Attendance PUSH Communication Protocol

PUSH SDK

Date: March 2020

Software Version: 2.4.1

Doc Version: 3.7

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.

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. 26, 188 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.

To know more about our global branches, visit 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 Attendance 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**
Bold font	Used to identify software interface names e.g. **OK**, **Confirm**, **Cancel**
>	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.

Edit history

Date	Version	Description	Modifier	Note
2020/03/20	V3.7	1. Add hybrid identification protocol 2. Modify the initialization information exchange protocol 3. Modify the push configuration information protocol 4. Modify the protocol for issuing comparison photos 5. Add query unified template protocol 6. Add clear unified template protocol 7. Add the Heartbeat protocol	eirc.cao	
2019/08/02	V3.6	1. Add exception log protocol	darren.li	
2019/05/30	V3.5	1. Add the credentials protocol: ①Upload identity card attendance record protocol ②Upload identity card attendance record photo protocol ③Identity card blacklist issue protocol	eirc.cao	
2018/10/08	V3.4	1. Communication encryption added 2 protocols: ① Exchange public key protocol ② Exchange factor protocol 2. Support communication encryption version description: ①Attendance PUSH: 2.4.0 and above 3. Details of communication encryption are shown in appendix 8.	Yan Guangtian	
2018/8/9	V3.3	1. TransFlag added two: ①11 (Work code, WORKCODE) ②12 (Comparison photo, BioPhoto) 2. Online registration card ENROLL_MF 3. Online registration of face, palm print (unified templates) ENROLL_BIO 4. Upload unified templates to add visible light face Type=9 5. Online update 6. Background verification 7. Add the following parameters:	Yan Guangtain/ guodong.wang	

		① BioPhotoFun is used to mark comparison photo ② BioDataFun identifies visible light face templates ③ VisilightFun to identify visible light devices 8. Add comparison photo protocol		
2017/11/10	V3.2	1. Description of serial number 2. Add the initial request reply content to support the BIODATA table	darren.li	
2017/9/8	The first version	1. Perfect the list of error codes, distinguish common error codes and special command errors 2. Add unified Templates (currently applied to the palm template) 3.Add Pushing Configuration Information (to be customized) 4. Set up the new user authentication mode 5. Add data packaging and uploading agreement (for customization) 6. Extend the PUTFILE command and support synchronous data protocol 7. Modify the format of upload operation record protocol	XSEN	

Table of Contents

1 Abstract

The Push protocol is a data protocol which is defined based on the Hyper Text Transmission Protocol (HTTP). Established on a TCP/IP connection, the Push protocol is applicable to the data interchange between a server and a ZKTeco attendance device or a ZKTeco access control device, and defines the transmission formats of data (including user information, biological recognition templates, and attendance records) and the command format for control equipment. Currently, ZKTeco supports servers such as the WDMS, ZKECO, ZKNET, and ZKBioSecurity3.0, as well as third-party servers such as the ESSL from India.

1.1 Features

- Active uploading of new data
- Resuming transmission from breakpoint
- The client initiates all behaviors such as uploading data or performing commands issued by the server

1.2 Encoding

Most data transmitted via the protocol is consisted of ASCII characters, but individual fields involve coding, for example, the user name. Therefore, the following rules are made for data of this type.

- For Chinese data, the GB2312 encoding is used
- For other languages, the UTF-8 encoding is used
- Currently, the following data involves this encoding
- User names in a user information table
- Content of the short messages in a short message table

1.3 Introduction to HTTP Protocol

Since the Push protocol is a data protocol defined based on the HTTP protocol, a brief introduction to the HTTP is given hereby. Skip this part if you are already familiar with it.

The HTTP is a request/response protocol. The format of a request sent by a client to a server is a request method, a URI and a protocol version number, and then a MIME-like message containing modifiers, client information and a possible message body. The format of a response sent by the server to the client is a status line followed by a MIME-like message containing server information, entity meta-information and possible entity-body content. The status line contains the protocol version number of the message and a success code or error code. The following is an example.

A request from the client:

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

A response from the server:

HTTP/1.1 200 OK
Server: nginx/0.8.12
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 under a TCP/IP connection. The default port is TCP 80, but other ports can also be used. However, the HTTP protocol might also be implemented via other protocols. Only reliable transmission is expected from the HTTP (Note: HTTP is usually established on a transport layer protocol), therefore, any protocol providing such guarantee can be used.

2 Definitions

In this document, the format of definition reference is: ${ServerIP}

- ServerIP: The IP address of the server
- ServerPort: A port of the server
- XXX: An unknown value
- Value 1\Value 2\Value 3\……\Value n: Value 1\Value 2\Value 3\……\Value n
- Required: Mandatory
- Optional: Selectable
- SerialNumber: Serial number (it can be formed by characters, numbers, or combination of charcters+numbers)
- NUL: Null (\0)
- SP: A space
- LF: A line break (\n)
- HT: A tab character (\t)
- DataRecord: A data record
- CmdRecord: A command record
- CmdID: The ID of a command
- CmdDesc: Command description
- Pin: ID
- Time: Attendance time
- Status: Attendance status
- Verify: Verification mode
- Workcode: A workcode
- Reserved: A reserved field
- OpType: An operation type
- OpWho: An operator
- OpTime: Operation time
- BinaryData: A binary data flow
- TableName: The name of a data table
- SystemCmd: A system command
- Key: A key
- Value: A value
- FilePath: A file path
- URL: A resource location

3 Functions

The following functions supported by the Push protocol are described from the view of a client.

- Initializing Information Exchange
- Uploading Update Information
- Uploading Data
- Downloading Command
- Command Reply
- Remote Attendance

3.1 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 server issues the following two parameters to the device through the [Initialization Information Exchange] interface: MultiBioDataSupport, MultiBioPhotoSupport.
2. The device uploads the following 5 parameters to the server through the [Pushing Configuration Information] interface: MultiBioDataSupport, MultiBioPhotoSupport, MultiBioVersion, MaxMultiBioDataCount, MaxMultiBioPhotoCount. See [Pushing Configuration Information] interface description for details.
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=0:1:0:0:0:0:0:0:0:1, MultiBioPhotoSupport =0:0:0:0:0:0:0:0:0:1

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

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 device can upload the quantity of photos/ templates saved by the current device in real time through the [Pushing Configuration Information] interface.

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].

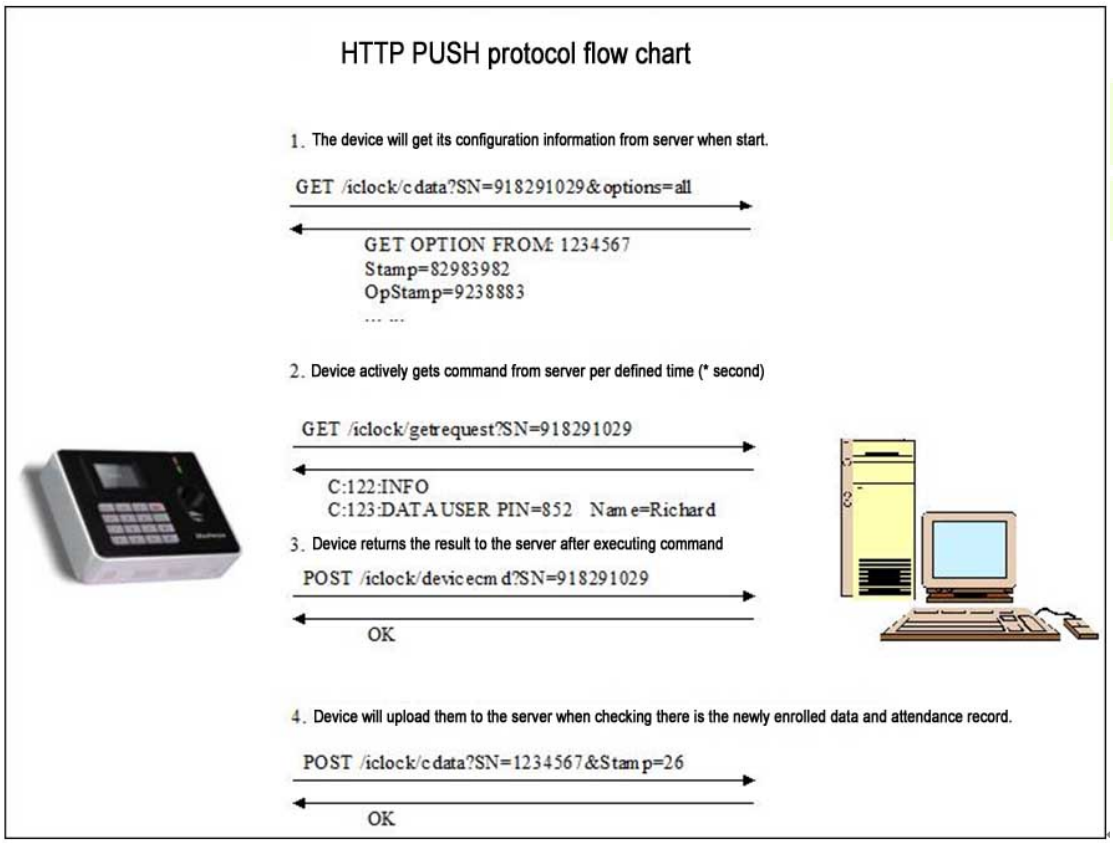
Hybrid identification protocol provides optimization strategies:

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.

4 Process

Between a client and a server that both use the Push protocol, a request of "Initialization Information Exchange" must be firstly initiated by the client successfully and then other functions can be used, such as uploading data, obtaining server commands, uploading update information, and replying server commands. These functions are not necessarily in order but dependent to the development of the client application, as shown in the figure below.



5 Initialization Information Exchange

The client initiates a request to and sends corresponding configuration information to the server, and the server replies to the client with corresponding configuration information after receiving the request. Only when the client obtains the corresponding configuration information, the exchange is successful. The configuration information is exchanged in a specified format as shown below:

A request message from the client:

```
GET  /iclock/cdata?SN=${SerialNumber}&options=all&pushver=${XXX}&language=${XXX}&pushcommkey=${XXX} HTTP/1.1
Host: ${ServerIP}: ${ServerPort}
……
```

Annotation:

HTTP request method: GET method

URI: /iclock/cdata

HTTP protocol version: 1.1

Client configuration information:

SN: ${Required} Client's serial number

options: ${Required} Obtaining server configuration parameters, and only the value "all" is available currently

pushver: ${Optional} latest Push protocol version of the device supported by a newly-developed client software, and is of the 2.2.14 version or higher. See “Appendix 6”.

language: ${Optional} languages supported by the client, better supported by a newly developed client so that the server knows the language the current equipment uses. See "Appendix 2".

pushcommkey: ${Optional} ciphertext information for binding the client and the server, allowing the software to determine whether the equipment is authorized or not. The value differs fordifferent equipment. This parameter needs to be supported by the client only when it is supported by the server.

