

# Security PUSH Communication Protocol PUSH SDK

Date: January 2021

PUSH Protocol Version: 3.1.2

Doc Version: 2.3

English

Thank you for choosing our product. Please read the instructions carefully before operation. Follow these instructions to ensure that the product is functioning properly. The images shown in this manual are for illustrative purposes only.



For further details, please visit our Company's website  
[www.zkteco.com](http://www.zkteco.com).

## Copyright © 2021 ZKTECO CO., LTD. All rights reserved.

Without the prior written consent of ZKTeco, no portion of this manual can be copied or forwarded in any way or form. All parts of this manual belong to ZKTeco and its subsidiaries (hereinafter the "Company" or "ZKTeco").

## Trademark

**ZKTeco** is a registered trademark of ZKTeco. Other trademarks involved in this manual are owned by their respective owners.

## Disclaimer

This manual contains information on the operation and maintenance of the ZKTeco equipment. The copyright in all the documents, drawings, etc. in relation to the ZKTeco supplied equipment vests in and is the property of ZKTeco. The contents hereof should not be used or shared by the receiver with any third party without express written permission of ZKTeco.

The contents of this manual must be read as a whole before starting the operation and maintenance of the supplied equipment. If any of the content(s) of the manual seems unclear or incomplete, please contact ZKTeco before starting the operation and maintenance of the said equipment.

It is an essential pre-requisite for the satisfactory operation and maintenance that the operating and maintenance personnel are fully familiar with the design and that the said personnel have received thorough training in operating and maintaining the machine/unit/equipment. It is further essential for the safe operation of the machine/unit/equipment that personnel has read, understood and followed the safety instructions contained in the manual.

In case of any conflict between terms and conditions of this manual and the contract specifications, drawings, instruction sheets or any other contract-related documents, the contract conditions/documents shall prevail. The contract specific conditions/documents shall apply in priority.

ZKTeco offers no warranty, guarantee or representation regarding the completeness of any information contained in this manual or any of the amendments made thereto. ZKTeco does not extend the warranty of any kind, including, without limitation, any warranty of design, merchantability or fitness for a particular purpose.

ZKTeco does not assume responsibility for any errors or omissions in the information or documents which are referenced by or linked to this manual. The entire risk as to the results and performance obtained from using the information is assumed by the user.

ZKTeco in no event shall be liable to the user or any third party for any incidental, consequential, indirect, special, or exemplary damages, including, without limitation, loss of business, loss of profits, business interruption, loss of business information or any pecuniary loss, arising out of, in connection with, or

relating to the use of the information contained in or referenced by this manual, even if ZKTeco has been advised of the possibility of such damages.

This manual and the information contained therein may include technical, other inaccuracies or typographical errors. ZKTeco periodically changes the information herein which will be incorporated into new additions/amendments to the manual. ZKTeco reserves the right to add, delete, amend or modify the information contained in the manual from time to time in the form of circulars, letters, notes, etc. for better operation and safety of the machine/unit/equipment. The said additions or amendments are meant for improvement /better operations of the machine/unit/equipment and such amendments shall not give any right to claim any compensation or damages under any circumstances.

ZKTeco shall in no way be responsible (i) in case the machine/unit/equipment malfunctions due to any non-compliance of the instructions contained in this manual (ii) in case of operation of the machine/unit/equipment beyond the rate limits (iii) in case of operation of the machine and equipment in conditions different from the prescribed conditions of the manual.

The product will be updated from time to time without prior notice. The latest operation procedures and relevant documents are available on <http://www.zkteco.com>

If there is any issue related to the product, please contact us.

## ZKTeco Headquarters

**Address** ZKTeco Industrial Park, No. 32, Industrial Road,  
Tangxia Town, Dongguan, China.

**Phone** +86 769 - 82109991

**Fax** +86 755 - 89602394

For business related queries, please write to us at: [sales@zkteco.com](mailto:sales@zkteco.com).

To know more about our global branches, visit [www.zkteco.com](http://www.zkteco.com).

## About the Company

ZKTeco is one of the world's largest manufacturer of RFID and Biometric (Fingerprint, Facial, Finger-vein) readers. Product offerings include Access Control readers and panels, Near & Far-range Facial Recognition Cameras, Elevator/floor access controllers, Turnstiles, License Plate Recognition (LPR) gate controllers and Consumer products including battery-operated fingerprint and face-reader Door Locks. Our security solutions are multi-lingual and localized in over 18 different languages. At the ZKTeco state-of-the-art 700,000 square foot ISO9001-certified manufacturing facility, we control manufacturing, product design, component assembly, and logistics/shipping, all under one roof.

The founders of ZKTeco have been determined for independent research and development of biometric verification procedures and the productization of biometric verification SDK, which was initially widely applied in PC security and identity authentication fields. With the continuous enhancement of the development and plenty of market applications, the team has gradually constructed an identity authentication ecosystem and smart security ecosystem, which are based on biometric verification techniques. With years of experience in the industrialization of biometric verifications, ZKTeco was officially established in 2007 and now has been one of the globally leading enterprises in the biometric verification industry owning various patents and being selected as the National High-tech Enterprise for 6 consecutive years. Its products are protected by intellectual property rights.

## About the Manual

This manual introduces the Security PUSH Communication Protocol.

All figures displayed are for illustration purposes only. Figures in this manual may not be exactly consistent with the actual products.






## Document Conventions

Conventions used in this manual are listed below:

### GUI Conventions

For Software	
Convention	Description
<b>Bold font</b>	Used to identify software interface names e.g. <b>OK</b> , <b>Confirm</b> , <b>Cancel</b>
>	Multi-level menus are separated by these brackets. For example, File > Create > Folder.
For Device	
Convention	Description
< >	Button or key names for devices. For example, press <OK>
[ ]	Window names, menu items, data table, and field names are inside square brackets. For example, pop up the [New User] window
/	Multi-level menus are separated by forwarding slashes. For example, [File/Create/Folder].

### Symbols

Convention	Description
	This implies about the notice or pays attention to, in the manual
	The general information which helps in performing the operations faster
	The information which is significant
	Care taken to avoid danger or mistakes
	The statement or event that warns of something or that serves as a cautionary example.

## Revision History

Date	Version	Description	Revised By	Remarks
2021/01/06	V2.3	1. Add event code 70-79, 111-121, 219, 224, 243-248 in Appendix 2  2. Add VMSUserName, VMSPasswd parameters to upload in 7.4 Registration  3. Add new access control verification method rules  4. Add temperature measurement protocol  5. Add QR code encryption protocol  6. Add remote registration protocol	Cao Yanming	
2020/07/30	V1.8	1. Add subcontracting upgrade protocol switch parameter: SubcontractingUpgradeFunOn in 7.4 Registration  2. Add subcontracting upgrade protocol in 12.4.1 Upgrade  3. Add description of sitecode and linkid fields in 10.2 Upload Real-time Events	Cao Yanming	
2020/04/14	V1.7	1. Add event code 69, 233, 234, 237, 238, 239, 240, 241, 242 in Appendix 2 Description of Real-time Events	Cao Yanming	
2020/03/31	V1.6	1. Add position value 47, 48, 49 in 7.4 Registration  2. Add event code 63, 64, 65, 66, 67, 68, 109, 110, 235, 236 in Appendix 2 Description of Real-time Events  3. Add the 6 <sup>th</sup> ~8 <sup>th</sup> digits of the alarm field in	Cao Yanming	

		10.3 Upload Real-time Status		
2020/03/20	V1.5	1. Add hybrid identification protocol 2. Add deliver unified template protocol 3. Add delete unified template protocol 4. Modify the get comparison photo count protocol 5. Add get unified template count protocol 6. Modify the query comparison photo protocol 7. Add query unified template protocol 8. Modify the deliver comparison photo protocol 9. Add URL mode to deliver user photo protocol	Cao Yanming	
2019/08/02	V1.4	1. Added an error logging protocol	Li Xianping	
2019/05/10	V1.3	1. Added a protocol for pushing device parameters	Yan Guangtian	
2019/02/19	V1.2	1. Added a protocol for acquiring and querying comparison photos 2. Modified the document format and some errors	Yan Guangtian	
2018/10/10	V1.1	1. Added two communication encryption protocols: (1) exchange public key protocol (2) exchange factor protocol 2. Described the version of the supported communication encryption protocol: (1) Access control PUSH: 3.1.1 or later 3. Described the supported communication	Yan Guangtian	

		encryption protocol in details (see Appendix 16)		
2018/8/29	V1.0	1. Added protocols for visible light face recognition	Yan Guangtian	
2018/4/16	First edition		Li Xianping	



## Table of Contents

1 Abstract.....	12
2 Features .....	12
3 Encoding .....	12
4 About HTTP .....	12
5 Definition .....	14
6 Functions.....	15
6.1 Specification of Hybrid Identification Protocol .....	15
7 Process .....	18
7.1 Initialize Information Interaction .....	19
7.2 Exchange Public Keys (scenarios in which communication encryption is supported) .....	22
7.3 Exchange Factors (scenarios in which communication encryption is supported) .....	23
7.4 Registration .....	25
7.5 Download Configuration Parameters.....	33
7.6 Upload Device Parameters.....	36
8 Authorization.....	38
9 Heartbeat .....	39
10 Upload .....	40
10.1 Upload Method.....	40
10.2 Upload Real-time Events .....	40
10.3 Upload Real-time Status.....	43
10.4 Upload Returned Result of the Command .....	44
10.5 Upload User Information.....	46
10.6 Upload the Identity Card Information.....	48
10.7 Upload Fingerprint Template.....	52
10.8 Upload Comparison Photo.....	55
10.9 Upload Snapshot.....	57
10.10 Upload User Photo.....	59
10.11 Upload Integrated Template.....	61

10.12 Upload Error Log.....	64
11 Download.....	66
11.1 Download Cache Command.....	66
12 Details of Server Commands .....	68
12.1 Data.....	68
12.1.1 Update.....	68
12.1.2 Delete .....	100
12.1.3 Count .....	119
12.1.4 Query .....	124
12.2 Account .....	133
12.2.1 Pull SDK Device .....	133
12.2.2 Controller .....	135
12.3 Test Host.....	137
12.4 Control Devide .....	137
12.4.1 Upgrade.....	139
12.5 Configuration Class .....	143
12.5.1 Set Options .....	143
12.5.2 Get Options .....	146
12.6 Remote Registration .....	150
13 Remote Identification .....	151
Appendixes.....	154
Appendix 1 Description of Returned Values of Commands .....	154
Appendix 2 Description of Real-time Events.....	156
Appendix 3 Description of Verification Mode Code .....	162
Appendix 4 Language Number .....	163
Appendix 5 Algorithm to Convert to Values in Seconds.....	164
Appendix 6 Algorithm to Convert to Date .....	164
Appendix 7 Device Types and Corresponding Models .....	165
Appendix 8 APB Values .....	165
Appendix 9 Interlock Value.....	166

Appendix 10 Table of Access Control Parameters.....	166
Appendix 11 Table of Reader Parameters .....	167
Appendix 12 Table of Device Parameters.....	167
Appendix 13 Protocol Version Rule .....	168
Appendix 14 Parameter CmdFormat .....	169
Appendix 15 Multi-level Control Design sketch of multi-level control.....	170
Appendix 16 Data Encryption Schemes Data encryption and key exchange scheme .....	171
Appendix 17 Error Codes of Error Logs .....	174
Appendix 18 Biometric Type Index.....	175

## 1 Abstract

The PUSH protocol is a data protocol defined on the basis of the hypertext transfer protocol (HTTP) and is established on TCP/IP connections. It mainly controls the data interaction between the server and ZKTeco's devices, such as the T&A devices and access control devices. It defines the transmission format of data (including user information, biometric templates, and access control records) and the command format of control devices. At present, the protocol supports ZKTeco's WDMS, ZKECO, ZKNET, ZKBioSecurity3.0, and other servers as well as third-party servers such as Indian ESSL.

## 2 Features

- New data is actively uploaded.
- All actions, such as data upload and command delivered by the server, are all initiated by the client.

## 3 Encoding

Most of the data transmitted over the protocol are ASCII characters, but some fields, such as the user name, need to be encoded. This type of data is prescribed as follows:

- When the data is in Chinese, encode it with the GB2312 character set.
- When the data is in other languages, encode it with the UTF-8 character set.
- Currently, the following data should be encoded in this mode.
- User name in the user information table.

## 4 About HTTP

The PUSH protocol is a data protocol defined on the basis of HTTP. The following describes what is HTTP briefly. If you are familiar with HTTP, skip this section.

HTTP is a request/response protocol. The client sends to the server a request method, a URI, and a protocol version number, and then sends a MIME-like message that contains request modifiers, client information, and possible message body immediately. The server sends a status line to the client and then a MIME-like message that contains the server information, the entity metadata, and possible entity body immediately. The status line contains the protocol version number of the request and a success or failure error code. The following is an example:

**Client request:**

GET http://113.108.97.187:8081/iclock/accounts/login/?next=/iclock/data/iclock/ HTTP/1.1

User-Agent: Fiddler

Host: 113.108.97.187:8081

**Response from the server:**

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Date: Fri, 10 Jul 2015 03:53:16 GMT

Content-Type: text/html; charset=utf-8

Transfer-Encoding: chunked

Connection: close

Content-Language: en

Expires: Fri, 10 Jul 2015 03:53:16 GMT

Vary: Cookie, Accept-Language

Last-Modified: Fri, 10 Jul 2015 03:53:16 GMT

ETag: "c487be9e924810a8c2e293dd7f5b0ab4"

Pragma: no-cache

Cache-Control: no-store

Set-Cookie: csrftoken=60fb55cedf203c197765688ca2d7bf9e; Max-Age=31449600; Path=/  
Set-Cookie: sessionid=06d37fdc8f36490c701af2253af79f4a; Path=/  
0

HTTP communication usually occurs over TCP/IP connections. The default port is TCP 80, but other ports can also be used. However, HTTP communication can also be implemented over other protocols. HTTP communication only expects a reliable transmission and generally is established on the transmission layer protocol. Therefore, any protocol providing such a guarantee can be used.

## 5 Definition

The definition referred to in the document is in the format of \${ServerIP}.

- ServerIP is the IP address of the server.
- ServerPort is the port of the server.
- XXX is an unknown value.
- Value1\Value2\Value3.....\Valuen means value 1\value 2\value 3.....\value n.
- Required means a required value.
- Optional means an optional value.
- SerialNumber is the serial number which can be letters, numbers, or a combination of letters and numbers.
- NUL: null (\0).
- SP means a space.
- LF means a line feeder (\n).
- CR means a carriage return (\r).
- HT means a horizontal tab (\t).
- DataRecord means a data record.
- CmdRecord means a command record.
- CmdID means the command number.
- CmdDesc means the command description.
- Reserved means a reserved field.
- BinaryData means a binary data stream.
- Base64Data means a Base64-based data stream.
- TableName means the name of the data table.
- Key means the key.
- Value means the value.
- FilePath means the file path.
- URL means the resource location.
- Cond: means the condition.

## 6 Functions

The following describes functions supported by the PUSH protocol for the client.

- [Initialize information interaction](#)
- [Upload](#)
- [Download](#)
- [Server command details](#)
- [Appendixes](#)

### 6.1 Specification of Hybrid Identification Protocol

With more and more types of biometrics, the instructions issued by different types of biometrics are also different, making software docking protocols very difficult.

In order to simplify the development process, the specifications for biological template/ photo issue/ upload/ query/ delete are unified.

#### **Hybrid identification protocol docking process:**

- 1) The device pushes the following 5 parameters to the server through registration interface: MultiBioDataSupport, MultiBioPhotoSupport, MultiBioVersion, MaxMultiBioDataCount, MaxMultiBioPhotoCount. See [Registration] interface description for details.
- 2) The server issues the following two parameters to the device through the [Download Configuration Parameters] interface: MultiBioDataSupport, MultiBioPhotoSupport.
- 3) Both the device and the server will determine the finally supported hybrid identification template/ photo type based on the MultiBioDataSupport and MultiBioPhotoSupport parameters pushed by each other.

For example:

Device side: MultiBioDataSupport=0:1:0:0:0:0:0:0:1, MultiBioPhotoSupport =0:0:0:0:0:0:0:0:1

Server side: MultiBioDataSupport=0:0:0:0:0:0:0:0:1, MultiBioPhotoSupport= 0:0:0:0:0:0:0:0:1

The device supports fingerprint templates, visible light face templates, and visible light face photos. The software supports face templates and visible light face photos. Because the software does not support fingerprint templates, finally after the device docking with the software, it only support visible light face templates and visible light face photos.

**Hybrid identification protocol unified upload/ issue bio-templates format:**

After successfully connecting to the hybrid identification protocol, a unified template format can be used for the types supported by the device and the server.

1) The server issues the templates to the device

Unified use of [Issue Unified Templates] interface.

2) The server issues the photos to the device

Unified use of [Issue Comparison Photos] interface.

3) The server queries the template data

Unified use of [Query Unified Templates] interface.

4) The sever queries the quantity of templates

Unified use of [Query the Quantity of Unified Templates] interface.

5) The device uploads the templates to the server

Unified use of [Upload Unified Templates] interface.

6) The device uploads the comparison photos to the server

Unified use of [Upload Comparison Photos] interface.

**Hybrid identification protocol unified upload templates/ photos quantity interface:**

1) For devices that support hybrid identification protocol, the maximum number of templates/ photos supported by the current device will be pushed to the server at the registration interface: MaxMultiBioDataCount, MaxMultiBioPhotoCount.

2) The server can get the number of photos/templates saved by the current device by [Get comparison photo count] and [Get unified template count].

**Hybrid identification protocol specification real-time upload of unified templates and photos:**

1) The bio-templates/ comparison photos registered by the device will be uploaded to the server in real time.

Upload interface refer to [upload unified templates] and [upload comparison photos].

2) You can use PostBackTmpFlag to specify whether you want the device to return the unified templates when the software issues the comparison photos.

For specific interface, please refer to [Issue Comparison Photos].

**Hybrid identification protocol provides optimization strategies:**

For devices that support both templates and photos issuing, the server can determine the device



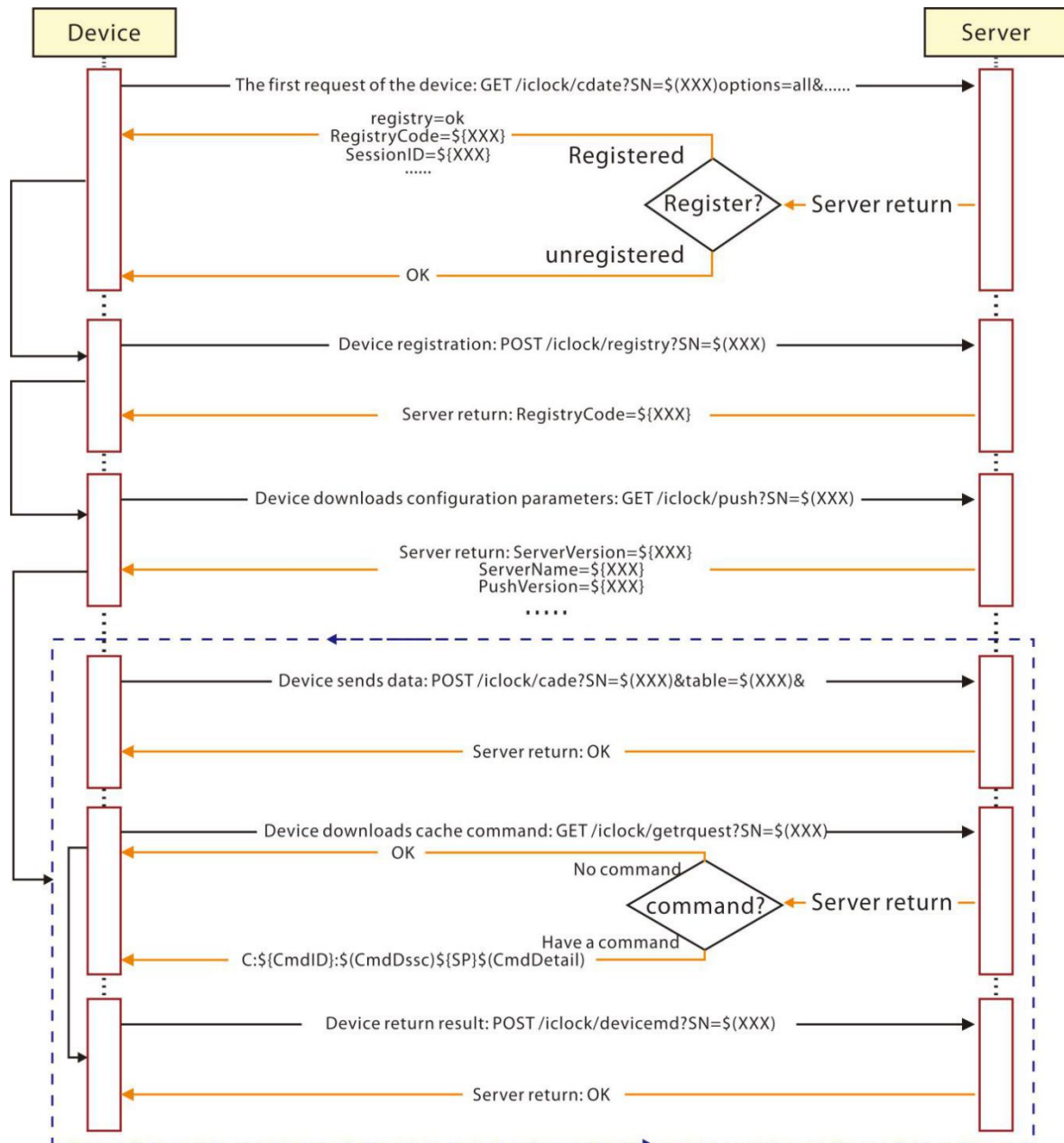
template version number based on the MultiBioVersion parameter uploaded by the device. If the server has saved the template of the current version number, the template can be issued first instead of comparison photos.

**Note:** To issue the comparison photos, the device needs to extract photos into templates, which is less efficient than directly issuing templates.

ZKTECO

## 7 Process

In PUSH mode, the client must first initiate a request to initialize information interaction and can use other functions only after the request is successful, such as uploading data, obtaining server commands, uploading update information, and replying to the server commands. These functions can be used in any sequence, specifically depending on the development of the client applications, as shown in the following figure:



## 7.1 Initialize Information Interaction

After the server receives the request sent by the client, two kinds of procedures are available depending on whether the device has been registered. If the device has been registered, the server returns the registration code and its configuration parameters and completes the connection. If the device is not registered, the client needs to initiate a registration request and send device parameters to the server. After the device is registered, the server returns the registration code. Then, the client sends a request to download the server configuration parameters.

The interaction is successful after the client obtains the configurations. The details are as follows:

Connection request of the client.

Application scenario:

If the device has not connected to the software, it needs to send a connection request to create a connection.

Format of the client's connection request message: (compatible with the old protocol):

```
GET /iclock/cdata?SN=$(SerialNumber)&pushver=${XXX}&options=all&${XXX}=${XXX} HTTP/1.1
```

Host: \${ServerIP}:\${ServerPort}

...

Note:

HTTP request method: GET

URI: /iclock/cdata

HTTP version: 1.1

Client configurations:

SN: \${Required} is the SN of the client (the same below).

pushver: \${Optional} is the latest version of the PUSH protocol of the device. For details, see Appendix 13.

options: \${Required} means to obtain the server configuration parameters. Currently, only the value all is supported.

Host header: \${Required} (the same below)

Other headers: \${Optional}

If the device has been registered, the next step is to send a request to download the configuration parameters; otherwise, the next step is to send a registration request and then send a request to download the configuration parameters after the device is registered. Currently, the client needs to be registered for each connection.

\${XXX}: It is pushed only when this parameter has been set for the device, for example, DeviceType.

If the device is not registered, the format of the normal server response is as follows:

```
HTTP/1.1 200 OK
```

```
Server: Apache-Coyote/1.1
Content-Length: ${XXX}
Date: ${XXX}
OK
```

If the device has been registered, the format of the normal server response is as follows:

```
HTTP/1.1 200 OK
Server: ${XXX}
Content-Length: ${XXX}
Date: ${XXX}
...
registry=ok
RegistryCode=${XXX}
ServerVersion=${XXX}
ServerName=${XXX}
PushProtVer=${XXX}
ErrorDelay=${XXX}
RequestDelay=${XXX}
TransTimes=${XXX}
TransInterval=${XXX}
TransTables=${XXX}
Realtime=${XXX}
SessionID=${XXX}
TimeoutSec=${XXX}
```

Note:

registry: OK means that the device has been registered.

RegistryCode: A random number generated by the server, up to 32 bytes.

ServerVersion: The version of the server

ServerName: The name of the server

PushProtVer: The protocol version based on which the server is developed. For details, see Appendix 13.

ErrorDelay: The interval (in seconds) at which the client reconnects to the server after networking failure. The recommended range is the 30s to 300s. If no value is configured at the client, the default value 30s are used.

RequestDelay: The interval (in seconds) at which the client sends the command acquiring request. If no value is configured at the client, the default value 30s are used.

TransTimes: The time point in which the newly transmitted data is regularly checked. If no value is configured, the default value 12:30;14:30 is used.

**TransInterval:** The interval (in minutes) at which the system checks whether any new data needs to be transmitted. If no value is configured at the client, the default value 2 min is used.

**TransTables:** The new data that needs to be checked and uploaded. The default value is User Transaction, which means the user and access control records need to be automatically uploaded.

**Realtime:** Specifies whether the client transmits new records in real time. 1 means that any new data is transmitted to the server. 0 means that the data is transmitted according to the values of TransTimes and TransInterval. If no value is configured at the client, the default value 1 is used.

**SessionID:** The ID of the PUSH communication session. This field is used to calculate the new Token for all subsequent requests from the device.

**TimeoutSec:** The network timeout time. If no value is configured at the client, the default value 10s is used.

If the device has been registered, the software will and must return the registry and RegistryCode; otherwise, the software does not return the registry and RegistryCode.

**HTTP status line:** It is defined by the standard HTTP.

**HTTP response header:**

**Date header:** \${Required} This header is used to synchronize the server time in the GMT format. For example, Date: Fri, 03 Jul 2015 06:53:01 GMT

**Content-Length header:** According to HTTP 1.1, this header is generally used to specify the length of the response entity. If you do not know the length, you can also set Transfer-Encoding to chunked. Both Content-Length and Transfer-Encoding are headers defined by standard HTTP, which are not described in detail here.

Example:

**Connection request of the client:**

GET /iclock/cdata?SN=3383154200002&pushver=3.0.1&options=all HTTP/1.1

Host: 192.168.213.17:8088

User-Agent: iClock Proxy/1.09

Connection: starting

Accept: application/push

Accept-Charset: UTF-8

Accept-Language: VoicePrint-CN

**If the device is not registered, the normal server response is as follows:**

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Content-Length: 2

Date: Mon, 09 Jan 2017 02:20:19 GMT

OK

## 7.2 Exchange Public Keys (scenarios in which communication encryption is supported)

Application scenario:

The device pushes its public key to the server and receives the server's public key from the server.

Client request message:

```
POST /iclock/exchange?SN=${SerialNumber}&type=publickey
Host: ${ServerIP}:${ServerPort}
Content-Length: ${XXX}
...
PublicKey=${XXX}
```

Note:

```
HTTP request method: POST
URI: /iclock/exchange
HTTP version: 1.1
Host header: ${Required}
Other headers: ${Optional}
PublicKey: Call the device public key that is returned by the encryption library.
```

Response from the server:

```
HTTP/1.1 200 OK
Server: ${XXX}
Set-Cookie: ${XXX}; Path=/; HttpOnly
Content-Type: application/push;charset=UTF-8
Content-Length: ${XXX}
Date: ${XXX}
PublicKey=${XXX}
```

Note:

PublicKey: The server's public key returned by the server.

Example:

**Client request message:**

```
POST /iclock/exchange?SN=ODG7030067031100001&type=publickey HTTP/1.1
Cookie: token=e6bc9a9c2f9ce675e3548d5aeda2777e
Host: 192.168.52.44:8088
User-Agent: iClock Proxy/1.09
Connection: starting
```

Accept: application/push

Accept-Charset: UTF-8

Accept-Language: zh-CN

Content-Type: application/push;charset=UTF-8

Content-Language: zh-CN

Content-Length: 3580

PublicKey=DMCtR13RwiGI4M9TRn/3xEmkddz2lqoZR7zUrOMhOc3FLvhLtpIW3REOSaKT4A9WO0ONt3V+mVb0W3Ka3NeCTWljf9LpIV1EyJloZwXspGroPMTEWitLE+LLsrO1r47OQRr62j5YSViUDKgZLVCvEek2iJ+3D181Z3qxV7a7WloQ9DUGiPaU8gml4cmiyqQimxlQ1wwMcMpcIFOlSx7UjCG+D41dM/vh5UZxrQwn7liMOmNdFXIB+TOjaJ+4K/n3TDjubrbebx6H2+nErH1mBuCCSNIKfwc5eakNfXPuqgBNGqCFJojgcQiOySquaQ2DFXdUwYBLIURDnBLf+TtoSh4=

#### Response from the server:

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Content-Length: 590

Date: Mon, 10 Sep 2018 10:10:55 GMT

PublicKey=DMCvFlzRwiGI4M9TRn/3xEmkddz2lqoZR7zUrOMhOc3FLvhLtpIYu3ZMNSeKVLcZUv4iHNNxzI9B8SfuVSxXAwijYAj6Wg4YyxTj4stv4K7q54sUCikb1CbQ/H0m9QZGyhM1WjrHhpxT/CsOAquEy/2gxfrSt3nai38Hb/8QoTHvnXJR2EVpcY6u47jBeGiXM3ZUQgCtcdB7JBXsOr71XWEsLX1fIC3GofGCy0g0bUkumWJfNkwBwFwZb95o6klDi8uP/wU+DS1uLs1VcCN0WNTX+DCajyzcYvecR8cgbs0F1QfMmiRr/dYAOkwF/bSMYuLkd+o6FmLBAh9keFtgkFa+PC5RIFGrmxpJx4lMoLfaNqUNwyAuRdKevYBDUrRhGwgtwo/BRGUoWCeOB4YP/gHHGro0M8f3/HISqliuT55Xks/Btp1tpfO/OeJjELUA9Yu0o4TQlNm19PuOGsYhipM9NeGGexKjtotHotLT4Ccs004nAf7TltDavoPvVGJGiDbnN7l8wsUCsqcCRsiKhpmON2waLjdFa8PNJ62N6DI6QRPKn9XLnIDfdtKSq5Vgn

## 7.3 Exchange Factors (scenarios in which communication encryption is supported)

Application scenario:

The device pushes its factors to the server and receives the server's factors from the server.

Client request message:

POST /iclock/exchange?SN=\${SerialNumber}&type=factors

Host: \${ServerIP}: \${ServerPort}

Content-Length: \${XXX}

...

Factors=\${XXX}

**Note:**

HTTP request method: POST

URI: /iclock/exchange

HTTP version: 1.1

Host header: \${Required}

Other headers: \${Optional}

Factors: Call the device factors that are returned by the encryption library.

**Response from the server:**

HTTP/1.1 200 OK

Server: \${XXX}

Set-Cookie: \${XXX}; Path=/; HttpOnly

Content-Type: application/push;charset=UTF-8

Content-Length: \${XXX}

Date: \${XXX}

Factors=\${XXX}

**Note:**

Factors: The server's factors returned by the server.

**Example:****Client request message:**

POST /iclock/exchange?SN=ODG7030067031100001&type=factors HTTP/1.1

Cookie: token=e6bc9a9c2f9ce675e3548d5aeda2777e

Host: 192.168.52.44:8088

User-Agent: iClock Proxy/1.09

Connection: starting

Accept: application/push

Accept-Charset: UTF-8

Accept-Language: zh-CN

Content-Type: application/push;charset=UTF-8

Content-Language: zh-CN

Content-Length: 352

Factors=D/VtnltwfrABVfyAJku6jKdobqCUIz2eTJ0yXlwl8ZMi5wSylQePPhVDGQvrppcssZ/zgX6qAWSKUXV  
IGRXNQ6kBk7+Kc6Sal090cvvMeLHCc1h69TlsbfkMtLGd1npscnDmmOoC19qqIbtTha9zNHmf38dHDkGZf3  
vLv/I8iwbQV3RAX7MalBW4M+kYvbdYNgzR5kqG+wTZOet5QAf/8YoRg7qZmnkAG6sAYALQotTfwQcjGUct  
VoJONw8WshzkpdzgNsoo7UtYEmmhbEPHtqgaDN5OLexoE9u6lp3gMd+V2hf70rs+mVUYR6frkEKe/6rA+o  
Idguhb2lp6HnwfNQ==

**Response from the server:**

HTTP/1.1 200 OK



Server: Apache-Coyote/1.1

Content-Length: 180

Date: Mon, 10 Sep 2018 10:10:55 GMT

Factors=XQ10e0WiFtsIJW5ob221T/WCK42GXGP6mmiBB9yB93rD3CxIKZo3mavyqfTKFxtCn8AtkxL7MH4U  
eRvRnFTrv3Q4kKaYndiiphuvxOGQxrzcGGjH0sRzgPcTtAQu0U7A8vg2sMzZOxokqLuDVE5nlsx1/1V46wTK+  
oNU9q8fgKM=

## 7.4 Registration

Application scenario:

After the client sends a connection request, if no registration code is returned (which means that the device is not registered), the client needs to send a registration request to register the device.

Client request message:

POST /iclock/registry?SN=\${SerialNumber} HTTP/1.1

Host: \${ServerIP}:\${ServerPort}

Content-Length: \${XXX}

...

