	●
6.5 KillTag_6C	●	●	●	●	●	●	●	●	●
6.6.1.1 EasConfig_6C	●	●	●	●	●	●	●		
6.6.1.2 EasAlarm_6C	●	●	●	●	●	●	●		
6.6.2 QT_6C	●	●	●	●	●	●	●		
6.7 SelectTag_6C	●	●	●	●	●	●	●	●	●
6.8 FilterByTime	●	●	●	●	●	●	●	●	●
7 PowerOff	●	●	●	●	●	●	●	●	●
8.1 Gpi_800	●	●	●		●	●	●		
8.2 Gpo_800	●	●	●		●	●	●		
8.3 RXD_IOTriggerSignal_800	●	●	●		●	●	●		

Shenzhen HQ:
Corporate Headquarters
Invengo Information Technology Co., Ltd.
3/F, No.T2-B, High-tech Industrial Park South,
Shenzhen 518057, China

Tel: +86 800 830 7036
Fax: +86 755 2671 1693
Email: sales@invengo.cn
Website: www.invengo.cn

Singapore:
Invengo Technology Pte. Ltd.
10 Kallang Avenue, # 05-15 Aperia tower 2,
Singapore, 339510

Tel:+65 6702 3909
Email: invengo.sales@invengo.sg
Website: www.invengo.sg

Europe:
Invengo Technology BV
Belder 30-A, 4704RK Roosendaal,
The Netherlands

Tel: +31 88 6363 793
Fax: +31 88 6363 794
Email: web@invengo.eu

US:
Invengo Technology Corp
2700-160 Sumner Blvd.
Raleigh, NC 27616, USA

Tel: +1 919 890 0202
Toll Free: +1 855 379 2725
Email: sales@invengo.com
Website: www.invengo.com

Korea:
Invengo International Pte. Ltd (Korea)
30F ASEM Tower, 517 Yeongdong-daero,
Gangnam-gu, Seoul 135-798 Korea

Tel: +82 2 6001 3525
Fax: +82 2 6001 3003
Email: justin.kou@invengo.sg