Host header field: ${Required}

Other header fields: ${Optional}

A normal response from the server :

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

GET OPTION FROM: ${SerialNumber}${LF}${XXX}Stamp=${XXX}${LF}ErrorDelay=${XXX}${LF}Delay=${XXX}${LF}TransTimes=${XXX}${LF}TransInterval=${XXX}${LF}TransFlag=${XXX}${LF}TimeZone=${XXX}${LF}Realtime=${XXX}${LF}Encrypt=${XXX}${LF}ServerVer=${XXX}${LF}PushProtVer=${XXX}${LF}PushOptionsFlag=${XXX}${LF}PushOptions=${XXX}

Annotation:

HTTP status line: Defined according to the standard HTTP protocol

HTTP response header field:

Date header field: ${Required} This header field is used for server time synchronization in GMT time format, for example, Date: Fri, 03 Jul 2015 06:53:01 GMT

Content-Length header field: Based on the HTTP 1.1 protocol, this header field is usually used to specify the data length of a response entity. If the entity size is uncertain, header fields Transfer-Encoding: chunked, Content-Length and Transfer-Encoding are supported, all of which are standard definitions of the HTTP protocol.

Server configuration information:

The description in the first line must be this: GET OPTION FROM: ${SerialNumber}, with the ${LF} separating configuration information.

${SerialNumber} is the serial number of the request initiated by the client. The configuration information is in key=value pairs, with a ${LF} separating two configurations.

${XXX}Stamp: Timestamps for all kinds of data types, currently supporting the following:

${XXX}	Data type
ATTLOG	Attendance record
OPERLOG	Operation log
ATTPHOTO	Attendance photo
BIODATA	Unified Templates
IDCARD	Identity card information
ERRORLOG	Error log

Purpose of timestamp mark design: When the client uploads data, the corresponding timestamp mark is uploaded. The server is responsible for recording this mark. When the equipment reboots, the client initiates a request for initialization of information exchange, and the server sends a series of marks to the client, realizing the function of resuming transmission from breakpoint.

Timestamp mark flaw: As time modification is permitted and the uncertainty of time change is possible, the client may not correctly determine which data has been uploaded to the server and which has not, and this leads to server data loss.

Application of timestamp on server: Currently, the server has only one application of the timestamp mar

k. When the server needs to reupload all corresponding data, it sets the corresponding timestamp mark t o 0. See this function at "Get Command – Control Command – Check Data Update".

Timestamp discard at client side: In the Push design for new framework firmware, no timestamp is used to mark a cut-off point of data uploading. However, for compatibility with old servers, timestamp marks are also sent. Actually, it realizes only the function of data reuploading when the mark is set to 0, so the server does not need to differentiate whether the client has discarded a timestamp or not.

ErrorDelay: Interval time for the client to reconnect to the server after networking connection failure, and the recommended value is 30~300s.

Delay: Interval for the client to connect to the server when the networking is normal (s), that is, the functi on of requesting "Get Command" by client. The recommended value is 2~60s. When a rapid response is r equired, a smaller value can be set, but this will increase the pressure on the server.

TransTimes: Time at which the client checks for and transmits new data regularly (in a 24-hour format: ho ur: minute) and multiple times are separated by semicolons. Up to 10 times are supported. For example, TransTimes=00: 00;14: 00

TransInterval: Interval for the client to check and transmit new data (in minute), and no check is perform ed when it is set to 0. For example, TransInterval=1

TransFlag: Identifying the data to be uploaded by the client automatically to the server, and two formats are supported.

Format I: TransFlag=1111000000……, each digit representing a data type. 0 for forbidding automatic up loading of this data type, 1 for allowing automatic uploading of this data type.

Data type on each digit

1	Attendance record
2	Operation log
3	Attendance photo
4	Enrolling a new fingerprint
5	Enrolling a new user
6	Fingerprint image
7	Changing user information
8	Changing a fingerprint
9	New enrolled face
10	User picture
11	Work code
12	Comparison photo

Format II: TransFlag=TransData AttLog${HT}OpLog${HT}AttPhoto……

Data types marked by strings

AttLog	Attendance log
OpLog	Operation log
AttPhoto	Attendance photo
EnrollUser	Enrolling a new user
ChgUser	Changing user information

EnrollFP	Enrolling a new fingerprint
ChgFP	Changing a fingerprint
FPImag	Fingerprint image
FACE	New enrolled face
UserPic	User picture
WORKCODE	Work code
BioPhoto	Comparison photo

During new client development: Please support both formats simultaneously. When the server sends dat a in format I with all values set to 0 (TransFlag=0000000000), only uploading attendance photos is suppo rted.

During new server development: Only format II needs to be supported.

TimeZone: Specify the time zone where the server is located, primarily for server time synchronization. S ee the Date header field in [Get Command](#downloadcmd). This value is an integerand designed to sup port a whole time zone, half time zone and 1/4 time zone.

For -12 < TimeZone < 12, it is a whole time zone in the unit of hour. For example, TimeZone=4 means th e East 4 zone.

For TimeZone > 60 or TimeZone < -60, it can mean a half time zone or 1/4 time zone in the unit of minut e. For example, TimeZone=330 means a half of the East 5 time zone.

Realtime: Whether the client transmits new records in real time.1 means that data is transmitted to the s erver as soon as it is generated, while 0 means data is transmitted at the time defined by the TransTimes and TransInterval.

Encrypt: Whether to transmit data after encryption, to support the occasion of communicationencryptio n, this parameter should be set to 1.

EncryptFlag: The identity of data encryption.

Example: EncryptFlag = 10000000

Bit	Date type
1	attendance record

Currently, only version 2.3.0 of this protocol is supported, and only the encryption of attendance records is supported. Rc4 is used for encryption.

ServerVer: Protocol version and time (format to be determined), which are supported by the server, and i t must be set to 2.2.14 or above for a newly-developed server.

PushProtVer: The server is developed according to which protocol version, please refer to (appendix 6).

PushOptionsFlag: Whether the software supports the device push configuration parameter request, 0 is not supported, 1 is supported, and it is not supported by default when it is not set.

PushOptions: The software requires the device to push the parameter list, format :PushOptions=key1,key

2,key3,……,keyN, such as PushOptions=FingerFunOn,FaceFunOn.

ATTPHOTOBase64: Attendance photo base64 identity. 1: base64 encoding, other occasions is not base64 encoding.

MultiBioDataSupport: Supports multi-modal biometric 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: 0, indicating support for fingerprint photo and near-infrared face photo.

Example:

A request from the client:

GET /iclock/cdata?SN=0316144680030&options=all&pushver=2.2.14&language=83&pushcommkey=4a9594af164f2b9779b59e8554b5df26 HTTP/1.1
Host: 58.250.50.81: 8011
User-Agent: iClock Proxy/1.09
Connection: close
Accept: */*

A response from the server:

HTTP/1.1 200 OK
Server: nginx/1.6.0
Date: Fri, 03 Jul 2015 06: 53: 01 GMT
Content-Type: text/plain
Content-Length: 190
Connection: close
Pragma: no-cache
Cache-Control: no-store

GET OPTION FROM: 0316144680030
ATTLOGStamp=None
OPERLOGStamp=9999
ATTPHOTOStamp=None
ErrorDelay=30
Delay=10
TransTimes=00: 00;14: 05
TransInterval=1
TransFlag=TransData AttLog OpLog AttPhoto EnrollUser ChgUser EnrollFP ChgFP UserPic
TimeZone=8
Realtime=1
Encrypt=None

6 Exchange of Public Keys (where encryption of communications is supported)

The functional device pushes the public key of the device and receives the public key of the server returned by the server.

A request message from the client:

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

Annotation:

HTTP request method: POST method

URI: /iclock/ exchange

HTTP protocol version: 1.1

Host header field: ${Required}

Other header fields: ${Optional}

PublicKey: The device PublicKey returned by calling the encryption library.

A normal 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}
```

Annotation:

PublicKey: The server PublicKey returned by the server.

7 Exchange factor (where communication encryption is supported)

This function pushes the device factor and receives the server factor returned by the server.

A request message from the client:

```
POST /iclock/exchange?SN=$(SerialNumber)&type=factors
Host: ${ServerIP}:${ServerPort}
Content-Length: ${XXX}
……
Factors=${XXX}
```

Annotation:

HTTP request method: POST method

URI: /iclock/ exchange

HTTP protocol version: 1.1

Host header field: ${Required}

Other header fields: ${Optional}

Factors: The device factor returned by calling the encryption library.

A normal 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}
```

Annotation:

Factors: The server factor returned by the server.

8 Pushing Configuration Information

The functional device proactively pushes relevant configuration information, which can be designated by the device or the server (see "PushOptions" in "Exchanging Initialization Information" for more information). Any change to configuration information is proactively pushed to the server.

Request message from the client.

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

Host: ${ServerIP}:${ServerPort}

Content-Length: ${XXX}

……

${key}=${Value},${key}=${Value},${key}=${Value}……,${key}=${Value}

UserPicURLFunOn: Supports issuing user photos by URL.

Hybrid identification protocol adds the following $ {key}:

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: 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: 0, indicating support for fingerprint algorithm10.0 and near-infrared face algorithm7.0.

MultiBioCount: Supports multi-modal biometric data version 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: 100: 200: 0: 0: 0: 0: 0: 0: 0, indicating support for 100 fingerprints and 200 near-infrared faces.

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: 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: 0, indicating support for the maximum number of fingerprint photos is 10000 and the maximum number of near-infrared face photos is 2000.

Annotation:

HTTP Request Method: POST method

URI: /iclock/cdata

HTTP Version: 1.1

Client Configuration Information:

table=options

Host Header Field: ${Required}

Other Header Field: ${Optional}

Normal response message from the server

HTTP/1.1 200 OK

Content-Length: ${XXX}

……

OK

Example

Request from the client:

POST /iclock/cdata?SN=0316144680030&table=options HTTP/1.1

Host: 58.250.50.81:8011

Content-Length: 26

User-Agent: iClock Proxy/1.09

Connection: close

Accept: */*

FingerFunOn=1,FaceFunOn=1

Response from the server:

HTTP/1.1 200 OK

Server: nginx/1.6.0

Date: Tue, 30 Jun 2015 01:24:26 GMT

Content-Type: text/plain

Content-Length: 2

Connection: close

Pragma: no-cache

Cache-Control: no-store

OK

9 Uploading Update Information

This function multiplexes the Download Command (#downloadcmd) request and adds parameters in its URL to mainly upload the client's firmware version number, number of enrolled users, number of enrolled fingerprints, number of attendance records, IP address of equipment, fingerprint algorithm version, face algorithm version, number of faces required for face enrollment, number of enrolled faces, and marked information about functions supported by the equipment.