DeviceType=acc,~DeviceName=\${XXX},FirmVer=\${XXX},PushVersion=\${XXX},MAC=\${XXX},CommType=\${XXX},MaxPackageSize=\${XXX},LockCount=\${XXX},ReaderCount=\${XXX},AuxInCount=\${XXX},AuxOutCount=\${XXX},MachineType=\${XXX},~IsOnlyRFMachine=\${XXX},~MaxUserCount=\${XXX},~MaxAttLogCount=\${XXX},~MaxUserFingerCount=\${XXX},MThreshold=\${XXX},IPAddress=\${XXX},NetMask=\${XXX},GATEIPAddress=\${XXX},~ZKFPVersion=\${XXX},~REXInputFunOn=\${XXX},~CardFormatFunOn=\${XXX},~SupAuthzizeFunOn=\${XXX},~ReaderCFGFunOn=\${XXX},~ReaderLinkageFunOn=\${XXX},~RelayStateFunOn=\${XXX},~Ext485ReaderFunOn=\${XXX},~TimeAPBFunOn=\${XXX},~CtlAllRelayFunOn=\${XXX},~LossCardFunOn=\${XXX},SimpleEventType=\${XXX},VerifyStyles=\${XXX},EventTypes=\${XXX},DisableUserFunOn=\${XXX},DeleteAndFunOn=\${XXX},LogIDFunOn=\${XXX},DateFmtFunOn=\${XXX},DelAllLossCardFunOn=\${XXX},AutoClearDay=\${XXX},FirstDelayDay=\${XXX},DelayDay=\${XXX},StopAllVerify=\${XXX},FvFunOn=\${XXX},FaceFunOn=\${XXX},FingerFunOn=\${XXX},CameraOpen=\${XXX},AccSupportFunList=\${XXX},AutoServerFunOn=\${XXX},DelayOpenDoorFunOn=\${XXX},UserOpenDoorDelayFunOn=\${XXX},MultiCardInterTimeFunOn=\${XXX},OutRelaySetFunOn=\${XXX},MachineTZFunOn=\${XXX},DSTFunOn=\${XXX},CardSiteCodeFunOn=\${XXX},MultiCardUserFunOn=\${XXX},UserNameFunOn=\${XXX},StringPinFunOn=\${XXX},MaxLockCount=\${XXX},MaxZigbeeCount=\${XXX},MaxMCUCardBits=\${XXX},SupportReaderType=\${XXX},CmdFormat=\${XXX},authKey=\${XXX},MultiStageControlFunOn=\${XXX},MasterControlOn=\${XXX},SubControlOn=\${XXX},BioPhotoFun=\${XXX},BioDataFun=\${XXX}

Note: HTTP request method: POST URI: /iclock/registry HTTP version: 1.1

DeviceType: T&A module: att, security module: acc

DeviceName: The name of the device

FirmVer: The version of the firmware

PushVersion: The version of PUSH SDK

MAC: The MAC address of the device

CommType: The communication type. The following types are supported:

Ethernet: wired communication

usb-4G-modem: communication over 4G networks

Serial-wireless: Wi-Fi communication over serial ports and Sdio

MaxPackageSize: The maximum size of the communication package

LockCount: The number of the doors

ReaderCount: The number of readers

AuxInCount: The number of auxiliary inputs

AuxOutCount: The number of auxiliary outputs

MachineType: The machine model. PULL SDK Device 101

~IsOnlyRFMachine: Specify whether only the enable/disable parameter of the RF card is supported.

~MaxUserCount: The maximum number of users or cards

~MaxAttLogCount: The maximum number of attendance records

~MaxUserFingerCount: The maximum number of fingers of a single user

MThreshold: Finger 1: The N match threshold

IPAddress: The IP address of the wired network

NetMask: The subnet mask of the wired network

GATEIPAddress: The gateway address of the wired network

~ZKFPVersion: The version of the finger algorithm

~REXInputFunOn: Specify whether to enable the exit door locking function. This parameter is not supported by the PULL SDK device.

~CardFormatFunOn: Specify whether to enable the custom card format function for the Wiegand reader.

~SupAuthrizeFunOn: Specify whether to enable the super user function.

~Ext485ReaderFunOn: The external reader 485

~TimeAPBFunOn: Specify whether to enable the APB time function.

~CtlAllRelayFunOn: Specify whether to enable the All Relay function.

~LossCardFunOn: Specify whether to enable the loss card function.

SimpleEventType: Event merging

VerifyStyles: The supported verification mode. It contains 32 bits in total. Each of the first 16 bits corresponds to a different combination of verification modes. For details, see Appendix 3 Description of Verification Mode Code. 1 indicates that the verification mode is supported, and 0 indicates that the verification is not supported. 200 indicates an access control event.

EventTypes: The supported event type, which contains 32 bytes (0 to 31) in total. Each byte contains 8 bits, represented by 2 hexadecimal numbers, a total of 256 bits (0 to 255). For the function represented by each bit, see Appendix 2 Description of Real-time Events. 1 indicates that the event is supported and 0 indicates that the event is not supported.

Take BF0FE03D300001000000000070000000000000000000000077002001000000 as an example. The 0th byte is BF, which is 11111101 in binary mode (the lower bit is placed first), the 6th bit is 0 and the other bits are 1. It means that in the function controlled by the first eight bits, only the linkage event represented by the 6th bit is not supported, and other events are supported.

DisableUserFunOn: Specify whether to enable the blacklist function.

DeleteAndFunOn: Specify whether to enable the AND deletion function.

LogIDFunOn: Specify whether to enable the ID logging function.

DateFmtFunOn: Specify whether to enable the date format function (the expiry date in the control user table).

DelAllLossCardFunOn: Specify whether to enable the function of deleting all blacklists.

**AutoClearDay:** The interval at which the event logs are automatically cleared

FirstDelayDay: The time for authorizing an unauthenticated user to open the door for the first time after registration

DelayDay: The additional time for authorizing an unauthenticated user to open the door for the first time after registration

**StopAllVerify:** Stop all the verification functions and prohibit opening the door with the card.

FvFunOn: Specify whether to enable the finger vein recognition function.

FaceFunOn: Specify whether to enable the face recognition function.

FingerFunOn: Specify whether to enable the finger recognition function.

CameraOpen: Specify whether to enable the camera function.

**AutoServerFunOn:** Specify whether to enable the remote identification function.

DelayOpenDoorFunOn: Specify whether to enable the delay open door function. This function is not supported.

UserOpenDoorDelayFunOn: Specify whether to enable the function of delaying the door open time for a user. This function is not supported.

MultiCardInterTimeFunOn: Specify whether to enable the function of setting the interval for multiple cards. This function is not supported.

OutRelaySetFunOn: Specify whether to enable the configuration output function. This function is not supported.

MachineTZFunOn: Specify whether to enable the function of setting a time zone for the machine. This function is not supported.

DSTFunOn: Specify whether to enable the DLST function. This function is not supported.

CardSiteCodeFunOn: Specify whether to enable the card sitecode function.

**MulCardUserFunOn:** Specify whether to enable the multi-card user function, which is used by the device to judge the service logic.

UserNameFunOn: Specify whether to enable the user name function.

StringPinFunOn: Specify whether to support the string-type user ID.

MaxLockCount: The maximum number of doors

MaxZigbeeCount: The maximum number of Zigbee

MaxMCUCardBits: The maximum number of card bits in binary

SupportReaderType: The type of supported reader

CmdFormat: The format of the command. For details, see Appendix 14.

MultiStageControlFunOn: The multi-level control parameter. For details, see Appendix 15.

MasterControlOn: The multi-level control parameter. For details, see Appendix 15.

SubControlOn: The multi-level control parameter. For details, see Appendix 15.

BioPhotoFun: Specify whether to support the comparison photo parameter.

BioDataFun: Specify whether to support the visible light face template parameter.

UserPicURLFunOn: Supports issuing user photos by URL.

MultiBioDataSupport: Supports multi-modal bio-template parameters. The type is defined bit by bit. Different types are separated by colons, 0 means not supported, 1 means supported. The supported version number, such as: 0: 1: 1: 0: 0: 0: 0: 0: 0, indicating support for fingerprint template and near-infrared face template.

MultiBioPhotoSupport: Supports multi-modal biometric photo parameters. The type is defined bit by bit. Different types are separated by colons, 0 means not supported, 1 means supported. The supported version number, such as: 0: 1: 1: 0: 0: 0: 0: 0: 0, indicating support for fingerprint photo and near-infrared face photo.

MultiBioVersion: The multi-modal biometric data version. Different types are separated by colons, 0 means not supported, 1 means supported. The supported version number, such as: 0: 10: 0: 7: 0: 0: 0: 0: 0, indicating support for fingerprint algorithm10.0 and near-infrared face algorithm7.0.

MaxMultiBioDataCount: Supports maximum number of multi-modal bio-templates. The type is defined bit by bit. Different types are separated by colons, 0 means not supported, 1 means supported. The supported maximum number of templates, such as: 0: 10000: 2000: 0: 0: 0: 0: 0: 0, indicating support for the maximum number of fingerprint templates is 10000 and the maximum number of near-infrared face templates is 2000.

MaxMultiBioPhotoCount: Supports maximum number of multi-modal biometric photos. The type is defined bit by bit. Different types are separated by colons, 0 means not supported, 1 means supported. The supported maximum number of photos, such as: 0: 10000: 2000: 0: 0: 0: 0: 0: 0, indicating support for the maximum number of fingerprint photos is 10000 and the maximum number of near-infrared face photos is 2000.

authKey: The communication key.

SubcontractingUpgradeFunOn: Subcontract upgrade protocol function switch parameter.

IRTempDetectionFunOn: The infrared temperature detection function is turned on.

MaskDetectionFunOn: Mask detection function is on.

VMSUserName: vms login user account

VMSPasswd: vms login password

NewVFStyles: The device supports new verification methods

A total of 16 digits, currently only supports up to 10 digits, issued as a string.

```
typedef enum _VERIFY_STYLE_E
{
    VF_STYLE_NONE = 0, //0: Invalid bit
    VF_STYLE_FACE = 1, //1: Face
    VF_STYLE_PALM_PRINT = 2, //2: Palmprint
    VF_STYLE_PALM_VEIN = 3, //3: Palm vein
    VF_STYLE_FP = 4, //4: Fingerprint
    VF_STYLE_VEIN = 5, //5: Finger vein
    VF_STYLE_VP = 6, //6: Vocal print
    VF_STYLE_IRIS = 7, //7: Iris
    VF_STYLE_RETINA = 8, //8: Retina
    VF_STYLE_PW = 9, //9: Password
    VF_STYLE_PIN = 10, //10: User ID
    VF_STYLE_RF = 11, //11: Card
}VERIFY_STYLE_E;
```

Such as:

The device supports face, palm vein, fingerprint, password, UserID, card, then upload 0000111000011010

The device supports face, palm vein, password, card, then upload 0000101000001010

IsSupportQRcode: Whether the device supports the QR code function, the value is as follows:

0: QR code is not supported

1: Only supports QR code display

2: Only supports QR code recognition

3: Support QR code display + QR code recognition function

QRCodeEnable: Whether to enable the QR code function (0: off, 1: on)

QRCodeDecryptFunList: QR Code Decryption Function Parameter. This parameter is used to determine which decryption methods the device specifically supports, and the function support parameters are obtained according to the bits (the parameter cannot be modified in software). If it is not transmitted, it is not supported by default.

The position value refers to the string position of the **QRCodeDecryptFunList** parameter value, starting from 0 and from left to right, as follows:

Position value	Meaning	Remarks
0	Device supports scheme 1 decryption	0 not supported; 1 supported
1	Device supports scheme 2 decryption	0 not supported; 1 supported
2	Device supports scheme 3 decryption	0 not supported; 1 supported

Remarks:

Scheme 1: Use the date of the day as the key and use the AES256 algorithm for encryption (the key is fixed)

Scheme 2: the system randomly generates a key and encrypts it with the AES256 algorithm (the key is not fixed)

Scheme 3: The system randomly generates a key and encrypts it with the RSA1024 algorithm (public and private keys are not fixed)

For example:

QRCodeDecryptFunList=101, which means that the device supports scheme one and three decryption methods.

AccSupportFunList: Obtain the parameter supported by the function based on the bit. This parameter cannot be modified in the software. If no value is set, 0 is used for all the bits.

Note: The position value means the value of the bit in the string of the AccSupportFunList value. It starts at 0 and is counted from the left to the right.

Position	Value	Remarks
0	0: not supported; 1: supported.	Reader disabling
1	0: not supported; 1: supported.	485 reader encryption
2	0: not supported; 1: supported.	Door locking
3	0: not supported; 1: supported.	Emergency double On or Off
4	0: not supported; 1: supported.	Read/green Wiegand reader indicator corresponding to the door sensor/relay status
5	0: not supported; 1: supported.	Long name
6	0: not supported; 1: supported.	Wiegand format with sitecode
7	0: not supported; 1: supported.	Multiple cards for one user. This parameter is used by the software to judge the service logic. When the value is 1, the software delivers the device setting MulCardUserFunOn=1.
8	0: not supported; 1: supported.	The auxiliary input and exit button are controlled by time
9	0: not supported; 1: supported.	Automatically delete the data of temporary user each day
10	0: not supported; 1: supported.	Support the wg test function
11	0: not supported; 1: supported.	Support the ACCOUNT command
12	0: not supported; 1: supported.	Different verification modes for different users in different periods of time
13	0: not supported; 1: supported.	Control the 485 reader Led
14	0: not supported; 1: supported.	Whether the door supports multiple Wiegand formats.
Reserved		
15	0: not supported; 1: supported.	The door id in the privilege table indicates one door
16	0: not supported; 1: supported.	Independent user super privilege table
17	0: not supported; 1: supported.	Support adding door properties
18	0: not supported; 1: supported.	Reader properties
19	0: not supported; 1: supported.	Auxiliary input property table
20	0: not supported; 1: supported.	Auxiliary output property table

21	0: not supported; 1: supported. Door parameter table
22	0: not supported; 1: supported. APB table
23	0: not supported; 1: supported. Interlock table
24	0: not supported; 1: supported. Wireless-serial function
25	0: not supported; 1: supported. 4G-USB communication function
26	0: not supported; 1: supported. Whether the table supports the DevID field, or whether the control protocol contains the DevID field
27	0: not supported; 1: supported. Whether the slave is supported
28	0: not supported; 1: supported. Whether dual network cards are supported
29	0: not supported; 1: supported. Whether the authorization table is supported
30	0: not supported; 1: supported. Whether identity card registration is supported
31	0: not supported; 1: supported. Whether the master-slave switch is supported
32	0: not supported; 1: supported. Whether the device supports mixed readers (Wiegand/485 reader)
33	0: not supported; 1: supported. Whether Wiegand data processing, byte swap, and bit reversal are supported
34	0: not supported; 1: supported. Whether desfire control card is supported
35	0: not supported; 1: supported. Whether the two-dimensional code command can be delivered
36	0: not supported; 1: supported. Whether zksq200 can be added to the advertisement delivery command
37	0: not supported; 1: supported. Whether zksq200 can be added to the indoor device
38	0: not supported; 1: supported. Whether comparison photos can be uploaded
39	0: not supported; 1: supported. Whether the integrated visible light face template can be uploaded
40	0: not supported; 1: supported. Whether the device can be remote registered
41	0: not supported; 1: supported. Whether the device supports setting the verification result format
42	0: not supported; 1: supported. Whether the device supports setting the verification server
43	0: not supported; 1: supported. Whether the device supports delivering the resource files, such as voice files, boot screen, welcome page, and screensaver page
44	0: not supported; 1: supported. Whether the device can connect to an AI device (for non-inbio5-series devices that do not support the reader property table, such as inbioPro)
45	0: not supported; 1: supported. Whether the device supports the expansion board dm10
46	0: not supported; 1: supported. Whether the device supports the expansion board aux485
47	0: not supported; 1: supported. Whether the device supports the expansion board ex0808
48	0: not supported; 1: supported. Whether the device supports the function of allowing super users to pass through when the door is locked

49      0: not supported; 1: supported. Whether the device supports osdp protocol expansion board

The normal server response is as follows:

```
HTTP/1.1 200 OK
Server: ${XXX}
Set-Cookie: ${XXX}; Path=/; HttpOnly
Content-Type: application/push;charset=UTF-8
Content-Length: ${XXX}
Date: ${XXX}
RegistryCode=${XXX}
```

Note:

RegistryCode: A random number generated by the server, up to 32 bytes.  
If registration fails, 406 instead of the registration code is returned.

Example:

**Client request message:**

```
POST /iclock/registry?SN=3383154200002 HTTP/1.1
Host: 192.168.213.17:8088
User-Agent: iClock Proxy/1.09
Connection: starting
Accept: application/push
Accept-Charset: UTF-8
Accept-Language: zh-CN
Content-Type: application/push;charset=UTF-8
Content-Language: zh-CN
Content-Length: 855
DeviceType=acc,~DeviceName=F20/M,FirmVer=Ver 8.0.1.3-20151229,PushVersion=Ver
2.0.22-20161201,CommType=ethernet,MaxPackageSize=2048000,LockCount=1,ReaderCount=2,AuxInCo
unt=0,AuxOutCount=0,MachineType=101,~IsOnlyRFMachine=0,~MaxUserCount=50,~MaxAttLogCount=
10,~MaxUserFingerCount=10,MThreshold=60,IPAddress=192.168.213.221,NetMask=255.255.255.0,GATEI
PAddress=192.168.213.1,~ZKFPVersion=10,IclockSvrFun=1,OverallAntiFunOn=0,~REXInputFunOn=0,~Ca
rdFormatFunOn=0,~SupAuthrizeFunOn=0,~ReaderCFGFunOn=0,~ReaderLinkageFunOn=0,~RelayStateF
unOn=1,~Ext485ReaderFunOn=0,~TimeAPBFunOn=0,~CtlAllRelayFunOn=0,~LossCardFunOn=0,SimpleE
ventType=1,VerifyStyles=ff7f0000,EventTypes=BF0FE03D300001000000000007000000000000000000
00077002001000000,DisableUserFunOn=0,DeleteAndFunOn=0,LogIDFunOn=0,DateFmtFunOn=0,DelAll
LossCardFunOn=0,AutoClearDay=0,FirstDelayDay=0,DelayDay=0,StopAllVerify=0,FvFunOn=0,FaceFunO
n=0,FingerFunOn=1,CameraOpen=1,AccSupportFunList=0101010000111000000111010100011010,Auto
ServerFunOn=1,DelayOpenDoorFunOn=1,UserOpenDoorDelayFunOn=1,MultiCardInterTimeFunOn=1,O
```



```
utRelaySetFunOn=1,MachineTZFunOn=1,DSTFunOn=1,CardSiteCodeFunOn=1,MulCardUserFunOn=1,Us
erNameFunOn=1,StringPinFunOn=1,MaxLockCount=4,MaxZigbeeCount=3,MaxMCUCardBits=37,Suppor
tReaderType=1,authKey=dassas
```

**Response from the server:**

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Set-Cookie: JSESSIONID=30BFB04B2C8AECC72C01C03BFD549D15; Path=/; HttpOnly

Content-Type: text/plain;charset=ISO-8859-1

Content-Length: 23

Date: Mon, 09 Jan 2017 01:31:59 GMT

RegistryCode=Uy47fxftP3

## 7.5 Download Configuration Parameters

Application scenario:

After the registration code is returned after the registration request is successful in the previous step, the device needs to actively obtain the server configuration parameters and then the whole initialization process is finished.

Client request message:

POST /iclock/push?SN=\${SerialNumber} HTTP/1.1

Cookie: token=\${XXX}, timestamp=\${XXX}

Host: \${ServerIP}: \${ServerPort}

...

Response from the server:

HTTP/1.1 200 OK

Server: \${XXX}

Content-Type: application/push;charset=UTF-8

Content-Length: \${XXX}

Date: \${XXX}

...

ServerVersion=\${XXX}

ServerName=\${XXX}

ErrorDelay=\${XXX}

RequestDelay=\${XXX}

TransTimes=\${XXX}

TransInterval=\${XXX}

TransTables=\${XXX}

Realtime=\${XXX}

```
SessionID=${XXX}  
TimeoutSec=${XXX}  
BioPhotoFun=${XXX}  
BioDataFun=${XXX}  
MultiBioDataSupport=${XXX}  
MultiBioPhotoSupport=${XXX}  
QRCodeDecryptKey=${XXX}  
QRCodeDecryptType=${XXX}
```

**Note:**

HTTP request method: GET

URI: /iclock/push

HTTP version: 1.1

ServerVersion: The version of the server

ServerName: The name of the server

ErrorDelay: The interval (in seconds) at which the client reconnects to the server after networking failure. The recommended range is 30s to 300s. If no value is configured at the client, the default value 30s is used.

RequestDelay: The interval (in seconds) at which the client sends the command acquiring request. If no value is configured at the client, the default value 30s are used.

TransTimes: The time point in which the newly transmitted data is regularly checked. If no value is configured, the default value 12:30; 14:30 is used.

TransInterval: The interval (in minutes) at which the system checks whether any new data needs to be transmitted. If no value is configured at the client, the default value 2 min is used.

TransTables: The new data that needs to be checked and uploaded. The default value is User Transaction, which means the user and access control records need to be automatically uploaded.

Realtime: Specifies whether the client transmits new records in real time. 1 means that any new data is transmitted to the server. 0 means that the data is transmitted according to the values of TransTimes and TransInterval. If no value is configured at the client, the default value 1 is used.

SessionID: The ID of the PUSH communication session. This field is used to calculate the new Token for all subsequent requests from the device.

TimeoutSec: The network timeout time. If no value is configured at the client, the default value 10s is used.

BioPhotoFun: Specify whether to support the comparison photo parameter.

BioDataFun: Specify whether to support the visible light face template parameter.

MultiBioDataSupport: Supports multi-modal bio-template parameters. The type is defined bit by bit. Different types are separated by colons, 0 means not supported, 1 means supported. The supported version number, such as: 0: 1: 1: 0: 0: 0: 0: 0: 0: 0, indicating support for fingerprint template and near-infrared face template.

**MultiBioPhotoSupport:** Supports multi-modal biometric photo parameters. The type is defined bit by bit. Different types are separated by colons, 0 means not supported, 1 means supported. The supported version number, such as: 0: 1: 1: 0: 0: 0: 0: 0: 0, indicating support for fingerprint photo and near-infrared face photo.

**NewVFStyles:** If the server supports the new verification method, it will reply to the value of this parameter uploaded by the device in the registration interface. If the server does not support the new verification method, the device can be compatible with the old verification method.

**IRTempUnitTrans:** Specifies the unit of temperature upload (specifies the temperature unit of ConvTemperature in real-time events)

0: The temperature is uploaded in Celsius

1: The temperature is uploaded in Fahrenheit

**QRCodeDecryptType:** QR code decryption method, currently supports three methods, value 0, 1, 2.

1: Use scheme one, use today's date as the key and use AES256 algorithm to encrypt (the key is fixed)

2: Use scheme two, the system randomly generates a key and uses AES256 algorithm for encryption (the key is not fixed)

3: Use scheme three, the system randomly generates a key and uses RSA1024 algorithm for encryption (public and private keys are not fixed)

**QRCodeDecryptKey:** QR code key

**Note:**

The token calculation method: The client encrypts the values of RegistryCode, SerialNumber, and SessionID with the MD5 algorithm and then converts the calculated value to a hexadecimal string. The hexadecimal string is the token.

**Example:**

**Client request message:**

POST /iclock/push?SN=3383154200002 HTTP/1.1

Cookie: token=9d41643bfba01fe8b49143c21defc20a

Host: 192.168.213.17:8088

User-Agent: iClock Proxy/1.09

Connection: starting

Accept: application/push

Accept-Charset: UTF-8

Accept-Language: zh-CN

Content-Type: application/push;charset=UTF-8

Content-Language: zh-CN

Content-Length: 0

**Response from the server:**

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

```
Content-Type: text/plain;charset=ISO-8859-1
Content-Length: 218
Date: Mon, 09 Jan 2017 01:31:59 GMT
ServerVersion=3.0.1
ServerName=ADMS
PushVersion=3.0.1
ErrorDelay=60
RequestDelay=2
TransTimes=00:0014:00
TransInterval=1
TransTables=User Transaction
Realtime=1
SessionID=30BFB04B2C8AECC72C01C03BFD549D15
TimeoutSec=10
```

## 7.6 Upload Device Parameters

Application scenario:

To enable the device to actively push its parameters, set PushOptionsFlag to 1.

Client request message:

```
POST /iclock/cdata?SN=${SerialNumber}&table=options HTTP/1.1
Cookie: token=${XXX}, timestamp=${XXX}
Host: ${ServerIP}: ${ServerPort}
...
Parm1=${XXX}, Parm2=${XXX}, Parm3=${XXX}, Parm3=${XXX}, Parm4=${XXX}...
```

Response from the server:

```
HTTP/1.1 200 OK
Server: ${XXX}
Content-Length: ${XXX}
Date: ${XXX}
OK
```

Note:

```
HTTP request method: POST
URI: /iclock/cdata
HTTP version: 1.1
Parm1: A required parameter, same to Parm2 and other parameters.
```

**Note:**

The token calculation method: The client encrypts the values of RegistryCode, SerialNumber, and SessionID with the MD5 algorithm and then converts the calculated value to a hexadecimal string. The hexadecimal string is the token.

**Example:****Client request message:**

POST /iclock/cdata?SN=AJI6181160001&table=options HTTP/1.1

Cookie: token=50104b7ee71d8bb2ea10d8f1736b1bf8

Host: 192.168.52.44:8088

User-Agent: iClock Proxy/1.09

Connection: starting

Accept: application/push

Accept-Charset: UTF-8

Accept-Language: zh-CN

Content-Type: application/push; charset=UTF-8

Content-Language: zh-CN

~DeviceName=ProBio,MAC=00:17:61:11:ad:4f,TransactionCount=23,~MaxAttLogCount=10,UserCount=7,  
~MaxUserCount=100,PhotoFunOn=1,~MaxUserPhotoCount=3000,FingerFunOn=1,FPVersion=10,~MaxFi  
ngerCount=40,FPCount=4,FaceFunOn=1,FaceVersion=7,~MaxFaceCount=2000,FaceCount=3,FvFunOn=  
0,FvVersion=3,~MaxFvCount=10,FvCount=0,PvFunOn=0,PvVersion=5,~MaxPvCount=,PvCount=0,Langu  
age=69,IPAddress=192.168.52.71,~Platform=ZMM220\_TFT,~OEMVendor= ZKTeco Inc. ,FWVersion=Ver  
8.0.4.1-20180102,PushVersion=Ver 2.0.34-20180822

**Response from the server:**

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Content-Length: 2

Date: Wed, 22 Aug 2018 03:14:29 GMT

OK

## 8 Authorization

- Only requests of the master control device are authorized. For the definition of the master control device, see Appendix 15.
- The master control device actively pushes the sub-control authorization table to the software.

- When the device is started, the authorization table is pushed to the software.
- Any change to the authorization table is pushed to the software.
- Protocol:
- POST  
/iclock/cdata?SN=123456789&type=registry&table=tabledata&tablename=DeviceAuthorize&count=1
- Host: 113.108.97.187:8081
- DeviceAuthorize SN=%?\tOnline=%?\tlsAuthorize=%?\tAuthorizeInfo=%?\tRegisterInfo=%?
- Note
- SN: The serial number of the secondary controller
- Online: Indicate whether the secondary controller is online
- IsAuthorize: Indicate whether it is authorized. 0 means Unauthorized, 1 means Authorization in Progress, and 2 means Authorized.
- AuthorizeInfo: The simple device information pushed by the secondary controller when it is not authorized
- RegisterInfo: The device registration information pushed by the secondary controller after it is authorized

- The software authorizes the secondary controller based on the pushed authorization table.

- The software delivers a command to authorize the secondary controller.
- Based on the authorization delivered by the software, BioIR9000 allows the specified secondary controller to push the registration information.
- The secondary controller pushes the registration information of the corresponding device to BioIR9000.
- BioIR9000 immediately pushes the registration information of the secondary controller to the software.
- Protocol:
- C:295:DATA UPDATE DeviceAuthorize SN=%?\tlsAuthorize=1

- After receiving the registration information of the secondary controller, the software delivers the final authorization command.

- After receiving the registration information of the secondary controller, the software delivers the final authorization command.
- Based on the authorization delivered by the software, BioIR9000 allows access from the specified secondary controller.

- Authorization is completed.
- Protocol:
- C:296:DATA UPDATE DeviceAuthorize SN=%?\tlsAuthorize=2

- The software deletes the authorized secondary controller.

- Protocol:
- C:295:DATA DELETE DeviceAuthorize SN=%?

## 9 Heartbeat

Application scenario:

It is used to retain the heartbeat with the server. During the upload of massive data, the ping command is used to retain the heartbeat. After the massive data is processed, the getrequest command is used to retain the heartbeat.

Client request message:

```
GET /iclock/ping?SN=${SerialNumber} HTTP/1.1
Cookie: token=${XXX}
Host: ${ServerIP}: ${ServerPort}
Content-Length: ${XXX}
...
```

Response from the server:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Length: ${XXX}
Date: ${XXX}
OK
```

Note:

```
HTTP request method: POST
URI: /iclock/ping
HTTP version: 1.1
```

Example:

**Client request message:**

```
GET /iclock/ping?SN=3383154200002 HTTP/1.1
Cookie: token=cb386eb5f8219329db63356fb262ddff
Host: 192.168.213.17:8088
User-Agent: iClock Proxy/1.09
```

Connection: starting  
Accept: application/push  
Accept-Charset: UTF-8  
Accept-Language: zh-CN  
Content-Type: application/push;charset=UTF-8  
Content-Language: zh-CN  
**Response from the server:**  
HTTP/1.1 200 OK  
Server: Apache-Coyote/1.1  
Content-Length: 2  
Date: Tue, 10 Jan 2017 07:42:41 GMT  
OK

## 10 Upload

### 10.1 Upload Method

Real-time upload:

Messages are uploaded immediately. The device supports this mode by default and you can control the upload mode on the server. For details, see the parameter Realtime in Initialize Information Interaction.

Upload at the interval:

Messages are uploaded at an interval. You can set the specific interval on the server. For details, see the parameter TransInterval in Initialize Information Interaction.

Upload at a specific time:

Messages are uploaded at a specific time point. You can set the specific time point on the server. For details, see the parameter TransTimes in Initialize Information Interaction.

### 10.2 Upload Real-time Events

Real-time events are events generated by the device in real time. Considering network delay, all the events generated in the 15s are generally called real-time events. For the code and description of real-time events, see Appendix 2.

Application scenario:

When a real-time event occurs, the device needs to immediately upload the real-time event information to the software.

Client request message:

POST /iclock/cdata?SN=\$(SerialNumber)&table=rtlog HTTP/1.1



```

Cookie: token=${XXX}
Host: ${ServerIP}: ${ServerPort}
Content-Length: ${XXX}
...
${DataRecord}

```

Response from the server:

```

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Length: ${XXX}
Date: ${XXX}
OK

```

Note:

HTTP request method: POST

URI: /iclock/cdata

HTTP version: 1.1

Table name: rtlog

Request entity: \${DataRecord}, the real-time event data in the format of:

```

time=${Time}${HT}pin=${Pin}${HT}cardno=${XXX}${HT}sitecode=${XXX}${HT}linkid=${XXX}${HT}eventaddr=${XXX}${HT}event=${XXX}${HT}inoutstatus=${XXX}${HT}verifytype=${XXX}${HT}index=${xxx}${HT}maskflag=${xxx}${HT}temperature=${xxx}${HT}convtemperature=${xxx}

```

time: The time, in the format of XXXX-XX-XX XX:XX:XX

pin: The user ID

cardno: The card number

sitecode: Location code

linkid: Linkage event ID. For example, the event that the person has not registered will not trigger the door to open. If the software is set to open the door when the person unregistered event triggers, then the door will be opened as long as the device has a linked event.

eventaddr: The event point. The default value is doorid.

event: The event code. See Appendix 2.

inoutstatus: The in/out status. 0 indicates In, and 1 indicates Out.

verifytype:

If both the device and software support the upload of the new verification method rules:

"And" verification method-The verification is successful, and the string of the current verification method is returned. If the verification fails, the reply will be compared with the current verification method, the string of the wrong verification method. For example, card + password + face, if the verification is successful, it will reply the verification method 0000101000000011. If the verification method is Card + fingerprint, but the face is verified, the reply is 0000000000000010. If the verification method is card + fingerprint, but the card and palmprint are verified, then reply 0000000000000100.

“Or” verification method-Return the current verification type, face verification will reply 00000000000000010, palmprint verification will reply 0000000000000100, 0000000000010000 for fingerprint verification, 0000001000000000 for password verification, 0000010000000000 for User ID verification, and 0000100000000000 for card verification.

If the device supports the new verification method but the software does not support it, verifytype uses the original verification method, see Appendix 3.

maskflag: Value 0 or 1, 1 means wearing a mask

temperature: The value is the temperature data with a decimal point, for example: 36.2

convtemperature: The value is the temperature data with a decimal point. If the server does not send the IRTempUnitTrans parameter, then the unit of the temperature upload is subject to the IRTempUnit parameter.

index: The access control record ID. Each access control record has a unique ID in the device.

Example:

**Client request message:**

POST /iclock/cdata?SN=3383154200002&table=rtlog HTTP/1.1

Cookie: token=cb386eb5f8219329db63356fb262ddff

Host: 192.168.213.17:8088

User-Agent: iClock Proxy/1.09

Connection: starting

Accept: application/push

Accept-Charset: UTF-8

Accept-Language: zh-CN

Content-Type: application/push;charset=UTF-8

Content-Language: zh-CN

Content-Length: 99

time=2017-01-10 11:49:32      pin=0      cardno=0      eventaddr=1 event=27      inoutstatus=1

verifytype=0      index=21

**Response from the server:**

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Content-Length: 2

Date: Tue, 10 Jan 2017 03:49:32 GMT

OK

## 10.3 Upload Real-time Status

The real-time status means the current status or logic status of the physical devices, such as the relay, door sensor, barrier, and alarm.

Application scenario:

It is used to regularly upload the access control status to change to the access control status of the current device to the software.

Client request message:

```
POST /iclock/cdata?SN=${SerialNumber}&table=rtstate HTTP/1.1
```

```
Cookie: token=${XXX}
```

```
Host: ${ServerIP}:${ServerPort}
```

```
Content-Length: ${XXX}
```

```
...
```

```
${DataRecord}
```

Response from the server:

```
HTTP/1.1 200 OK
```

```
Server: Apache-Coyote/1.1
```

```
Content-Length: ${XXX}
```

```
Date: ${XXX}
```

```
OK
```

Note:

HTTP request method: POST

URI: /iclock/cdata

HTTP version: 1.1

Table name: rtstate

Request entity: \${DataRecord}, the real-time status data, in the format of:

```
time=${XXX}${HT}sensor=AABB${HT}relay=CC${HT}alarm=DDEEFFGGHHIIJJKK
```

time: The current time, in the format of XXXX-XX-XX XX:XX:XX

sensor: Indicate the door sensor status. Each door occupies two binary bits, AA indicates doors 1-4 and BB indicates doors 5-8. Door 1 occupies the 1st and 2nd bits of the first byte, and so on. 0b00 indicates that the current door sensor type is set to No Door Sensor, 0b01 indicates that the current door is closed (with the door sensor), and 0b10 indicates that the door is open (without the door sensor).

relay: Indicate the relay status. Each door occupies two binary bits. 0b0 indicates that the relay is connected and 0b1 indicates that the relay is disconnected. The first bit indicates the relay status of door 1, and so on.

alarm: Indicate the alarm status. Four doors occupy one byte, with each door occupying 8 binary bits. It can indicate up to 8 alarms. DD indicates the alarm status of the door 1, and so on. The alarms are

defined as follows:

- First bit: Accidental door opening event
- Second bit: Tamper alarm
- Third bit: Duress password alarm
- Fourth bit: Duress fingerprint alarm
- Fifth bit: Door sensor timeout alarm
- Sixth bit: Mains power failure alarm
- Seventh bit: Battery power failure alarm
- Eighth bit: Reader disassembly alarm

Example:

**Client request message:**

POST /iclock/cdata?SN=3383154200002&table=rtstate HTTP/1.1

Cookie: token=cb386eb5f8219329db63356fb262ddff

Host: 192.168.213.17:8088

User-Agent: iClock Proxy/1.09

Connection: starting

Accept: application/push

Accept-Charset: UTF-8

Accept-Language: zh-CN

Content-Type: application/push;charset=UTF-8

Content-Language: zh-CN

Content-Length: 69

time=2017-01-10 15:42:40    sensor=00    relay=00    alarm=0200000000000000

**Response from the server:**

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Content-Length: 2

Date: Tue, 10 Jan 2017 07:42:41 GMT

OK

## 10.4 Upload Returned Result of the Command

Application scenario:

It is used to return the execution result to the server after the command delivered by the server is executed.

Client request:

POST /iclock/devicecmd?SN=\${SerialNumber} HTTP/1.1

Cookie: token=\${XXX}

Host: \${ServerIP}:\${ServerPort}

Content-Length: 35

...

\${DataRecord}

Response from the server:

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Content-Type: text/plain;charset=ISO-8859-1

Content-Length: \${XXX}

Date: \${XXX}

OK

Note:

HTTP request method: POST

URI: /iclock/devicecmd

HTTP version: 1.1

Request entity: \${DataRecord}, the returned command result, in the format of:

ID=\${XXX}&Return=\${XXX}&CMD=\${XXX}&SN=\${XXX}

ID: The command ID which is carried by the cache command delivered by the software

Return: The returned command execution result

It is returned based on the multi-level control parameters. See Appendix 15.

If the master controller receives the sub-controller command delivered by the server and returns -5000, the command has been received and waits to be executed.

If the master controller command, non-multi-level device command, or sub-controller command delivered by the server is executed, the returned value is a general error code (see Appendix 1). A value equal to or greater than 0 means that the command is successfully executed. A value is smaller than 0 means that the command execution failed.

CMD: The command type

SN: The SN of the sub-controller. SN is returned only when the command is executed by a sub-controller. For the definition of the sub-controller, see Appendix 15.

Multiple records carried in the command delivered by the software are connected by \${LF}.

Example:

#### Client request:

POST /iclock/devicecmd?SN=3383154200002 HTTP/1.1

Cookie: token=1637f0b091af92b0cc66b8eede0ae48e

Host: 192.168.213.17:8088

User-Agent: iClock Proxy/1.09

Connection: starting

Accept: application/push  
 Accept-Charset: UTF-8  
 Accept-Language: zh-CN  
 Content-Type: application/push;charset=UTF-8  
 Content-Language: zh-CN  
 Content-Length: 35  
 ID=1&Return=0&CMD=DATA UPDATE

**Response from the server:**

HTTP/1.1 200 OK  
 Server: Apache-Coyote/1.1  
 Content-Type: text/plain;charset=ISO-8859-1  
 Content-Length: 2  
 Date: Wed, 11 Jan 2017 01:30:08 GMT  
 OK

## 10.5 Upload User Information

Application scenario:

The device actively uploads the user information. Generally after a user is registered, the device automatically uploads the user information to the server.

Client request message:

POST /iclock/cdata?SN=\${SerialNumber}&table=tabledata&tablename=user&count=\${XXX} HTTP/1.1  
 Cookie: token=\${XXX}  
 Host: {ServerIP}:{ServerPort}  
 Content-Length: \${XXX}  
 \${DataRecord}

Note:

HTTP request method: POST

URI: /iclock/cdata

HTTP version: 1.1

Table type: tabledata

Table name: user

count: The number of users of which information is uploaded in the current message

Request entity: \${DataRecord}, the user data, in the format of:

user

uid=\${XXX}\${HT}cardno=\${XXX}\${HT}pin=\${XXX}\${HT}password=\${XXX}\${HT}group=\${XXX}\${HT}starttime=\${XXX}\${HT}endtime=\${XXX}\${HT}name=\${XXX}\${HT}privilege=\${XXX}\${HT}disable=\${XXX}\${HT}verify=\${XXX}

uid: The user's ID in the device

cardno: The card number, which is an unsigned integer. Values delivered by the software can be uploaded in two formats:

a. Hexadecimal data, in the format of [%02x%02x%02x%02x], which means the first to the fourth bytes from left to right. For example, if the card number is 123456789, then Card=[15CD5B07].

b. A string. If the card number is 123456789, then Card=123456789.

pin: The user ID

password: The user password, up to 6 bits

group: The access control group of the user

starttime: The start time of the user validity period

If the DateFmtFunOn value is 1, for example, starttime=583512660, starttime is calculated according to the algorithm in Appendix 5 and the specific time in the format of YYYYMMDDHHMMSS is obtained according to the method in Appendix 6.

If the DateFmtFunOn value is 0, the format is YYYYMMDD.

endtime: The end time of the user validity period

If the DateFmtFunOn value is 1, for example, endtime=583512660, endtime is calculated according to the algorithm in Appendix 5 and the specific time in the format of YYYYMMDDHHMMSS is obtained according to the method in Appendix 6.

If the DateFmtFunOn value is 0, the format is YYYYMMDD.

name: The user name. When the device language is Chinese, it is encoded with the GB2312 character set; otherwise, it is encoded with the UTF-8 character set.

privilege: The user privilege

disable: Whether it is a blacklist. 0: whitelist; 1: blacklist.

verify: The supported verification mode

Multiple records are connected by \${LF}.

Response from the server:

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Content-Length: \${XXX}

Date: \${XXX}

user=\${XXX}

Note:

user: The number of users received by the server this time

Example:

#### Client request message:

POST /iclock/cdata?SN=3383154200002&table=tabledata&tablename=user&count=1 HTTP/1.1

Cookie: token=af65a75608cf5b80fbb3b48f0b4df95a

Host: 192.168.213.17:8088

User-Agent: iClock Proxy/1.09

Connection: starting

Accept: application/push

Accept-Charset: UTF-8

Accept-Language: zh-CN

Content-Type: application/push;charset=UTF-8

Content-Language: zh-CN

Content-Length: 104

user uid=5 cardno= pin=4 password= group=1 starttime=0 endtime=0 name= privilege=0  
disable=0 verify=0

**Response from the server:**

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Content-Length: 6

Date: Thu, 12 Jan 2017 00:49:31 GMT

user=1

## 10.6 Upload the Identity Card Information

Application scenario:

The device actively uploads the identity card information. Generally after an identity card is registered, the device automatically uploads the identity card to the server.

Client request message:

POST /iclock/cdata?SN=\${SerialNumber}&table=tabledata&tablename=identitycard&count=\${XXX}

HTTP/1.1

Cookie: token=\${XXX}

Host: {ServerIP}:{ServerPort}

Content-Length: \${XXX}

\${DataRecord}

Note:

HTTP request method: POST

URI: /iclock/cdata

HTTP version: 1.1

Table type: tabledata

Table name: identitycard

count: The number of users of which information is uploaded in the current message

Request entity: \${DataRecord}, the user data, in the format of:



**identitycard**

pin=\${XXX}\$(HT)SN\_Num=\${XXX}\$(HT)ID\_Num=\${XXX}\$(HT)DN\_Num=\${XXX}\$(HT)Name=\${XXX}\$(HT)Gender=\${XXX}\$(HT)Nation=\${XXX}\$(HT)Birthday=\${XXX}\$(HT)Valid\_Info=\${XXX}\$(HT)Address=\${XXX}\$(HT)Additional\_Info=\${XXX}\$(HT)Issuer=\${XXX}\$(HT)Photo=\${XXX}\$(HT)PhotoJPG=\${XXX}\$(HT)FP\_Template1=\${XXX}\$(HT)FP\_Template2=\${XXX}\$(HT)Reserve=\${XXX}\$(HT)Notice=\${XXX}

**Note:**

Pin: The user ID

SN\_Num: The physical number of the identity card

ID\_Num: The number of the citizen identity card

DN\_Num: The SN of the resident identity card (the card management number)

Name: The name, UTF-8-encoded

Gender: The gender code

1: Male

2: Female

Nation: The nationality code

0: Decoding Error

1: Han

2: Mongol

3: Hui

4: Tibetan

5: Uyghur

6: Miao

7: Yi

8: Zhuang

9: Buyi

10: Korean

11: Manchu

12: Dong

13: Yao

14: Bai

15: Tujia

16: Hani

17: Kazak

18: Dai

19: Li

20: Lisu

21: Va

22: She

- 23: Gaoshan
- 24: Lahu
- 25: Shui
- 26: Dongxiang
- 27: Naxi
- 28: Jingpo
- 29: Kirgiz
- 30: Tu
- 31: Daur
- 32: Mulao
- 33: Qiang
- 34: Blang
- 35: Salar
- 36: Maonan
- 37: Kelao
- 38: Xibe
- 39: Achang
- 40: Pumi
- 41: Tajik
- 42: Nu
- 43: Uzbek
- 44: Russian
- 45: Ewenki
- 46: De'ang
- 47: Baoan
- 48: Yugu
- 49: Jing
- 50: Tatar
- 51: Drung
- 52: Oroqen
- 53: Hezhe
- 54: Monba
- 55: Lhoba
- 56: Jino
- 57: Encoding Error
- 97: Other
- 98: Foreign Origin

Birthday: The date of birth, in the format of yyyyMMdd

Valid\_Info: The validity period, start time and end time, in the format of yyyyMMddyyyyMMdd

Address: The address, UTF-8-encoded

Additional\_Info: The additional information read by the machine, UTF-8-encoded

Issuer: The issuing authority, UTF-8-encoded

Photo: The photo data stored in the identity card, which is encrypted and converted to the base64-based data before transmission.

PhotoJPG: The photo data stored in the identity card, which is decrypted and converted to the base64-based data before transmission.

FP\_Template1: Features of the finger 1.

FP\_Template2: Features of the finger 2.

Reserve: A reserved field

Notice: The remarks, UTF-8-encoded

Multiple records are connected by \${LF}.

Response from the server:

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Content-Length: \${XXX}

Date: \${XXX}

identitycard=\${XXX}

Note:

identitycard: The number of identity cards received by the server this time

Example:

#### Client request message:

POST /iclock/cdata?SN=3383154200002&table=tabledata&tablename=identitycard&count=1 HTTP/1.1

Cookie: token=af65a75608cf5b80fbb3b48f0b4df95a

Host: 192.168.213.17:8088

User-Agent: iClock Proxy/1.09

Connection: starting

Accept: application/push

Accept-Charset: UTF-8

Accept-Language: zh-CN

Content-Type: application/push; charset=UTF-8

Content-Language: zh-CN

Content-Length: 104

identitycard pin=1 SN\_Num=26121292460088826975 IDNum=210102199212182xxxx DNum=

Name=Zhangsan Gender=1 Nation=1 Birthday=19911218 ValidInfo=2013091220230912

Address=xxxxx, Tangxia Town, Dongguan, Guangdong AdditionalInfo= Issuer=xxx Public Security

Bureau Photo=V0xmAH4AMgAA/apmyEd8  
 PhotoJPG=Y0tKqWNPzyEd8Pn0EhRsgAAjeWPxiUzLaPU1w=  
 FPTemplate1=QwGIEgELUQAAAAAAAAAAAAAAAAACgBmnlcAf////////0wZqfwuJK78PzJg/lw  
 FPTemplate2=QwGIEgEQUAAAAAAAAAAAAAAAAADIBmkbyAP////////3UYCPxXQs38qEVW/rpLovwyUBv8W  
 V8 Reserve= Notice=  
**Response from the server:**  
 HTTP/1.1 200 OK  
 Server: Apache-Coyote/1.1  
 Content-Length: 6  
 Date: Thu, 12 Jan 2017 00:49:31 GMT  
 identitycard=1

## 10.7 Upload Fingerprint Template

Application scenario:

The device actively uploads the fingerprint template. Generally after a user fingerprint is registered, the device automatically uploads the fingerprint to the server.

Client request message:

POST /iclock/cdata?SN=\${SerialNumber}&table=tabledata&tablename=templatev10&count=\${XXX}  
 HTTP/1.1  
 Cookie: token=\${XXX}  
 Host: \${ServerIP}: \${ServerPort}  
 Content-Length: \${XXX}  
 ...  
 \${DataRecord}

Note:

HTTP request method: POST  
 URI: /iclock/cdata  
 HTTP version: 1.1  
 count: The number of fingerprint templates in one request  
 Table type: tabledata  
 Table name: user  
 Request entity: \${DataRecord}, the fingerprint template, in the format of:  
 templatev10  
 size=\${XXX}\${HT}uid=\${XXX}\${HT}pin=\${XXX}\${HT}fingerprintid=\${XXX}\${HT}valid=\${XXX}\${HT}template=\${X  
 XX}\${H  
 T}resverd=\${XXX}\${HT}endtag=\${XXX}  
 templatev10: Indicate that the data type is a fingerprint template. **The supported fingerprint**

**algorithm version is no later than 10.0.**

size: The size of the base64-encoded fingerprint template

uid: The fingerprint template ID in the device

pin: The ID of the user corresponding to the fingerprint template

fingerid: The fingerprint ID of each user, which is used to distinguish different fingers. 0 to 9 indicate a common fingerprint. When 16 is added to a common fingerprint ID, such as 6+16=22, it indicates a duress fingerprint. 16 must be added when duress fingerprints in the device are uploaded to the software.

valid: The duress fingerprint mark in the device. 0 indicates an invalid fingerprint, 1 indicates a common fingerprint, and 3 indicates a duress fingerprint. The software distinguishes common fingerprints from duress fingerprints by fingerid. If the valid value is 3 when the fingerprint is uploaded to the software, change the value to 1 first.

template: The fingerprint template data. Before a fingerprint template is transmitted, the original binary fingerprint template must be base64-encoded.

resverd:

endtag:

Multiple records are connected by \${LF}.

Response from the server:

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Content-Length: \${XXX}

Date: \${XXX}

templatev10=\${XXX}

Note:

templatev10: The number of fingerprint templates using the 10.0 algorithm received by the server

Example:

**Client request message:**

POST /iclock/cdata?SN=3383154200002&table=tabledata&tablename=templatev10&count=2 HTTP/1.1

Cookie: token=af65a75608cf5b80fbb3b48f0b4df95a

Host: 192.168.213.17:8088

User-Agent: iClock Proxy/1.09

Connection: starting

Accept: application/push

Accept-Charset: UTF-8

Accept-Language: zh-CN

Content-Type: application/push; charset=UTF-8

Content-Language: zh-CN

Content-Length: 3900

templatev10 size=1808 uid=4 pin=4 fingerid=7 valid=1  
template=TAITUzlxAAAFsksECAUHCc7QAAAdS2kBAAAAhfvcU0o3AIAPswD/AltFJQBWAG0PcABfSo0Pog  
BhAE0PeEpnAIEPqACjAAIFcAB0AAQPvwCBSOEPvACTANEPGUqcAOoPSgBkAGtEOwCpAO0OAAcPso8PF  
gC9AKwPWUrBAGcPLAAPAFpFjQDNAHYPEADIShoPagDRAKcPf0riAFcPLQAtAFxElADtAGEPpwDqSkoP3A  
DxAOMP/krzACMPmgA8ABRFHgD8ANkO+AAFS00PLAACAZUOx0oDAZgOdADPATdFvAAOAY8OnAAKSz  
oPyAAVAU8NT0oWAT8PowDkAQNF3wAhAYENUQApSxIPQQAwwAfsPBUs5AYIOUAD4AS5FEAE9AYwO8QB  
ES0AOewBHAd8PR0pRATIPGwCWAV9FahcnAzN/9wRDQT+H/+9bh+JrUMtkBbIAFH4s/XNDXH1GAPr51YZ  
Qy/ePqYVZhap9qDfHAOf5oYvaCB7Ep/zegJt8RIE7Q5aEgYEmBCORoDUjdV8bLQ8ngAM0PAbG+bj+fPtnQ  
vN1aSNCFEMVE8ef9Yf5wf6EDm9SzPaZDwXjRIJsQVAPUFOTdXpSiL0D6F4YXgQDkBNH4/z/Cbem5bMP9le  
GuYiBgmUmtLZgD933wXjoBPSxOQsdEdJ0DPv/ohfewPsIB5+ltLY8C9rwjP9s9QhALATt95EDJPVnMm9+lfq  
W9t8CUEfP+RPgYRVY/RRJCPmaApf7MO20sNDvgQ2LD25rZfYBTkbRgk3805c8PEpCg70yxgjYAodP0PuHt  
gjJQQAavloUg3FOQVahMBKQ/8FxYoHQz4IAHoHCY5TCkokD/04wMAF/m2KYAsAKhYDIIFJRAELK/TBMTp  
Hx30TAAky/cP1/IYRwP/AcQYAlzKGi8NpCwBaNsNCxQF2BgBSOH29wQFKtjsTfQ0A0VDxHUDAVf8LAO1U9  
XE8WwQAuVvJQQIKn16Mc8GAB2QCSqdhAzZMDMWYZ8xThHfACQC8Y4PKaMEKAIFmw0/EBsEHAKtnCTv  
B+hMIAHloff9GcQIKdHQGXsH/BDb6QgF2gobCwUbBAEp/hQA2CgC3h4U0wsE/BAAsSWdeWQESjun//4L+  
bbVC/0wGAL9XDPIlwYAv5cTgcEQSguc58AvwPZgXYr8ZBEAwatWwnfBw8H/wsCF1QC25o1xg3HB/IEJB  
f+sDP9DSxbFGr+jaf5MNv7/OlVitxcABb7i/pP/NwtHRMBNBQCdwWzOEwDRypeEBcKDtMOTZwQAKQ5gak  
QBkMwDLv8FMsSKmgcA2cwW8jwASmfQacKICMWJ1T3AxMHBkwTF19RQWw0AaNVkB8GBFsBbDwB35KX  
FicLEUMHCEgBU6nKNxcbBwcN8VsI5RwFm6977wDr8K3L+BACZ7QPnDgXY8WffYMLCacDGi8P/wp8DADr  
yG7UKAF7zU8M6w8fCgwsAZPNMBMRsw8MGAN3zJDBH0sU9KBqc2+vw8QuxXLB/8EEfxzBw4FAJ33DD  
h2AUqU+S3GwgPFnPhZwQUQHABWBlwBWogBMKEIEPwDVTFZBxA/BEy2TgJaKgVWgyYF1ZIDasKUGRDT  
Clv+xlndXp+ywclBgvgqlw//DwH8Q1XQlfonBwYh+weMJFRMTQMPAlOHCEEJQQnDEwQMqiRo4igQQpx8D  
wAb8GlsUlaTB/nSuwoUliIPA/8HCr/sCWqYIDJAZCdWYK13DjF4gEXRrpMR9d8CihMKGBmLHi/7Bwf39+cEQ  
Sgo2nQQQVEDogwBaf0oewnME1f9hN0tSQgALQ8QABUFEUg== resverd= endtag=  
templatev10 size=1928 uid=3 pin=4 fingerid=6 valid=1  
template=TOVTUzlxAAAFpqqECAUHCc7QAAAdp2kBAAAAhUs1zKYXAPUPzgD2AP+pCgE6AAYOFgBDpg  
EPtQBSAL0Pk6ZVAPYP8ACRAAOpmABoAPQPDABopgcPNgCCACsO2KaHAIYPqQBNAPiprACTAHYPMQCR  
phMPrwCeAMcPN6aIAOoOEQBvAOetQQCqAGoO2wC3pmQLpAcYALMPI6a4AOoLZgAEAPWpwwDJAA0P  
xAHRphwOqADWAM4PEqbcAOIPuQAnAISpowDIAAMPjwDjpukPYQDqACsPq6buABEP/wA6ACapaAAAAf  
MP1AADp2UPkQALAbIPqqYPASAPbgDRAfGplQAVARYPvAAZp/8PFQAgAalOaKYmAf8P9wDiASiplQAqAS  
8O3AA2p+IOpgAzAf0OZ6Y0Af4P3ADyATGpbAA4AXgO9gBMp+UPZQBMAa8Ps6ZUAS4NoACZAUkQwQB  
dASMNAP/CUtp/vf1BByeG26fyCgMPNQIT/Pcuh4T6eDJ1AAWPJr76DgheASIATKdzfbr5ooDnkUOs2wXKhu  
eK1HenJnIRtlU5/vcJQdG9e1YABY/KDyqpOY81BtWOblDopPyGtXy5+1/5rCaDgl4FOH90hADY4HbWACpp8  
YdopDAL1gYqiyoluCjcBxKZpv/aD+61xAiRjWEDt/rjJGd/XXCZebYcnFJM+HmGof5z9nJeGARf/loCyBBINiQL  
pvoiC3LzwKdABrr0Vf8jhllmPWNxew5YEA/jDiMQpfjR9YABJQWws6WH7HQwgAHxNUIXVB0OFoADELtwBvv  
geDiYYPF8QGeYwVX1/AVIm3An2JXoOSYtYp7yWOGinEHYIkdnleFkSOgFmfULb5PbRbuJuqQ/ATAP6LRO+  
JMCS/VHj3DvwsHjtYw+9ASBQAQMF4k4Apr8BdML/wMAAx6Fq/0sLAOfCgMfK/GvAbwMAKGxyZgwA8BR3  
wKNWbmcQAQQeg34HUsVZcVQSAQsmRXFh+8P/a2UGAdcsf1IEgEMNICv/4FYxUPBVAUAekYFZysKARVH

```
kwbAjP4GALIQ9y+CCAU0VHBq/1cNxbFQ1v/CwGvB/zjGx6UB9FUJwgrFnWZSwcH9//5RyACQz3DBWsDB/
gD+N6wBnWr0wP86wPlmSgcAzW0A9EYWpxR+l4WWwQXAxWRkbQQArYQ/MAimpll6wsBpl8BloAGtigBK
/sgArDZ2c37AwMC+AwVekxDBAwAZUlf7rwGxk/r+//BUHacNk5NtecM7wcRma2vA/cXDBBYEsJWew/+Hw
FxbvYDgCpl3cEa3bPbgkAs5r97jf7qQGrn3p+agXBxlhdCACzoAaRNgcMobB6wsDBzwCtFPw1//7AV8YAk
B92wg8Aabw1wDiJ/v82wAQA08BnYP4KAGLDab1kUqoBdMV0icl7wcVn/vwJAMHKzP5E8AcAPNBkcwYLB
QrSBv7/OP+GBgUC13qEwAwAadkMWP7/wDZCDcW25CClc8NnwBLFTeFBVf9GwP7+OjtRtwFk5uvAODr9
+1IH/8lxCwBy5wxb/j7/VQgAmO1oZHFXBwCy7cz9+lJB/xAAqu5JxMdZw39w/sljyQC0VBI4NcBGGsQS8gRr
wnnAfsKx/4P8NREAbf7w8P34Z/81wEDBA9QCBIT/EBCrC4xUwYJkT/0/CRCzyBf4Zv8ywgqQsdceNllzBxCXF
gzkoQ22fxv9/f78O/3FoxFwlv3/Ks0Qdl4B/xUfCBCsKmFZwET/BRD77yk4oxH2Ky1VBdWYKYv8HwQQZTQy/
/+iEaU2Ojke1aozlikGEGk5bQdRAraCOYDHQ//BENicMEwGEH8/rMLFIACQjKN6xza/rcRNUfnwVs5wvhm/fz
+/jYD1Y9O6vwDEINMXgQFFcVPaf/A/RrVAVxPwYH/wUb9jv74mv5LX1JCAM5DBKYBC0VS
resverd=          endtag=
```

### Response from the server:

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Content-Length: 13

Date: Thu, 12 Jan 2017 00:49:32 GMT

templatev10=2

## 10.8 Upload Comparison Photo

Application scenario:

The visible light device automatically uploads the registered user comparison photos to the server.

Client request message:

```
POST /iclock/cdata?SN=${SerialNumber}&table=tabledata&tablename=biophoto&count=${XXX}
```

HTTP/1.1

Cookie: token=\${XXX}

Host: \${ServerIP}:\${ServerPort}

Content-Length: \${XXX}

...

\${DataRecord}

Note:

HTTP request method: POST

URI: /iclock/cdata

HTTP version: 1.1

count=\${XXX}: The number of comparison photos in one request

Host header: \${Required}

Content-Length header: \${Required}

Other headers: \${Optional}

Request entity: \${DataRecord}, the comparison photo data, in the format of

biophoto=\${SP}pin=\${XXX}\${HT}filename=\${XXX}\${HT}type=\${XXX}\${HT}size=\${XXX}\${HT}content=\${XXX}

Note:

pin=\${XXX}: The user ID.

filename=\${XXX}: The file name of the biometric photo. Currently, only photos in JPG format are supported.

type=\${XXX}: The biometric type.

Value Meaning

0 General

1 Fingerprint

2 Face (Near Infrared)

3 VoicePrint

4 Iris

5 Retina

6 Palm Print

7 Finger Vein

8 Palm

9 Visible Light Face

size=\${XXX}: The length of the base64-encoded biometric photo

content=\${XXX}: The original binary biometric photo must be base64-encoded before transmission.

Multiple records are connected by \${LF}.

Response from the server:

HTTP/1.1 200 OK

Content-Length: \${XXX}

...

biophoto=\${XXX}

Note:

HTTP status line: It is defined by the standard HTTP.

HTTP response header:

Content-Length header: According to HTTP 1.1, this header is generally used to specify the length of the response entity. If you do not know the length, you can also set Transfer-Encoding to chunked. Both Content-Length and Transfer-Encoding are headers defined by standard HTTP, which are not described in detail here.

Response entity: When the server receives the data and processes it normally, it returns biophoto=\${XXX}, in which \${XXX} indicates the number of records that are successfully processed. When an error occurs, the server returns the error description.



Example:

**Client request:**

POST /iclock/cdata?SN=0316144680030&table=tabledata&tablename=biophoto&count=1 HTTP/1.1

Host: 58.250.50.81:8011

User-Agent: iClock Proxy/1.09

Connection: close

Accept: \*/\*

Content-Length: 95040

biophoto pin=123      filename=123.jpg      type=9      size=95040      content=AAAA.....

**Response from the server:**

HTTP/1.1 200 OK

Server: nginx/1.6.0

Date: Thu, 30 Jul 2015 07:25:38 GMT

Content-Type: text/plain

Content-Length: 4

Connection: close

Pragma: no-cache

Cache-Control: no-store

biophoto=1

## 10.9 Upload Snapshot

Application scenario:

The visible light device automatically uploads snapshots to the server.

Client request message:

POST /iclock/cdata?SN=\${SerialNumber}&table=tabledata&tablename=ATTPHOTO&count=\${XXX}

HTTP/1.1

Cookie: token=\${XXX}

Host: \${ServerIP}:\${ServerPort}

Content-Length: \${XXX}

...

\${DataRecord}

Note:

HTTP request method: POST

URI: /iclock/cdata

HTTP version: 1.1

count=\${XXX}: The number of snapshots in one request

Host header: \${Required}

Content-Length header: `${Required}`

Other headers: `{Optional}`

Request entity: `{DataRecord}`, the attendance photo data, in the format of

pin=\${XXX}\${HT}sn=\${XXX}\${HT}size=\${XXX}\${HT}photo=\${XXX}

Note:

pin=\${XXX}: The name of the attendance photo.

sn=\${XXX}: The device SN.

size=\${XXX}: The original size of the snapshot.

photo=\${XXX}: The original binary biometric photo must be base64-encoded before transmission.

Multiple records are connected by `{LF}`.

Response from the server:

HTTP/1.1 200 OK

Content-Length: \${XXX}

...

ATTTPHOTO=\${XXX}

Note:

HTTP status line: It is defined by the standard HTTP.

HTTP response header:

Content-Length header: According to HTTP 1.1, this header is generally used to specify the length of the response entity. If you do not know the length, you can also set Transfer-Encoding to chunked. Both Content-Length and Transfer-Encoding are headers defined by standard HTTP, which are not described in detail here.

Response entity: When the server receives the data and processes it normally, it returns

ATTPHOTO=\${XXX}, in which  $\{XXX\}$  indicates the number of photos that are successfully processed.

When an error occurs, the server returns the error description.

Example:

**Client request:**

POST /iclock/cdata?SN=1809140006&table=tabledata&tablename=ATTPHOTO&count=1 HTTP/1.1

Host: 58.250.50.81:8011

User-Agent: iClock Proxy/1.09

Connection: close

Accept: \*/\*

Content-Length: 30327

pin=20181003143617-22.jpg sn=1809140006 size=22701

photo=/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDABsSFBCUERsXFhceHBsgKElrKCUIKFE6PTBCY  
FVIZF9VXvtqeJmBanGQc1tdhbWGkJ6jq62rZ4C8ybqmx5moq6T/2wBDArWeHigjKE4rK06kbl1upKSkpKSk  
pKSkpKSqpKSqpKSqpKSqpKSqpKSqpKSqpKSqpKSqpKSqpKSqpKT/wAARCAUAAAtADASIAAhEBAxEB

```
B/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgclCQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQI
DAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUp
TVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDx
MXGx8jJytLT1NXW19jZ2uHi4+TI5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEBAQEBAQAAAAAAAAECAw
QFBgclCQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAECAxEEBSExBhJBUQdhcRMiMoEIFEKRobHBCSMzUv
AVYnLRChYkNOEl8RcYGRomJygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goO
EhYaHilmKkpOUlZaXmJmaoqOkpaanqKmqsro0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4uPk5ebn6On
q8vP09fb3+Pn6/9oADAMBAAIRAxEAPwBXXIPFVpl+fbHWrn1pjJnOKgZ
```

#### Response from the server:

HTTP/1.1 200 OK

Server: nginx/1.6.0

Date: Thu, 30 Jul 2015 07:25:38 GMT

Content-Type: text/plain

Content-Length: 10

Connection: close

Pragma: no-cache

Cache-Control: no-store

ATTPHOTO=1

## 10.10 Upload User Photo

Application scenario:

The device automatically uploads user photos to the server.

Client request message:

POST /iclock/cdata?SN=\${SerialNumber}&table=tabledata&tablename=userpic&count=\${XXX} HTTP/1.1

Cookie: token=\${XXX}

Host: \${ServerIP}:\${ServerPort}

Content-Length: \${XXX}

...

\${DataRecord}

Note:

HTTP request method: POST

URI: /iclock/cdata

HTTP version: 1.1

count=\${XXX}: The number of user photos in one request

Host header: \${Required}

Content-Length header: \${Required}

Other headers: \${Optional}

Request entity: `$(DataRecord)`, the user photo data, in the format of:

userpic pin=\${XXX}\${HT}filename=\${XXX}\${HT}size=\${XXX}\${HT}content=\${XXX}

Note:

pin=\${XXX}: The user ID.

filename=\${XXX}: The file name of the user photo. Currently, only photos in JPG format are supported.

size=\${XXX}: The length of the base64-encoded user photo.

content=\${XXX}: The original binary user photo must be base64-encoded before transmission.

Multiple records are connected by `${LF}`.

Response from the server:

HTTP/1.1 200 OK

Content-Length: \${XXX}

...