A request message from the client:

```
Get /iclock/getrequest?SN=${SerialNumber}&INFO=${Value1},${Value2},${Value3},${Value4},${Value5},
${Value6},${Value7},${Value8},${Value9},${Value10}
Host: ${ServerIP}: ${ServerPort}
……
```

Annotation:

HTTP request method: GET method

URI: /iclock/getrequest

HTTP protocol version: 1.1

Client configuration information:

SN: ${Required} Client's serial number

${Value1}: Firmware version number

${Value2}: Number of enrolled users

${Value3}: Number of enrolled fingerprints

${Value4}: Number of attendance records

${Value5}: IP address of Equipment

${Value6}: Version of fingerprint algorithm

${Value7}: Version of face algorithm

${Value8}: Number of faces required for face enrollment

${Value9}: Number of enrolled faces

${Value10}: Identifier of functions supported by the equipment in the format of 101 with every digit representing a function, 0—Not supporting this function, 1—Supporting this function.

Description of function on each digit

1	Fingerprint function
2	Face function
3	User photo function
4	Comparison photo function (comparison photo function is supported, th

e parameter BioPhotoFun needs to be set to 1)

5 Visible light face template function (face template function issupported, the parameter BioDataFun needs to be set to 1)

(Push the first 3 digits by default, and push 5 digits when VisilightFun is set to 1)

Host header field: ${Required}

Other header fields: ${Optional}

For server responses, see Download Command.

Example

A request from the client:

GET /iclock/getrequest?SN=0316144680030&INFO=Ver%202.0.12-20150625,0,0,0,192.168.16.27,10,7,15,0,111 HTTP/1.1
Host: 58.250.50.81: 8011
User-Agent: iClock Proxy/1.09
Connection: close
Accept: */*

A response from the server:

HTTP/1.1 200 OK
Server: nginx/1.6.0
Date: Tue, 30 Jun 2015 01: 24: 26 GMT
Content-Type: text/plain
Content-Length: 2
Connection: close
Pragma: no-cache
Cache-Control: no-store

OK

10 Heartbeat

Used to maintain a heartbeat with the server. When processing big data upload, use ping to keep the heartbeat. When big data is processed, use getrequest to keep the heartbeat.

A request message from the client:

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

A response from the server:

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

Annotation:

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

Example

```
A request from the client:
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

A 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

11 Uploading Data

The data to be uploaded automatically can be set on the server. (For details, see the "TransFlag" parameter in "Initialization Information Exchange".)

11.1 Uploading Mode

Realtime uploading

Interval uploading

Timed uploading

Real-time \ interval \ timed three upload modes, if support real-time, interval \ timed

mode does not work.

Realtime uploading: This is supported by the equipment by default and can be controlled by the server. (For details, see the "Realtime" parameter in “Initializing Information Exchange”).

Interval uploading: The server can control specific interval time. (For details, see the "TransInterval" parameter in “Initializing Information Exchange”.)

Timed uploading: The server can control specific upload timing. (For details, see the "TransTimes" parameter in “Initializing Information Exchange".)

11.2 Uploading Attendance Record

A request message from the client:

```
POST /iclock/cdata?SN=${SerialNumber}&table=ATTLOG&Stamp=${XXX} HTTP/1.1
Host: ${ServerIP}: ${ServerPort}
Content-Length: ${XXX}
……

${DataRecord}
```

Annotation:

HTTP request method: POST method

Used URI: /iclock/cdata

HTTP protocol version: 1.1

Client configuration information:

SN: ${Required} Serial number of the client table=ATTLOG: ${Required} Indicating that the uploaded data is attendance records.

Stamp: ${Optional} Latest timestamp at which the attendance record is uploaded to the server. (For details, see the "Stamp" or "ATTLOGStamp" parameter in "Initializing Information Exchange".)

Host header field: ${Required}

Content-Length header field: ${Required}

Other header fields: ${Optional}

Request entity: ${DataRecord}, attendance record data, in the following format:

${Pin}${HT}${Time}${HT}${Status}${HT}${Verify}${HT}${Workcode}${HT}${Reserved}${HT}${Reserved}
Note:
${Time}: Verification time, in the format of XXXX-XX-XX XX: XX: XX. For example, 2015-07-29 11: 11: 11, with ${LF} used to connect multiple records.

A normal response message from the server:

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

OK: ${XXX}
```

Annotation:

HTTP status line: Defined with standard HTTP protocol

HTTP response header field:

Content-Length header field: According to the HTTP 1.1, this header field is generally used to specify the data length of the response entity. If the response entity size is uncertain, head fields of Transfer-Encoding: chunked, Content-Length and Transfer-Encoding are supported, all of which are standard definitions of HTTP and are not described in details here.
Response entity: When the server normally receives data and successfully processes data, OK: ${XXX} is r eplied. ${XXX} represents the number of records successfully processed. When an error occurs, the error description is replied.

Example

A request from the client:

```
POST /iclock/cdata?SN=0316144680030&table=ATTLOG&Stamp=9999 HTTP/1.1
Host: 58.250.50.81: 8011
User-Agent: iClock Proxy/1.09
Connection: close
Accept: */*
Content-Length: 315

1452    2015-07-30 15: 16: 28 0   1   0   0   0
1452    2015-07-30 15: 16: 29 0   1   0   0   0
1452    2015-07-30 15: 16: 30 0   1   0   0   0
1452    2015-07-30 15: 16: 31 0   1   0   0   0
1452    2015-07-30 15: 16: 33 0   1   0   0   0
1452    2015-07-30 15: 16: 34 0   1   0   0   0
```

```
1452    2015-07-30 15: 16: 35 0   1   0   0   0
8965    2015-07-30 15: 16: 36 0   1   0   0   0
8965    2015-07-30 15: 16: 37 0   1   0   0   0
```

A 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

OK: 9
```

11.3 Uploading Attendance Photo

The configuration PushProtVer parameter sent by the server for initialization information exchange is greater than or equal to version 2.2.14.

A request message from the client:

```
POST /iclock/cdata?SN=${SerialNumber}&table=ATTPHOTO&Stamp=${XXX} HTTP/1.1
Host: ${ServerIP}: ${ServerPort}
Content-Length: ${XXX}
……

${DataRecord}
```

Annotation:

HTTP request method: POST method

URI: /iclock/fdata or /iclock/cdata

HTTP protocol version: 1.1

Client configuration information:

SN: ${Required} Serial number of the client table=ATTPHOTO: ${Required}

Stamp: ${Optional} Latest timestamp at which the attendance photo is uploaded to the server. (For details, see the "ATTPHOTOStamp" parameter in "Initializing Information Exchange".) Host header field: ${Required}

Content-Length header field: ${Required}

Other header fields: ${Optional}

Request entity: ${DataRecord}, attendance photo data, in the following format:

PIN=${XXX}${LF}SN=${SerialNumber}${LF}size=${XXX}${LF}CMD=uploadphoto${NUL}${BinaryData}

Note:

PIN=${XXX}: Filename of the attendance photo, with only the jpg format supported currently.

SN=${XXX}: Serial number of the client.

size=${XXX}: Original size of the attendance photo

${BinaryData}: Binary dataflow of the original photo.

Transmission of multiple records is not supported in attendance photos.

A normal response message from the server:

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

OK
```

Annotation:

HTTP status line: Defined with standard HTTP protocol

HTTP response header field:

Content-Length header field: According to the HTTP 1.1, this header field is usually used to specify the data length of the response entity. If the response entity size is uncertain, head fields of Transfer-Encoding: chunked, Content-Length and Transfer-Encoding are supported, all of which are standard definitions of HTTP and are not described in details here.

Response entity: When the server normally receives data and successfully processes data, OK is replied. When an error occurs, the error description is replied.

Example:

A request from the client:

```
POST /iclock/cdata?SN=0316144680030&table=ATTPHOTO&Stamp=9999 HTTP/1.1
Host: 58.250.50.81: 8011
User-Agent: iClock Proxy/1.09
Connection: close
Accept: */*
Content-Length: 1684

PIN=20150731103012-123.jpg SN=0316144680030 size=9512 CMD=uploadphoto${NUL}${BinaryData}
```

A 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: 2
Connection: close
Pragma: no-cache
Cache-Control: no-store

OK

11.4 Uploading Operation Record

The configuration PushProtVer parameter sent by the server for initialization information exchange is greater than or equivalent to version 2.2.14.

A request message from the client:

POST /iclock/cdata?SN=${SerialNumber}&table=OPERLOG&Stamp=${XXX} HTTP/1.1

Host: ${ServerIP}: ${ServerPort}

Content-Length: ${XXX}
……

${DataRecord}

Annotation:

HTTP request method: POST method

URI: /iclock/cdata

HTTP protocol version: 1.1

Client configuration information:

SN: ${Required} Serial number of the client

table=OPERLOG: ${Required}

Stamp: ${Optional} Latest timestamp at which the attendance record is uploaded to the server. (For details, see the "OPERLOGStamp" parameter in "Initializing Information Exchange".) Host header field: ${Required}
Content-Length header field: ${Required}

Other header fields: ${Optional}

Request entity: ${DataRecord}, operation record data, in the following format:

OPLOG${SP}${OpType}${HT}${OpWho}${HT}${OpTime}${HT}${Value1}${HT}${Value2}${HT}${Value3}${HT}${Reserved}

${OpType}: Operation code. See Appendix 3.

${Value1}, ${Value2}, ${Value3}, ${Reserved}: Operand 1, 2, 3 and 4. See Appendix 4.

Note:

${LF} is used to connect multiple records.

A normal response message from the server:

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

OK: ${XXX}
```

Annotation:

HTTP status line: Defined with standard HTTP protocol

HTTP response header field:

Content-Length header field: According to the HTTP 1.1, this header field is generally used to specify the data length of the response entity. If the response entity size is uncertain, head fields of Transfer-Encoding: chunked, Content-Length and Transfer-Encoding are supported, all of which are standard definitions of HTTP and are not described in details here. Response entity: When the server normally receives data and successfully processes data, OK: ${XXX} is replied. ${XXX} represents the number of records successfully processed. When an error occurs, the error description is replied.

Example

A request from the client:

```
POST /iclock/cdata?SN=0316144680030&table=OPERLOG&Stamp=9999 HTTP/1.1
Host: 58.250.50.81: 8011
User-Agent: iClock Proxy/1.09
Connection: close
Accept: */*
Content-Length: 166

OPLOG 4 14 2015-07-30 10: 22: 34   0   0   0   0
```

A 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: 3
```

```
Connection: close
Pragma: no-cache
Cache-Control: no-store

OK: 1
```

11.5 Uploading User Information

The configuration PushProtVer parameter sent by the server for initialization information exchange is greater than or equals to version 2.2.14.

A request message from the client:

```
POST /iclock/cdata?SN=${SerialNumber}&table=OPERLOG&Stamp=${XXX} HTTP/1.1
Host: ${ServerIP}: ${ServerPort}
Content-Length: ${XXX}
……

${DataRecord}
```

Annotation:

HTTP request method: POST method

URI: /iclock/cdata

HTTP protocol version: 1.1

Client configuration information:

SN: ${Required} Serial number of the client

table=OPERLOG: ${Required}

Stamp: ${Optional} Latest timestamp at which user information is uploaded to the server. (For details, see the "OPERLOGStamp" parameter in "Initializing Information Exchange".)

Host header field: ${Required}

Content-Length header field: ${Required}

Other header fields: ${Optional}

Request entity: ${DataRecord}, fingerprint template data, in the following format:

USER${SP}PIN=${XXX}${HT}Name=${XXX}${HT}Pri=${XXX}${HT}Passwd=${XXX}${HT}Card=${XXX}${HT}Grp=${XXX}${HT}TZ=${XXX}${HT}Verify=${XXX}${HT}ViceCard=${XXX}

Note:

Name=${XXX}: User name. When the equipment is in Chinese, the GB2312 code is used. When the equip

ment is in another language, the UTF-8 code is used.

Card=${XXX}: User card number (main card), supporting only two formats.

a. hexadecimal data, in the format of [%02x%02x%02x%02x], representing the first, second, third and f ourth digit from left to right. For example, if the card number is 123456789, this is Card=[15CD5B07]

b. string data. If the card number is 123456789, this is: Card=123456789

TZ=${XXX}: Information on number of the time period used by the user, in the format of XXXXXXXXXXXX XXXX. Digits 1-4 describe whether the group time period is used, digits 5-8 description use personal time period 1, digits 9-12 description use personal time period 2, and digits 13-16 description use personal ti me period 3.

For example: 0000000000000000 represents use of the group time period.

0001000200000000 represents using personal time period, with personal time period 1 using the time in formation of time period numbered 2.

0001000200010000 represents using personal time period, with personal time period 1 using the time in formation of time period numbered 2 and personal time period 2 using the time information of time peri od numbered 1.

Verify=${XXX} : User verification mode, does not contain the field, is null, or is set to -1(use group verificat ion mode, if there is no access group, group verification mode is 0), otherwise see (appendix 7)

ViceCard=${XXX} : User card number (secondary card), string data. If the card number is 123456789, Vice Card=123456789.

${LF} is used to connect multiple records.

A normal response message from the server:

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

OK: ${XXX}
```

Annotation:

HTTP status line: Defined with standard HTTP protocol

HTTP response header field:

Content-Length header field: According to the HTTP 1.1, this header field is generally used to specify the data length of the response entity. If the response entity size is uncertain, head fields of Transfer-Encodin g: chunked, Content-Length and Transfer-Encoding are supported, all of which are standard definitions of HTTP and are not described in details here.

Response entity: When the server normally receives data and successfully processes data, OK: ${XXX} is r

eplied. ${XXX} represents the number of records successfully processed. In case of an error, an error desc ription is replied.

Example

```
POST /iclock/cdata?SN=0316144680030&table=OPERLOG&Stamp=9999 HTTP/1.1
Host: 58.250.50.81: 8011
User-Agent: iClock Proxy/1.09
Connection: close
Accept: */*
Content-Length: 166

USER PIN=36234 Name=36234 Pri=0 Passwd= Card=133440 Grp=1 TZ=0001000000000000
USER PIN=36235 Name=36235 Pri=0 Passwd= Card=133441 Grp=1 TZ=0001000000000000
```

A 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

OK: 2
```

11.6 Uploading Identity Card Information

The configuration PushProtVer parameter sent by the server for initialization information exchange is greater than or equals to version 2.3.0.

A request message from the client:

```
POST /iclock/cdata?SN=${SerialNumber}&table=IDCARD&Stamp=${XXX} HTTP/1.1
Host: ${ServerIP}: ${ServerPort}
Content-Length: ${XXX}
……

${DataRecord}
```

Annotation:

HTTP request method: POST method

URI: /iclock/cdata

HTTP protocol version: 1.1

Client configuration information:

SN: ${Required} Serial number of the client

table=IDCARD: ${Required}

Stamp: ${Optional} Latest timestamp at which the identity card information is uploaded to the server. (not used)

Host header field: ${Required}

Content-Length header field: ${Required}

Other header fields: ${Optional}

Request entity: ${DataRecord}, user information data, in the following format:

IDCARD${SP}PIN=${XXX}${HT}SNNum=${XXX}${HT}IDNum=${XXX}${HT}DNNum=${XXX}${HT}Name=${XXX}${HT}Gender=${XXX}${HT}Nation=${XXX}${HT}Birthday=${XXX}${HT}ValidInfo=${XXX}${HT}Address=${XXX}${HT}AdditionalInfo=${XXX}${HT}Issuer=${XXX}${HT}Photo=${XXX}${HT}FPTemplate1=${XXX}${HT}FPTemplate2=${XXX}${HT}Reserve=${XXX}${HT}Notice=${XXX}

Note:

PIN=${XXX} : User ID. If the user's information is not bound to the identity card, then the value of PIN is 0.

SNNum=${XXX} : Physical card number of identity card

IDNum=${XXX} : Citizen id number

DNNum=${XXX} : Identity card serial number (card body management number)

Name=${XXX} : Id Name, using utf-8 encoding

Gender=${XXX} : Gender code

1," male "

2," female"

Nation=${XXX}: Ethnic code

0,"Decoding error"

1," Han"

2," Mongol"

3,"Hui"

4," Tibetan"

5," Uighur"

6,"Miao"

7,"Yi"

8,"Zhuang"

9,"Buyi"

10,"Korean"

11,"Manchu"

12,"Dong"

13,"Yao"

14,"Bai"

15,"Tujia"

16,"Hani"

17,"Kazakh"

18,"Dai"

19,"Li"

20,"Lisu"

21,"Wa"

22,"She"

23,"Gaoshan"

24,"Lahu"

25,"Shui"

26,"Dongxiang"

27,"Naxi"

28,"Jingpo"

29,"Kirghiz"

30,"Du"

31,"Daur"

32,"Mulam"

33,"Qiang"

34,"Blang"

35,"Salar"

36,"Maonan"

37,"Gelao"

38,"Xibe"

39,"Achang"

40,"Pumi"

41,"Tajik"

42,"Nu"

43,"Uzbek"

44,"Russian"

45,"Evenki"

46,"De'ang"

47,"Bonan"

48,"Yugur"

49,"Gin"

50,"Tatar"

51,"Drung"

52,"Oroqin"

53,"Hezhen"

54,"Menba"

55,"Lhoba"

56,"Jino"

57,"Coding error"

97,"Other"

98," Foreign origin"

Birthday=${XXX} : Date of birth (format: yyyyMMdd)

ValidInfo=${XXX} : Period of validity, start date and end date (format: yyyyMMddyyyyMMdd)

Address=${XXX}: Address, encoded in UTF-8

AdditionalInfo=${XXX}: Machine read appends address, encoded in UTF-8

Issuer = ${XXX}: Issuing authority, use UTF-8 encoding.

Photo=${XXX} : Photo data stored by identity card, which is encrypted and converted into base64 data c ontent for transmission.

FPTemplate1=${XXX} : Fingerprint 1_ fingerprint characteristic data, and converted into base64 data content for transmission.

FPTemplate2=${XXX} : Fingerprint 2_ fingerprint characteristic data, and converted into base64 data content for transmission.

Reserve=${XXX}: Reserve field

Notice=${XXX} : Note information, encoded in UTF-8.

${LF} is used to connect multiple records.

A normal response message from the server:

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

OK: ${XXX}
```

Annotation:

HTTP status line: Defined with standard HTTP protocol

HTTP response header field:

Content-Length header field: Based on HTTP 1.1, this header field is usually used to specify the data length of the response entity. If the response entity size is uncertain, head fields of Transfer-Encoding: chunked, Content-Length and Transfer-Encoding are supported, all of which are standard definitions of HTTP and are not described in details here.

Response entity: When the server normally receives data and successfully processes data, OK: ${XXX} is replied. ${XXX} represents the number of records successfully processed. In case of an error, an error description is replied.

Example

A request from the client:

```
POST /iclock/cdata?SN=0316144680030&table=IDCARD&Stamp=9999 HTTP/1.1
Host: 58.250.50.81:8011
User-Agent: iClock Proxy/1.09
Connection: close
Accept: */*
Content-Length: 658

 IDCARD PIN=2 SNNum=xxxxxxxx460088xxxxxx IDNum=xxxxxx19911218xxxx DNNum= Name=Zhang San Gender=1 Nation=1 Birthday=19911218 ValidInfo=2017091520270915 Address= Province xx City xx County xxx Village xxx Group xx AdditionalInfo= Issuer= County xxx public security bureau Photo=V0xmAH4AMgAA/4UYUV+sjnpymK1Boqvz3UCBevbbHnYikGyH1XA7Emt2agF0HFhDc4Bxzeg/jH0Yp8Ngl1861Y812K1AOUIRgy1Z5TEuSG1GV4MwlAB3qY0tKqWNPzyEd8Pn0EhRsgAAjeWP
```

```
xiUzLaPU1w	FPTemplate1=QwGIEgELUQAAAAAAAAAAAAAC9AAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA4	FPTemplate2=QwGIEgEQUAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAD8	Reserve=	Notice=

A 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

OK: 1
```

11.7 Uploading Identity Card Attendance Record

The configuration PushProtVer parameter sent by the server for initialization information exchange is greater than or equal to version 2.4.0.

A request message from the client:

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

Host: ${ServerIP}:${ServerPort}

Content-Length: ${XXX}

……


${DataRecord}
```

Annotation:

```
HTTP request method: POST method

URI: /iclock/cdata

HTTP protocol version: 1.1

Client configuration information:

SN: ${Required} Serial number of the client

table= ATTLOG: ${Required} The uploaded data is the attendance record of identity card.

Stamp: ${Optional} Latest timestamp at which the identity card attendance record is uploaded to the ser
```


ver. (For details, see the “Stamp” or "ATTLOGStamp" parameter in "Initializing Information Exchange".)

Host header field: ${Required}

Content-Length header field: ${Required}

Other header fields: ${Optional}

Request entity: ${DataRecord}, upload identity card attendance record, in the following format

${Pin}${HT}${Time}${HT}${Status}${HT}${Verify}${HT}${Workcode}${HT}${Reserved1}${HT}${Reserved2}${HT}${IDNum}${HT}${Type}

IDNum: Id number

Type: Record Type (0 means attendance, 1 means verification)

The Type value is 0, and the content of the attendance record is defined in accordance with the attendance agreement.

The Type value is 1, STATUS: 0 - success, 1 - failure, 2 – blacklist

VERIFY :1- face, 2- face + fingerprint, 3- fingerprint + face

Other content is defined in accordance with standard protocols.