```
userpic=${XXX}
```

Note:

HTTP status line: It is defined by the standard HTTP.

HTTP response header:

Content-Length header: According to HTTP 1.1, this header is generally used to specify the length of the response entity. If you do not know the length, you can also set Transfer-Encoding to chunked. Both Content-Length and Transfer-Encoding are headers defined by standard HTTP, which are not described in detail here.

Response entity: When the server receives the data and processes it normally, it returns userpic=\${XXX}, in which \${XXX} indicates the number of photos that are successfully processed. When an error occurs, the server returns the error description.

Example:

**Client request:**

POST /iclock/cdata?SN=1809140006&table=tabledata&tablename=userpic&count=1 HTTP/1.1

Host: 58.250.50.81:8011

User-Agent: iClock Proxy/1.09

Connection: close

Accept: \*/\*

Content-Length: 29848

```
userpic pin=1      filename=1.jpg      size=22320
```

content=/9j/4AAQSkZJRgABAQAAQABAAQ/2wBDABsSFBCUERsXFhceHBsgKElrKCUIKFE6PTBC

YFVIZF9VXVtqeJmBanGQC1tdhbWGkJ6jq62rZ4C8ybmX5moq6T/2wBDARweHigjKE4rK06kbl1upKSkpKSkpKS  
pKSkpKS  
EB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9A  
QIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdIs

UpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usL  
 DxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEBAQEBAQAAAAAAAAEC  
 AwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAECAxEEBSExBhJBUQdhcRMiMoEIFEKRobHBCSMz  
 UvAVYnLRChYkNOEl8RcYGRomJygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6g  
 oOEhYaHilmKkpOUlZaXmJmaoqOkpaanqKmqsRO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4uPk5ebn6  
 Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwDUooooQCiiigAooooAK

**Response from the server:**

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Content-Type: text/plain; charset=utf-8

Connection: close

Content-Length: 24

Date: Thu, 08 Nov 2018 06:16:41 GMT

userpic=1

## 10.11 Upload Integrated Template

Application scenario:

New biometric templates, such as integrated palm templates, will be uploaded and downloaded in a unified format and distinguished by the data type.

Client request message:

POST /iclock/cdata?SN=\${SerialNumber}&table=tabledata&tablename=biodata&count=\${XXX} HTTP/1.1

Cookie: token=\${XXX}

Host: \${ServerIP}:\${ServerPort}

Content-Length: \${XXX}

...

\${DataRecord}

Note:

HTTP request method: POST

URI: /iclock/cdata

HTTP version: 1.1

count=\${XXX}: The number of integrated templates in one request

Host header: \${Required}

Content-Length header: \${Required}

Other headers: \${Optional}

Request entity: \${DataRecord}, the integrated template, in the format of:

biodata pin=\${XXX}\${HT}no=\${XXX}\${HT}index=\${XXX}\${HT}valid=\${XXX}\${HT}duress=\${XXX}\${HT}

type=\${XXX}\${HT}majorver=\${XXX}\${HT}minorver=\${XXX}\${HT}format=\${XXX}\${HT}tmp=\${XXX}

**Note:**

pin=\${XXX}: The user ID.

no=\${XXX}: The number of the specific biont. The default value is 0.

[Fingerprint] The number ranges from 0 to 9, corresponding to the little finger, ring finger, middle finger, index finger, and thumb of the left hand, and thumb, index finger, middle finger, ring finger, and little finger of the right hand respectively.

[Finger Vein] Same to the fingerprints.

[Face] All is 0.

[Iris] 0 is the iris of the left eye and 1 is the iris of the right eye.

[Palm] 0 is the palm of the left hand and 1 is the palm of the right hand.

index=\${XXX}: The number of the specific biont template. Multiple templates may be stored for one finger. It is calculated from 0.

valid=\${XXX}: Whether it is valid. 0: Invalid; 1: Valid. The default value is 1.

duress=\${XXX}: Whether it is duress. 0: Not Duress; 1: Duress. The default value is 0.

type=\${XXX}: The biometric type

Value	Meaning
0	General
1	Fingerprint
2	Face
3	VoicePrint
4	Iris
5	Retina
6	Palm Print
7	Finger Vein
8	Palm
9	Visible Light Face

majorver=\${XXX}: The major version number. For example, for the fingerprint algorithm 10.3, the major version number is 10 and the minor version number is 3.

[Fingerprint] 9.0, 10.3, and 12.0

[Finger Vein] 3.0

[Face] 5.0, 7.0, and 8.0

[Palm] 1.0

minorver=\${XXX}: The minor version number. For example, for the fingerprint algorithm 10.3, the major version number is 10 and the minor version number is 3.

[Fingerprint] 9.0, 10.3, and 12.0

[Finger Vein] 3.0

[Face] 5.0, 7.0, and 8.0

[Palm] 1.0 pin=\${XXX}: The name of the attendance photo.

format=\${XXX}: The template format. The fingerprint template formats include ZK, ISO, and ANSI.

[Fingerprint]

Value	Format
0	ZK
1	ISO
2	ANSI

[Finger Vein]

Value	Format
0	ZK

[Face]

Value	Format
0	ZK

[Palm]

Value	Format
0	ZK

tmp=\${XXX}: The template data. The original binary fingerprint template must be base64-encoded.

Multiple records are connected by \${LF}.

Response from the server:

HTTP/1.1 200 OK

Content-Length: \${XXX}

...

biodata=\${XXX}

Note:

HTTP status line: It is defined by the standard HTTP.

HTTP response header:

Content-Length header: According to HTTP 1.1, this header is generally used to specify the length of the response entity. If you do not know the length, you can also set Transfer-Encoding to chunked. Both Content-Length and Transfer-Encoding are headers defined by standard HTTP, which are not described in detail here.

Response entity: When the server receives the data and processes it normally, it returns biodata=\${XXX}, in which \${XXX} indicates the number of integrated templates that are successfully processed. When an error occurs, the server returns the error description.

Example:

#### Client request:

POST /iclock/cdata?SN=5165181600015&table=tabledata&tablename=biodata&count=1 HTTP/1.1

Host: 58.250.50.81:8011

User-Agent: iClock Proxy/1.09

Connection: starting

Accept: application/push

Accept-Charset: UTF-8

Accept-Language: zh-CN

Content-Type: application/push;charset=UTF-8

Content-Language: zh-CN

Content-Length: 3564

biodata pin=1 no=0 index=0 valid=1 duress=0 type=8 majorver=5 minorver=0  
format=0

tmp=apUBD+cAC44IAAEAAJfKWIBWAQQFAAAAAAAAEAcHAcQAAAA3PJc4PO4W/AxrTnYNXQ7cJ0+ObT  
0vBE81KyGPkzSHuh7ul+wP647kWNnMTDA8WMwlanRcNTpzTjc2hzUGZ488FweSBlaN0kY2GdZeNHIVXCs  
x81nI5M0kZ2YzJD2kMxaWrBIWM9hWXEt4VzILNP4JzMdLF6OUY5Kn8GPepNRjUp+0Z2BPOkd5STsfeZuXa  
xaKzU0WvnUk06a2L1Prlu1jzwLvYf3U6wBrBG80yLbMc+g0DPL9MClyPTCNk50Fzws8xBsN

### Response from the server:

HTTP/1.1 200 OK

Server Apache-Coyote/1.1 is not blacklisted

Server: Apache-Coyote/1.1

Content-Length: 9

Date: Fri, 19 Oct 2018 06:55:30 GMT

biodata=1

## 10.12 Upload Error Log

Application scenario:

The device automatically uploads error logs to the server. The value of PushProtVer delivered by the server is greater to or equal to 3.1.2.

Client request message:

POST /iclock/cdata?SN=\${SerialNumber}&table=tabledata&tablename=errorlog&count=\${XXX} HTTP/1.1

Cookie: token=\${XXX}

Host: \${ServerIP}:\${ServerPort}

Content-Length: \${XXX}

...

\${DataRecord}

Note:

HTTP request method: POST

URI: /iclock/cdata

HTTP version: 1.1

count=\${XXX}: The number of error logs in one request



Host header: \${Required}

Content-Length header: \${Required}

Other headers: \${Optional}

Request entity: \${DataRecord}, the error log, in the format of:

errorlog

errcode=\${XXX}\${HT}errmsg=\${XXX}\${HT}dataorigin=\${XXX}\${HT}cmdid=\${XXX}\${HT}additional=\${XXX}

Note:

errcode=\${XXX}: The error code. See Appendix 17.

errmsg=\${XXX}: The error message

dataorigin=\${XXX}: The data source. Dev indicates that the data is sourced from the device, and cmd indicates that the data is delivered by the device through a command.

cmdid=\${XXX}: The number of the command delivered by the software

additional=\${XXX}: The base64-encoded additional information. The native data is in JSON format.

Multiple records are connected by \${LF}.

Response from the server:

HTTP/1.1 200 OK

Content-Length: \${XXX}

...

errorlog=\${XXX}

Note:

HTTP status line: It is defined by the standard HTTP.

HTTP response header:

Content-Length header: According to HTTP 1.1, this header is generally used to specify the length of the response entity. If you do not know the length, you can also set Transfer-Encoding to chunked. Both Content-Length and Transfer-Encoding are headers defined by standard HTTP, which are not described in detail here.

Response entity: When the server receives the data and processes it normally, it returns errorlog=\${XXX}, in

which \${XXX} indicates the number of error logs that are successfully processed. When an error occurs, the server returns the error description.

Example:

#### Client request:

POST /iclock/cdata?SN=5165181600015&table=tabledata&tablename=errorlog&count=1 HTTP/1.1

Host: 58.250.50.81:8011

User-Agent: iClock Proxy/1.09

Connection: starting

Accept: application/push

Accept-Charset: UTF-8  
Accept-Language: zh-CN  
Content-Type: application/push;charset=UTF-8  
Content-Language: zh-CN  
Content-Length: 3564  
errorlog errcode=D01E0001 errmsg= dataorigin=cmd cmdid=123 additional=

**Response from the server:**

HTTP/1.1 200 OK  
Server Apache-Coyote/1.1 is not blacklisted  
Server: Apache-Coyote/1.1  
Content-Length: 9  
Date: Fri, 19 Oct 2018 06:55:30 GMT  
errorlog=1

## 11 Download

### 11.1 Download Cache Command

Application scenario:

The client obtains generated commands from the server.

Client request message:

GET /iclock/getrequest?SN=\${SerialNumber} HTTP/1.1  
Cookie: token=\${XXX}  
Host: \${ServerIP}: \${ServerPort}  
...

Note:

HTTP request method: GET  
URI: /iclock/getrequest  
HTTP version: 1.1

Response from the server:

HTTP/1.1 200 OK  
Server: Apache-Coyote/1.1  
Content-Type: text/plain;charset=UTF-8  
Content-Length: \${XXX}  
Date: \${XXX}  
\${CmdRecord}

**Note:**

`${CmdRecord}`: The server response entity: the command, in the format of

C: `${CmdID}:${CmdDesc}${SP}${CmdDetail}`

`CmdID`: The command ID

`CmdDesc`: The command description. Commands are divided into data commands, control commands, and configuration commands.

`CmdDetail`: The command details

Multiple records are connected by `${LF}`.

For more details about commands delivered by the server, see [Details of Server Commands](#).

**Example:****Client request message:**

GET /iclock/getrequest?SN=3383154200002 HTTP/1.1

Cookie: token=1637f0b091af92b0cc66b8eede0ae48e

Host: 192.168.213.17:8088

User-Agent: iClock Proxy/1.09

Connection: starting

Accept: application/push

Accept-Charset: UTF-8

Accept-Language: zh-CN

**Response from the server:**

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Content-Type: text/plain;charset=UTF-8

Content-Length: 99

Date: Wed, 11 Jan 2017 01:30:03 GMT

C:223:SET OPTIONS IPAddress=192.168.213.222,GATEIPAddress=192.168.213.1,NetMask=255.255.255

## 12 Details of Server Commands

To operate on a device, the server must generate a command and then send the command to the device after the device sends a request. The execution results of commands are already described in Upload Returned Result of the Command. The formats of the requests sent by the client and the server response are already described in Download Cache Command. The following describes the command format in detail.

### 12.1 Data

It is used to operate on the device data.

#### 12.1.1 Update

It is used to add or update data to the device. If the device does not have the primary key data, you can run this command to add the data to the device; otherwise, you can run this command to update the data to the device.

Format:

##### **Format of a single data record:**

C:\${CmdID}:DATA\$(SP)UPDATE\$(SP)table name\$(SP)field name1=value1\$(HT)field name2=value2\$(HT)field name3=value3

##### **Format of multiple data records:**

C:\${CmdID}:DATA\$(SP)UPDATE\$(SP)table name\$(SP)field name1=value1\$(HT)field name2=value2\$(HT)field name3=value3\$(LF)  
field name4=value4\$(HT)field name5=value5\$(HT)field name6=value6

Wherein, 123 indicates the first data record, 456 indicates the second data record, and they are separated by the line feeder \$(LF).

##### 12.1.1.1 Deliver User Information

Application scenario:

The server delivers the user information to the client. It is generally used to synchronize the user information edited on the server to the client.

The command format is as follows:

##### **The server delivers the command:**

C:\${CmdID}:DATA\$(SP)UPDATE\$(SP)user\$(SP)CardNo=\${XXX}\$(HT)Pin=\${XXX}\$(HT>Password=\${XXX}\$(HT)Group=\${XXX}\$(HT)StartTime=\${XXX}\$(HT)EndTime=\${XXX}\$(HT)Name=\${XXX}\$(HT)Privilege=\${XXX}

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA UPDATE

**Note:**

user: The table name, which is the user

CardNo: The card number, which is an unsigned integer. Values delivered by the software can be delivered in two formats:

- a. Hexadecimal data, in the format of [%02x%02x%02x%02x], which means the first to the fourth bytes from left to right. For example, if the card number is 123456789, then Card=[15CD5B07].
- b. A string. If the card number is 123456789, then Card=123456789.
- c. When MulCardUserFunOn is enabled (refer to the value of AccSupportFunList), this table field is invalid, and the number of MulCardUser protocol storage card is used.

Pin: The user ID

Password: The password

Group: The user group. By default, the user belongs to group 1.

StartTime: The start time of the user validity period:

If the DateFmtFunOn value is 1, for example, StartTime=583512660, StartTime is calculated according to the algorithm in Appendix 5.

If the DateFmtFunOn value is 0, the format is YYYYMMDD.

If the value is 0, this user's start time has no limit.

EndTime: The end time of the user validity period:

If the DateFmtFunOn value is 1, for example, EndTime=583512660, EndTime is calculated according to the algorithm in Appendix 5.

If the DateFmtFunOn value is 0, the format is YYYYMMDD.

If the value is 0, this user's end time has no limit.

Name: The user name. When the device language is Chinese, it is encoded with the GB2312 character set; otherwise, it is encoded with the UTF-8 character set.

Privilege: The user privilege values have the following meanings respectively:

- |    |                     |
|----|---------------------|
| 0  | Common User         |
| 2  | Registrar           |
| 6  | Administrator       |
| 10 | User-defined        |
| 14 | Super Administrator |

Multiple records are connected by \${LF}.

**Example:**

**The software delivers the information about a user, of which the user ID is 1 and the password is 234. The user has no card, belongs to the default group, and has the privilege of a common user. No validity period is set for the user:**

C:295:DATA UPDATE user CardNo= Pin=1 Password=234 Group=0 StartTime=0 EndTime=0  
Name= Privilege=0

**The client uploads the successful execution result:**

ID=295&Return=0&CMD=DATA UPDATE

### 12.1.1.2 Deliver Extended User Information

Application scenario:

The server delivers the extended user information to the client. It is generally used to synchronize the user information edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)UPDATE\$(SP)extuser\$(SP)Pin=\${XXX}\$(HT)FunSwitch=\${XXX}\$(HT)FirstName=\${X  
XX}\$(HT)LastName=\${XXX}\$(HT)PersonalVS=\${XXX}

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

extuser: The table name, which is the extended user

Pin: The user ID

FunSwitch: The function switch. Each bit controls a specific function.

00000000 00000000 00000000 10001000

The bits have the following meanings from low to high:

1: The switch for the driving duration of the extended lock.

2: The switch for the temporary user (visitor)

For example, FunSwitch=1 (00000001) means enabling the function of driving duration for the extended lock.

FunSwitch=2 (00000010) means enabling the temporary user function.

FunSwitch=3 (00000011) means enabling the above two functions simultaneously.

FirstName: Reserved, not used yet

LastName: Reserved, not used yet

PersonalVS: The personal verification mode

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:500:DATA UPDATE extuser Pin=1 FunSwitch=1 FirstName= LastName= PersonalVS=0

**The client uploads the successful execution result:**

```
ID=500&Return=0&CMD=DATA
```

### 12.1.1.3 Deliver MulCardUser Information

Application scenario:

The server delivers MulCardUser information to the client. It is generally used to synchronize the MulCardUser information edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

```
C:${CmdID}:DATA$(SP)UPDATE$(SP)mulcarduser$(SP)Pin=${XXX}$(HT)CardNo=${XXX}$(HT)LossCardFlag=${XXX}$(HT)CardType=${XXX}
```

**The client uploads the execution result:**

```
ID=${CmdID}&Return=${XXX}&CMD=DATA
```

Note:

mulcarduser: The table name, which is the extended user

Pin: The user ID

CardNo: The card number. It is stored as a string. The value delivered by the software is a hexadecimal string. For example, CardNo=DF349C3BEA5277ED

LossCardFlag: The card loss flag. 0 indicates that the card is in normal use, and 1 indicates that the card is lost.

CardType: The main card or sub-card. 0 indicates the main card, and 1 indicates a sub-card.

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

```
C:501:DATA UPDATE mulcarduser Pin=1 CardNo=CC9932DD LossCardFlag=0 CardType=0
```

**The client uploads the successful execution result:**

```
ID=501&Return=0&CMD=DATA
```

### 12.1.1.4 Deliver Identity Card Information

Application scenario:

The server delivers the identity card information to the client. It is generally used to synchronize the identity card information edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)UPDATE\$(SP)identitycard\$(SP)Pin={\$XXX}\$(HT)SN\_Num={\$XXX}\$(HT)ID\_Num={\$XXX}\$(HT)DN\_Num={\$XXX}\$(HT)Name={\$XXX}\$(HT)Gender={\$XXX}\$(HT)Nation={\$XXX}\$(HT)Birthday={\$XXX}\$(HT)Valid\_Info={\$XXX}\$(HT)Address={\$XXX}\$(HT)Additional\_Info={\$XXX}\$(HT)Issuer={\$XXX}\$(HT)Photo={\$XXX}\$(HT)PhotoJPG={\$XXX}\$(HT)FP\_Template1={\$XXX}\$(HT)FP\_Template2={\$XXX}\$(HT)Reserve={\$XXX}\$(HT)Notice={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

**Note:**

identitycard: The table name, which is the identity card information

Pin: The user ID

SN\_Num: The physical number of the identity card

ID\_Num: The number of the citizen identity card

DN\_Num: The SN of the resident identity card (the card management number)

Name: The name

Gender: The gender code

Nation: The nationality code

0: Decoding Error

1: Han

2: Mongol

3: Hui

4: Tibetan

5: Uyghur

6: Miao

7: Yi

8: Zhuang

9: Buyi

10: Korean

11: Manchu

12: Dong

13: Yao

14: Bai

15: Tujia

16: Hani

17: Kazak

18: Dai

19: Li

20: Lisu



21: Va  
22: She  
23: Gaoshan  
24: Lahu  
25: Shui  
26: Dongxiang  
27: Naxi  
28: Jingpo  
29: Kirgiz  
30: Tu  
31: Daur  
32: Mulao  
33: Qiang  
34: Blang  
35: Salar  
36: Maonan  
37: Kelao  
38: Xibe  
39: Achang  
40: Pumi  
41: Tajik  
42: Nu  
43: Uzbek  
44: Russian  
45: Ewenki  
46: De'ang  
47: Baoan  
48: Yugu  
49: Jing  
50: Tatar  
51: Drung  
52: Oroqen  
53: Hezhe  
54: Monba  
55: Lhoba  
56: Jino  
57: Encoding Error  
97: Other

**98: Foreign Origin**

Birthday: The date of birth

Valid\_Info: The validity period

Address: The address

Additional\_Info: The additional address read by the machine

Issuer: The issuing authority

Photo: The photo data is encrypted and converted to the base64-based data before transmission.

PhotoJPG: The photo data is decrypted and converted to the base64-based data before transmission.

FP\_Template1: Features of the finger 1.

FP\_Template2: Features of the finger 2.

Reserve: A reserved field

Notice: The remarks

Example:

**The server delivers:**

C:502:DATA UPDATE identitycard Pin=1 SN\_Num=26121292460088826975

IDNum=210102199212182xxxx DNum= Name=Zhang San Gender=1 Nation=1

Birthday=19911218 ValidInfo=2013091220230912 Address=xxxxx, Tangxia Town, Dongguan,

Guangdong AdditionalInfo= Issuer=xxx Public Security Bureau Photo=V0xmAH4AMgAA/apmyEd8

PhotoJPG=Y0tKqWNPzyEd8Pn0EhRsgAAjeWPxiUzLaPU1w=

FPTemplate1=QwGIEgELUQAAAAAAAAAAAAAAAAACgBmnlcAf////////0wZqfwuJK78PzJg/lw

FPTemplate2=QwGIEgEQUAAAAAAAAAAAAAAAAADIBmkbyAP////////3UYCPxXQs38qEVW/rpLovwyUBv8W

V8 Reserve= Notice=

**The client uploads the successful execution result:**

ID=502&Return=0&CMD=DATA

### 12.1.1.5 Deliver User's Access Control Privilege

Application scenario:

The server delivers the user's access control privilege to the client. Generally, the privilege is delivered together with the user information.

The command format is as follows:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)UPDATE\$(SP)userauthorize\$(SP)Pin=\${XXX}\$(HT)AuthorizeTimezoneId=\${XXX}\$(HT)AuthorizeDoorId=\${XXX}\$(HT)DevID=\${XXX}

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

**Note:**

userauthorize: The table name, which is the user's access control privilege

AuthorizeTimezoneId: The ID of the user's time rule

AuthorizeDoorId: The ID of the user's door

It indicates to which doors of the device the privilege can be applied. The value is encoded in binary. Each door is represented by a binary bit. If the bit value is 1, the user has the privilege of this door. For details, see the following:

1 indicates LOCK1.

2 indicates LOCK2.

3 indicates LOCK1 and LOCK2.

4 indicates LOCK3.

5 indicates LOCK1 and LOCK3.

6 indicates LOCK2 and LOCK3.

7 indicates LOCK1, LOCK2, and LOCK3.

8 indicates LOCK4.

9 indicates LOCK1 and LOCK4.

10 indicates LOCK2 and LOCK4.

11 indicates LOCK1, LOCK2, and LOCK4.

12 indicates LOCK3 and LOCK4.

13 indicates LOCK1, LOCK3, and LOCK4.

14 indicates LOCK2, LOCK3, and LOCK4.

15 indicates LOCK1, LOCK2, LOCK3, and LOCK4.

If the four doors are numbered as 1, 2, 3, and 4 respectively, the value 15 obtained according to the following calculation formula:

$$1 << (1-1) + 1 << (2-1) + 1 << (3-1) + 1 << (4-1) = 15$$

DevID: The device ID. The value of AccSupportFunList counts from 0, and the value of the 26th bit is used to control the function.

Multiple records are connected by \${LF}.

Example:

**The server delivers the information of a user of which user ID is 1, the time rule ID is 1, and the door ID is 1:**

C:296:DATA UPDATE userauthorize Pin=1 AuthorizeTimezoneId=1 AuthorizeDoorId=1

DevID=1

**The client uploads the successful execution result:**

ID=296&Return=0&CMD=DATA

### 12.1.1.6 Deliver Holiday

Application scenario:

The server delivers the holiday data to the client. It is generally used to synchronize the holiday data edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)UPDATE\$(SP)holiday\$(SP)Holiday=\${XXX}\$(HT)HolidayType=\${XXX}\$(HT)Loop=\${XXX}

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

holiday: The table name, which is the holiday

Holiday: 20100101 means January 1, 2010.

HolidayType: The holiday type. Values are 1, 2, and 3, of which the meaning should be defined as needed.

Loop: 1 indicates that the year is different but the month and the day are the same. 2 indicates that all the year, month, and day must be the same.

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:503:DATA UPDATE holiday Holiday=20100101 HolidayType=1 Loop=1

**The client uploads the successful execution result:**

ID=503&Return=0&CMD=DATA

### 12.1.1.7 Deliver Time Rule

Application scenario:

The server delivers a time rule to the client. Generally, it is used to automatically deliver the time rule edited on the server.

Format:

The server delivers the command:

```
C:{$CmdID}:DATA$(SP)UPDATE$(SP)timezone$(SP)Timezoneld={$XXX}$(HT)SunTime1={$XXX}$(HT)SunTime2={$XXX}$(HT)SunTime3={$XXX}$(HT)MonTime1={$XXX}$(HT)MonTime2={$XXX}$(HT)MonTime3={$XXX}$(HT)TueTime1={$XXX}$(HT)TueTime2={$XXX}$(HT)TueTime3={$XXX}$(HT)WedTime1={$XXX}$(HT)WedTime2={$XXX}$(HT)WedTime3={$XXX}$(HT)ThuTime1={$XXX}$(HT)ThuTime2={$XXX}$(HT)ThuTime3={$XXX}$(HT)FriTime1={$XXX}$(HT)FriTime2={$XXX}$(HT)FriTime3={$XXX}$(HT)SatTime1={$XXX}$(HT)SatTime2={$XXX}$(HT)SatTime3={$XXX}$(HT)Hol1Time1={$XXX}$(HT)Hol1Time2={$XXX}$(HT)Hol1Time3={$XXX}$(HT)Hol2Time1={$XXX}$(HT)Hol2Time2={$XXX}$(HT)Hol2Time3={$XXX}$(HT)Hol3Time1={$XXX}$(HT)Hol3Time2={$XXX}$(HT)Hol3Time3={$XXX}
```

The client uploads the execution result:

```
ID={$CmdID}&Return={$XXX}&CMD=DATA
```

Note:

timezone: The table name, which is time group

Timezoneld: The ID of the time rule

SunTime1-SatTime3: From Sunday to Saturday, three time periods each day

Hol1Time1-Hol3Time3: Three holidays, three time periods each holiday

Note: Each time period lasts from StartTime to EndTime on the software. Before the time rule is delivered to the device, the software converts the time rule to an integer based on the convention

StartTime<<16+EndTime. For example, the converted value of 8:30-12:00 is 830 << 16 + 1200, that is, 0x33e04b0.

Example:

**The server delivers a time rule, in which the time rule ID is 2, the time period 1 on Monday is 00:01-00:03, and the time period 2 is 10:00-11:00:**

```
C:307:DATA UPDATE timezone Timezoneld=2 SunTime1=0 SunTime2=0 SunTime3=0
```

```
MonTime1=65539 MonTime2=65537100 MonTime3=0 TueTime1=0 TueTime2=0 TueTime3=0
```

```
WedTime1=0 WedTime2=0 WedTime3=0 ThuTime1=0 ThuTime2=0 ThuTime3=0 FriTime1=0 FriTime2=0
```

```
FriTime3=0 SatTime1=0 SatTime2=0 SatTime3=0 Hol1Time1=0 Hol1Time2=0 Hol1Time3=0 Hol2Time1=0
```

```
Hol2Time2=0 Hol2Time3=0 Hol3Time1=0 Hol3Time2=0 Hol3Time3=0
```

**The client uploads the execution result:**

ID=307&Return=0&CMD=DATA

**12.1.1.8 Deliver Fingerprint Template**

Application scenario:

The server delivers a fingerprint template, if any, to the client when delivering the user information.

Format:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)UPDATE\$(SP)templatev10

Pin={\$XXX}\$(HT)FingerID=\$(HT)Valid={\$XXX}\$(HT)Template={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

templatev10: Indicate that the data type is fingerprint template. The supported fingerprint algorithm version is no latter than 10.0.

Pin: The user ID

FingerID: The finger number. The value ranges from 0 to 9.

Valid: Describe whether the template is valid and is duress. The value meanings are described below:

Value	Description
0	Invalid
1	Valid
3	Duress

Template: Before a fingerprint template is transmitted, the original binary fingerprint template must be base64-encoded.

Multiple records are connected by \${LF}.

Example:

**The server delivers a common fingerprint of which the user ID is 1 and the fingerprint ID is 6:**

C:333:DATA UPDATE templatev10 Pin=1 FingerID=6 Valid=1

Template=Si9TUzlxAAADbG8ECAUHCc7QAAAabbWkBAAAAG5Eac2wNAG4PmwDZAAVjrQAiAIIPWwAzBAIPsgCOAMgOWmyOAFgPwwBWAi5i2wCUAJsPTACUbF0PrwCqAKIPgGy8AFAPwwAHAINjZADMANQPPADdbKgPjgDhAIAOoGzmADAIOIQApaADJiqwD5ALIOYwAEbasM/wAJAWcPn2wVAYoNhAdfAfJhbQAIAdYPCgAkbZ0PbAA2ASAP0Gw4AZQPI2jqbyCDiIW1BQuYSH1i7Y4XXYE5/d6Xgu24q3p2M/ymh0hngIH19llorBJt/II0bg4eU//33jP73D7onn2jDvmapxjD9F9QX4XA/ManEgtvK04TZ3tIFlqVxQBEgd6C+e9BAGWC3AiC6WSCofgxIEd4mZdKCj8T0eComsm0hGr9H+4NKPCai55NHQmD9bIWxnx79YNfpgK94RjfiCZAIAvxAIRcMoEAHYBdL

kEA88FhpMEAKvNAzBmAaQLglhitQQDAhFtXAYAn9z9/icPANEZjMAEmcOuZ3sKAJcdsn53AQQAsCIJ/48L  
 A8QkfcLAwmavBwPOLf3+/cFGzQCZXHuDwHIHAGcyAJP/UxMA91pbw8MUw//EaVzCOsMQbPpmmmnBw  
 UHCwq3Bb2wVAQe4oHfohcT/eMHBtREDMYrgRjExK44QA8KPfcPF/8IEwPyvwcBg/xCBwoykG32MwYNneK  
 8HAzmRXsJrwRTF2JPwkMLEwv/CQcDBk8HB/sEIAJ+TVAtZBwDHlxz9VQZshplgfhkBwpyorsHBfpPCg6pna6  
 sKAKqqcMUGwVVQCwCCwFDABFtPYAG/wJPJxwfDwqjAw0cJAKsD/fiXWFQZAQe+dldq+sPDgMDCwLvA  
 WmUBr8sQInYHCwPE0UPF/8N2BXsJbKzQHP3AwjrBaGUBrtktfMCuGwJj26ISwUWXB8LArXZvZfKYADDcpA  
 jBw8LCw8UGwv2uecH+wEQJxY3IKsD9KXYPAFHnw0b99/z//sEFwFBwAAvpsG3CBcDBrKHDw3rAizrAWJM  
 GAJLvPcUAVRI9DQinZMDABcDBrsPExsOJUDv/wV4EEKAXF8GOBBO/Khf/xAkQqjLhkVz7/sBmBdWTS+zCV  
 wgQaUoi//+S/0QNEONMSIfCrKDBaVJCAM5DAmwBC0VS

**The client returns the execution result:**

ID=333&Return=0&CMD=DATA

### 12.1.1.9 Deliver First-card Opening Information

Application scenario:

The server delivers the first-card opening data to the client. It is generally used to synchronize the first-card opening data edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)UPDATE\$(SP)firstcard\$(SP)DoorID={\$XXX}\$(HT)TimezoneID={\$XXX}\$(HT)Pin={\$XX  
 X}\$(HT)DevID={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

firstcard: The table name, which is the first-card opening table

Pin: The user ID

DoorID: The door ID

TimezoneID: The time zone ID

DevID: The device ID. The value of AccSupportFunList counts from 0, and the value of the 26th bit is used to control the function.

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:504:DATA UPDATE firstcard DoorID=1 TimezoneID=1 Pin=1 DevID=1

**The client uploads the successful execution result:**

ID=504&Return=0&CMD=DATA

### 12.1.1.10 Deliver Multi-card Opening Information

Application scenario:

The server delivers the multi-card opening data to the client. It is generally used to synchronize the multi-card opening data edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

```
C:${CmdID}:DATA$(SP)UPDATE$(SP)multimcard$(SP)Index=${XXX}$(HT)DoorId=${XXX}$(HT)Group1=${XX}$(HT)Group2=${XXX}$(HT)Group3=${XXX}$(HT)Group4=${XXX}$(HT)Group5=${XXX}$(HT)DevID=${XX}X}
```

**The client uploads the execution result:**

```
ID=${CmdID}&Return=${XXX}&CMD=DATA
```

Note:

multimcard: The table name, which is the multi-card opening table

Index: The index

DoorId: The door ID

Group1: The group 1

Group2: The group 2

Group3: The group 3

Group4: The group 4

Group5: The group 5

DevID: The device ID. The value of AccSupportFunList counts from 0, and the value of the 26th bit is used to control the function.

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

```
C:505:DATA UPDATE multimcard Index=1 DoorId=1 Group1=1 Group2=3 Group3=
Group4= Group5= DevID=1
```

**The client uploads the successful execution result:**

```
ID=505&Return=0&CMD=DATA
```



### 12.1.1.11 Deliver Linkage Details

Application scenario:

The server delivers the linkage details to the client. It is generally used to synchronize the linkage details edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

```
C:${CmdID}:DATA$(SP)UPDATE$(SP)inoutfun$(SP)Index=${XXX}$(HT)EventType=${XXX}$(HT)InAddr=${XXX}$(HT)OutType=${XXX}$(HT)OutAddr=${XXX}$(HT)OutTime=${XXX}$(HT)Reserved=${XXX}$(HT)InDevID=${XXX}$(HT)OutDevID=${XXX}
```

**The client uploads the execution result:**

```
ID=${CmdID}&Return=${XXX}&CMD=DATA
```

Note:

inoutfun: The table name, which is the linkage details table

Index: The index

EventType: The event triggered

InAddr: The position where the event is triggered, which is the door ID or the auxiliary input ID. The value ranges from 1 to 4. 0 means any position.

OutType: The output type. 0 means lock and 1 means auxiliary output.

OutAddr: The position of the output action. The value of the lock output ranges from 1 to 4. The value of auxiliary output ranges from 1 to 4.

OutTime: The time of the output action. 0 means closed. 1 to 254 means the lock is opened for n seconds, and 255 means normal open.

Reserved: Reserved for alignment. The low four bits indicate the linkage type. 0: door linkage; 1: reader linkage; and 2: IPC linkage. The high four bits indicate the action type. 16: Lock and 32: Unlock.

InDevID: The device ID. The value of AccSupportFunList counts from 0, and the value of the 26th bit is used to control the function.

OutDevID: The device ID. The value of AccSupportFunList counts from 0, and the value of the 26th bit is used to control the function.

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

```
C:506:DATA UPDATE inoutfun Index=1 EventType=204 InAddr=0 OutType=0  
OutAddr=1 OutTime=0 Reserved=0 InDevID=14 OutDevID=14
```

**The client uploads the successful execution result:**