Example

A request from the client:

```
POST /iclock/cdata?SN=0316144680030&table=ATTLOG&Stamp=9999 HTTP/1.1
Host: 58.250.50.81:8011
User-Agent: iClock Proxy/1.09
Connection: close
Accept: */*
Content-Length: 315

1452   2015-07-30 15:16:28   0   1   0   0   0   210218199105072345   0
1452   2015-07-30 15:16:29   0   1   0   0   0   210218199103062104   0
1452   2015-07-30 15:16:30   0   1   0   0   0   210218199411212642   0
1452   2015-07-30 15:16:31   0   1   0   0   0   210218199207123075   0
1452   2015-07-30 15:16:33   0   1   0   0   0   210218199512012332   0
1452   2015-07-30 15:16:34   0   1   0   0   0   210218199011304365   0
1452   2015-07-30 15:16:35   0   1   0   0   0   210218199806068325   0
8965   2015-07-30 15:16:36   0   1   0   0   0   210218199310094316   0
8965   2015-07-30 15:16:37   0   1   0   0   0   210218199708167443   0
```

A 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

OK: 9
```

11.8 Uploading Identity Card Attendance Photo

The configuration PushProtVer parameter sent by the server for initialization information exchange is greater than or equal to version 2.4.0.

A request message from the client:

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

Host: ${ServerIP}:${ServerPort}

Content-Length: ${XXX}

……


${DataRecord}
```

Annotation:

HTTP request method: POST method

URI: /iclock/fdata or /iclock/cdata

HTTP protocol version: 1.1

Client configuration information:

SN: ${Required} Serial number of the client

table= ATTPHOTO: ${Required}

Stamp: ${Optional} Latest timestamp at which the identity card attendance photo is uploaded to the ser ver. (For details, see the "ATTPHOTOStamp" parameter in "Initializing Information Exchange".)

Host header field: ${Required}

Content-Length header field: ${Required}

Other header fields: ${Optional}

Request entity: ${DataRecord}, upload identity card attendance photo, in the following format:

PIN=${XXX}${LF}SN=${SerialNumber}${LF}size=${XXX}${LF}CMD=uploadphoto${NUL}${BinaryData}

Note:

PIN= time - photo type – User ID - id number.jpg

Photo type:

0: attendance successful photo

1: attendance failed photo

2: blacklist photo

3: verification successful photo

4: verification failed photo

SN=${XXX} : Cient series number

Size =${XXX} : Original size of attendance photo

${BinaryData} : Original image BinaryData stream

Example

A request from the client:

POST /iclock/cdata?SN=0316144680030&table=ATTPHOTO&Stamp=9999 HTTP/1.1
Host: 58.250.50.81:8011
User-Agent: iClock Proxy/1.09
Connection: close
Accept: */*
Content-Length: 1684

PIN=20160615093758-0-1457-210218199011304365.jpg SN=0316144680030 size=9512 CMD=uploadphoto${NUL}${BinaryData}

A 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

OK: 9

11.9 Uploading Fingerprint Template

The configuration PushProtVer parameter sent by the server for initialization information exchange is greater than or equal to version 2.2.14.

A request message from the client:

```
POST /iclock/cdata?SN=${SerialNumber}&table=OPERLOG&Stamp=${XXX} HTTP/1.1
Host: ${ServerIP}: ${ServerPort}
Content-Length: ${XXX}
……

${DataRecord}
```

Annotation:

HTTP request method: POST method

URI: /iclock/cdata

HTTP protocol version: 1.1

Client configuration information:

SN: ${Required} Serial number of the client

table=OPERLOG: ${Required}

Stamp: ${Optional} Latest timestamp at which the fingerprint template is uploaded to the server. (For details, see the "OPERLOGStamp" parameter in "Initializing Information Exchange".)

Host header field: ${Required}

Content-Length header field: ${Required}

Other header fields: ${Optional}

Request entity: ${DataRecord}, fingerprint template data, in the following format:

FP${SP}PIN=${XXX}${HT}FID=${XXX}${HT}Size=${XXX}${HT}Valid=${XXX}${HT}TMP=${XXX}

Note:

Size=${XXX}: Length after base64 coding of the fingerprint template

TMP=${XXX}: When the fingerprint template is transmitted, base64 coding needs to be conducted for the original binary fingerprint template.

${LF} is used to connect multiple records.

A normal response message from the server:

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


OK: ${XXX}
```

Annotation:

HTTP status line: Defined with standard HTTP protocol

HTTP response header field:

Content-Length header field: Based on HTTP 1.1, this header field is usually used to specify the data length of the response entity. If the response entity size is uncertain, head fields of Transfer-Encoding: chunked, Content-Length and Transfer-Encoding are supported, all of which are standard definitions of HTTP and are not described in details here.

Response entity: When the server normally receives data and successfully processes data, OK: ${XXX} is replied. ${XXX} represents the number of records successfully processed. In case of an error, an error description is replied.

Example

A request from the client:

```
POST /iclock/cdata?SN=0316144680030&table=OPERLOG&Stamp=9999 HTTP/1.1
Host: 58.250.50.81: 8011
User-Agent: iClock Proxy/1.09
Connection: close
Accept: */*
Content-Length: 4950

FP PIN=2	FID=0	Size=1124	Valid=1 TMP=SghTUzIxAAADS00ECAUHCc7QAAAnSnkBAAAAg/YUfEsyAI
EPHgH6ALFHRQBBAPkP8wBAS2UPEwBTACYPe0tYAHkIjACuAHdIeQBtAGwEUAB1S20DhAB+AK8EXUuP
AOoPJABwANVENQDCANsPZQDbSx8PbwDeACwPz0vjAJ8PdwArAPFELAD5AMwPvQASSvMKMgAwAQk
PSUE2DkcXQ0uCQ1B4AJT7GZuC3GyNySrvjoKT7X77SkYkB9L6MQhMCV5G1PUR+T0FfPiGTqMABQHp+X
gBhclzg397xf0iD5CkQAXvErv3q4PQZ940xfmXzBb5bcher2e7PQkLAXyf8gJ78nP7iwFIQmrXcwKn31Lfiwo
BIDQBAjrbrAdLOBFwwv8ExVYStv7ABQBOE7+JCEt/GIBs/4SqDwPoJ4yHwMDEBcHDicB/DQCrLkbAwgnFc
nwGAHn2g4iICQCsNoPCnm4RS75CjJ0rgwFqwS4HADFDZmwEAwJcRjrABwA1jWTCtMX+whMAw4+JwY
nC+//Aw3pCwsK0wRIAxFKJB8PAsfzCw3WLwaENAz9besHBi4eqBwOPW4PBwsEoygBxKnvCwcLAwqx4W
FIB02WGwl86boCLw4b/ZMFx3ADXJ4FSi3X/wqt2whjD/wkAc261eMKKigQBG249lwUD0nP9QxEAdbZww
Yt5wcHCwME6wcMuBQCeeANl1gCQMWjC/v+DwL90UbQCAIF7cMLNAJo2CMHA+/xTzQCCy2jA/pPABM
XTh2ZXEQEfnkCn/8OLwWVdSQMB18tDiwsBF8s9RZFcFUvqza3C/3gAwMOMw1lxZ8laxeXM7cFyxMDCpL
vAdCDD/n4HAKMcHsMH/gsAcNreO/z/tfxC/wsAdBvk/bf9/T5KFAAM4KfPxMHFjP/CB8DDi8FzCgBs4p+Ew
hHBGwDr4qQEUcA2wsPBwMF4B8DCEML+wAQAei/wH1wB6/Gma8AFxI2lwIPAg1gYxej46Ut8n8PBwqrB
bgYJEHsO4vk8/cMkBRCIF1NpxhCGe0jBERDTMGxri4/CrGb/FBAaMqiLccDAwMTFAMLAi8PB/8HAA9WLO
gj/DxDPPqtA/8GJtcRCDBDEj55Bi/7JyZMEEdFRQ7T4
FP PIN=2	FID=1	Size=2120	Valid=1 TMP=T3dTUzIxAAAGNDsECAUHCc7QAAAeNWkBAAAAhtlDsjQjA
```

KIPYgDiAHo7NgA1AHAPZgBSNKQPdwBWAOAPVjRYABMPsACiADM76wB8ALkO2QB4NFgPVQCCAP0O2
TSTALoOYABeAEo7TgCdAFMPwQGmNNgNwgCiAH0PnDSpAEQPCgFuAEI5eQCtAEcPIwCpNMoNEgGwA
PkNvTSyADoP+ABxAEY/KgC1ANUPVQC/NMMP7gC8AAwNHTTAAM0O9AAEAEc5JgDCANQO7QDKNN4O
zgDOAH4OHjTQANUODgAdAMc6ZgDgAEsPNgDmNMMOKQDiAKMOEjTlAOUOzgAhAM86QQDmAGUP
RQDoNFIPaQDwAAIP6DT3AD8OBgEzALk6uAD3AMQOCAAHNTQOPwAHAaEPDTUGAUlOtQDMAUY7oQ
AMAU8PEAAJNbQO8gAZAXgPxDQfATsOCgHaAcQ63AApATYOZAAqNToMmAAvAY0M/TQuAcANyQD1
AcY60gA2AT8OaAA/NcELOgA7ASYPlDQ8AWQLJQCEAfQ7hQBEAd8OKwBCNb8ORABMASoPdDROAeUO
nACUAco5Lv0jb0N/vw8xbVZpOl8G9+JPA0Zbg6sF2hafgw2db/Kq3hanMJLdJ1p/HQLu48rxWbIqIj8Lee9U
5wASmAL+8DIEdPjjI1MEKQj1H1OD0qPInf2aGYFzANzakIW6/UoNkQruW6gFTYfW/PMF5zDAjd2O3PosZb
vLGlVugNb5YXuLNMx1+QARCTT6k7VQf7qL1lpZi2c0ufv9+NHwGXoLNOR66PcwC7wAWjsUDD0UBfargL
TBFRd1CKH4be/q65T3cYRJ+3fqD7LgB4KAmYEciic2SSdlEZmBvBlfM/iXpQJeAHuJr7rAC04HdXxwdkBTDH
5Bglr9MH1HtmT/BYXRkmT/c7TgcsH23gAPAw2o9IWBgqaEzH27xWgGLRFyFNOMc7QwfgGD9f4l/i+/dH0
pivH+2AaHtkgGpfi9efR7vsJofaXaLA5M5Eu+oAJxBhkGHYN3sbSN2IMdCCwHZ7FUej1fyfXU8mW17ByJhE
maQBKYMGKTkXwN/XAXnzZXgG6IPQ3n/JE4hAmKgZ4bpGpP24wPjfRRa9Hpz8AEIEwBAungoQU0ZwUA
wAwAaAcidMBEwDgKAH0IIctEQv4SAHzOEzP0/0NAW8L9xgCfPx3/CgDDC+L/U8tKDQBnDAk4RvgMw/3A
FgCNyRcwH1VYVDyfB8XQChnBVf8SADfU/cbK/v7/O8H8BcL4y0gDAOQkLToLBgMb/TwwQQnFuCAZS1X
+CwAm4vdKyf/AKwUAX+2AdiEBUS4A/yeD/VdyZWADADMztMAeNFc1APwwMDvA+WdZW//AxQTFND9
ZZAgApVcwOlXG9AYAelgg/ur/EzRUWRD+/v31wDEHTcE4GgADm+lV9Pz9wP0wKTpDxnhoEAAHaOL9O/
vK/v7+/jQHxAVuZ5X9wgUAsaw0TCYBCHLkR/4FISx1Oh8AAX7wBkcozy8wRTbC/j7CQBgCARCAVsHBABy
0X40EAOuAg3YANEWDaZ3CCMVYjwP6RjYEAD5KXI4wAUSPVlYFxUqJePxPHQAEkRL++GP8+/4j/v+O/8fK
/cFW/ykOxd+QDv5Xc8DCwJULBmufSTj//cDqBQZ/nlNBBwBQZEz4y/8yBADDpvHA+CQBl6tDUlE+wccHw
fsUAJyrhVVdyv3E///9/wX7+2MCAQevRsHDAQqbQf9SBgB4dUZD9AsAv7U3QQXAxUQEALq2QGjdAB+Jx/
7///7+0vz4HcH+/8H9/wX++TcBCL9AwQXF7cZ0/sLBCAD9AkD79Xb/IAACygUoxsvB/f/9+/04/fjK/v3AwP//
Bf3E9MH9/v3+HcUP0PfD/MD+//86/fvP/SErS//AOz74MAAS3El9BMUl510xBgD/40AH/Hk/AWfmUP5VhCIA
NN3nQMDBV8AAOt5j/0AKAIA0UEV0/P0GAOv6g8RAMRE9CmLAI8EQtDhCcwkQuAzyQPvJ+/8FEJwPlf5w
MxHWEzf9wgT/wjcQFRhQwQbV0xxyY/0DEMMihcEFJNwvNMEDEBY5RvUEEJVBVvmTBRY5Tf1RChBCpw
DG9MHAwMBKAA==
FP PIN=3 FID=0 Size=1592 Valid=1 TMP=TetTUzIxAAAEqKsECAUHCc7QAAAcqWkBAAAAhFUooagrA
KAPQADpAGKnzgAuAKkPtQA1qIsPnQBGAO0PTqhKAGIPlgCrADanNwByAEIPngB6qDQOfgCLAGENJKiN
AD0PiABTAKimdACXAB0OVgCmqKwOigCnAFgOtqi0AKsPiAByAJimyQC3AKsPjQDEqKwPngDBAFIPrqjD
AKIPaQAPABGnrwDQAIwPAQDQqJsP6wDUAF4PXKjgAJ8PwgAhAIOm2QDrAIwPTgDrqJIPQwD8AFYPdqj
/AJQP9gDBAYmnugAGAXwPrwANqZMP6QAJAUYOWagTAYsP+AD2AWqmVAA0AX8PUQA8qYAP6ABD
AbIPKdPn+18ZTgTm3GePXwtHgjd7mih7jAplJYui+i6h1lRzIwfYrpLTDpojnwVODhLjqxleOdv2Afg5iMsIJlU
6C2uTeQgwABW4PHhhcX2DhRNxqCgS0PMpAGTwKaDYAkEWqQB88vGuWAIdE0oIJ5kGuRof7PcJDhDuZ
KcUGLX4xXs/aVLRqOyV9v326BRpuqzzQhBmApMXuSc/DZL1wezA/p1XgAbS8Jr0nANWq9IlCgnK9TsCQa
xMC9kDZQxXAjelz/Xm8gv4DABRoH8C8fiq+fcX2VDY+5L5dfoD79dlz/XX7LfvZvOHr94H6QrL9xbvuTys+2
sgRAHHpSVRDQCpBCtrlmTEpgGQBxw4/55qYK0BcQgP/zHBAIWgEScTAF0Jxvz7V/5HwP5rZ5QMBDYIHv5
KwGKRBAQSCydZBQDDzifE7wUAtQ0nYNMAQLgBwP3+QV6hUl30CgAlJ+D8O/3EVf4wFwAqKiL9+Vf/MT
hTZcAEwY+gAWwsj5LDQAYElSxndMINAG0tl/1j/8L/XQzFbDYhwJ9pwnsFxdI2j1MJAHQzHoNU+64BRkpr
wsS2DwTmS/38//wwkMPGxgwAnEorwjv/YGjAwYQOAEmKZ8ckksLBwsDAzwCxwSj/wVjCcdQAU8VWxM
PCwcIHw/tqwMDC/UsMxTlqfif9/MD9/qoMBPRuT8LDxMBYwWiuAZdwMFLAwwBl3UHDw8DDB8Uzcu6R
wsIGADmzQ8UgCQBleCfDiWEfqA6Fyf////T++VX+/f7/wP4F/8RowItvBgBaQDTGIAwAe4ipwjvIzW2hfhUAF
o8DQfpX//39/S7/7nwIqByPQ8LCiAV+xKABI489jMK2BAQdkDSDFQCIV6v7IMfHw8TCwgb/xtTAhQUAdZv
nw/poCgAVoD2SuWQDqLe5IHSFCsURv5jCacHB/sPGAKZrFsAGAEzE4cF+rwFsyRPAhjsGBKbLKcLAiAnFa8
q2wv/Cw1MGxcjSv8HCRg0AWyEXftZHSQQA3O3KWQ2ojvADaEf/xgBDVRLABRAnAMxHArh3AAl+wATVv
wKoQAUQKQgTAPzGrhFuCQZpwMgQ/7qHdsDCQ/4FwRW4LhYJwf/COsJUaMNDQBAQOuYJdFfAZmb+Y
AjV/C7YM2QJEQMtk/77Vf//UgUQWfH9UKcR9TdwLcCeUm2vEZA7fVJdyRDg7nvA/2jC/gY/Drg7XAPAbcI7
/cb6QwALQwAADKcPCQ==

A 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


OK: 3
```

11.10 Uploading Face Template

The configuration PushProtVer parameter sent by the server for initialization information exchange is larger than or equal to version 2.2.14.

A request message from the client:

```
POST /iclock/cdata?SN=${SerialNumber}&table=OPERLOG&Stamp=${XXX} HTTP/1.1
Host: ${ServerIP}: ${ServerPort}
Content-Length: ${XXX}
……

${DataRecord}
```

Annotation:

HTTP request method: POST method

URI: /iclock/cdata

HTTP protocol version: 1.1

Client configuration information:

SN: ${Required} Serial number of the client

table=OPERLOG: ${Required}

Stamp: ${Optional} Latest timestamp at which the face template is uploaded to the server. (For details, see the "OPERLOGStamp" parameter in "Initializing Information Exchange".)

Host header field: ${Required}

Content-Length header field: ${Required}

Other header fields: ${Optional}

Request entity: ${DataRecord}, face template data, in the following format:

FACE${SP}PIN=${XXX}${HT}FID=${XXX}${HT}SIZE=${XXX}${HT}VALID=${XXX}${HT}TMP=${XXX}

Note:

SIZE=${XXX}: Length after base64 coding of the face template

TMP=${XXX}: When the face template is transmitted, sixteen bytes (of random content) need to be added as the prefix of the original binary face template before base64 coding is conducted.

${LF} is used to connect multiple records.

A normal response message from the server:

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

OK: ${XXX}
```

Annotation:

HTTP status line: Defined with standard HTTP protocol

HTTP response header field:

Content-Length header field: Based to the HTTP 1.1, this header field is generally used to specify the data length of the response entity. If the response entity size is uncertain, head fields of Transfer-Encoding: chunked, Content-Length and Transfer-Encoding are supported, all of which are standard definitions of HTTP and are not described in details here.

Response entity: When the server normally receives data and successfully processes data, OK is sent: ${XXX} is replied. ${XXX} represents the number of records successfully processed. When an error occurs, the error description is replied.

Example:

A request from the client:

```
POST /iclock/cdata?SN=0316144680030&table=OPERLOG&Stamp=9999 HTTP/1.1
Host: 58.250.50.81: 8011
User-Agent: iClock Proxy/1.09
Connection: close
Accept: */*
Content-Length: 1684

FACE PIN=306    FID=2   SIZE=1648   VALID=1 TMP=AAAAAAAAAAAAAAAAAAAAAFpLRmlYATFLFLToAU
QBQ1Mg+fgXuia23BDrNtwSfgJ8g74H3YHmXlkFpgetB5eH5yXuBvMLoa6wSx9HNgK7RP80v1i+LLY8nCn
7PXmD7w15Bp8N1wm/A78PowejZx9jJyWnBZ88K5wVfDDcNTjifGlvox9iD8sf1g37B70Fk4WRl5RrKq8uD2
MngRexMxk5cbDiH+c3xj+CV8Zf1idaDfWbkB8Rnwt/AV8Du0SvAddBywHMQ9MVysfFkNENfZD9FJ9jrnGe
BD5Kcwp7CVySfJzOE+wZxjWFVY6fgreXHBd6B4ov4BXBX+GuIZ4pazBTlNiG8kf4h/DHxGaFxYe+yh+O1slC
DsPcweuB/SHnA+UnqwG3AvnA88DZg/vhmaaV4dsWzwerBn1jLcN8wu/ErITiR+YHVsc1wy2wdaC5uEmxK
bwZ+AGeB5fFt6WLVa8kq/gvqqv8LvwsBAACAgIHAQABAAEGAUxyAQkcIP9SAAohGhchAQAAEAEAAQYF
ASBPAgYGB1YKAQEMBgAACg8HFQ0DDAoEAg8CCBQDAxtLDwM40CUFEBNVSQYDDRNJJg8CAAcJFBM
MAQQiYA8MAwhZHAcEehMCACYEAAwACQsJBQAFBAYxAAknpwQGJ6EiBAUPIxwFBwQPOgQCBAARGz
8WCAIAGUQWBggLhyMIBJQIBAIiAgUEAgMPAwUACQAFUAABE/8EBwkQEgABCRECBwQFFEYNCAcCFiJ3
```

YwUACTpKLgwBDn8yDApjFAIBEwAAAwACBgEAAAECAUMBAwBCAAACAQIBAAEABAMBAQUEAQMCBR
0x/z4CAhUhSk8GAAcIDhUBNgsLDQMBAQAAAAYDAAAAAAEEAQAHBwEGQWEIAAJe/zQBAAYY/30GAA
ABAAICAgAABAMAAAEFFAgAAAoEAAEUAgABAQEABwEECQMWGAIBDSIHASH/HQQIClQeBgYMCgsIAw
EBDSB0IBEJAjLbQgkIB245BQYKCAAEAAEDAwEDDgcKAQIBAhwABB+4DAQOoiAFCAgXFwMBByNDCgYB
ABUQGSYDAwItYhMIATG9KQwLV0oCBBkEAQgBDggHAQADBwJDAxAS6AIGFxYXBgMiRxQNCgMXvCMO
BQIXGrxABgkGJQ4YBQEQHxUJBUgNCgggAwIBAgMKAAAAAgMCJwADAzUAAAEAAwAAAB4BAAEAAAA
AAAAAA1j/JgQGAxf/fAMBAgkLAAEEAgIAAAAEAAEAAwEAAAABAAQAAAAAAAAABAAAAARgQAQABAjP/
KQMAAAIDAQEBBAIEBAoCAAoUAgMBRAYCACoBAQAAAAsDAgAAAgEJAAMDHgABEiwNAQQnRA8DAw
c5lDICAQAIETAbAAEGM00eAQApYCIKA/81CwU5DAIGAAEEBQQABQACKAAGG2MEAxkpFQMBCTkKAAU
CFSYJAQABDByEGwIBAjUrIQUDO/8eCwSMRAQACAQABQEABgAAAAMAAA8AAAMuAAAAAwgCBQMFC
QcCBBNECgMAAAQvZhABAAhZ/DIBACL/XQgDJiEHAgwFAQEAAAcDAAAAAQEDAAACAAAAAwEAAgEF
CQEBBAEHIAIAAQELFRQEAwICS/8rBQEDRVQUBAYODwQAAAgAAAANAAAAAAAA

A 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

OK: 1

11.11 Uploading Finger Vein Template

The configuration PushProtVer parameter sent by the server for initialization information exchange is larger than or equal to version 2.2.14.

A request message from the client:

```
POST /iclock/cdata?SN=${SerialNumber}&table=OPERLOG&Stamp=${XXX} HTTP/1.1
Host: ${ServerIP}: ${ServerPort}
Content-Length: ${XXX}
……

${DataRecord}
```

Annotation:

HTTP request method: POST method

URI: /iclock/cdata

HTTP protocol version: 1.1

Client configuration information:

SN: ${Required} Serial number of the client

table=OPERLOG: ${Required}

Stamp: ${Optional} Latest timestamp at which the face template is uploaded to the server. (For details, see the "OPERLOGStamp" parameter in "Initializing Information Exchange".)

Host header field: ${Required}

Content-Length header field: ${Required}

Other header fields: ${Optional}

Request entity: ${DataRecord}, face template data, in the following format:

FVEIN${SP}Pin=${XXX}${HT}FID=${XXX}${HT}Index=${XXX}${HT}Size=${XXX}${HT}Valid=${XXX}${HT}Tmp=${XXX}

Note:

Pin=${XXX} : User ID

FID=${XXX} : Finger number, (0~9)

Index=${XXX} : One finger has multiple finger vein templates, and Index is the number of finger vein template (0~2).

SIZE=${XXX}: Length after base64 coding of the finger vein template binary data

Valid=${XXX} : Valid identification of the finger vein template, the values are as follows:

Value	Description
0	invalid template
1	normal template

Tmp=${XXX}: Base64 encoding of the original binary finger vein template is needed when transferring the finger vein template.

${LF} is used to connect multiple records.

A normal response message from the server:

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

OK: ${XXX}
```

Annotation:

HTTP status line: Defined with standard HTTP protocol

HTTP response header field:

Content-Length header field: Based to the HTTP 1.1, this header field is generally used to specify the data length of the response entity. If the response entity size is uncertain, head fields of Transfer-Encoding: chunked, Content-Length and Transfer-Encoding are supported, all of which are standard definitions of HTTP and are not described in details here.

Response entity: When the server normally receives data and successfully processes data, OK is sent: ${XXX} is replied. ${XXX} represents the number of records successfully processed. When an error occurs, the error description is replied.

Example:

A request from the client:

POST /iclock/cdata?SN=0316144680030&table=OPERLOG&Stamp=9999 HTTP/1.1

Host: 58.250.50.81:8011

User-Agent: iClock Proxy/1.09

Connection: close

Accept: */*

Content-Length: 1698

FVEIN Pin=306 FID=2 Index=0 Size=1648 Valid=1 Tmp=AAAAAAAAAAAAAAAAAAAAAFpLRmlYA
TFLFLToAUQBQ1Mg+fgXuia23BDrNtwSfgJ8g74H3YHmXlkFpgetB5eH5yXuBvMLoa6wSx9HNgK7RP80v1i
+LLY8nCn7PXmD7w15Bp8N1wm/A78PowejZx9jJyWnBZ88K5wVfDDcNTjifGIvox9iD8sf1g37B70Fk4WRl5
RrKq8uD2MngRexMxk5cbDiH+c3xj+CV8Zf1idaDfWbkB8Rnwt/AV8Du0SvAddBywHMQ9MVysfFkNENfZD
9FJ9jrnGeBD5Kcwp7CVySfJzOE+wZxjWFVY6fgreXHBd6B4ov4BXBX+GuIZ4pazBTINiG8kf4h/DHxGaFxYe
+yh+O1slCDsPcweuB/SHnA+UnqwG3AvnA88DZg/vhmaaV4dsWzwerBn1jLcN8wu/ErITiR+YHVsc1wy2w
daC5uEmxKbwZ+AGeB5fFt6WLVa8kq/gvqqv8LvwsBAACAgIHAQABAAEGAUxyAQkcIP9SAAohGhchAQA
AEAEAAQYFASBPAgYGB1YKAQEMBgAACg8HFQ0DDAoEAg8CCBQDAxtLDwM40CUFEBNVSQYDDRNJJg
8CAAcJFBMMAQQiYA8MAwhZHAcEehMCACYEAAwACQsJBQAFBAYxAAknpwQGJ6EiBAUPIxwFBwQPOg
QCBAARGz8WCAIAGUQWBggLhyMIBJQlBAIiAgUEAgMPAwUACQAFUAABE/8EBwkQEgABCRECBwQFFEY
NCAcCFiJ3YwUACTpKLgwBDn8yDApjFAIBEwAAAwACBgEAAAECAUMBAwBCAAACAQIBAAEABAMBAQ
UEAQMCBR0x/z4CAhUhSk8GAAcIDhUBNgsLDQMBAQAAAAYDAAAAAAEEAQAHBwEGQWEIAAJe/zQBA
AYY/30GAAABAAICAgAABAMAAAEFFAgAAAoEAAEUAgABAQEABwEECQMWGAIBDSIHASH/HQQIClQeB
gYMCgsIAwEBDSB0IBEJAjLbQgkIB245BQYKCAAEAAEDAwEDDgcKAQIBAhwABB+4DAQOoiAFCAgXFwM
BByNDCgYBABUQGSYDAwItYhMIATG9KQwLV0oCBBkEAQgBDggHAQADBwJDAxAS6AIGFxYXBgMiRxQN
CgMXvCMOBQIXGrxABgkGJQ4YBQEQHxUJBUgNCgggAwIBAgMKAAAAAgMCJwADAzUAAAEAAwAAAB
4BAAEAAAAAAAAAA1j/JgQGAxf/fAMBAgkLAAEEAgIAAAAEAAEAAwEAAAABAAQAAAAAAAABAAAAAR
gQAQABAjP/KQMAAAIDAQEBBAIEBAoCAAoUAgMBRAYCACoBAQAAAAsDAgAAAgEJAAMDHgABEiwNA
QQnRA8DAwc5lDICAQAIETAbAAEGM00eAQApYCIKA/81CwU5DAIGAAEEBQQABQACKAAGG2MEAxkpF
QMBCTkKAAUCFSYJAQABDByEGwIBAjUrIQUDO/8eCwSMRAQACAQABQEABgAAAAMAAA8AAAMuAAA
AAwgCBQMFCQcCBBNECgMAAAQvZhABAAhZ/DIBACL/XQgDJiEHAgwFAQEAAAcDAAAAAQEDAAACAA

AAAwEAAgEFCQEBBAEHIAIAAQELFRQEAwICS/8rBQEDRVQUBAYODwQAAAgAAAANAAAAAAAA

A 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

OK: 1

11.12 Uploading Unified Templates

If the PushProtVer is greater than or equal to 2.2.14 in configurations distributed by the server, a unified format should be used for the uploading or downloading of new biological identification templates. The Type in data is used to identify the type of biological identification templates. The unified format applies to the palm template among others.

Request message from the client

POST /iclock/cdata?SN=${SerialNumber}&table=BIODATA&Stamp=${XXX} HTTP/1.1

Host: ${ServerIP}:${ServerPort}

Content-Length: ${XXX}

……

${DataRecord}

Note:

HTTP Request Method: POST method

URI: /iclock/cdata

HTTP Version: 1.1

Client Configuration Information

SN: ${Required} represents the series number of the client.

table=BIODATA: ${Required}

Stamp: ${Optional} represents the latest time stamp for the delivery of a unified template to the server (unavailable).

Host Header Field: ${Required}

Content-Length Header Field： ${Required}

Other Header Field: ${Optional}

Request Entity: ${DataRecord}, data about the unified templates in the following data format:

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}

Note:

Pin=${XXX}: Employee No.

No=${XXX}: Number of specific biological individual, 0 by default.

[Fingerprints] No.: 0 to 9, corresponding to the little finger/the fourth finger/the middle finger/the index finger/the thumb on the left hand, and the thumb/the index finger/the middle finger/the fourth finger/the little finger on the right hand.

[Finger Vein]: The same as [Fingerprints].

[Face]: 0

[Irises]: 0 for the left eye and 1 for the right eye.

[Palms]: 0 for the left hand and 1 for the right hand.

Index=${XXX}: Template No. of a specific biological individual, for example, multiple templates stored for a finger that counts from 0.

Valid=${XXX}: Identifier of validity, 0: Invalid and 1: Valid, with 1 as the default.

Duress=${XXX}: Identifier of duress, 0: Under no duress and 1: Under duress, with 0 as the default.

Type=${XXX}: Type of biological identification

Value Meaning

0 Universal

1 Fingerprint

2 Face

3 Voiceprint

4 Iris

5 Retina

6 Palmprint

7 Finger vein

8 Palm

9 Visible light face

MajorVer=${XXX}：For example, for the fingerprint algorithm version 10.3, the major version is 10 and th e minor version is 3.

[Fingerprints]: 9.0 , 10.3 and 12.0.

[Finger Vein]: 3.0

[Face]: 5.0, 7.0 and 8.0

[Palms]: 1.0

MinorVer=${XXX}：For example, for the fingerprint algorithm version 10.3, the major version is 10 and t he minor version is 3.

[Fingerprints]: 9.0 and 10.3.

[Finger Vein]: 3.0

[Face]: 5.0, 7.0 and 8.0

[Palms]: 1.0

Format=${XXX}: Template format, for example, the ZK\ISO\ANSI format for fingerprints.

[Fingerprints]

Value Format

0 ZK

1 ISO

2 ANSI

[Finger Vein]

Value Format

0 ZK

[Face]

Value Format

0 ZK

[Palms]

Value Format

0 ZK

Tmp=${XXX}: Template data, with base64 encoding required for raw binary fingerprint templates.

${LF} is used to connect multiple entries.

Normal response message from the server

HTTP/1.1 200 OK

Content-Length: ${XXX}

……

OK:${XXX}

Note:

HTTP Status Line: The standard HTTP definition is used.

HTTP Response Header Field:

Content-Length Header field: According to HTTP 1.1, the data length of the specified response entity in t he header field is usually used. If the length of the response entity is uncertain, Transfer-Encoding: chunk ed, Content-Length and Transfer-Encoding are also supported, whose header fields are all in compliance with the standard HTTP definition and require no elaboration here.