```
ID=506&Return=0&CMD=DATA
```

### 12.1.1.12 Deliver scheduled Output Information

Application scenario:

The server delivers the scheduled output information to the client. It is generally used to synchronize the scheduled output information edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)UPDATE\$(SP)outrelaysetting\$(SP)Num=\${XXX}\$(HT)OutType=\${XXX}\$(HT)ActionType=\${XXX}\$(HT)TimezoneId=\${XXX}\$(HT)DevID=\${XXX}

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

outrelaysetting: The table name, which is the scheduled output table

Num: The output point or the number of the primary key of the auxiliary output property table

OutType: 0 indicates door and 1 indicates auxiliary output.

ActionType: 0 indicates none, 2 indicates normally closed, and 1 indicates normal open.

TimezoneId: The ID of the time zone

DevID: The device ID. The value of AccSupportFunList counts from 0, and the value of the 26th bit is used to control the function.

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:507:DATA UPDATE outrelaysetting Num=1 OutType=1 ActionType=0 TimezoneId=1  
DevID=14

**The client uploads the successful execution result:**

ID=507&Return=0&CMD=DATA

### 12.1.1.13 Deliver DLST Information

Application scenario:

The server delivers the DLST information to the client. It is generally used to synchronize the DLST information edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)UPDATE\$(SP)DSTSetting\$(SP)Year={\$XXX}\$(HT)StartTime={\$XXX}\$(HT)EndTime={\$XXX}\$(HT)Loop={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

DSTSetting: The table name, which is the DLST table

Year: The year

StartTime: The start time, in the format of MMIIWWHH, wherein MM indicates the month, II indicates the week, WW indicates the weekday, and HH indicates the hour.

EndTime: The end time, in the format of MMIIWWHH, wherein MM indicates the month, II indicates the week, WW indicates the weekday, and HH indicates the hour.

Loop: Whether it is a loop

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:508:DATA UPDATE DSTSetting Year=2018 StartTime=03020517 EndTime=08020508

Loop=1

**The client uploads the successful execution result:**

ID=508&Return=0&CMD=DATA

### 12.1.1.14 Deliver Device Property

Application scenario:

The server delivers the device properties to the client. It is generally used to synchronize the device properties edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)UPDATE\$(SP)DevProperty\$(SP)ID={\$XXX}\$(HT)Type={\$XXX}\$(HT)BusType={\$XXX}\$(HT)MachineType={\$XXX}\$(HT)Address={\$XXX}\$(HT)Mac={\$XXX}\$(HT)IPAddress={\$XXX}\$(HT)SN={\$XXX}\$(HT)IsMaster={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

DevProperty: The table name, which is the device property table

ID: The device ID

Type: The device type. 0: Door Unit, 1: Zigbee, and 2: BioCom48. Currently, all is 0.

BusType: The bus type. 0: 485 A, 1: 485B, 2: ZIGBEE, and 3: TCP/IP.

MachineType: The machine type. See Appendix 7.

Address: The device address

Mac: The device MAC address

IPAddress: The device IP address

SN: The device SN

IsMaster: Whether it is a master. 0: Master and 1: Slave.

Multiple records are connected by \${LF}.

Example:

#### The server delivers:

C:509:DATA UPDATE DevProperty ID=14 Type=0 BusType=0 MachineType=28 Address=0

Mac= IPAddress=192.168.223.222 SN=DDG7050067042500078 IsMaster=0

#### The client uploads the successful execution result:

ID=509&Return=0&CMD=DATA

### 12.1.1.15 Deliver Device Parameter

Application scenario:

The server delivers the device parameters to the client. It is generally used to synchronize the device parameters edited on the server to the client.

The command format is as follows:

#### The server delivers the command:

C:\${CmdID}:DATA\$(SP)UPDATE\$(SP)DevParameters\$(SP)ID=\${XXX}\$(HT)Name=\${XXX}\$(HT)Value=\${XXX}

#### The client uploads the execution result:

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

DevParameters: The table name, which is the device parameter table

ID: The device ID

Name: The parameter name. See Appendix 12.

Value: The parameter value

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:510:DATA UPDATE DevParameters ID=14    Name=IsSupportReaderEncrypt Value=0

**The client uploads the successful execution result:**

ID=510&Return=0&CMD=DATA

### 12.1.1.16 Deliver Door Property

Application scenario:

The server delivers the door properties to the client. It is generally used to synchronize the door properties edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)UPDATE\$(SP)DoorProperty\$(SP)ID=\${XXX}\$(HT)DevID=\${XXX}\$(HT)Address=\${XXX}\$(HT)Disable=\${XXX}

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

DoorProperty: The table name, which is the door property table

ID: The door ID

DevID: The slave ID

Address: The lock relay. 1: LOCK1, 2: LOCK2, 3: LOCK3, and 4: LOCK4

Disable: Enable or disable. 0: Enable. 1: Disable

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:511:DATA UPDATE DoorProperty ID=1    DevID=14    Address=1    Disable=0

**The client uploads the successful execution result:**

ID=511&Return=0&CMD=DATA

### 12.1.1.17 Deliver Door Parameter

Application scenario:

The server delivers the door parameters to the client. It is generally used to synchronize the door parameters edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)UPDATE\$(SP)DoorParameters\$(SP)ID={\$XXX}\$(HT)Name={\$XXX}\$(HT)Value={\$XX  
X}\$(HT)DevID={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

DoorParameters: The table name, which is the door parameter table

ID: The door ID

Name: The parameter name. See Appendix 10.

Value: The parameter value

DevID: The ID of the device property

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:512:DATA UPDATE DoorParameters ID=1 Name=MaskFlag Value=0 DevID=14

**The client uploads the successful execution result:**

ID=512&Return=0&CMD=DATA

### 12.1.1.18 Deliver Reader Property

Application scenario:

The server delivers the reader properties to the client. It is generally used to synchronize the reader properties edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)UPDATE\$(SP)ReaderProperty\$(SP)ID={\$XXX}\$(HT)DoorID={\$XXX}\$(HT)Type={\$XXX  
}\$(HT)Address={\$XXX}\$(HT)IPAddress={\$XXX}\$(HT)Port={\$XXX}\$(HT)MAC={\$XXX}InOut={\$XXX}\$(HT)Disa  
ble={\$XXX}\$(HT)VerifyType={\$XXX}\$(HT)Multicast={\$XXX}\$(HT)DevID={\$XXX}\$(HT)OfflineRefuse={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

ReaderProperty: The table name, which is the reader property table

ID: The reader ID

DoorID: The door ID

Type: The reader type, which is calculated by binary bits. 0: Disable. 1: 485 Reader. 2: Wiegand Reader. 3: 485 Reader or Wiegand Reader. 4: Network Reader. 8: Zigbee Reader

Address: The reader address, applicable for the Wiegand reader and 485 reader

IPAddress: The reader IP address

Port: The reader port

MAC: The reader MAC address

InOut: The input or output reader. 0: input and 1: Output

Disable: Enable or disable. 0: Enable. 1: Disable

VerifyType: The reader verification mode

Multicast: The multicast address

DevID: The device ID

OfflineRefuse: Whether offline access is supported. 0: Accept and 1: Refuse

Multiple records are connected by \${LF}.

Example:

#### The server delivers:

C:513:DATA UPDATE ReaderProperty ID=1 DoorID=1 Type=2 Address=1 IPAddress=  
Port=4376 MAC= InOut=0 Disable=0 VerifyType=0 Multicast= DevID=14  
OfflineRefuse=0

#### The client uploads the successful execution result:

ID=513&Return=0&CMD=DATA

### 12.1.1.19 Deliver Reader Parameter

Application scenario:

The server delivers the reader parameters to the client. It is generally used to synchronize the reader parameters edited on the server to the client.

The command format is as follows:

#### The server delivers the command:

C:\${CmdID}:DATA\$(SP)UPDATE\$(SP)ReaderParameters\$(SP)ID=\${XXX}\$(HT)DevID=\${XXX}\$(HT)Name=\${X  
XX}\$(HT)Value=\${XXX}

#### The client uploads the execution result:

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

ReaderParameters: The table name, which is the reader parameter table

ID: The reader ID

DevID: The device ID

Name: The parameter name. See Appendix 11.

Value: The parameter value

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:514:DATA UPDATE ReaderParameters ID=1 DevID=1 Name=MaskFlag Value=1

**The client uploads the successful execution result:**

ID=514&Return=0&CMD=DATA

### 12.1.1.20 Deliver Auxiliary Input Property

Application scenario:

The server delivers the auxiliary input properties to the client. It is generally used to synchronize the auxiliary input properties edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)UPDATE\$(SP)AuxIn\$(SP)ID=\${XXX}\$(HT)DevID=\${XXX}\$(HT)Address=\${XXX}\$(HT)Disable=\${XXX}

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

AuxIn: The table name, which is the auxiliary input property table

ID: The auxiliary input ID

DevID: The device ID

Address: Auxiliary input. 1: AuxIn1 and 2: AuxIn2

Disable: Enable or disable. 0: Enable. 1: Disable

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:515:DATA UPDATE AuxIn ID=1 DevID=14 Address=1 Disable=0

**The client uploads the successful execution result:**

ID=515&Return=0&CMD=DATA

### 12.1.1.21 Deliver Auxiliary Output Property

Application scenario:



The server delivers the auxiliary output properties to the client. It is generally used to synchronize the auxiliary output properties edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)UPDATE\$(SP)AuxOut\$(SP)ID={\$XXX}\$(HT)DevID={\$XXX}\$(HT)Address={\$XXX}\$(HT)Disable={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

AuxOut: The table name, which is the auxiliary output property table

ID: The auxiliary output ID

DevID: The device ID

Address: The auxiliary output relay. 1: AuxOut1

Disable: Enable or disable. 0: Enable. 1: Disable

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:516:DATA UPDATE AuxOut ID=1    DevID=14    Address=1    Disable=0

**The client uploads the successful execution result:**

ID=516&Return=0&CMD=DATA

### 12.1.1.22 Deliver Default Wiegand Format

Application scenario:

The server delivers the default Wiegand format to the client. It is generally used to synchronize the default Wiegand format edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)UPDATE\$(SP)DefaultWGFormat\$(SP)ID={\$XXX}\$(HT)CardBit={\$XXX}\$(HT)SiteCode={\$XXX}\$(HT)FormatName={\$XXX}\$(HT)CardFormat={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

DefaultWGFormat: The table name, which is the default Wiegand format table

ID: The index

CardBit: The number of bits

SiteCode: The site code

FormatName: The user-defined Wiegand name

CardFormat: The Wiegand format

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:517:DATA UPDATE DefaultWGFormat ID=2 CardBit=26 SiteCode=0 FormatName=Wiegand  
26 CardFormat=pcccccccccccccccccccccp:eeeeeeeeeeeeeeeeoooooooooooo

**The client uploads the successful execution result:**

ID=517&Return=0&CMD=DATA

### 12.1.1.23 Deliver Wiegand Format

Application scenario:

The server delivers the Wiegand format to the client. It is generally used to synchronize the Wiegand format edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)UPDATE\$(SP)WGFormat\$(SP)ID=\${XXX}\$(HT)CardBit=\${XXX}\$(HT)SiteCode=\${XXX}\$(HT)FormatName=\${XXX}\$(HT)CardFormat=\${XXX}

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

WGFormat: The table name, which is the Wiegand format table

ID: The index

CardBit: The number of bits

SiteCode: The site code

FormatName: The user-defined Wiegand name

CardFormat: The Wiegand format

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:518:DATA UPDATE WGFormat ID=2 CardBit=26 SiteCode=0 FormatName=Wiegand 26  
CardFormat=pcccccccccccccccccccccp:eeeeeeeeeeeeeeeeoooooooooooo

**The client uploads the successful execution result:**

ID=518&Return=0&CMD=DATA

**12.1.1.24 Deliver Wiegand Format of Wiegand Reader**

Application scenario:

The server delivers the Wiegand format of the Wiegand reader to the client. It is generally used to synchronize the Wiegand format of the Wiegand reader edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)UPDATE\$(SP)ReaderWGFormat\$(SP)ReaderID={\$XXX}\$(HT)DevID={\$XXX}\$(HT)WGFormatID={\$XXX}\$(HT)ParityVerifyDisable={\$XXX}\$(HT)ReversalType={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

ReaderWGFormat: The table name, which is the table of the Wiegand format of the Wiegand reader

ReaderID: The reader ID

DevID: The device ID. The value of AccSupportFunList counts from 0, and the value of the 26th bit is used to control the function.

WGFormatID: The index of the Wiegand format table

ParityVerifyDisable: Whether to disable the parity check. 1: Disable and 0: Enable

ReversalType: 1: Invert all bits. 2: Invert 0 and 1. 3: Exchange low bits with high bits. 4: Exchange a low bit with a high bit.

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:519:DATA UPDATE ReaderWGFormat ReaderID=1 DevID=14 WGFormatID=2

ParityVerifyDisable=0 ReversalType=0

**The client uploads the successful execution result:**

ID=519&Return=0&CMD=DATA

**12.1.1.25 Deliver Input Control (by Time Period) Information**

Application scenario:

The server delivers the input control (by time period) information to the client. It is generally used to synchronize the input control (by time period) information edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)UPDATE\$(SP)InputIOSetting\$(SP)Number={\$XXX}\$(HT)InType={\$XXX}\$(HT)TimeZoneld={\$XXX}\$(HT)DevID={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

InputIOSetting: The table name, which is the table of input control by time period

Number: The input ID, which is the door ID in the door property table or the ID in the auxiliary input property table

InType: 0: Exit button. 1: Auxiliary input

TimeZoneld: The ID of the time zone

DevID: The device ID. The value of AccSupportFunList counts from 0, and the value of the 26th bit is used to control the function.

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:520:DATA UPDATE InputIOSetting Number=1 InType=0 TimeZoneld=1 DevID=14

**The client uploads the successful execution result:**

ID=520&Return=0&CMD=DATA

### 12.1.1.26 Deliver Verification Modes in Different Time Periods

Application scenario:

The server delivers the verification modes in different time periods to the client. It is generally used to synchronize the verification modes edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)UPDATE\$(SP)DiffTimezoneVS\$(SP)TimezoneID={\$XXX}\$(HT)SunTime1={\$XXX}\$(HT)SunTime1VSUser={\$XXX}\$(HT)SunTime1VSDoor={\$XXX}\$(HT)SunTime2={\$XXX}\$(HT)SunTime2VSUser={\$XXX}\$(HT)SunTime2VSDoor={\$XXX}\$(HT)SunTime3={\$XXX}\$(HT)SunTime3VSUser={\$XXX}\$(HT)SunTime3VSDoor={\$XXX}\$(HT)MonTime1={\$XXX}\$(HT)MonTime1VSUser={\$XXX}\$(HT)MonTime1VSDoor={\$XXX}\$(HT)MonTime2={\$XXX}\$(HT)MonTime2VSUser={\$XXX}\$(HT)MonTime2VSDoor={\$XXX}\$(HT)MonTime3={\$XXX}\$(HT)MonTime3VSUser={\$XXX}\$(HT)MonTime3VSDoor={\$XXX}

```

=${XXX}$(HT)MonTime3VSUser=${XXX}$(HT)MonTime3VSDoor=${XXX}$(HT)TueTime1=${XXX}$(HT)TueTime1VSUser=${XXX}$(HT)TueTime1VSDoor=${XXX}$(HT)TueTime2=${XXX}$(HT)TueTime2VSUser=${XXX}$(HT)TueTime2VSDoor=${XXX}$(HT)TueTime3=${XXX}$(HT)TueTime3VSUser=${XXX}$(HT)TueTime3VSDoor=${XXX}$(HT)WedTime1=${XXX}$(HT)WedTime1VSUser=${XXX}$(HT)WedTime1VSDoor=${XXX}$(HT)WedTime2=${XXX}$(HT)WedTime2VSUser=${XXX}$(HT)WedTime2VSDoor=${XXX}$(HT)WedTime3=${XXX}$(HT)WedTime3VSUser=${XXX}$(HT)WedTime3VSDoor=${XXX}$(HT)ThuTime1=${XXX}$(HT)ThuTime1VSUser=${XXX}$(HT)ThuTime1VSDoor=${XXX}$(HT)ThuTime2=${XXX}$(HT)ThuTime2VSUser=${XXX}$(HT)ThuTime2VSDoor=${XXX}$(HT)ThuTime3=${XXX}$(HT)ThuTime3VSUser=${XXX}$(HT)ThuTime3VSDoor=${XXX}$(HT)FriTime1=${XXX}$(HT)FriTime1VSUser=${XXX}$(HT)FriTime1VSDoor=${XXX}$(HT)FriTime2=${XXX}$(HT)FriTime2VSUser=${XXX}$(HT)FriTime2VSDoor=${XXX}$(HT)FriTime3=${XXX}$(HT)FriTime3VSUser=${XXX}$(HT)FriTime3VSDoor=${XXX}$(HT)SatTime1=${XXX}$(HT)SatTime1VSUser=${XXX}$(HT)SatTime1VSDoor=${XXX}$(HT)SatTime2=${XXX}$(HT)SatTime2VSUser=${XXX}$(HT)SatTime2VSDoor=${XXX}$(HT)SatTime3=${XXX}$(HT)SatTime3VSUser=${XXX}$(HT)SatTime3VSDoor=${XXX}$(HT)Hol1Time1=${XXX}$(HT)Hol1Time1VSUser=${XXX}$(HT)Hol1Time1VSDoor=${XXX}$(HT)Hol1Time2=${XXX}$(HT)Hol1Time2VSUser=${XXX}$(HT)Hol1Time2VSDoor=${XXX}$(HT)Hol1Time3=${XXX}$(HT)Hol1Time3VSUser=${XXX}$(HT)Hol1Time3VSDoor=${XXX}$(HT)Hol2Time1=${XXX}$(HT)Hol2Time1VSUser=${XXX}$(HT)Hol2Time1VSDoor=${XXX}$(HT)Hol2Time2=${XXX}$(HT)Hol2Time2VSUser=${XXX}$(HT)Hol2Time2VSDoor=${XXX}$(HT)Hol2Time3=${XXX}$(HT)Hol2Time3VSUser=${XXX}$(HT)Hol2Time3VSDoor=${XXX}$(HT)Hol3Time1=${XXX}$(HT)Hol3Time1VSUser=${XXX}$(HT)Hol3Time1VSDoor=${XXX}$(HT)Hol3Time2=${XXX}$(HT)Hol3Time2VSUser=${XXX}$(HT)Hol3Time2VSDoor=${XXX}$(HT)Hol3Time3=${XXX}$(HT)Hol3Time3VSUser=${XXX}$(HT)Hol3Time3VSDoor=${XXX}

```

**The client uploads the execution result:**

ID={CmdID}&Return=\${XXX}&CMD=DATA

**Note:**

DiffTimezoneVS: The table name, which is the table of verification modes in different time periods

TimezoneID: The index ID

SunTime1-SatTime3: From Sunday to Saturday, three time periods each day

Hol1Time1-Hol3Time3: Three holidays, three time periods each holiday

SunTime1VSUser-SatTime3VSUser: From Sunday to Saturday, three time periods each day, the verification modes in different time periods when the user is different

Hol1Time1VSUser-Hol3Time3VSUser: Three holidays, three time periods each holiday, the verification modes in different time periods when the user is different

SunTime1VSDoor-SatTime3VSDoor: From Sunday to Saturday, three time periods each day, the verification modes in different time periods when the door is different

Hol1Time1VSDoor-Hol3Time3VSDoor: Three holidays, three time periods each holiday, the verification modes in different time periods when the door is different

Note: Each time period lasts from StartTime to EndTime on the software. Before the time rule is delivered to the device, the software converts the time rule to an integer based on the convention

StartTime<<16+EndTime. For example, the converted value of 8:30-12:00 is 830 << 16 + 1200, that is, 0x33e04b0.

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

```
C:521:DATA UPDATE DiffTimezoneVS TimezoneID=1  SunTime1=2359  SunTime1VSUser=255
SunTime1VSDoor=255  SunTime2=0  SunTime2VSUser=255  SunTime2VSDoor=255  SunTime3=0
SunTime3VSUser=255  SunTime3VSDoor=255  MonTime1=2359  MonTime1VSUser=7
MonTime1VSDoor=5  MonTime2=0  MonTime2VSUser=255  MonTime2VSDoor=255
MonTime3=0  MonTime3VSUser=255  MonTime3VSDoor=255  TueTime1=2359
TueTime1VSUser=255  TueTime1VSDoor=255  TueTime2=0  TueTime2VSUser=255
TueTime2VSDoor=255  TueTime3=0  TueTime3VSUser=255  TueTime3VSDoor=255
WedTime1=2359  WedTime1VSUser=255  WedTime1VSDoor=255  WedTime2=0
WedTime2VSUser=255  WedTime2VSDoor=255  WedTime3=0  WedTime3VSUser=255
WedTime3VSDoor=255  ThuTime1=2359  ThuTime1VSUser=255  ThuTime1VSDoor=255
ThuTime2=0  ThuTime2VSUser=255  ThuTime2VSDoor=255  ThuTime3=0  ThuTime3VSUser=255
ThuTime3VSDoor=255  FriTime1=2359  FriTime1VSUser=255  FriTime1VSDoor=255  FriTime2=0
FriTime2VSUser=255  FriTime2VSDoor=255  FriTime3=0  FriTime3VSUser=255  FriTime3VSDoor=255
SatTime1=2359  SatTime1VSUser=255  SatTime1VSDoor=255  SatTime2=0  SatTime2VSUser=255
SatTime2VSDoor=255  SatTime3=0  SatTime3VSUser=255  SatTime3VSDoor=255  Hol1Time1=2359
Hol1Time1VSUser=255  Hol1Time1VSDoor=255  Hol1Time2=0  Hol1Time2VSUser=255
Hol1Time2VSDoor=255  Hol1Time3=0  Hol1Time3VSUser=255  Hol1Time3VSDoor=255  Hol2Time1=2359
Hol2Time1VSUser=255  Hol2Time1VSDoor=255  Hol2Time2=0  Hol2Time2VSUser=255
Hol2Time2VSDoor=255  Hol2Time3=0  Hol2Time3VSUser=255  Hol2Time3VSDoor=255  Hol3Time1=2359
Hol3Time1VSUser=255  Hol3Time1VSDoor=255  Hol3Time2=0  Hol3Time2VSUser=255
Hol3Time2VSDoor=255  Hol3Time3=0  Hol3Time3VSUser=255  Hol3Time3VSDoor=255
```

**The client uploads the successful execution result:**

```
ID=521&Return=0&CMD=DATA
```

### 12.1.1.27 Deliver Verification Modes for Different Doors in Different Periods

Application scenario:

The server delivers the verification modes for different doors in different time periods to the client. It is generally used to synchronize the verification modes edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)UPDATE\$(SP)DoorVSTimezone\$(SP)DoorID={\$XXX}\$(HT)TimezoneID={\$XXX}\$(HT)DevID={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

DoorVSTimezone: The table name, which is the table of verification modes for different doors in different time periods

DoorID: The door ID

TimezoneID: Correspond to DiffTimezoneVS

DevID: The device ID. The value of AccSupportFunList counts from 0, and the value of the 26th bit is used to control the function.

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:522:DATA UPDATE DoorVSTimezone DoorID=4 TimezoneID=1 DevID=14

**The client uploads the successful execution result:**

ID=522&Return=0&CMD=DATA

### 12.1.1.28 Deliver Verification Modes for Different Users in Different Periods

Application scenario:

The server delivers the verification modes for different users in different time periods to the client. It is generally used to synchronize the verification modes edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)UPDATE\$(SP)PersonalVSTimezone\$(SP)PIN={\$XXX}\$(HT)DoorID={\$XXX}\$(HT)TimezoneID={\$XXX}\$(HT)DevID={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

PersonalVSTimezone: The table name, which is the table of verification modes for different users in different time periods

PIN: The user ID

DoorID: The door ID

TimezoneID: Correspond to DiffTimezoneVS

DevID: The device ID. The value of AccSupportFunList counts from 0, and the value of the 26th bit is used to control the function.

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:523:DATA UPDATE PersonalVSTimezone PIN=1 DoorID=4 TimezoneID=1 DevID=14

**The client uploads the successful execution result:**

ID=523&Return=0&CMD=DATA

### 12.1.1.29 Deliver Super User Privilege

Application scenario:

The server delivers the super user privilege to the client. It is generally used to synchronize the super user privilege edited on the server to the client.

The command format is as follows:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)UPDATE\$(SP)SuperAuthorize\$(SP)Pin=\${XXX}\$(HT)DoorID=\${XXX}\$(HT)DevID=\${XXX}

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

SuperAuthorize: The table name, which is the super user privilege table

Pin: The user ID

DoorID: The door ID

DevID: The device ID. The value of AccSupportFunList counts from 0, and the value of the 26th bit is used to control the function.

Multiple records are connected by \${LF}.

Example:

**The server delivers:**

C:524:DATA UPDATE SuperAuthorize Pin=1 DoorID=4 DevID=14

**The client uploads the successful execution result:**

ID=524&Return=0&CMD=DATA



### 12.1.1.30 Deliver Comparison Photo

Application scenario:

The server delivers comparison photos.

The command format is as follows:

**The server delivers the command:**

C:{\$CmdID}:DATA{\$SP}UPDATE{\$SP}biophoto{\$SP}PIN={\$XXX}{\$HT}Type={\$XXX}{\$HT}Size={\$XXX}{\$HT}Content={\$XXX}{\$HT}Format={\$XXX}{\$HT}Url={\$XXX}{\$HT}PostBackTmpFlag={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

PIN={\$XXX}: The user ID

Type={\$XXX}: The biometric type:

Value Meaning

0 General

1 Fingerprint

2 Face (Near Infrared)

3 Voice Print

4 Iris

5 Retina

6 Palm Print

7 Finger Vein

8 Palm

9 Visible Light Face

Size={\$XXX}: The length of the base64-encoded binary biometric photo.

Content={\$XXX}: The original binary biometric photo must be base64-encoded before transmission.

Url={\$XXX}: The path where the file is stored on the server. Currently, only photos in JGP format are supported.

Format={\$XXX}: The delivery mode. 0: base64. 1: URL

PostBackTmpFlag={\$XXX}: Whether to return the template data after image conversion (0: not required, 1: required). No PostBackTmpFlag parameter, it is not required to return by default.

Multiple records are connected by {\$LF}.

Note:

When URL is a relative path, use the relative path directly.

### 12.1.1.31 Deliver User Photo

Application scenario:

The server delivers the user photo.

The command format is as follows:

### The server delivers the command:

```
C:${CmdID}:DATA${SP}UPDATE${SP}userpic${SP}pin=${XXX}${HT}size=${XXX}${HT}format=${xxx}${HT}url
=${xxx}${HT}content=${XXX}
```

### The client uploads the execution result:

ID=\${CmdID}&amp;Return=\${XXX}&amp;CMD=DATA

Note:

pin=\${XXX}: The user ID.

size=\${XXX}: The length of the base64-encoded binary user photo.

format=\${xxx}: Delivery mode, 0: base64 mode, 1: url mode.

url=\${xxx}: Server user photo storage address.

Multiple records are connected by `${LF}`.

Example:

### The server delivers:

C:524:DATA UPDATE userpic pin=2 size=138620

`content=/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAOEBAOEBAOEBAOEBAO`

[illegible]

EBAQEBAQEBAQH/wAARCAGQAoADASIAAhEBAxEB/8QAHwAAAQUBAQEBAQ.....

**The client uploads the successful execution result:**

ID=524&amp;Return=0&amp;CMD=DATA

### 12.1.1.32 Deliver Unified Template

Application scenario:

The server delivers the united template.

The command format is as follows:

### The server delivers the command:

C:{\$CmdID}:DATA{\$SP}UPDATE{\$SP}biodata{\$SP}pin={\$XXX}{\$HT}No={\$XXX}{\$HT}index={\$XXX}{\$HT}valid={\$XXX}{\$HT}duress={\$XXX}{\$HT}type={\$XXX}{\$HT}majorver={\$XXX}{\$HT}minorver={\$XXX}{\$HT}format={\$XXX}{\$HT}tmp={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&return={\$XXX}&CMD=DATA

**Note:**

pin={\$XXX}: The user ID.

no={\$XXX}: The number of specific biont. The default value is 0.

[Fingerprint] The number ranges from 0 to 9, corresponding to the little finger, ring finger, middle finger, index finger, and thumb of the left hand, and thumb, index finger, middle finger, ring finger, and little finger of the right hand respectively.

[Finger Vein] Same to the fingerprints.

[Face] All is 0.

[Iris] 0 is the iris of the left eye and 1 is the iris of the right eye.

[Palm] 0 is the palm of the left hand and 1 is the palm of the right hand.

index={\$XXX}: The number of the specific biont template. Multiple templates may be stored for one finger. It is calculated from 0.

valid={\$XXX}: Whether it is valid. 0: Invalid; 1: Valid. The default value is 1.

duress={\$XXX}: Whether it is duress. 0: Not Duress; 1: Duress. The default value is 0.

type={\$XXX}: The biometric type

Value Meaning

0 General

1 Fingerprint

2 Face

3 VoicePrint

4 Iris

5 Retina

6 Palm Print

7 Finger Vein

8 Palm

9 Visible Light Face

majorver={\$XXX}: The major version number. For example, for the fingerprint algorithm 10.3, the major version number is 10 and the minor version number is 3.

[Fingerprint]: 9.0, 10.3 and 12.0

[Finger Vein]: 3.0

[Face]: 5.0, 7.0 and 8.0

[Palm]: 1.0

minorver=\${XXX}: The minor version number. For example, for the fingerprint algorithm 10.3, the major version number is 10 and the minor version number is 3.

[Fingerprint]: 9.0, 10.3 and 12.0

[Finger Vein]: 3.0

[Face]: 5.0, 7.0 and 8.0

[Palm]: 1.0

pin=\${XXX} : Name of attendance photo.

format=\${XXX}: The template format. The fingerprint template formats include ZK, ISO, and ANSI.

[Fingerprint]

Value Format

0 ZK

1 ISO

2 ANSI

[Finger Vein]

Value Format

0 ZK

[Face]

Value Format

0 ZK

[Palm]

Value Format

0 ZK

tmp=\${XXX}: The template data. The original binary fingerprint template must be base64-encoded.

Multiple records are connected by \${LF}.

Example:

#### The server delivers:

C:524:DATA UPDATE biodata pin=2 no=0 index=0 valid=1 duress=0 type=9 majorver=5 minorver=8  
format=0

tmp=AAAAA.....

#### The client uploads the successful execution result:

ID=524&return=0&CMD=DATA

## 12.1.2 Delete

Application scenario:

This command is used to control the deletion of device data on the software, for example, the user and fingerprint template.

Format:

C:\$(CmdID):DATA\$(SP)DELETE\$(SP)\$(TableName)\$(SP)\$(Cond)

Note:

\$(Cond): The deletion condition list, in the format of:

Condition field name1=value1\$(HT)condition field name2=value2\$(HT)condition field name3=value3

When the deletion condition list is \*, it means to clear the table. When the deletion condition list contains multiple conditions, the conditions are in AND relationship.

Multiple conditions are separated by \$(LF).

### 12.1.2.1 Delete User Information

Application scenario:

It is generally used to delete a user or all users.

Format:

**The server delivers the command:**

C:\$(CmdID):DATA\$(SP)DELETE\$(SP)user\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

user: The table name, which is the user information

\$(Cond): Generally, Pin=x means deleting the user of which ID is x, and \* means deleting all users.

Note:

Generally, when deleting a user, the server deletes the user's verification mode and access control privilege together. Therefore, when delivering the command to delete a user, the server also delivers the command to delete the user's access control privilege and verification mode (such as a fingerprint).

Example:

**Delete the user of which ID is 1 (can delete the user data together):**

C:335:DATA DELETE userauthorize Pin=1

C:336:DATA DELETE templatev10 Pin=1

C:337:DATA DELETE user Pin=1

Wherein, commands 1 and 2 are used to delete the access control privilege and the fingerprint template respectively. Command 3 is used to delete the user.

**The client returns the execution result:**

```
ID=335&Return=0&CMD=DATA
```

```
ID=336&Return=0&CMD=DATA
```

```
ID=337&Return=0&CMD=DATA
```

### 12.1.2.2 Delete Extended User Information

Application scenario:

It is generally used to delete an extended user or all extended users.

Format:

**The server delivers the command:**

```
C:${CmdID}:DATA$(SP)DELETE$(SP)extuser$(SP)$(Cond)
```

**The client uploads the execution result:**

```
ID=${CmdID}&Return=${XXX}&CMD=DATA
```

Note:

extuser: The table name, which is the extended user information

\$(Cond): Generally, Pin=x means deleting the extended user of which ID is x, and \* means deleting all extended users.

Example:

**Delete an extended user of which ID is 1:**

```
C:337:DATA DELETE extuser Pin=1
```

**The client returns the execution result:**

```
ID=337&Return=0&CMD=DATA
```

### 12.1.2.3 Delete Multi-card User

Application scenario:

It is generally used to delete a multi-card user or all multi-card users.

Format:

**The server delivers the command:**

```
C:${CmdID}:DATA$(SP)DELETE$(SP)multipcarduser$(SP)$(Cond)
```

**The client uploads the execution result:**

```
ID=${CmdID}&Return=${XXX}&CMD=DATA
```

Note:

multipcarduser: The table name, which is the multi-card user

\$(Cond): Generally, Pin=x means deleting the multi-card user of which ID is x, and \* means deleting all multi-card users.

Example:

**Delete the multi-card user of which ID is 1:**

C:337:DATA DELETE mulcarduser Pin=1

**The client returns the execution result:**

ID=337&Return=0&CMD=DATA

### 12.1.2.4 Delete User's Access Control Privilege

Application scenario:

After deleting a user, the server also deletes the user's access control privilege.

Format:

**The server delivers the command:**

C:(CmdID):DATA\$(SP)DELETE\$(SP)userauthorize\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

userauthorize: The table name, which is the user's access control privilege

\$(Cond): Generally, Pin=x means deleting the access control privilege of the user of which ID is x, and \* means deleting the access control privilege of all users.

Example:

Delete the access control privilege of the user of which ID is 1:

C:335:DATA DELETE userauthorize Pin=1

The client uploads the execution result:

ID=335&Return=0&CMD=DATA

### 12.1.2.5 Delete Access Control Record

Application scenario:

It is generally used to delete all the access control records of the device.

Format:

**The server delivers the command:**

C:({CmdID}):DATA\$(SP)DELETE\$(SP)transaction\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

transaction: The table name, which is the access control record

\$(Cond): \* means deleting all the records.

Example:

Delete all the access control records:

C:123:DATA DELETE transaction \*

The client uploads the execution result:

ID=123&Return=0&CMD=DATA

### 12.1.2.6 Delete Fingerprint Template

Application scenario:

Generally, it is used to delete a specified fingerprint template of a device used on the server or delete a user as well as the user's fingerprint template.

Format:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)DELETE\$(SP)templatev10\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

templatev10: The table name, which is the user's fingerprint template

\$(Cond): Pin=x means deleting all the fingerprint templates of the user of which ID is x. FingerID=x means deleting a specified fingerprint template.

\* means deleting all the users.

Example:

**Delete the fingerprint template of a user of which ID is 1 and fingerprint ID is 6:**

C:342:DATA DELETE templatev10 Pin=1 FingerID=6

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA



### 12.1.2.7 Delete Holiday

Application scenario:

It is generally used to delete all the holidays of the device.

Format:

**The server delivers the command:**

C:(CmdID):DATA\$(SP)DELETE\$(SP)holiday\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

holiday: The table name, which is the holiday

\$(Cond): \* means deleting all the holidays.

Example:

**Delete all the holidays:**

C:123: DATA DELETE holiday \*

**The client uploads the execution result:**

ID=123&Return=0&CMD=DATA

### 12.1.2.8 Delete Time Period

Application scenario:

It is generally used to delete the data in a time period or the data in all time periods.

Format:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)DELETE\$(SP)timezone\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

holiday: The table name, which is the time period

\$(Cond): Timezoneld=x means deleting the time period of which Timezoneld is x.

\* means deleting all the time periods.

Example:

**Delete a time period of which Timezoneld is 1:**

C:342:DATA DELETE timezone Timezoneld=1

**The client uploads the execution result:**

```
ID=342&Return=0&CMD=DATA
```

### 12.1.2.9 Delete First-card Opening Record

Application scenario:

It is generally used to delete a first-card opening record or all the first-card opening records.

Format:

**The server delivers the command:**

```
C:${CmdID}:DATA$(SP)DELETE$(SP)firstcard$(SP)$(Cond)
```

**The client uploads the execution result:**

```
ID=${CmdID}&Return=${XXX}&CMD=DATA
```

Note:

firstcard: The table name, which is the first-card opening record

\$(Cond): DoorID=x means deleting the first-card opening record of which DoorID is x.

Pin=x means deleting the first-card opening record of which Pin is x.

\* means deleting all the first-card opening records.

Example:

**Delete the first-card opening record of which DoorID or Pin is 1:**

```
C:342:DATA DELETE firstcard DoorID=1
```

```
C:343: DATA DELETE firstcard Pin=1
```

**The client uploads the execution result:**

```
ID=342&Return=0&CMD=DATA
```

```
ID=343&Return=0&CMD=DATA
```

### 12.1.2.10 Delete Multi-card Opening Record

Application scenario:

It is generally used to delete a multi-card opening record or all the multi-card opening records.

Format:

**The server delivers the command:**

```
C:${CmdID}:DATA$(SP)DELETE$(SP)multimcard$(SP)$(Cond)
```

**The client uploads the execution result:**

```
ID=${CmdID}&Return=${XXX}&CMD=DATA
```

Note:

multimcard: The table name, which is the multi-card opening table  
\$(Cond): Index=x means deleting the multi-card opening record of which Index is x.  
\* means deleting all the multi-card opening records.

Example:

**Delete the multi-card opening record of which Index is 1:**

C:342:DATA DELETE multimcard Index=1

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

### 12.1.2.11 Delete Linkage Details

Application scenario:

It is generally used to delete a linkage detail or all the linkage details.

Format:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)DELETE\$(SP)inoutfun\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

inoutfun: The table name, which is the linkage details table  
\$(Cond): Index=x means deleting the multi-card opening record of which Index is x.  
\* means deleting all the multi-card opening records.

Example:

**Delete the linkage details of which Index is 1:**

C:342:DATA DELETE inoutfun Index=1

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

### 12.1.2.12 Delete Scheduled Output

Application scenario:

It is generally used to delete a scheduled output or all the scheduled outputs.

Format:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)DELETE\$(SP)outrelaysetting\$(SP)\$(Cond)

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

outrelaysetting: The table name, which is the scheduled output table  
 \$(Cond): Num=x means deleting the scheduled output of which Num is x.  
 \* means deleting all the scheduled outputs.

Example:

**Delete all the scheduled outputs:**

C:342:DATA DELETE outrelaysetting Num=1

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

### 12.1.2.13 Delete DLST Record

Application scenario:

It is generally used to delete all the DLST records.

Format:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)DELETE\$(SP)DSTSetting\$(SP)\$(Cond)

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

DSTSetting: The table name, which is the DLST table  
 \$(Cond): \* means deleting all the DLST records.

Example:

**Delete the scheduled output of which Num is 1:**

C:342: DATA DELETE DSTSetting \*

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

### 12.1.2.14 Delete Device Property

Application scenario:

It is generally used to delete all the device properties.

Format:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)DELETE\$(SP)DevProperty\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

DevProperty: The table name, which is the device property table

\$(Cond): \* means deleting all the device properties.

Example:

**Delete all the device properties:**

C:342: DATA DELETE DevProperty \*

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

### 12.1.2.15 Delete Device Parameter

Application scenario:

It is generally used to delete all the device parameters.

Format:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)DELETE\$(SP)DevParameters\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

DevParameters: The table name, which is the device parameter table

\$(Cond): \* means deleting all the device parameters.

Example:

**Delete all the device parameters:**

C:342:DATA DELETE DevParameters \*

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

### 12.1.2.16 Delete Door Property

Application scenario:

It is generally used to delete all the door properties.

Format:

**The server delivers the command:**

C:\${CmdID};DATA\$(SP)DELETE\$(SP)DoorProperty\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

DoorProperty: The table name, which is the door property table  
\$(Cond): \* means deleting all the door properties.

Example:

**Delete all the door properties:**

C:342:DATA DELETE DoorProperty \*

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

### 12.1.2.17 Delete Door Parameter

Application scenario:

It is generally used to delete all the door parameters.

Format:

**The server delivers the command:**

C:\${CmdID};DATA\$(SP)DELETE\$(SP)DoorParameters\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

DoorParameters: The table name, which is the door parameter table  
\$(Cond): \* means deleting all the door parameters.

Example:

**Delete all the door parameters:**

C:342:DATA DELETE DoorParameters \*

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

### 12.1.2.18 Delete Reader Property

Application scenario:

It is generally used to delete all the reader properties.

Format:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)DELETE\$(SP)ReaderProperty\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

ReaderProperty: The table name, which is the reader property table  
\$(Cond): \* means deleting all the reader properties.

Example:

**Delete all the reader properties:**

C:342:DATA DELETE ReaderProperty \*

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

### 12.1.2.19 Delete Reader Parameter

Application scenario:

It is generally used to delete all the reader parameters.

Format:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)DELETE\$(SP)ReaderParameters\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

ReaderParameters: The table name, which is the reader parameter table  
\$(Cond): \* means deleting all the reader parameters.

Example:

**Delete all the reader parameters:**

C:342:DATA DELETE ReaderParameters \*

**The client uploads the execution result:**

```
ID=342&Return=0&CMD=DATA
```

### 12.1.2.20 Delete Auxiliary Input

Application scenario:

It is generally used to delete all the auxiliary inputs.

Format:

**The server delivers the command:**

```
C:${CmdID}:DATA$(SP)DELETE$(SP)AuxIn$(SP)$(Cond)
```

**The client uploads the execution result:**

```
ID=${CmdID}&Return=${XXX}&CMD=DATA
```

Note:

AuxIn: The table name, which is the auxiliary input table

\$(Cond): \* means deleting all the auxiliary inputs.

Example:

**Delete all the auxiliary inputs:**

```
C:342: DATA DELETE AuxIn *
```

**The client uploads the execution result:**

```
ID=342&Return=0&CMD=DATA
```

### 12.1.2.21 Delete Auxiliary Output

Application scenario:

It is generally used to delete all the auxiliary outputs.

Format:

**The server delivers the command:**

```
C:${CmdID}:DATA$(SP)DELETE$(SP)AuxOut$(SP)$(Cond)
```

**The client uploads the execution result:**

```
ID=${CmdID}&Return=${XXX}&CMD=DATA
```

Note:

AuxOut: The table name, which is the auxiliary output table

\$(Cond): \* means deleting all the auxiliary outputs.

Example:



**Delete all the auxiliary outputs:**

C:342:DATA DELETE AuxOut \*

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

### 12.1.2.22 Delete Default Wiegand Format

Application scenario:

It is generally used to delete a default Wiegand format or all the default Wiegand formats.

Format:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)DELETE\$(SP)DefaultWGFormat\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

DefaultWGFormat: The table name, which is the default Wiegand format table

\$(Cond): ID=x means deleting the default Wiegand format of which ID is x.

CardBit=x means deleting the default Wiegand format of which CardBit is x.

\* means deleting all the default Wiegand formats.

Example:

**Delete the Wiegand format of which ID is 1 or CardBit is 26.**

C:342:DATA DELETE DefaultWGFormat ID=1

C:343:DATA DELETE DefaultWGFormat CardBit=26

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

ID=343&Return=0&CMD=DATA

### 12.1.2.23 Delete Wiegand Format

Application scenario:

It is generally used to delete a Wiegand format or all the Wiegand formats.

Format:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)DELETE\$(SP)WGFormat\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

WGFormat: The table name, which is the Wiegand format table

\$(Cond): ID=x means deleting the Wiegand format of which ID is x.

CardBit=x means deleting the Wiegand format of which CardBit is x.

\* means deleting all the Wiegand formats.

Example:

**Delete the Wiegand format of which ID is 1 or CardBit is 26.**

C:342:DATA DELETE WGFormat ID=1

C:343:DATA DELETE WGFormat CardBit=26

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

ID=343&Return=0&CMD=DATA

### 12.1.2.24 Delete Reader Wiegand Format

Application scenario:

It is generally used to delete the Wiegand format of a reader or the Wiegand formats of all readers.

Format:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)DELETE\$(SP)WGFormat\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

WGFormat: The table name, which is the reader Wiegand format table

\$(Cond): ReaderID=x means deleting the Wiegand format of the reader of which ReaderID is x.

DevID=x means deleting the Wiegand format of the reader of which DevID is x.

\* means deleting the Wiegand formats of all the readers.

Example:

**Delete the Wiegand format of the reader of which ReaderID is 1 or DevID is 24:**

C:342:DATA DELETE WGFormat ReaderID=1

C:343:DATA DELETE WGFormat DevID=24

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

ID=343&Return=0&CMD=DATA

### 12.1.2.25 Delete Input Control (by Time Period)

Application scenario:

It is generally used to delete all the input control (by time period) data.

Format:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)DELETE\$(SP)InputIOSetting\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

InputIOSetting: The table name, which is the table of input control by time period

\$(Cond): \* means deleting all the input control (by time period) data.

Example:

**Delete all the input control (by time period) data:**

C:342:DATA DELETE InputIOSetting \*

**The client uploads the execution result:**

ID=342&Return=0&CMD=DAT

### 12.1.2.26 Delete Verification Modes in Different Time Periods

Application scenario:

It is generally used to delete a verification mode in different time periods or all the verification modes in different time periods.

Format:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)DELETE\$(SP)DiffTimezoneVS\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

DiffTimezoneVS: The table name, which is the table of verification modes in different time periods

\$(Cond): TimezoneID=x means deleting the verification mode in different time periods of which TimezoneID is x.

\* means deleting all the verification modes in different time periods.

Example:

**Delete the verification mode in different time periods of which TimezoneID is 1:**

C:342:DATA DELETE DiffTimezoneVS TimezoneID=1

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

### 12.1.2.27 Delete a Door's Verification Modes in Different Time Periods

Application scenario:

It is generally used to delete a door's verification modes in different time periods or all the doors' verification modes in different time periods.

Format:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)DELETE\$(SP)DoorVSTimezone\$(SP)\$(Cond)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

DoorVSTimezone: The table name, which is the table of the door's verification modes in different time periods

\$(Cond): DoorID=x means deleting the verification modes in different time periods of the door of which DoorID is x.

\* means deleting all the doors' verification modes in different time periods.

Example:

**Delete the verification modes in different time periods of the door of which DoorID is 1:**

C:342:DATA DELETE DoorVSTimezone DoorID=1

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

### 12.1.2.28 Delete User's Verification Modes in Different Time Periods

Application scenario:

It is generally used to delete a user's verification modes in different time periods or all the users' verification modes in different time periods.

Format:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)DELETE\$(SP)PersonalVSTimezone\$(SP)\$(Cond)

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

PersonalVSTimezone: The table name, which is the table of the user's verification modes in different time periods

\$(Cond): PIN=x means deleting the verification mode in different time periods of a user of which PIN is x.

PIN=x and DoorID=x mean deleting the verification mode in different time periods of a user of which PIN is x and DoorID is x.

\* means deleting all the users' verification modes in different time periods.

Example:

**Delete the verification modes in different time periods of the user of which PIN is 1 or PIN and DoorID are 1:**

C:342:DATA DELETE PersonalVSTimezone PIN=1

C:343:DATA DELETE PersonalVSTimezone PIN=1 DoorID=1

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

ID=343&Return=0&CMD=DATA

## 12.1.2.29 Delete Super User Privilege

Application scenario:

It is generally used to delete the privilege of a super user or all the super users.

Format:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)DELETE\$(SP)SuperAuthorize\$(SP)\$(Cond)

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

SuperAuthorize: The table name, which is the super user privilege table

\$(Cond): Pin=x means deleting the privilege of the super user of which Pin is x.

\* means deleting the privilege of all the super users.

Example:

**Delete the privilege of the super user of which Pin is 1:**

C:342:DATA DELETE SuperAuthorize Pin=1

**The client uploads the execution result:**

ID=342&Return=0&CMD=DATA

### 12.1.2.30 Delete Comparison Photo

Application scenario:

It is generally used to delete the user comparison photo.

Format:

**The server delivers the command:**

C:\${CmdID}:DATA\${SP}DELETE\${SP}biophoto\${SP}PIN=\${XXX}

**The client uploads the execution result:**

ID=\${XXX}&Return=\${XXX}&CMD=DATA

Note:

PIN=\${XXX}: The user ID

### 12.1.2.31 Delete User Photo

Application scenario:

It is used to delete the user photo.

The command format is as follows:

**The server delivers the command:**

C:\${CmdID}:DATA\${SP}DELETE\${SP}userpic\${SP}pin=\${XXX}

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

pin=\${XXX}: The user ID.

### 12.1.2.32 Delete Unified Template

Application scenario:

It is used to delete the unified template.

The command format is as follows:

**The server delivers the command:**

C:{\$CmdID}:DATA{\$SP}DELETE{\$SP}biodata{\$SP}Type={\$XXX}

C:{\$CmdID}:DATA{\$SP}DELETE{\$SP}biodata{\$SP}Type={\$XXX}{\$SP}pin={\$XXX}{\$SP}no={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&return={\$XXX}&CMD=DATA

Note:

type={\$XXX}: The biometric type

Value Meaning

0 General

1 Fingerprint

2 Face

3 VoicePrint

4 Iris

5 Retina

6 Palm Print

7 Finger Vein

8 Palm

9 Visible Light Face

pin={\$XXX}: The user ID.

no={\$XXX}: The number of specific biont. The default value is 0.

[Fingerprint] The number ranges from 0 to 9, corresponding to the little finger, ring finger, middle finger, index finger, and thumb of the left hand, and thumb, index finger, middle finger, ring finger, and little finger of the right hand respectively.

[Finger Vein] Same to the fingerprints.

[Face] All is 0.

[Iris] 0 is the iris of the left eye and 1 is the iris of the right eye.

[Palm] 0 is the palm of the left hand and 1 is the palm of the right hand.

## 12.1.3 Count

Application scenario:

It is used to count the number of records in a specified table or the number of records that meet the specified condition in a specified table.

Format:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)COUNT\$(SP)\$(TableName)\$(SP)\$(Cond)

The client uploads the statistical result:

POST

/iclock/querydata?SN=\$(SerialNumber)&type=count&cmdid={\$CmdID}&tablename=user&count={\$XXX}  
&packcnt={\$XXX}&packidx={\$XXX} HTTP/1.1

...

\$(TableName)={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA

Note:

\$(Cond): The filter condition list. Condition field name1=value1\t condition field name2=value2\t  
Condition field name3=value3. When the filter condition list is x, the whole table is counted. When the list  
contains multiple filter conditions, the conditions are in AND relationship.

/iclock/querydata: The URI of the data that is uploaded by the client and meets the condition

type=count: The type of data to be uploaded by the client is statistics.

count: The number of data entries returned.

packcnt: The total number of packages. When there is a great amount of data, the data needs to be  
divided into different packages. This value indicates how many packages the data is divided into.

packidx: The package ID, which indicates which of all packages is currently being sent.

\$(TableName)={\$XXX}: The left parameter indicates the table name, and the right value indicates the  
number of data entries in the table that meet the condition.

The sub-command COUNT contains three interaction processes: The client gets the command, the server  
returns the COUNT command, the client returns the statistical result, the server returns OK, the client  
returns the command execution result, and then the server returns OK. The details are not repeated  
below, and only the key points will be listed.

### 12.1.3.1 Get the User Count

Application scenario:

It is generally used to get the number of users of the device.

Format:

**The server delivers the command:**

C:{\$CmdID}:DATA\$(SP)COUNT\$(SP)user\$(SP)\$(Cond)

**The client uploads the statistical result:**

user={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=DATA COUNT



Note:

user: The table name, which is the number of obtained user information tables

\$(Cond): Generally no condition is set, which means getting the number of all users.

Example:

**The server delivers the command:**

C:288:DATA COUNT user

**The client uploads the statistical result:**

POST

/iclock/querydata?SN=3383154200002&type=count&cmdid=288&tablename=user&count=1&packcnt=1  
&packidx=1 HTTP/1.1

Cookie: token=af65a75608cf5b80fbb3b48f0b4df95a

Host: 192.168.213.17:8088

Content-Length: 6

...

user=5

**The client uploads the execution result:**

ID=288&Return=0&CMD=DATA COUNT

### 12.1.3.2 Get Access Control Record Count

Application scenario:

It is used to get the number of access control records from the client.

Note:

The interaction process and command format are the same as those of the command to get the user count. You only need to replace the table name with "transaction".

### 12.1.3.3 Get Fingerprint Template Count

Application scenario:

It is used to get the number of fingerprint templates from the client.

Note:

The interaction process and command format are the same as those of the command to get the user count. You only need to replace the table name with "template10".

### 12.1.3.4 Get Comparison Photo Count

Application scenario:

It is generally used to get the number of comparison photos from the device.

Format:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)COUNT\$(SP)biophoto\$(SP)Type=\${XXX}\$(HT)\$(Cond)

**The client uploads the statistical result:**

biophoto=\${XXX}

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA COUNT

Note:

biophoto: The table name, which is the number of obtained comparison photos

Type=\${XXX}: The biometric type

Value	Meaning
0	General
1	Fingerprint
2	Face
3	VoicePrint
4	Iris
5	Retina
6	Palm Print
7	Finger Vein
8	Palm
9	Visible Light Face

\$(Cond): Generally no condition is set, which means getting the number of all comparison photos.

Example:

**The server delivers the command:**

Does not support hybrid identification protocol, only supports the number of visible light face comparison photos:

C:80:DATA COUNT biophoto

After supporting the hybrid identification protocol, you must specify the type of comparison photo.

C:80:DATA COUNT biophoto Type=9

**The client uploads the statistical result:**

POST

/iclock/querydata?SN=1809140005&type=count&cmdid=80&tablename=biophoto&count=1&packcnt=1

&packidx=1 HTTP/1.1

Cookie: token=af65a75608cf5b80fbb3b48f0b4df95a

Host: 192.168.213.17:8088

Content-Length: 10

...

biophoto=2

**The client uploads the execution result:**

ID=80&Return=0&CMD=DATA COUNT

### 12.1.3.5 Get Unified Template count

Application scenario:

It is generally used to get the number of device unified templates of the specified type.

Format:

**The server delivers the command:**

C:\${CmdID};DATA\$(SP)COUNT\$(SP)biodata\$(SP)Type=\${XXX}\$(HT)\$(Cond)

**The client uploads the statistical result:**

biodata=\${XXX}

**The client uploads the execution result:**

ID=\${CmdID}&return=\${XXX}&CMD=DATA COUNT

Note:

biophoto: The table name, which is the number of obtained comparison photos

Type=\${XXX}: The biometric type

Value Meaning

0 General

1 Fingerprint

2 Face

3 VoicePrint

4 Iris

5 Retina

6 Palm Print

7 Finger Vein

8 Palm

9 Visible Light Face

\$(Cond): Generally no condition is set, which means getting the number of all comparison photos.

Example:

**The server delivers the command:**

C:80:DATA COUNT biodata

**The client uploads the statistical result:**

P POST

/iclock/querydata?SN=1809140005&type=count&cmdid=80&tablename=biodata&count=1&packcnt=1&packidx=1 HTTP/1.1

Cookie: token=af65a75608cf5b80fbb3b48f0b4df95a

Host: 192.168.213.17:8088

Content-Length: 10

.....

biodata=2

**The client uploads the execution result:**

ID=80&Return=0&CMD=DATA COUNT

## 12.1.4 Query

**The server actively requires the client to upload data that meets the query condition.**

Format:

**The server delivers the command:**

C:\${CmdID}:DATA\$(SP)QUERY\$(SP)tablename=\$(XXX),fielddesc=\$(XXX),filter=\$(XXX)

**The client uploads the data that meets the condition:**

POST

/iclock/querydata?SN=\$(SerialNumber)&type=tabledata&cmdid=\$(XXX)&tablename=\$(TableName)&count=\$(XXX)&packcnt=\$(XXX)&packidx=\$(XXX) HTTP/1.1

...

\$(DataRecord)

**Response from the server:**

\$(TableName)=\$(XXX)

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=DATA

Note:

tablename is the name of the table to query

fielddesc is a field. When this field is set to \*, all fields of the table are queried; otherwise, the specified field is queried.

The filter is the query condition. When this field is set to \*, the data is not filtered. When this field is set to NewRecord and tablename is transaction, new records are queried.

On some devices, when this field is set to starttime=\tendtime=, records in the specified time period are queried.

/iclock/querydata: The URI of the data that is uploaded by the client and meets the condition

type: The data type. If it is set to tabledata in the QUERY command, it means the data in the table.

count: The number of data entries returned.

packcnt: The total number of packages. When there is a great amount of data, the data needs to be divided into different packages. This value indicates how many packages the data is divided into.

packidx: The package ID, which indicates which of all packages is currently being sent.

\$(DataRecord): The data that meets the condition and needs to be uploaded. The specific format depends on the table name.

return: The returned value is the quantity of this query.

### 12.1.4.1 Query User

Application scenario:

The server actively requires the client to upload the specified user. It is generally used to get all the users from the device.

Format:

#### The server delivers the command:

C:415:DATA\$(SP)QUERY\$(SP)tablename=user,fielddesc=\$(XXX),filter=\$(XXX)

#### The client uploads the data:

POST

/iclock/querydata?SN=\$(SerialNumber)&type=tabledata&cmdid={CmdID}&tablename=user&count=\$(XX  
X)&packcnt=\$(XXX)&packidx=\$(XXX) HTTP/1.1

...

\$(DataRecord)

#### Response from the server:

user=\$(XXX)

#### The client uploads the successful execution result:

ID=\${CmdID}&Return=\$(XXX)&CMD=DATA QUERY

Note:

tablename: When it is set to user, it means the user table and the data processed is the user data.

\$(DataRecord): When the table name is user, the data format is the user data format. For details, see Upload User Information.

Example:

#### The server delivers the command:

C:415:DATA QUERY tablename=user,fielddesc=\*,filter=\*

**The client uploads the data of three users:**

POST

/iclock/querydata?SN=3383154200002&type=tabledata&cmdid=415&tablename=user&count=3&packcnt=1&packidx=1 HTTP/1.1

...

user uid=1 cardno= pin=1 password= group=1 starttime=0 endtime=0 name= privilege=0 disable=0 verify=0

user uid=2 cardno= pin=2 password=1 group=1 starttime=0 endtime=0 name= privilege=0 disable=0 verify=0

user uid=3 cardno= pin=3 password= group=1 starttime=0 endtime=0 name= privilege=0 disable=0 verify=0

**The server responds that it has received the data of the three user:**

user=3

**The client uploads the successful execution result:**

ID=415&Return=3&CMD=DATA QUERY

### 12.1.4.2 Query Fingerprint Template

Application scenario:

The server actively requires the client to upload the specified fingerprint template. It is generally used to get all the fingerprint templates after getting all the users.

Note:

The command is the same as the command to query users, except that you need to replace the table name with templatev10. The \$(DataRecord) format is the fingerprint template format. For details, see Upload Fingerprint Template.

### 12.1.4.3 Query Access Control Record

Application scenario:

The server actively requires the client to upload the specified access control record. It is generally used to get all access control records or the access control record in the specified range after the ACCOUNT command failed.

Format:

**The server receives the command:**

C:415:DATA\$(SP)QUERY\$(SP)tablename=transaction,fielddesc=\$(XXX),filter=\$(XXX)

**The client uploads the data:**

POST

/iclock/querydata?SN=\${SerialNumber}&type=tabledata&cmdid={CmdID}&tablename=transaction&count=\${(XXX)}&packcnt=\${(XXX)}&packidx=\${(XXX)} HTTP/1.1

...

\$(DataRecord)

**Response from the server:**

transaction=\${(XXX)}

**The client uploads the successful execution result:**

ID=\${CmdID}&Return=\${(XXX)}&CMD=DATA

**Note:**

The command format is same to the format of the command used to query users. You need to replace the table name with transaction.

\$(DataRecord): Its format is same to the format of the access control record, which is:

transaction\$(SP)cardno=\${(XXX)}\$(HT)pin=\${(XXX)}\$(HT)verified=\${(XXX)}\$(HT)doorid=\${(XXX)}\$(HT)eventtype=\${(XXX)}\$(HT)inoutstate=\${(XXX)}\$(HT)time\_second=\${(XXX)}\$(HT)index=\${(XXX)}\$(HT)sitecode=\${(XXX)}\$(HT)devid=\${(XXX)}\$(HT)maskflag=\${(xxx)}\$(HT)temperature=\${(xxx)}\$(HT)convtemperature=\${(xxx)}

transaction: The table name, which means that the data type is access control record

cardno: The card number

pin: The user ID

verified: The verification mode code. For details, see Appendix 3 Description of Verification Mode Code.

doorid: The door ID

eventtype: The event type

inoutstate: The in/out status. 0 indicates In, and 1 indicates Out.

time\_second: The time stamp of the record, for example, time\_second=583512660, in which time\_second is calculated according to the algorithm in Appendix 5, and the specific time yyyy-mm-dd-hh-mm-ss is calculated according to the method in Appendix 6.

index: The ID of the access control record

sitecode: The site code

devid: The device ID

maskflag: Value 0 or 1, 1 means wearing a mask

temperature: The value is the temperature data with a decimal point, for example: 36.2

convtemperature: The value is the temperature data with a decimal point. If the server does not send the IRTempUnitTrans parameter, then the unit of the temperature upload is subject to the IRTempUnit parameter.

Multiple records are connected by \${LF}.

**Note:**

When the filter is set to NewRecord, new records that are not uploaded are queried.

Example:

**The server delivers the command:**

C:415:DATA QUERY tablename=user,fielddesc=\*,filter=\*

**The client uploads the access control record:**

POST

/iclock/querydata?SN=3383154200002&type=tabledata&cmdid=415&tablename=transaction&count=1&packcnt=1&packidx=1 HTTP/1.1

...

transaction cardno=0 pin=0 verified=200 doorid=1 eventtype=100 inoutstate=2  
time\_second=548003688 index=92 sitecode=0 devid=1 sn=3383154200002

**Response from the server:**

transaction=1

**The client uploads the successful execution result:**

ID=415&Return=1&CMD=DATA

### 12.1.4.4 Query Wi-Fi List

Application scenario:

The software delivers a command to query the Wi-Fi list. After searching for surrounding Wi-Fis, the device uploads the SSIDs and other information to the software.

Format:

**The server delivers the command:**

C:415:DATA\$(SP)QUERY\$(SP)tablename=[APList],fielddesc=\$(XXX),filter=\$(XXX)

**The client uploads the data:**

POST

/iclock/querydata?SN=\$(SerialNumber)&type=vmtable&cmdid={CmdID}&tablename=APList&count=\$(XX)&packcnt=\$(XXX)&packidx=\$(XXX) HTTP/1.1

...

\$(DataRecord)

**Response from the server:**

APList=\$(XXX)

**The client uploads the successful execution result:**

ID=\${CmdID}&Return=\$(XXX)&CMD=DATA

Note:

tablename: [APList], which is a Wi-Fi virtual table

\$(DataRecord): The Wi-Fi data format is as follows:

APList\$(SP)ecn=\$(XXX)\$(HT)ssid=\$(XXX)\$(HT)rssi=\$(XXX)\$(HT)mac=\$(XXX)



APList: A Wi-Fi virtual table  
 ecn: Direct congestion flag  
 ssid: The Wi-Fi user name  
 rssi: The signal  
 mac: The MAC address

Example:

**The server delivers the command:**

C:415:DATA QUERY tablename=[APList],fielddesc=\*,filter=\*

**The data uploaded by the client:**

**Upload the request protocol:**

POST

/iclock/querydata?SN=\$(SerialNumber)&type=vmtable&cmdid=415&tablename=APList&count=%d&packcnt=%d&packidx=%d HTTP/1.1

Cookie: token=\${XXX}

Host: \${ServerIP}:\${ServerPort}

Content-Length: \${XXX}

...

APList ecn=%?\tssid=%?\trssi=%?\tmac=%?

APList ecn=%?\tssid=%?\trssi=%?\tmac=%?

APList ecn=%?\tssid=%?\trssi=%?\tmac=%?

**Response from the server:**

APList=3

**The client uploads the successful execution result:**

ID=415&Return=3&CMD=DATA

### 12.1.4.5 Query Comparison Photo

Application scenario:

The server actively requires the client to upload the specified comparison photo. It is generally used to get all the comparison photos from the device.

Format:

**The server delivers the command:**

Does not support hybrid identification protocol, only supports obtaining visible light face comparison photos:

C:99:DATA\$(SP)QUERY\$(SP)tablename=biophoto,fielddesc=\$(XXX),filter=\$(XXX)

After supporting the hybrid identification protocol, you must specify the type of comparison photo:

C:99:DATA\$(SP)QUERY\$(SP)tablename=biophoto,fielddesc=\$(XXX),filter=Type=\$(XXX)\$(HT)\$(XXX)

Type=\${XXX}: The biometric type

Value Meaning

0 General

1 Fingerprint

2 Face

3 VoicePrint

4 Iris

5 Retina

6 Palm Print

7 Finger Vein

8 Palm

9 Visible Light Face

#### **The client uploads the data:**

POST

/iclock/querydata?SN=\$(SerialNumber)&type=tabledata&cmdid={CmdID}&tablename=biophoto&count=\$(XXX)&packcnt=\$(XXX)&packidx=\$(XXX) HTTP/1.1

...

\$(DataRecord)

#### **Response from the server:**

biophoto=\$(XXX)

#### **The client uploads the successful execution result:**

ID=\${CmdID}&Return=\$(XXX)&CMD=DATA QUERY

Note:

tablename: When it is set to biophoto, it means the comparison photo table and the data processed is the comparison photo data.

\$(DataRecord): When the table name is biophoto, the data format is the format of the comparison photo data. For details, see Upload Comparison Photo.

Example:

#### **The server delivers the command:**

C:99:DATA QUERY tablename=biophoto,fielddesc=\*,filter=\*

#### **The client uploads the data of two comparison photos:**

POST

/iclock/querydata?SN=1809140005&type=tabledata&cmdid=99&tablename=biophoto&count=1&packcnt=2&packidx=1 HTTP/1.1

...

biophoto pin=123 filename=123.jpg type=9 size=95040 content=AAAA.....

**The server responds that it has received the data of one comparison photo:**

biophoto=1

POST

/iclock/querydata?SN=1809140005&type=tabledata&cmdid=99&tablename=biophoto&count=1&packcnt=2&packidx=2

HTTP/1.1

...

biophoto pin=124 filename=124.jpg type=9 size=95040 content=AAAA.....

**The server responds that it has received the data of one comparison photo:**

biophoto=1

**After the client uploads all the comparison photos, the result is returned.**

ID=99&Return=2&CMD=DATA QUERY

### 12.1.4.6 Query Unified Template

Application scenario:

The server actively requires the client to upload the specified unified template data. It is generally used to get the unified template data of the device specified type.

Format:

**The server delivers the command:**

Does not support hybrid identification protocol, only supports obtaining visible light face unified templates:

C:99:DATA\$(SP)QUERY\$(SP)tablename=biodata,fielddesc=\$(XXX),filter=\$(XXX)

After supporting the hybrid identification protocol, you must specify the type of unified templates:

C:99:DATA\$(SP)QUERY\$(SP)tablename=biodata,fielddesc=\$(XXX),filter=Type=\$(XXX)\$(HT)\$(XXX)

Type=\$(XXX): The biometric type

Value Meaning

0 General

1 Fingerprint

2 Face

3 VoicePrint

4 Iris

5 Retina

6 Palm Print

7 Finger Vein

8 Palm

9 Visible Light Face

**The client uploads the data:**

POST

/iclock/querydata?SN=\$(SerialNumber)&type=tabledata&cmdid={CmdID}&tablename=biodata&count=\$(XXX)&packcnt=\$(XXX)&packidx=\$(XXX) HTTP/1.1

.....

\$(DataRecord)

**Response from the server:**

biodata=\$(XXX)

**The client uploads the successful execution result:**

ID=\${CmdID}&return=\$(XXX)&CMD=DATA QUERY

Note:

tablename: When it is set to biodata, it means the unified template table and the data processed is the unified template data.

\$(DataRecord): When the table name is biodata, the data format is the format of the unified template data. For details, see Upload Unified Template.

Example:

**The server delivers the command:**

C:99:DATA QUERY tablename=biodata,fielddesc=\*,filter=Type=9

**The client uploads the data of two unified templates:**

POST

/iclock/querydata?SN=1809140005&type=tabledata&cmdid=99&tablename=biodata&count=1&packcnt=2&packidx=1 HTTP/1.1

.....

biodata pin=1 no=0 index=0 valid=1 duress=0 type=1 majorver=10 minorver=0 format=0  
tmp=apUBD.....

**The server responds that it has received the data of one unified template:**

biodata=1

POST

/iclock/querydata?SN=1809140005&type=tabledata&cmdid=99&tablename=biodata&count=1&packcnt=2&packidx=2

HTTP/1.1

.....

biodata pin=123 no=0 index=0 valid=1 duress=0 type=1 majorver=10 minorver=0 format=0  
tmp=AAAAA.....

**The server responds that it has received the data of one unified template:**

biodata=1

**After the client uploads all the unified templates, the result is returned.**

ID=99&Return=2&CMD=DATA QUERY

**The server specifies the query condition:**

Query conditions must include Type, different query conditions are separated by {HT}.

**Query visible light face template of User PIN = 1**

C:99:DATA QUERY tablename=biodata,fielddesc=\\\*,filter=Type=9\\tPin=1

**Query fingerprint template of the right index finger of User PIN = 1**

C:99:DATA QUERY tablename=biodata,fielddesc=\\\*,filter=Type=1\\tPin=1\\tNo=6

## 12.2 Account

This command is currently used to upload the access control records for software account checking only on the server. Account checking means that the software compares its access control records with those uploaded by the device, to check whether any access control record is missing.

### 12.2.1 Pull SDK Device

Format:

**The server delivers the command:**

C:{\$CmdID}:ACCOUNT\$(SP)transaction\$(SP)ContentType=tgz\$(SP)MaxIndex=0

**The client uploads the data that meets the condition:**

POST

/iclock/querydata?SN=\$(SerialNumber)&type=tabledata&cmdid={\$CmdID}&tablename=transaction&count={\$XXX}&datafmt=3&Stamp=9999

...

name=transaction.tgz

size={\$XXX}

datacode=base64

datatype=tgz\$(NULL)\$(DataRecord)

**Response from the server:**

transaction={\$XXX}

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=ACCOUNT&transaction&StartTime={\$XXX}&EndTime={\$XXX}&RecordSum={\$XXX}

Note:

transaction: The table name, which is the access control record

ContentType: The text format. tgz means a compressed package.

MaxIndex: Access control records of which the ID is greater than this value need to be uploaded. When the value is 0, all access control records need to be uploaded.

/iclock/querydata: The URI of the access control record that is uploaded by the client and meets the condition

type: The type of the data to upload. If it is set to tabledata in the ACCOUNT command, it means the data in the table.

tablename: The table name. Currently, the ACCOUNT command is used only for the access control records, and therefore all the table names in the ACCOUNT command is transaction.

count: The number of data entries uploaded

datafmt: The format of the data to upload. Currently, it is fixed to 3 in this command, which means a compressed package.

Stamp: The timestamp. Currently, it is fixed to 9999 in this command.

name: The name of the compressed package. transaction.tgz means a compressed package of access control records.

size: The size of the compressed package

datacode: The data format used when the compressed package is uploaded. base64 means that the compressed package is base64-encoded before upload.

datatype: The data type. tgz means a compressed package.

\$(DataRecord): Multiple access control records need to be packaged into a compressed package and then base64-encoded before upload. For the format details, see Upload Access Control Record.

transaction=\${XXX}: \${XXX} is the number of access control records uploaded by the client.

return: The number of access control records that meet the condition

CMD: It is fixed to ACCOUNT in the ACCOUNT command.

StartTime: The start time of the access control record. It is reserved currently and left blank during upload.

EndTime: The end time of the access control record. It is reserved currently and left blank during upload.

RecordSum: The number of access control records that meet the condition

Example:

**Upload all access control records in the form of a compressed package:**

**The server delivers the command:**

C:291:ACCOUNT transaction ContentType=tgz MaxIndex=0

**The client uploads a piece of data consisting of 41 access control records:**

POST

/iclock/querydata?SN=\$(SerialNumber)&type=tabledata&cmdid=291&tablename=transaction&count=41  
&datafmt=3&Stamp=9999

...

name=transaction.tgz

size=636

datacode=base64

datatype=tgz\0 [base64 data stream]

**Response from the server:**

transaction=41

**The client uploads the command execution result:**

ID=291&Return=41&CMD=ACCOUNT&transaction&StartTime=&EndTime=&RecordSum=41

## 12.2.2 Controller

Format:

**The server delivers the following three types of commands:**

- (1) C:\${CmdID}:ACCOUNT\$(SP)transaction\$(SP)MaxIndex=\${XXX}
- (2) C:\${CmdID}:ACCOUNT\$(SP)transaction\$(SP)StartIndex=\${XXX}\$(HT)EndIndex=\${XXX}
- (3) C:\${CmdID}:ACCOUNT\$(SP)transaction\$(SP)StartTime=\${XXX}\$(HT)EndTime=\${XXX}

**The client uploads the data that meets the condition:**

First, the device uploads the summary:

URL:

POST /iclock/cdata?SN=\$(SerialNumber)&cmdid=\${CmdID}&desc=overview HTTP/1.1

Content:

SumCount=\${XXX}

FileName=\${XXX}

ContentType=\${XXX}

FileCount=\${XXX}

**The server returns:**

OK

Then, the device uploads the access control records as files:

URL:

POST

/iclock/file?SN=\${SerialNumber}&cmdid=\${CmdID}&fileseq=\${XXX}&contenttype=tgz&table=transaction  
&count=\${XXX} HTTP/1.1

Content:

The content of transaction.tgz.

The following describes how transaction.tgz is generated:

a. Convert the count number of access control records into those in the push upload format and store them in transaction.txt, for example:

cardno=0	pin=0	verified=200	doorid=1	eventtype=100	inoutstate=2
time_second=548003687	index=92				
cardno=0	pin=0	verified=200	doorid=1	eventtype=100	inoutstate=2
time_second=548003688	index=93				
cardno=0	pin=0	verified=200	doorid=1	eventtype=100	inoutstate=2
time_second=548003689	index=94				

B. Convert transaction.txt to transaction.tgz.

**The server returns:**

OK

**The client uploads the execution result:**

ID=\${CmdID}&amp;Return=\${XXX}&amp;CMD=ACCOUNT

Note:

transaction: The table name, which is the access control record

MaxIndex: Access control records of which the ID is greater than this value need to be uploaded. When the value is 0, all access control records need to be uploaded.

StartIndex: Need to upload access control records of which the ID is greater than or equal to this value.

EndIndex: Need to upload access control records of which the ID is smaller than or equal to this value.

StartTime: Need to upload access control records of which the record time is greater than or equal to this value. The time format is XXXX-XX-XX XX:XX:XX, for example, 2018-02-22 16:19:20.



EndTime: Need to upload access control records of which the record time is smaller than or equal to this value. The time format is XXXX-XX-XX XX:XX:XX, for example, 2018-02-22 16:19:20.

SumCount: The total number of access control records

FileName: The file name

ContentType: The text format. tgz means a compressed package.

FileCount: The number of access control record sub-packages that meet the condition

fileseq: The index of the file sub-packages to upload, which starts from 1

## 12.3 Test Host

Application scenario:

The software delivers a network test command. After receiving the command, the device test the connectivity based on the IP address and port provided and returns the test result.

Format:

**The server delivers the command:**

C:295:Test Host Address=110.80.38.74:8092

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=Test(SP)Host

Note:

Address: The IP address and port of the test server.

## 12.4 Control Devive

Application scenario:

The control commands are used to control device startup, remote door opening, remote door closing, alarm canceling, enabling or disabling of Normal Open, and auxiliary outputs.

Format:

**The server delivers the command:**

C:\${CmdID}:CONTROL\$(SP)DEVICE\$(SP)AABBCCDDEE

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=CONTROL\$(SP)DEVICE

**Note:**

AA, BB, CC, and DD are four groups of two-byte strings. Each group of strings is the converted result of a one-byte integer after %02X conversion. Wherein, AA indicates control commands, which are described as follows:

| AA | BB | CC | DD | EE |

|-----|

| 01: Control the output | 00: Open all the doors. 01-10: The door ID | 01: Control the lock. 02: Control the auxiliary output. | 00: Off. FF: Normal open. 01-FF: The opening duration. | Operator |

| 02: Cancel the alarm. | 00: Cancel the alarms. 01-10: The door ID | | | Operator |

| 03: Restart the device. | 00: Restart the current device. 01-10: The slave device ID. | | | Operator |

| 04: Control Normal Open. | 00: All the doors. 01-10: The door ID | 00: Disable. 01: Enable. | | Operator |

| 05: Deliver the administrator to the door. | 00: All the doors. 01-10: The door ID | | | Operator |

| 06: Lock/unlock the door. | 00: Open all the doors. 01-10: The door ID | 00: Unlock. 01: Lock. | | Operator |

| 07: Control emergency dual-on or dual-off. | 01: Emergency dual-on. 02: Emergency dual-off. | | | Operator |

| 08: The Zigbee command | 01: Re-networking. 02: Stop re-networking. 03: Allow access. 04: Stop access. | | | Operator |

| 09: The Wiegand test command | 01: Start the Wiegand test. 02: Stop the Wiegand test. | | | Operator |

| 10 485: The lighting command | | | | Operator |

| 11: The network switchover command | 01: Switch from wired network to the 4G network. 02: Switch from the wired network to Wi-Fi. 03: Switch from 4G network to wired network. 04: Switch from Wi-Fi to wired network. | | | Operator |

| 12: The command of registering by identity card | 01: Enable the mode of registering by identity card. 00: Disable the mode of registering by identity card. | 00: Open all the doors. 01-10: The door ID | Timeout time | Operator |

| 13: The command of querying the sub-device connection status | | | | Operator |

Example:

**(1) The server delivers the command of opening Door 1 for 5s:**

C:221:CONTROL DEVICE 01010105

**(2) The server delivers the command of canceling the alarm of Door 1:**

C:222:CONTROL DEVICE 02010000

**(3) The server delivers the command of restarting the current device:**

C:223:CONTROL DEVICE 03000000

**(4) The server delivers the Wiegand test command:**

C:224:CONTROL DEVICE 0900000000

**After receiving the command, the client uploads the request protocol:**

POST /iclock/cdata?SN=\${SerialNumber}&table=testdata HTTP/1.1

Cookie: token=\${XXX}

Host: \${ServerIP}:\${ServerPort}

Content-Length: \${XXX}

...

type=wg\tbitscount=%?\tbits=%?\tsn=%?

**The server returns:**

OK

**(5) The server delivers the command of querying the sub-device connection status:**

C:225:CONTROL DEVICE 0D000000

**(6) The command for switching between networks:**

From wired network to 4G network: C:226:CONTROL DEVICE 0B010000

From wired network to Wi-Fi:

C:227:SET OPTIONS TempWirelessSSID=11,TempWirelessKey=11 (TempWirelessSSID: Wi-Fi SSID, the Wi-Fi key)

C:228:CONTROL DEVICE 0B020000

From 4G network to wired network: C:229:CONTROL DEVICE 0B030000

From Wi-Fi to wired network: C:230:CONTROL DEVICE 0B040000

**(7)The client uploads the successful execution result:**

ID=224&Return=0&CMD=CONTROL DEVICE

## 12.4.1 Upgrade

**Remote Firmware Upgrade**

Application scenario:

It is used to remotely upgrade the firmware of the client device from the server software.

**Method 1:**

Application scenario:

To remotely upgrade the client firmware, the client needs to be compatible with the controller and the new-architecture PULL SDK Device. The files to upgrade must be converted by the server to the specified format and then delivered to the client.

Format:

**The server delivers the command:**

C:\${CmdID};UPGRADE\$(SP)checksum=\${XXX},url=\${URL},size=\${XXX}

**The client downloads the upgrade package from the URL carried in the command:**

GET /iclock/file?SN=\$(SerialNumber)&url=\${URL} HTTP/1.1

...

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=UPGRADE

Note:

Checksum: The MD5 checksum

url: The address to download the upgrade file named emfw.cfg

size: The size of the original file

Note:

In this method, the firmware upgrade file is base64-encoded by the server before delivery. After receiving the file, the client needs to convert it into a binary file and name it as emfw.cfg.

Example:

**The server delivers the firmware upgrade command:**

C:384:UPGRADE

checksum=a5bf4dcd6020f408589224274aab132d,url=http\*//localhost\*8088\fireware\F20\admin\emfw.cfg,size=2312

**The client sends a request to download the upgrade package:**

GET /iclock/file?SN=3383154200002&url=http://192.168.213.17:8088/fireware/F20/admin/emfw.cfg  
HTTP/1.1

Cookie: token=af65a75608cf5b80fbb3b48f0b4df95a

Host: 192.168.213.17:8088

...

**The client uploads the successful execution result:**

ID=384&Return=0&CMD=UPGRADE

**Method 2:**

Application scenario:

The client directly obtains the upgrade file to upgrade the firmware remotely, without converting the file format.

Format:

**The server delivers the command:**

C:\${CmdID};UPGRADE\$(SP)type=1,checksum=\${XXX},size=\${XXX},url=\${URL}

**The client sends a request to download the upgrade package:**

GET /iclock/file?SN=\$(SerialNumber)&url=\${URL} HTTP/1.1

Cookie: token=af65a75608cf5b80fbb3b48f0b4df95a

Host: 192.168.213.17:8088

...

**The client uploads the execution result:**

ID=\${CmdID}&Return=\${XXX}&CMD=UPGRADE

Note:

type: 1 means obtaining the upgrade file from the URL. Currently, only 1 is supported.

Checksum: The MD5 checksum

url: The address to download the upgrade file named emfw.cfg

size: The size of the upgrade package

Note:

In this method, the client directly obtains the firmware upgrade file, without the need of converting its format.

Example:

**The server delivers the command:**

C:123:UPGRADE

type=1,checksum=oqoier9883kjankdefi894eu,size=6558,url=http://192.168.0.13:89/data/emfw.cfg

**The client sends a request to download the upgrade package:**

GET /iclock/file?SN=3383154200002&url=http://192.168.0.13:89/data/emfw.cfg HTTP/1.1

Cookie: token=af65a75608cf5b80fbb3b48f0b4df95a

Host: 192.168.0.13:89

...

**The client uploads the successful execution result:**

ID=384&Return=0&CMD=UPGRADE

**Method 3:**

Application scenario:

Remotely upgrade the client's firmware, and obtain files in a subcontracted pull mode, without format conversion, and the client directly obtains the files. The subcontracting pull process refers to the Range protocol of HTTP, and the process is as follows:

- 1) The device side specifies RANGE in the HTTP Header: xx-xx specifies the byte range of the upgrade package to be pulled.
- 2) The software side parses the RANGE specified in the HTTP Header, and returns the upgrade package data in the specified range to the device side.
- 3) The device side pulls 1M of data each time by default, and the maximum one-time use of 1M memory, which solves the problem of insufficient memory caused by the device pulling large data packets at one time.

Format:

**The server delivers the command:**

C:{\$CmdID};UPGRADE\$(SP)checksum={\$XXX},size={\$XXX},url=\$(URL),supportsubcontracting={\$XXX}

**The client sends a request to download the upgrade package:**

GET \$(URL) HTTP/1.1

Cookie: token=af65a75608cf5b80fbb3b48f0b4df95a

Host: 192.168.213.17:8088

.....

**The client uploads the execution result:**

ID={\$CmdID}&Return={\$XXX}&CMD=UPGRADE

Note:

Checksum: The MD5 checksum

url: Represents the download resource address of the upgrade file, which can support both absolute path and relative path (if it is a relative path, the device will automatically splice into an absolute path according to the address of the currently connected cloud server)

size: The size of the upgrade package

supportsubcontracting: Represents whether the software supports the upgrade package subcontracting protocol (0: not supported, 1: supported)

Note:

In this method, the client directly obtains the firmware upgrade file, without the need of converting its format.

Example:

**The server delivers the command:**

C:123:UPGRADE

checksum=oqoier9883kjankdefi894eu,size=6558,url=http://192.168.0.13:89/data/emfw.cfg

**The client sends a request to download the upgrade package:**

GET http://192.168.0.13:89/data/emfw.cfg

Cookie: token=af65a75608cf5b80fbb3b48f0b4df95a

Host: 192.168.0.13:89

.....

**The client uploads the successful execution result:**

ID=123&Return=0&CMD=UPGRADE

## 12.5 Configuration Class

These commands are used to configure the client or obtain the client configurations.

### 12.5.1 Set Options

Application scenario:

It is used to set the client configuration parameters, such as the IP address, gateway, and lock driver time length.

Format:

**The server delivers the command:**

C:\$(CmdID):SET\$(SP)OPTIONS\$(SP)parameter-value list

**The client uploads the execution result:**

ID=\$(CmdID)&Return=0&CMD=SET OPTIONS

Note:

The parameter list is in the form of parameter name=parameter value. Multiple parameters are separated by commas, but the last parameter is not followed by a comma.

Example:

**Modify the IP address, gateway, and mask**

C:403:SET OPTIONS

IPAddress=192.168.213.221,GATEIPAddress=192.168.213.1,NetMask=255.255.255.0

**Set the server IP address and port for the device**

C:399:SET OPTIONS WebServerIP=192.168.213.221,WebServerPort=8088

**Synchronize the time to the client:**

C:401:SET OPTIONS DateTime=583080894

Note:

DateTime: The value means the current time of the server and is in seconds converted by using the method in Appendix 5.

Some devices stop time synchronization after receiving this command, while some devices immediately trigger the following request after executing this command. Compatibility should be considered.

Request:

GET /iclock/rtdata?SN=3383154200002&type=time HTTP/1.1

Cookie: token=af65a75608cf5b80fbb3b48f0b4df95a

Host: 192.168.213.17:8088

...

Response:

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Content-Type: text/plain;charset=UTF-8

Content-Length: 33

Date: Wed, 11 Jan 2017 01:30:03 GMT

DateTime=583050990,ServerTZ=+0800

Note:

DateTime: The value means a Greenwich mean time and is in seconds converted by using the method in Appendix 5.

ServerTZ: The time zone of the server

**Modify the communication password**

C:402:SET OPTIONS ComPwd=1

**Set access control parameters**

C:405:SET OPTIONS

Door1Drivertime=5,Door1KeepOpenTimeZone=0,Door1ValidTZ=1,Door1SupperPassWord=,Door1Interti



me=0,Door1VerifyType=0,Door1SensorType=1,Door1Detectortime=15,Door1CloseAndLock=0,Door1ForcePassWord=,Reader1IOState=0,WiegandIDIn=1,WiegandID=1

Door1Drivertime: Lock driver time length of Door 1

Door1KeepOpenTimeZone: Normal Open time period of Door 1

Door1ValidTZ: Activation validity period of Door 1

Door1SupperPassWord: Supper password

Door1Intertime: Punch interval (not applicable to the new-architecture device)

Door1VerifyType: Verification mode of Door 1

Door1SensorType: Door sensor type of Door 1. 0: none. 1: Normal open. 2: Normal close.

Door1Detectortime: Door sensor delay time of Door 1

Door1CloseAndLock: Support lock at door closing or not

Door1ForcePassWord: Duress password

Reader1IOState: In/out status of Reader 1

WiegandIDIn:

WiegandID:

### Access control new verification parameters

If the device supports the new verification method, the verification methods of Door1VerifyType and DoorVerifyType will not follow the rules described in Appendix 3, but the following new rules:

Supported verification methods, a total of 16 digits, currently only supports up to 7 digits, of which the 0th digit is the logical digit and is issued as a string.

```
typedef enum _VERIFY_TYPE_E
```

```
{
```

```
VF_TYPE_LOGIC = 0, //0: logical digit, 0: or; 1: and
```

```
VF_TYPE_FACE = 1, //1: face
```

```
VF_TYPE_PALM_PRINT = 2, //2: palmprint
```

```
VF_TYPE_PALM_VEIN = 3, //3: palm vein
```

```
VF_TYPE_FP = 4, //4: fingerprint
```

```
VF_TYPE_VEIN = 5, //5: finger vein
```

```
VF_TYPE_VP = 6, //6: vocal print
```

```
VF_TYPE_IRIS = 7, //7: iris
```

```
VF_TYPE_RETINA = 8, //8: retina
```

VF\_TYPE\_PW = 9, //9: password

VF\_TYPE\_PIN = 10, //10: User ID

VF\_TYPE\_RF = 11, //11: card

}VERIFY\_TYPE\_E;

Such as: card & password & face: 0000101000000011

Card | password | face: 0000101000000010

### QR code encryption parameters

C:402:SET OPTIONS QRCodeDecryptType=XXX,QRCodeDecryptKey=XXX

QRCodeDecryptType: QR code decryption method, currently supports three methods, value 0, 1, 2.

1: Use scheme one, use today's date as the key and use AES256 algorithm to encrypt (the key is fixed)

2: Use scheme two, the system randomly generates a key and uses AES256 algorithm for encryption (the key is not fixed)

3: Use scheme three, the system randomly generates a key and uses RSA1024 algorithm for encryption (public and private keys are not fixed)

QRCodeDecryptKey: QR code key

### The client uploads the successful execution result:

ID=405&Return=0&CMD=SET OPTIONS

## 12.5.2 Get Options

Application scenario:

It is used to obtain the device configurations.

Format:

### The server delivers the command:

C:\$(CmdID):GET\$(SP)OPTIONS\$(SP)parameter name list

### The client uploads the parameters to get:

POST

/iclock/querydata?SN=\$(SerialNumber)&type=options&cmdid=\$(CmdID)&tablename=options&count=1  
&packcnt=1&packidx=1 HTTP/1.1

...

Parameter-value list

**The client uploads the successful execution result:**

ID=\$(CmdID)&Return=0&CMD=GET OPTIONS

**Note:**

/iclock/querydata: The URI of the data that is uploaded by the client and meets the condition

type: "options" means the type of the data to be uploaded by the client is the parameter.

tablename. The table name. "options" means a parameter configuration table.

count: The number of data entries returned.

packcnt: The total number of packages. When there is a great amount of data, the data needs to be divided into different packages. This value indicates how many packages the data is divided into.

packidx: The package ID, which indicates which of all packages is currently being sent.

Multiple parameters are separated by commas, but the last parameter is not followed by a comma.

**Example:**

C:408:GET OPTIONS

~SerialNumber,FirmVer,~DeviceName,LockCount,ReaderCount,AuxInCount,AuxOutCount,MachineType,  
~IsOnlyRFMachine,~MaxUserCount,~MaxAttLogCount,~MaxFingerCount,~MaxUserFingerCount,MThres  
hold,NetMask,GATEIPAddress,~ZKFPVersion,SimpleEventType,VerifyStyles,EventTypes,ComPwd

~SerialNumber: The SN

FirmVer: The version of the firmware

DeviceName: The name of the device

LockCount: The number of doors

ReaderCount: The number of readers

AuxInCount: The number of auxiliary inputs

AuxOutCount: The number of auxiliary outputs

MachineType: The machine model. PULL SDK Device 101

~IsOnlyRFMachine: Specify whether only the enable/disable parameter of the RF card is supported.

~MaxUserCount: The maximum number of users or cards

~MaxAttLogCount: The maximum number of attendance records

~MaxUserFingerCount: The maximum number of fingers of a single user

MThreshold: The N match threshold

IPAddress: The IP address of the wired network

NetMask: The subnet mask of the wired network

GATEIPAddress: The gateway address of the wired network

~ZKFPVersion: The version of the finger algorithm

## SimpleEventType: Event merging

**VerifyStyles:** The supported verification mode. It contains 32 bits in total. Each of the first 16 bits corresponds to a different combination of verification modes. For details, see Appendix 3 Description of Verification Mode Code. 1 indicates that the verification mode is supported, and 0 indicates that the verification is not supported. 200 indicates an access control event.

**EventTypes:** The supported event type, which contains 32 bytes (0 to 31) in total. Each byte contains 8 bits, represented by 2 hexadecimal numbers, a total of 256 bits (0 to 255). For the function represented by each bit, see Appendix 2 Description of Real-time Events. 1 indicates that the event is supported and 0 indicates that the event is not supported.

Take BF0FE03D3000010000000000700000000000000000000000000077002001000000 as an example.  
The 0th byte is BF, which is 11111101 in binary mode (the lower bit is placed first), the 6th bit is 0 and other bits are 1. It means that in the function controlled by the first eight bits, only the linkage event represented by the 6th bit is not supported, and other events are supported.

ComPwd: The communication password

## C:409:GET OPTIONS

IclockSvrFun,OverallAntiFunOn,~REXInputFunOn,~CardFormatFunOn,~SupAuthrizeFunOn,~ReaderCFG  
 unOn,~ReaderLinkageFunOn,~RelayStateFunOn,~Ext485ReaderFunOn,~TimeAPBFFunOn,~CtlAllRelayFun  
 On,~LossCardFunOn,DisableUserFunOn,DeleteAndFunOn,LogIDFunOn,DateFmtFunOn,DelAllLossCardFu  
 nOn,DelayOpenDoorFunOn,UserOpenDoorDelayFunOn,MultiCardInterTimeFunOn,DSTFunOn,OutRelayS  
 etFunOn,MachineTZFunOn,AutoServerFunOn,PC485AsInbio485,MasterInbio485,RS232BaudRate,AutoSer  
 verMode,IPCLinkFunOn,IPCLinkServerIP

lclockSvrFun: Whether to enable ADMS. 1: Enable. 0: Disable.

OverallAntiFunOn: Specify whether to enable the regional APB function. It is not used.

~REXInputFunOn: Specify whether to enable the exit door locking function

~CardFormatFunOn: Specify whether the Wiegand reader supports the user-defined card format.

~SupAuthrizeFunOn: Specify whether to enable the super user function.

~ReaderCFGFunOn: Specify whether to enable the reader configuration function. It is not used.

~ReaderLinkageFunOn: Specify whether to enable the reader linkage function.

~RelayStateFunOn: Specify whether to support the relay status function.

~Ext485ReaderFunOn: Specify whether to enable the external reader 485 function.

~TimeAPBFunOn: Specify whether to enable the APB time function.

~CtlAllRelayFunOn: Specify whether to enable the All Relay function.

~LossCardFunOn: Specify whether to enable the function of deleting all blacklists.

DisableUserFunOn: Specify whether to enable the blacklist function.

DeleteAndFunOn: Specify whether to enable the AND deletion function.

LogIDFunOn: Specify whether to enable the ID logging function.

DateFmtFunOn: Specify whether to enable the date format function.

DelAllLossCardFunOn: Specify whether to enable the function of deleting all blacklists.

DelayOpenDoorFunOn: Specify whether to enable the delay open door function. This function is not supported.

UserOpenDoorDelayFunOn: Specify whether to enable the function of delaying the door open time for a user. This function is not supported.

MultiCardInterTimeFunOn: Specify whether to enable the function of setting the interval for multiple cards. This function is not supported.

DSTFunOn: Specify whether to enable the DLST function. This function is not supported.

OutRelaySetFunOn: Specify whether to enable the configuration output function. This function is not supported.

MachineTZFunOn: Specify whether to enable the function of setting a time zone for the machine. This function is not supported.

AutoServerFunOn: Specify whether to enable the remote identification function.

PC485AsInbio485: 485 reader/communication switchover.

MasterInbio485: 485 reader/communication switchover.

RS232BaudRate: RS232/485 baud rate.

AutoServerMode: Remote identification mode.

IPCLinkFunOn: Specify whether to enable the soft linkage with IPC. This function is not supported.

IPCLinkServerIP: The IPC address. This function is not supported.

C:410:GET OPTIONS FingerFunOn,FvFunOn,FaceFunOn,~MaxFace7Count,~MaxFvCount

FingerFunOn: Specify whether to enable the finger function.

FvFunOn: Specify whether to enable the finger vein function.

FaceFunOn: Specify whether to enable the face function.

MaxFace7Count: The maximum number of face templates.

MaxFvCount: The maximum number of finger veins.

## 12.6 Remote Registration

Application scenario:

Initiated by the server, the bio template is registered on the client.

Format:

### The server delivers the command:

C:\$(CmdID):ENROLL\_BIO type=%?\tno=%?\tpin=%?\tcardno=%?\tretry=%?\toverwrite=%?

### The client uploads the execution result:

ID=\${XXX}&Return=\${XXX}&CMD=ENROLL\_BIO

Note:

type: Biological template type

0: General template

1: Fingerprint

2: Face

3: Voice

4: Iris

5: Retina

6: Palm vein

7: Finger vein

8: Palmprint

9: Visible light face

no: Biological template specific individual number

[Fingerprint] The number is 0-9.

The corresponding fingers are:

Left hand: little finger/ring finger/middle finger/index finger/thumb

Right hand: thumb/index finger/middle finger/ring finger/little thumb

[Finger vein] Same as fingerprint

[Face & Visible Light Face] The number is 0

[Iris] The number is: 0 for the left eye, 1 for the right eye

[Palm vein] The number is: 0 for the left hand, 1 for the right hand, if not clearly defined, the default value is 0

pin: registered User ID

cardno: registered card number

retry: If the registration fails, the number of times to retry

overwrite: whether to overwrite

0 means that if the corresponding user exists, it will not be overwritten and return error

1 means that the corresponding user exists and covers

Example:

The software issues a visible light face with a registered User ID of 1, and retry at most 3 times during registration. If the user has a face, it will cover:

C:395:ENROLL\_BIO type=9 no=0 pin=1 retry=3 overwrite=1

**The client uploads the successful execution result:**

ID=395&return=0&CMD=ENROLL\_BIO

## 13 Remote Identification

The device sends the data that passes identification to the server. After identification, the server returns the result, the data that passes identification, and the control command, as shown below:

**Client request message:**

POST /iclock/cdata?SN=\${SerialNumber}&AuthType=device HTTP/1.1

Host: \${ServerIP}:\${ServerPort}

Content-Length: \${XXX}

...

time=\${XXX}\${HT}pin=\${XXX}\${HT}cardno=\${XXX}\${HT}addrtype=\${XXX}\${HT}eventaddr=\${XXX}\${HT}event=\${XXX}\${HT}inoutstatus=\${XXX}\${HT}verifytype=\${XXX}

Note:

HTTP request method: POST

URI: /iclock/cdata

HTTP version: 1.1

Client configurations:

SN: \${Required} is the client SN.

AuthType=device means globally remote APB.

Host header: \${Required}

Content-Length header: \${Required}

Other headers: \${Optional}

Request entity: The event record

time: The time, in the format of YYYY-MM-DD HH:MM:SS

pin: The user ID. 0 is used when it is left blank.

cardno: The card number

addrtype: The event point type, that is, by whom the event is generated

Value	Meaning
1	Device
2	Door
3	Reader
4	Auxiliary input
5	Auxiliary output

eventaddr: The event address. If the event is generated by the device, this value can be the device address. If the event is generated by the reader, this value can be the reader address. Currently, the door is the only option.

event: The event type

inoutstatus: The in or out the status, that is, entering or exiting the door.

verifytype: The verification type

### The normal server response is as follows:

When the user information is available, the response is as follows:

HTTP/1.1 200 OK

Date: \${XXX}

Content-Length: \${XXX}

...



```
AUTH=${XXX}${CR}${LF}time=${XXX}${HT}pin=${XXX}${HT}cardno=${XXX}${HT}addrtype=${XXX}${HT}eventaddr=${XXX}${HT}event=${XXX}${HT}inoutstatus=${XXX}${HT}verifytype=${XXX}${CR}${LF}CONTROL$(SP)DEVICE$(SP)AABBCCDDEE
```

Note:

The first line

AUTH: Whether the verification is successful or not

Value	Meaning
SUCCESS	The verification is successful.
FAILED	The verification failed.
TIMEOUT	The verification times out.

The second line

The requested original event record

The third line

The controller command after successful verification. For details, see CONTROL DEVICE.

## Appendixes

### Appendix 1 Description of Returned Values of Commands

Returned value	Description
---	---
>=0	Successful
-1	Failed to send the command
-2	No response to the command
-3	Insufficient cache
-4	Failed to decompress
-5	Incorrect length of read data
-6	The length after decompression does not meet the required one
-7	Repeated command
-8	Unauthorized connection
-9	Incorrect data. CRC failed
-10	Incorrect data. Failed to resolve dataapi
-11	Incorrect data parameter
-12	Command execution error
-13	Incorrect command. The command does not exist
-14	Incorrect communication password
-15	Failed to write the file
-16	Failed to read the file
-17	The file does not exist
-18	The device has insufficient space
-19	Checksum error
-20	The length of the received data is inconsistent with the specified data length
-21	No platform parameter is set on the device
-22	The received firmware platform for upgrade is not same to the local platform
-23	The firmware version for upgrade is earlier than that in the device
-24	Incorrect flag of the upgrade file
-25	The name of the received file for firmware upgrade is not emfw.cfg
-26	The length of the received fingerprint template is 0
-27	The PIN of the received fingerprint template is incorrect. Failed to find the user
-28	Door opening command is executed during the open time period
[-400,-100]	is the pullsdk internal returned value
-100	The table structure does not exist

-101	The conditional field in the table structure does not exist
-102	The total field count is inconsistent
-103	The field sequence is inconsistent
-104	Incorrect real-time event data
-105	Incorrect data for resolution
-106	The size of the delivered data exceeds 4 MB
-107	Failed to get the table structure
-108	Invalid OPTIONS
-201	LoadLibrary failed
-202	Failed to call the interface
-203	Communication initialization failed
-206	Failed to start the serial port agent program. A general cause is that the serial port does not exist or has been occupied
-301	Failed to get the TCP/IP version
[\*,-600] is a returned value exclusive to push communication [Returned value already mentioned above cannot be defined repeatedly]	
-601	Reserved. For firmware internal use only
-603	Internal error: invalid handle
-604	Insufficient memory
-609	Internal error: invalid handle
-612	The device failed to save the parameter
-614	Failed to remotely cancel the alarm
-615	Remote restart failed
-616	Remotely enabling or canceling normal open failed
-618	Incorrect URL format
-619	Firmware internal error: blank input parameter
-620	The format of the data delivered by the server is incorrect
-621	The table structure does not support this field
-628	Main failed to load the fingerprint to the memory
-629	Incorrect table name
-630	Failed to execute the shell command
-631	Incorrect base64 length
-632	Incorrect file name
-701	Reserved. For firmware internal use only
-702	Reserved. For firmware internal use only
-703	Reserved. For firmware internal use only
-704	Reserved. For firmware internal use only
-705	Reserved. For firmware internal use only

-706	Reserved. For firmware internal use only
-707	Reserved. For firmware internal use only
-708	The firmware does not support this command
-709	Failed to upload options
-720	Failed to upgrade the firmware
-721	The device downloads the file
-722	Failed to upload the data to the software
-723	Reserved. For firmware internal use only
-724	Reserved. For firmware internal use only
-725	It times out when the device waits for the result
-726	The server returns an HTTP error
-727	Reserved. For firmware internal use only
-728	Failed to upload the number of table data entries
-729	Reserved. For firmware internal use only
-730	No response from the server
-731	An error occurs when the server receives the data
-732	Reserved. For firmware internal use only
-801	Incorrect command format. The correct format is cmd:SN:command

## Appendix 2 Description of Real-time Events

Event code: 0-19 and 200-253 means a normal event. 20-99 means error events. 100-199 means alarm events.

The red font is the event code transplanted from the BEST protocol, which is currently only used on controller devices that support the BEST protocol.

Event code	Original description	Remarks
---	---	---
0	The door opens normally after punch	The door opens normally after verification  The event codes 0, 14, and 17 are merged and have the same meaning
1	Punch in the normal open time period	Verify in the normal open time period  The event codes 1 and 16 are merged and have the same meaning
2	The first user opens the door by punch	The first user opens the door  The event codes 2, 18, and 19 are merged and have the same meaning
3	Multiple users open the door by punch	Multiple users open the door  The event codes 3, 15, and 203 are merged and have the same meaning
4	Open the door by using the emergency password	Open the door by using the emergency password
5	Open the door in the normal open time period	Open the door in the normal open time period
6	Trigger linkage	Trigger linkage  The value of the verification mode in the linkage event record is the specific event type

| 7 | Cancel the alarm |Cancel the alarm||

| 8 | Open the door remotely |Open the door remotely ||

| 9 | Close the door remotely |Close the door remotely ||

| 10 | Disable the normal open time period of the day |Disable the normal open time period of the day ||

| 11 | Enable the normal open time period of the day |Enable the normal open time period of the day ||

| 12 | Enable auxiliary output remotely |Enable auxiliary output remotely ||

| 13 | Disable auxiliary output remotely |Disable auxiliary output remotely ||

| 14 | The door is normally opened by fingerprint |he door is normally opened after verification |The event codes 0, 14, and 17 are merged and have the same meaning|

| 15 | Multiple users open the door by fingerprint |Multiple users open the door |The event codes 3, 15, and 203 are merged and have the same meaning|

| 16 | Check the fingerprint in the normal open time period |Perform verification in the normal open time period |The event codes 1 and 16 are merged and have the same meaning|

| 17 | Open the door by card and fingerprint |Open the door by verification |The event codes 0, 14, and 17 are merged and have the same meaning|

| 18 | The first user opens the door by fingerprint |The first user opens the door |The event codes 2, 18, and 19 are merged and have the same meaning|

| 19 | The first user opens the door by card and fingerprint |The first user opens the door |The event codes 2, 18, and 19 are merged and have the same meaning|

| 20 | The punch interval is too short |The operation interval is too short |The event codes 20, 31, and 50 are merged and have the same meaning|

| 21 | Punch during the non-valid time period |Open the door by verification during the non-valid time period |The event codes 21, 35, and 49 are merged and have the same meaning|

| 22 | Invalid time period |Invalid time period ||

| 23 | Unauthorized access |Unauthorized access ||

| 24 | APB |APB ||

| 25 | Interlock |Interlock ||

| 26 | Multiple users perform verification by punch |Multiple users wait for verification |The event codes 26, 32, and 51 are merged and have the same meaning|

| 27 | The card is not registered |The user is not registered |The event codes 27, 30, and 34 are merged and have the same meaning|

| 28 | Door opening times out |Door opening times out ||

| 29 | The card privilege expires |The user privilege expires |The event codes 29, 33, and 53 are merged and have the same meaning|

| 30 | The password is incorrect |The user is not registered |The event codes 27, 30, and 34 are merged and have the same meaning|

| 31 | The fingerprint check interval is too short |The operation interval is too short |The event codes 20, 31, and 50 are merged and have the same meaning|

| 32 | Multiple users perform verification by fingerprint |Multiple users wait for verification |The event

codes 26, 32, and 51 are merged and have the same meaning|

| 33 | The fingerprint expires |The user privilege expires |The event codes 29, 33, and 53 are merged and have the same meaning|

| 34 | The fingerprint is not registered |The user is not registered |The event codes 27, 30, and 34 are merged and have the same meaning|

| 35 | Check the fingerprint during the non-valid time period |Open the door by verification during the non-valid time period |The event codes 21, 35, and 49 are merged and have the same meaning|

| 36 | Press the exit button during the non-valid time period |Press the exit button during the non-valid time period ||

| 37 | The door cannot be closed during the normal open time period | The door cannot be closed during the normal open time period ||

| 38 | The card has been reported as lost |The card has been reported as lost ||

| 39 | Blacklist |Blacklist ||

| 40 | Multi-user verification by fingerprint failed |Multi-user verification failed |The event codes 40, 48, and 52 are merged and have the same meaning|

| 41 | Incorrect verification mode |Incorrect verification mode ||

| 42 | Incorrect Wiegand format |Incorrect Wiegand format ||

||||

| 44 | Remote identification failed |Remote identification failed ||

| 45 | Remote identification times out |Remote identification times out ||

||||

| 47 | Failed to send the command |Failed to send the command||

| 48 | Multi-user verification by punch failed |Multi-user verification failed |The event codes 40, 48, and 52 are merged and have the same meaning|

| 49 | Open the door by password during the non-valid time period |Open the door by verification during the non-valid time period |The event codes 21, 35, and 49 are merged and have the same meaning|

| 50 | The password verification interval is too short |The operation interval is too short |The event codes 20, 31, and 50 are merged and have the same meaning|

| 51 | Multiple users perform verification by password |Multiple users wait for verification |The event codes 26, 32, and 51 are merged and have the same meaning|

| 52 | Multi-user verification by password failed |Multi-user verification failed |The event codes 40, 48, and 52 are merged and have the same meaning|

| 53 | The password expires |The user privilege expires |The event codes 29, 33, and 53 are merged and have the same meaning|

| 54 | Battery voltage is too low |Battery voltage is too low||

| 55 | Replace the battery immediately |Replace the battery immediately||

| 56 | Unauthorized operation |Unauthorized operation||

| 57 | Backup power supply |Backup power supply||

| 58 | Normal open alarm |Normal open alarm||

59	Unauthorized admin	Unauthorized admin	
60	The door is locked inside	The door is locked inside	
61	Repeat verification	Repeat verification	
62	Prohibited user	Prohibited user	
63	Door locked	Door locked	
64	The exit button is not operated within the time period	The exit button is not operated within the time period	
65	Auxiliary input is not operated within the time period	Auxiliary input is not operated within the time period	
66	Reader upgrade failed	Reader upgrade failed	
67	Remote comparison is successful (device is not authorized)	Remote comparison is successful (device is not authorized)	
68	High boggy temperature -Access Denied	High boggy temperature -Access Denied	
69	Without mask -Access Denied	Without mask -Access Denied	
70	The face comparison server communication is abnormal	The face comparison server communication is abnormal	
71	Face server responds abnormally	Face server responds abnormally	
72	Call unanswered	Call unanswered	
73	Invalid QR code	Invalid QR code	
74	QR code has expired	QR code has expired	
75	Combined verification timeout	Combined verification timeout	
76	Cannot get fingerprints in TITAN mode	Cannot get fingerprints in TITAN mode	
77	Internet cable not plugged in	Internet cable not plugged in	
78	Device hotspot disconnected abnormally	Device hotspot disconnected abnormally	
79	Mobile network disconnected abnormally	Mobile network disconnected abnormally	
100	Tamper alarm	Tamper alarm	
101	Open the door with the duress password	Alarm for opening the door with the duress password	
The event codes 101 and 103 are merged and have the same meaning			
102	The door is opened accidentally	The door is opened accidentally	
103	Open the door with the duress fingerprint	Alarm for opening the door with the duress fingerprint	
The event codes 101 and 103 are merged and have the same meaning			
104	Alarm for punch with an invalid card	Alarm for punch with an invalid card	
105	Network disconnected	Failed to connect to the server	
106	Battery power failure	Battery power failure	
107	Mains power failure	Mains power failure	
108	Failed to connect to the main controller	Failed to connect to the main controller	
109	Reader disassembly alarm	Reader disassembly alarm	
110	Reader offline alarm	Read head offline alarm	

111	POE power down	POE power down	
112	Extension board offline alarm	Extension board offline alarm	
113	Illegal security level	Illegal security level	
114	Line detection, fire alarm input disconnected	Line detection, fire alarm input disconnected	
115	Line detection, fire alarm input short circuit	Line detection, fire alarm input short circuit	
116	Line detection, auxiliary input disconnect	Line detection, auxiliary input disconnect	
117	Line detection, auxiliary input short circuit	Line detection, auxiliary input short circuit	
118	Line detection, the exit button is off	Line detection, the exit button is off	
119	Line detection, short circuit of exit button	Line detection, short circuit of exit button	
120	Line detection, door sensor disconnected	Line detection, door sensor disconnected	
121	Line detection, door sensor short circuit	Line detection, door sensor short circuit	
200	The door is open	The door is open	
201	The door is closed	The door is closed	
202	Open the door by pressing the exit button	Open the door by pressing the exit button	
203	Multiple users open the door by card and fingerprint	Multiple users open the door	The event codes 3, 15, and 203 are merged and have the same meaning
204	The door normal open time period ends	The door normal open time period ends	
205	Remotely enable normal open for the door	Remotely enable normal open for the door	
206	Start the device	Start the device	
207	Open the door by password	Open the door by password	
208	The super user opens the door	The super user opens the door	
209	Press the exit button (locked)	Press the exit button (locked)	
210	Start firefighting door opening	Start firefighting door opening	
211	The super user closes the door	The super user closes the door	
212	Enable the elevator control function	Enable the elevator control function	
213	Disable the elevator control function	Disable the elevator control function	
214	Successfully connected to the server	Successfully connected to the server	
215	The first card opens the door by password	The first card opens the door by password	
216	Open the door by password during the normal open time period	Open the door by password during the normal open time period	
217	Successfully connected to the main controller	Successfully connected to the main controller	
218	Pass by identity card	Pass by identity card	
219	Verification within the time period when the exit button is normally open	Verification within the time period when the exit button is normally open	
220	Disconnected from the auxiliary input point	Disconnected from the auxiliary input point	
221	The auxiliary input point is short-circuited	The auxiliary input point is short-circuited	
222	Remote identification is successful	Remote identification is successful	
223	Remote identification	Remote identification	



| 224 | Ring the doorbell | Ring the doorbell ||

| 225 | The auxiliary input point is normal |The auxiliary input point is normal ||

| 226 | Trigger the auxiliary input point |Trigger the auxiliary input point ||

| 227 | The door is in dual-on mode |The door is in dual-on mode ||

| 228 | The door is in dual-off mode |The door is in dual-off mode ||

| 229 | Regularly enable normal open for auxiliary output |Regularly enable normal open for auxiliary output ||

| 230 | Regularly disable normal open for auxiliary output |Regularly disable normal open for auxiliary output ||

| 231 | IPC firmware linkage event |IPC firmware linkage event |The value of the verification mode in the linkage event record is the specific event type|

| 232 | Passed the verification |Passed the verification ||

| 233 | Remote lock | Remote lock||

| 234 | Remote unlock | Remote unlock||

| 235 | Reader upgrade successfully | Reader upgrade successfully||

| 236 | Reader disassembly alarm release |Reader disassembly alarm release||

| 237 | Reader online | Reader online||

| ||| |

| 239 | Device call |Device call||

| 240 | call ended | call ended||

| 241 | Linked capture event | Linked capture event||

| 242 | Linked recording event | Linked recording event||

| 243 | Fire alarm input disconnected | Fire alarm input disconnected ||

| 244 | Fire alarm input short circuit | Fire alarm input short circuit ||

| 245 | Connect to hotspot | Connect to hotspot ||

| 246 | Mobile network connection is successful | Mobile network connection is successful ||

| 247 | Expansion board online | Expansion board online ||

| 248 | Plug in the internet cable | Plug in the internet cable ||

| |||| |

| 254 | Definition of the extended event number |Definition of the extended event number|0-255 are insufficient and therefore 254 is used to define the number bit of the extended event. This function is newly supported by the firmware|

| 255 | Door status |Door status ||

## Appendix 3 Description of Verification Mode Code

For example, the value of VerifyStyles is FB1E1F00 (11111011 00011110 00011111 00000000):

|---|---||Binary

|Binary value|1|1|1|1|1|0|1|1|0|0|0|1|1|1|1|0|...|

|position|0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|...|

For example, the position 15 corresponds to the binary value 0, which means the face verification mode is not supported. The value of the face verification mode is 15.

| Verification mode | Description |

|---|---|

| 0 | Vein, face, fingerprint, card, or password (automatically identified) |

| 1 | Fingerprint only |

| 2 | User ID |

| 3 | Password only |

| 4 | Card only |

| 5 | Fingerprint or password |

| 6 | Fingerprint or card |

| 7 | Card or password |

| 8 | User ID and fingerprint |

| 9 | Fingerprint and password |

| 10 | Card and fingerprint |

| 11 | Card and password |

| 12 | Fingerprint, password, and card |

| 13 | User ID, fingerprint, and password |

| 14 | User ID and fingerprint, or card and fingerprint |

| 15 | Face |

| 16 | Face and fingerprint |

| 17 | Face and password |

| 18 | Face and card |

| 19 | Face, fingerprint, and card |

| 20 | Face, fingerprint, and password |

| 21 | Finger vein |

| 22 | Finger vein and password |

| 23 | Finger vein and card |

| 24 | Finger vein, password, and card |

| 200 | Other |

## Appendix 4 Language Number

| Language Number | Meaning |

| ---- | ---- |

| 83 | Simplified Chinese |

| 69 | English |

| 97 | Spanish |

| 70 | French |

| 66 | Arabic |

| 80 | Portuguese |

| 82 | Russian |

| 71 | German |

| 65 | Farsi |

| 76 | Thai |

| 73 | Indonesian |

| 74 | Japanese |

| 75 | Korean |

| 86 | Vietnamese |

| 116 | Turkish |

| 72 | Hebrew |

| 90 | Czech |

| 68 | Dutch |

| 105 | Italian |

| 89 | Slovak |

| 103 | Greek |

| 112 | Polish |

| 84 | Traditional Chinese |

## Appendix 5 Algorithm to Convert to Values in Seconds

ZKTeco's algorithm to convert year, month, day, hour, minute, and second into values in seconds:

```
unsigned int OldEncodeTime(int year, int mon, int day, int hour, int min, int sec)
{
    unsigned int tt;
    tt = ((year - 2000) * 12 * 31 + ((mon - 1) * 31) + day - 1) * (24 * 60 * 60)
        +(hour * 60 + min) * 60 + sec;
    return tt;
}
```

## Appendix 6 Algorithm to Convert to Date

ZKTeco's algorithm to convert seconds to year, month, day, hour, minute, and second:

```
void OldDecodeTime(unsigned int tt, int *year, int *mon, int *day, int *hour, int *min, int *sec)
{
    *sec = tt % 60;
    tt /= 60;
    *min = tt % 60;
    tt /= 60;
    *hour = tt % 24;
    tt /= 24;
    *day = tt % 31 + 1;
    tt /= 31;
    *mon = tt % 12 + 1;
    tt /= 12;
    *year = tt + 2000;
}
```

## Appendix 7 Device Types and Corresponding Models

Device type value	Model
-----	----
1	C3-200
2	C3-400
3	C4-400
4	C3-100
5	C4-200 and INBIO280
6	C4 (four-door to two-door)
7	C3 (four-door to two-door)
8	C3-160 and INBIO160
9	C3-260 and INBIO260
10	C3-460 and INBIO460
23	C5-100
24	C5-200
25	C5-400
26	INBIO5-100
27	INBIO5-200
28	INBIO5-400
40	BIOIR9000
101	PULL SDK Device
102	Finger vein
103	iface7
301	Wireless lock

## Appendix 8 APB Values

APB value	Description
-----	----
0	No APB
1	APB between Door 1 and Door 2
2	APB between Door 3 and Door 4
3	APB between Door 1 and Door 2 and APB between Door 3 and Door 4
4	APB between Door 1 or 2 and Door 3 or 4
5	APB between Door 1 and Door 2 or 3
6	APB between Door 1 and Door 2 or 3 or 4
16	APB between readers 1
32	APB between readers 2
48	APB between each two of readers 1 and 2 respectively

64	APB between readers 3
80	APB between each two of readers 1 and 3 respectively
96	APB between each two of readers 2 and 3 respectively
112	APB between each two of readers 1, 2, and 3 respectively
128	APB between readers 4
144	APB between each two of readers 1 and 4 respectively
160	APB between each two of readers 2 and 4 respectively
176	APB between each two of readers 1, 2, and 4 respectively
196	APB between each two of readers 3 and 4 respectively
208	APB between each two of readers 1, 3, and 4 respectively
224	APB between each two of readers 2, 3, and 4 respectively
240	APB between each two of readers 1, 2, 3 and 4 respectively

## Appendix 9 Interlock Value

Interlock value	Description
0	None
1	Interlock between Door 1 and Door 2
2	Interlock between Door 3 and Door 4
3	Interlock among Door 1, Door 2, and Door 3
4	Interlock between Door 1 and Door 2, or between Door 3 and Door 4
5	Interlock among Door 1, Door 2, Door 3, and Door 4

## Appendix 10 Table of Access Control Parameters

Parameter name	Description
DoorAutoMatch	Automatic match of Wiegand format. 0: User-defined. 1: Auto
DoorDrivertime	Lock driver time length
DoorKeepOpenTimeZone	Normal open time period of the door. 0: Disable normal open
DoorCancelKeepOpenDelay	Cancel the normal open time (linux year) \* 10000 + (linux month) \* 100 + day
DoorValidTZ	Valid time period of the door
DoorSupplierPassWord	Emergency password, up to eight bits
DoorIntertime	Punch interval in seconds. Default: 2s
DoorVerifyType	Door opening mode. See Appendix 3 Verification Mode Code
DoorSensorType	Door sensor type. 0: None. 1: Normal open. 2: Normal close
DoorDetectortime	Door sensor delay in seconds. 0: No alarm for the delay. >0: Alarm for delay
DoorCloseAndLock	Lock at door closing. 1: Enable. 0: Disable. Default: 1
DoorReaderType	Reader type

| DoorForcePassWord | Duress password, up to 6 bits |

| DoorInTimeAPB | APB duration in seconds |

| DoorREXInput | Exit button status. 1: Do not lock (default). 0: Lock |

| DoorREXTimeOut | Exit button delay in seconds. The delay after the button switch event occurs when the door is locked |

| WiegandIDIn | Wiegand input type |

| WiegandID | Wiegand output type |

| DoorDelayOpenTime | Door open delay. The delay time after verification |

| ExtDoorDelayDrivertime | Extended pass time in seconds for the disabled |

| DoorMultiCardInterTime | Multi-user door opening interval in seconds |

| DoorFirstCardOpenDoor | First-card normal open. 0: Disable. 1: First-card normal open. 2: First-card activation |

| DoorMultiCardOpenDoor | Multi-card door opening. 0: Disable. 1: Enable |

| DoorDisable | Enable or disable the door. 1: Enable. 0: Disable. |

| DoorInOutAntiPassback | In/out APB for a single door. 0: No APB. 1: In APB. 2: Out APB. 3: In and out APB |

| DoorMaskFlag | Lock the door. 1: Lock |

## Appendix 11 Table of Reader Parameters

| Parameter name | Description |

| ---- | ---- |

| IdentityCardVerifyMode | 0: Normal. 1: Identity card |

## Appendix 12 Table of Device Parameters

| Parameter name | Description |

| ---- | ---- |

| DSTOn | DLST function |

| CurTimeMode | Current mode. 1: DLST. 2: Not DLST |

| DLSTMode | DLST mode |

| IPAddress | IP address |

| GATEIPAddress | Gateway |

| NetMask | Subnet mask |

| BackupTime | Regularly back up event records. Accuracy: hour |

| DelRecord | The number of records to delete after the number of event records exceeds the allowed maximum number |

| MachineTZ | Time zone |

| AntiPassback | APB. See Appendix 8. |

| InterLock | Interlock. See Appendix 9. |

## Appendix 13 Protocol Version Rule

- Published protocol versions:

3.0.1

3.1.1

3.1.2

Encryption protocol version: 3.1.1 or later

- Device:

The device pushes the current PUSH version to the server according to the following protocol:

GET /iclock/cdata?SN=\${SerialNumber}&options=all&pushver=\${XXX}

Based on the request, the server returns the version of the published protocol that is used by the server for development to the device.

PushProtVer=xxx: If this parameter is not returned, the device thinks that the protocol version of the server is 3.0.1 by default.

The device compares its current PUSH version with the protocol version returned by the server and uses the earlier version for interaction.

- Server:

The server sends the following request to get the version of PUSH used by the client. If no pushver field is returned, the server thinks that the device's PUSH version is 3.0.1 by default.

GET /iclock/cdata?SN=\${SerialNumber}&options=all&pushver=\${XXX}

The server needs to return the version of the publish protocol used by the software:

PushProtVer=xxx

The server compares the software's protocol version with the protocol version uploaded by the device and uses the earlier version for interaction.



## Appendix 14 Parameter CmdFormat

- The following describes the meaning of the values of the parameter CmdFormat. Its default value is not 0.

When the value is 0, the format is:

C:XXX:COMMAND.....

When the value is 1, the format is:

DataType=1,SN=%s,Priority=%d,CmdID=%s,CmdDesc=%s

Note:

DataType: The data format type, which is 1 by default

SN: The command is distributed to the specified SN.

Priority: The command priority, which is 0 by default

CmdID: The command ID

For example

DataType=1,SN=123456789,CmdID=295,CmdDesc=C:295:DATA UPDATE user CardNo= Pin=1  
Password=234 Group=0 StartTime=0 EndTime=0 Name= Privilege=0

DataType=1,SN=DDG7012017010600049,CmdID=295,CmdDesc=C:295:DATA UPDATE  
DoorParameters ID=1 Name=DoorAutoMatch Value=1 DevID=1

ID=1 Name=DoorDrivertime Value=5 DevID=1

ID=1 Name=DoorKeepOpenTimeZone Value=0 DevID=1

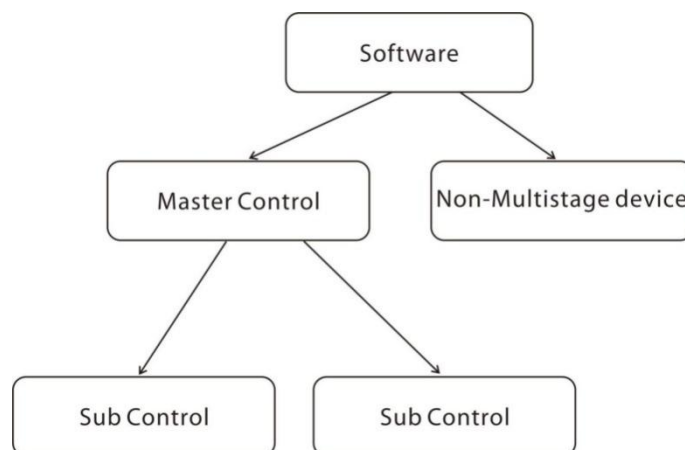
ID=1 Name=DoorValidTZ Value=1 DevID=1

ID=1 Name=DoorSupperPassWord Value= DevID=1

As for CmdFormat=1, the following prefix must be added to all C:XXX:COMMAND commands:

DataType=1,SN=%s,Priority=%d,CmdID=%s,CmdDesc=

## Appendix 15 Multi-level Control Design sketch of multi-level control



### Description of multi-level control parameters

- MultiStageControlFunOn // =1: Enable =0: Disable. Specify whether to enable the multi-level control function. The default value is 0.
- MasterControlOn // =1: Enable the main controller. =0: Disable the main controller. The default value is 0.
- SubControlOn // =1: Enable the sub-controller. =0: Disable the sub-controller. The default value is 0.

### Application scenario configurations

- Non-multi-level device. See Appendix 7.

• When MultiStageControlFunOn, SubControlOn, and MasterControlOn do not exist or their values equal to the following values

- MultiStageControlFunOn=1 or MultiStageControlFunOn=0
- SubControlOn=0
- MasterControlOn=0

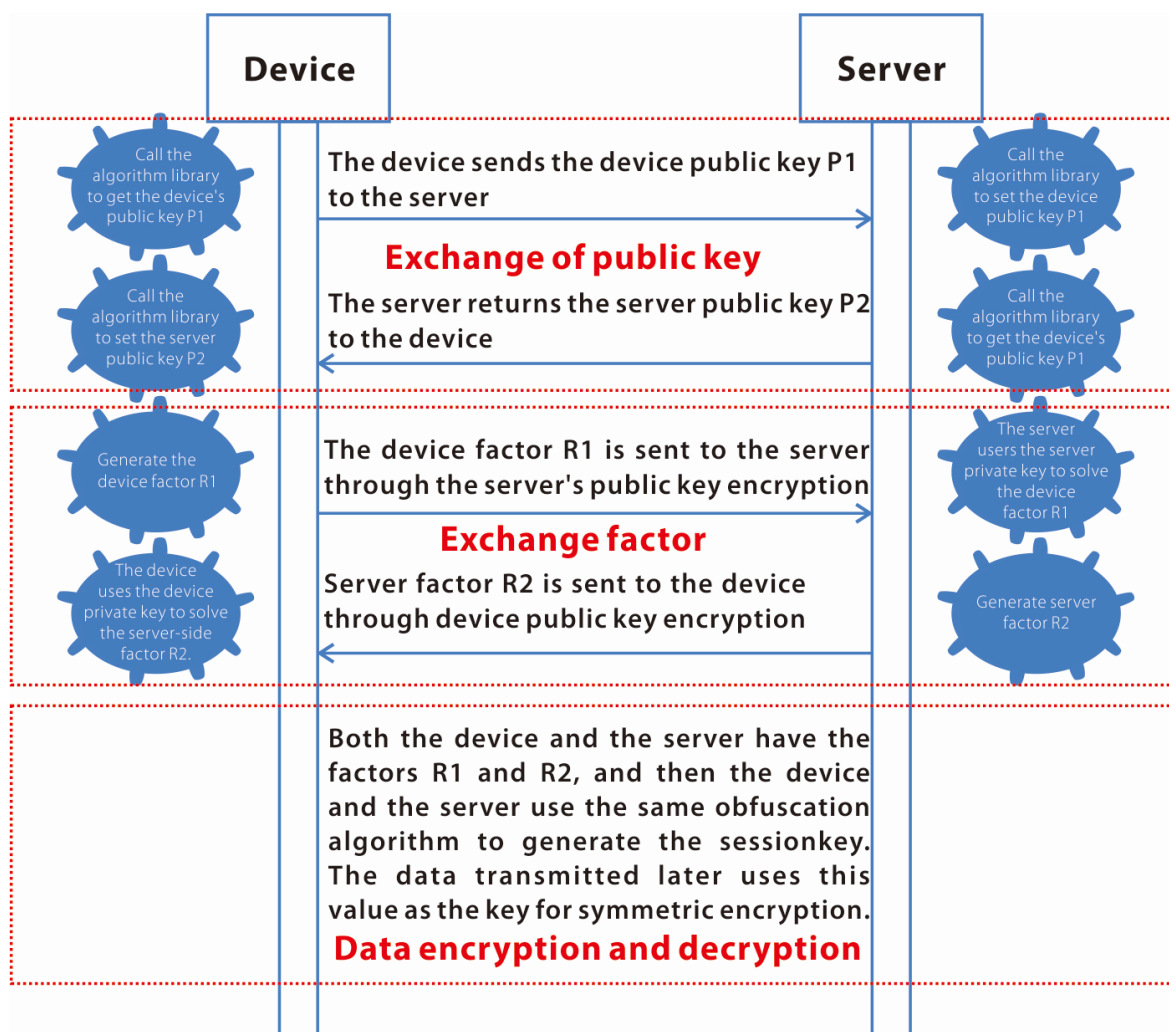
- Sub-controller

- MultiStageControlFunOn=1
- SubControlOn=1
- MasterControlOn=0

- Main controller

- MultiStageControlFunOn=1
- MasterControlOn=1
- SubControlOn=0

## Appendix 16 Data Encryption Schemes Data encryption and key exchange scheme



- Algorithm: The encryption algorithm libraries are encapsulated in a centralized manner. The device uses static algorithm libraries.
- Scheme:
  - Initialize asymmetric encrypted public and private keys when the device and server are reconnected.
  - The device and the server exchange the public keys.
    - The device sends its public key P1 to the server.
    - The server returns its public key P2 to the device.
    - The public keys are exchanged. Both the device and the server own the public keys P1 and P2.
  - The device and the server exchange the factors:
    - The device generates a factor R1 and encrypts it and then sends it to the server by using the server's public key.
    - The server decrypts the device factor R1 by using the server's private key.
    - The server generates a factor R2 and encrypts it and then sends it to the device by using the

device's public key.

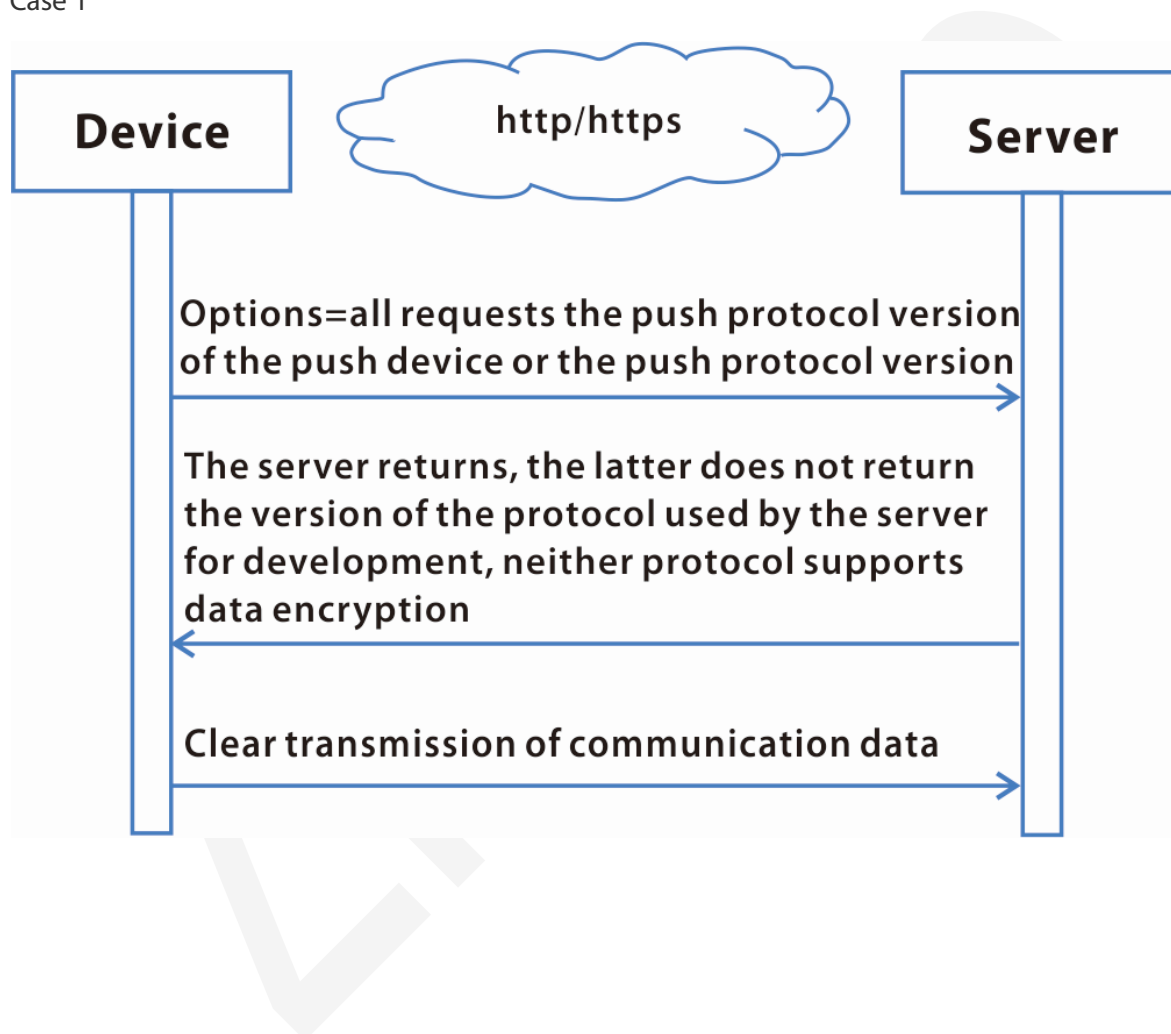
- ◆ The server decrypts the server factor R2 by using the device's private key.
- ◆ The factors are exchanged. Both the device and the server own the factors R1 and R2.

(d) Both the device and the server own the factors R1 and R2. Then, the device and the server use the same obfuscation algorithm to generate a sessionKey. All the data transmitted later uses this value as the private key for symmetric encryption.

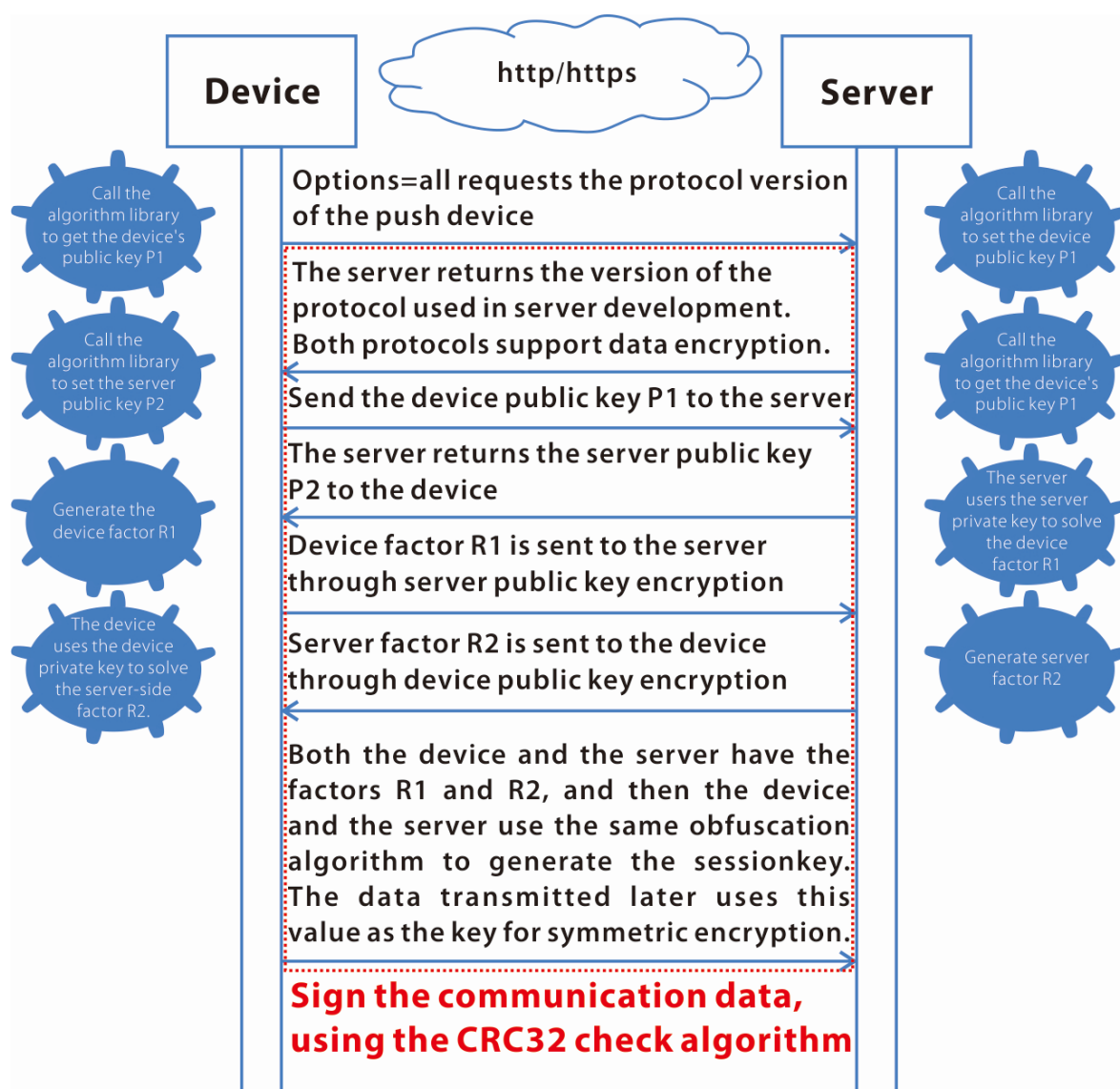
### Compatibility Scheme

The device and the server are compatible based on the protocol version, as follows:

- Case 1



- Case 2



Note:

(a) The device judges whether the transmission protocol is HTTPS or HTTP based on the set server address.

(b) The pushver field added in the existing first request protocol header of the device means the version of the current communication protocol of the device. The PushProtVer field added in the data returned by the server means the version of the protocol based on which the software is developed. The two protocol versions are compared and the device and the server use the earlier version for communication.

Case 1: When neither the protocol versions of the server and the device is supported, the data communication is transmitted in plain text.

Case 2: If one version protocol supports data encryption and the protocol versions of both the server and the device are supported, the data is encrypted for transmission.

The interaction sequence is as follows:

- ◆ The server and the device exchange their public keys P1 and P2 based on the new protocol.

- ◆ The server and the device exchange their factors R1 and R2 based on the new protocol.
- ◆ CRC32 check is performed for the communication data signature. When both the device and the server own the factors R1 and R2, the device and the server use the same obfuscation algorithm to generate a sessionKey. All the data transmitted later uses this value as the private key for symmetric encryption.

## Appendix 17 Error Codes of Error Logs

Error code	Description
00000000	Successful
D01E0001	Face detection failed
D01E0002	Face shielded
D01E0003	Insufficient clarity
D01E0004	Two large face angle
D01E0005	Life detection failed
D01E0006	Failed to extract the template

Define based on the error source, module, type, and error value.

Error source (first bit)

D: Error code returned from the device

S: Error code returned from the software

Module (the second to third bits)

Device:

01: PUSH communication module

02: Template processing module

03: Hardware interaction module

04: PULL communication module

05: Standalone communication module

06: Data relay module

07: License service module

Software:

To be confirmed

Type (the fourth bit)

E: ERROR

Error value (the fifth to the eighth bits)

Integer

## Appendix 18 Biometric Type Index

Index	0	1	2	3	4	5	6	7	8	9
Type	Common	Fingerprint	Near-infrared face	Voiceprint	Iris	Retina	Palmprint	Finger vein	Palm vein	Visible light face

Parameter	Description	Description
type	Biometric type  Type 1-8 belongs to near-infrared; Type 9 belongs to visible light.	0-Common  1-Fingerprint  2-Near-infrared face  3-Voiceprint  4-Iris  5-Retina  6-Palmprint  7-Finger vein  8-Palm vein  9-Visible light face
MultiBioPhotoSupport	Supports biometric photos	The type is defined bit by bit. Different types are separated by colons, 0 means not supported, 1 means supported.  Such as: 0: 1: 1: 0: 0: 0: 0: 0: 0: 0, indicating support for

		near-infrared fingerprint photo and face photo.
MultiBioDataSupport	Supports bio-templates	<p>The type is defined bit by bit. Different types are separated by colons, 0 means not supported, 1 means supported.</p> <p>Such as: 0: 1: 1: 0: 0: 0: 0: 0: 0: 0, indicating support for near-infrared fingerprint template and face template.</p>
MultiBioVersion	Supported algorithms	<p>The type is defined bit by bit. Different types are separated by colons, 0 means not supported, non-0 means supported version number.</p> <p>Such as: 0: 10: 0: 7: 0: 0: 0: 0: 0, indicating support for fingerprint algorithm10.0 and near-infrared face algorithm7.0.</p>
MaxMultiBioDataCount	Supports maximum number of bio-templates.	<p>The type is defined bit by bit. Different types are separated by colons, 0 means not supported, non-0 means supported maximum capacity.</p> <p>Such as: 0: 10000: 3000: 0: 0: 0: 0: 0: 0: 0, indicating support for the maximum number of fingerprint templates is 10000 and the maximum number of near-infrared face templates is 3000.</p>
MaxMultiBioPhotoCount	Supports maximum number of biometric photos.	<p>The type is defined bit by bit. Different types are separated by colons, 0 means not supported, non-0 means supported maximum capacity.</p> <p>Such as: 0: 10000: 3000: 0: 0: 0: 0: 0: 0: 0, indicating support for the maximum number of fingerprint photos is 10000 and the maximum number of near-infrared face photos is 3000.</p>



MultiBioDataCount	The current capacity of bio-templates	<p>The type is defined bit by bit. Different types are separated by colons.</p> <p>Such as: 0: 10000: 3000: 0: 0: 0: 0: 0: 0: 0, indicating the current number of fingerprint templates is 10000 and the current number of near-infrared face templates is 3000.</p>
MultiBioPhotoCount	The current capacity of biometric photos	<p>The type is defined bit by bit. Different types are separated by colons.</p> <p>Such as: 0: 10000: 3000: 0: 0: 0: 0: 0: 0: 0, indicating the current number of fingerprint photos is 10000 and the current number of near-infrared face photos is 3000.</p>

ZKTeco Industrial Park, No. 32, Industrial Road,  
Tangxia Town, Dongguan, China.

Phone : +86 769 - 82109991

Fax : +86 755 - 89602394

[www.zkteco.com](http://www.zkteco.com)