Response Entity: When data is received normally and processed successfully by the server, OK:${XXX} is r eturned, with ${XXX} representing the number of successfully processed record entries. When an error o ccurs, error description is simply returned.

Example

Request from the client:

POST /iclock/cdata?SN=0316144680030&table=BIODATA&Stamp=9999 HTTP/1.1

Host: 58.250.50.81:8011

User-Agent: iClock Proxy/1.09

Connection: close

Accept: */*

Content-Length: 1736

BIODATA Pin=306 No=0 Index=2 Valid=1 Duress=0 Type=8 MajorVer=1 MinorVer=0 Format=0 Tm p=AAAAAAAAAAAAAAAAAAAAAFpLRmlYATFLFLToAUQBQ1Mg+fgXuia23BDrNtwSfgJ8g74H3YHmXlkF pgetB5eH5yXuBvMLoa6wSx9HNgK7RP80v1i+LLY8nCn7PXmD7w15Bp8N1wm/A78PowejZx9jJyWnBZ88 K5wVfDDcNTjifGIvox9iD8sf1g37B70Fk4WRl5RrKq8uD2MngRexMxk5cbDiH+c3xj+CV8Zf1idaDfWbkB8Rn wt/AV8Du0SvAddBywHMQ9MVysfFkNENfZD9FJ9jrnGeBD5Kcwp7CVySfJzOE+wZxjWFVY6fgreXHBd6B4o v4BXBX+GuIZ4pazBTINiG8kf4h/DHxGaFxYe+yh+O1slCDsPcweuB/SHnA+UnqwG3AvnA88DZg/vhmaaV4 dsWzwerBn1jLcN8wu/ErlTiR+YHVsc1wy2wdaC5uEmxKbwZ+AGeB5fFt6WLVa8kq/gvqqv8LvwsBAACAgI

HAQABAAEGAUxyAQkcIP9SAAohGhchAQAAEAEAAQYFASBPAgYGB1YKAQEMBgAACg8HFQ0DDAoEAg
8CCBQDAxtLDwM40CUFEBNVSQYDDRNJJg8CAAcJFBMMAQQiYA8MAwhZHAcEehMCACYEAAwACQsJB
QAFBAYxAAknpwQGJ6EiBAUPIxwFBwQPOgQCBAARGz8WCAIAGUQWBggLhyMIBJQlBAIiAgUEAgMPAw
UACQAFUAABE/8EBwkQEgABCRECBwQFFEYNCAcCFiJ3YwUACTpKLgwBDn8yDApjFAIBEwAAAwACBgE
AAAECAUMBAwBCAAACAQIBAAEABAMBAQUEAQMCBR0x/z4CAhUhSk8GAAcIDhUBNgsLDQMBAQAAA
AYDAAAAAAEEAQAHBwEGQWEIAAJe/zQBAAYY/30GAAABAAICAgAABAMAAAEFFAgAAAoEAAEUAgAB
AQEABwEECQMWGAIBDSIHASH/HQQICIQeBgYMCgsIAwEBDSB0IBEJAjLbQgkIB245BQYKCAAEAAEDAwE
DDgcKAQIBAhwABB+4DAQOoiAFCAgXFwMBByNDCgYBABUQGSYDAwItYhMIATG9KQwLV0oCBBkEAQg
BDggHAQADBwJDAxAS6AIGFxYXBgMiRxQNCgMXvCMOBQIXGrxABgkGJQ4YBQEQHxUJBUgNCgggAwIB
AgMKAAAAAgMCJwADAzUAAAEAAwAAAB4BAAEAAAAAAAAAA1j/JgQGAxf/fAMBAgkLAAEEAgIAAAAE
AAEAAwEAAAABAAQAAAAAAAABAAAAARgQAQABAjP/KQMAAAIDAQEBBAIEBAoCAAoUAgMBRAYCAC
oBAQAAAAsDAgAAAgEJAAMDHgABEiwNAQQnRA8DAwc5lDICAQAIETAbAAEGM00eAQApYCIKA/81Cw
U5DAIGAAEEBQQABQACKAAGG2MEAxkpFQMBCTkKAAUCFSYJAQABDByEGwIBAjUrIQUDO/8eCwSMRA
QACAQABQEABgAAAAMAAA8AAAMuAAAAAwgCBQMFCQcCBBNECgMAAAQvZhABAAhZ/DIBACL/XQg
DJiEHAgwFAQEAAAcDAAAAAQEDAAACAAAAAwEAAgEFCQEBBAEHIAIAAQELFRQEAwICS/8rBQEDRVQU
BAYODwQAAAgAAAANAAAAAAAA

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

OK:1

11.13 Uploading User Photo

The configuration PushProtVer parameter sent by the server for initialization information exchange is greater than or equal to version 2.2.14.

A request message from the client:

```
POST /iclock/cdata?SN=${SerialNumber}&table=OPERLOG&Stamp=${XXX} HTTP/1.1
Host: ${ServerIP}: ${ServerPort}
Content-Length: ${XXX}
……
```

${DataRecord}

Annotation:

HTTP request method: POST method

URI: /iclock/cdata

HTTP protocol version: 1.1

Client configuration information:

SN: ${Required} Serial number of the client

table=OPERLOG: ${Required}

Stamp: ${Optional} Latest timestamp at which the user photo is uploaded to the server. (For details, see t he "OPERLOGStamp" parameter in "Initializing Information Exchange".)

Host header field: ${Required}

Content-Length header field: ${Required}

Other header fields: ${Optional}

Request entity: ${DataRecord}, fingerprint template data, in the following format:

USERPIC${SP}PIN=${XXX}${HT}FileName=${XXX}${HT}Size=${XXX}${HT}Content=${XXX}
Note:

FileName=${XXX}: Filename of the user photo, with only the jpg format supported currently.

Content=${XXX}: When the user photo is transmitted, base64 coding needs to be conducted for the origi nal binary user photo.

Size=${XXX}: Length of the user photo after base64 coding.

${LF} is used to connect multiple records.

A normal response message from the server:

HTTP/1.1 200 OK

Content-Length: ${XXX}

……

OK: ${XXX}

Annotation:

HTTP status line: Defined with standard HTTP protocol

HTTP response header field:

Content-Length header field: Based on the HTTP 1.1, this header field is usually used to specify the data l ength of the response entity. If the response entity size is uncertain, head fields of Transfer-Encoding: ch unked, Content-Length and Transfer-Encoding are supported, all of which are standard definitions of HT TP and are not described in details here.

Response entity: When the server normally receives data and successfully processes data, OK is sent: ${X XX} is replied. ${XXX} represents the number of records successfully processed. When an error occurs, the error description is replied.

Example:

```
POST /iclock/cdata?SN=0316144680030&table=OPERLOG&Stamp=9999 HTTP/1.1
Host: 58.250.50.81: 8011
User-Agent: iClock Proxy/1.09
Connection: close
Accept: */*
Content-Length: 1684

USERPIC PIN=123 FileName=123.jpg    Size=10 Content=AAAAAAAAAA……

A 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

OK: 1
```

11.14 Uploading Data Packets

The PushProtVer is greater than or equal to 2.2.14 in configurations distributed by the server

Request message from the client

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

Host: ${ServerIP}:${ServerPort}

Content-Length: ${XXX}

……
```

${DataPack}

Note:

HTTP Request Method: POST method

URI: /iclock/cdata

HTTP Version: 1.1

Client Configuration Information:

SN: ${Required} represents the series number of the client.

table=OPERLOG: ${Required}

ContentType: Entity data format, which currently supports the following

${Value} Meaning

tgz tgz as the compressed format of data packets

Host Header Field: ${Required}

Content-Length Header Field: ${Required}

Other Header Field: ${Optional}

Request Entity:${DataPack}. For the data format of data packets, refer to the format of other data types.

Multiple entries are connected with {LF} and then, packaged. For example,

Package the following data to transmit as entity data:

USER PIN=1 Name= Pri=0 Passwd=0 Card=89776433 Grp=1 TZ=0001000100000000 Verify=-1 ViceCard=123456789

FP PIN=1 FID=1 Size=1336 Valid=1 TMP=SqFTUzIxAAAD4uUECAUHCc7QAAAb42kBAAAAgw8hXuIvAPwP0ADwAI7tjwBKAH4PlgBU4vcPfABhAMcPKOJiAOsPPACmAGbtZABlAHEPdgBq4hEPzwB0AFUPiuJ/AAwPOABAAOvtTACYAF0PkQCi4uwPmgCiANgP1+KzACgPdwAMAFLtiADJADIPlgDP4kwPtADTAPQPdeLbADsPZQAhAELt6QD0AC4PQAD64rsP0wAFAfgPyeIiAb8PvgDhAUHssQAxAUgOXwBH48EONwBFAYkOaeJLAa0OVgCOASrsYgBOAZcOCwvAbJqCS383/64TXp0icG8MpYY8kwHt+JJZh674UljJ54v3nXwJc6d9rpwIcpIOgoAaBAx2g4Ava7MH4wylEA+fuY9ehY/9S5HAd2MROJPzBtVBExK278PNfvjk7RabcRgxIU8Q0gOw+iIXWgz3/UX02wMzCP8WzOyyHygKKQnm9N/wuhGqcuv/doGbeohqKBOSgJ79+X8tmo+EPIeh/d8FGuuciyaIpXtigDz2PyQ+tzuh+RpSZ9LsYJdVfSfc1wx1c3JPtXyRTgPCNgECMx1NwwBp6nx1wAMAmckGw+sBhg6AdIsECQOCK/dAVMARxd8vccHBhcLAcATCwrgHAFkwd8SXCwODMf1FQMBW1ADd1JL/w2fAeEbAUOoBikeAwJacCQNwSgAv/j0FxYpOn3kKAIBdAPjA/RxtFADyXZ4HwMAdwX/AhH/CQwcDz1/pwf3CJ84Ag90C/8E2O8HNAGKBdpbAQQsARWQDHTTAwP9EBcU6ZYnDwMIEAGGscYf2AfF7l//BQIlpn1v/CwCNfMb9RSJBwAwAhX1FkMIjkXYNAIyCzDVXHUrBEQA8hSI4LSMuwP3BQhLFUZYL/v79/f/B7i9WIwkASZhiwU58BOJMnFzAhMHVAFd/5ST+wP/+O8E1ogcAlqCXngEMA3ygFjY1/8AFwAviUaVed8FcygCfRB3//8A4/4r/wh8HANW0JEwFGAPzw9dU//7AO8H+oz//PsD+/8oAVSrd/v36l/06/8Mdwf8HAE/NloNp6AFzzVOdwKPACeKNzS0pc8A6BwO20UzBcf8DxX/X1v8HALXWLTo+w+gBdN5Aa3Q6xArie943VMB7zQBmCEJpcRgQMMLDPaL//v3+/P84/0Yc/EcGENQI/0DA9RE5G70+NTvA/xz9/f3///06wfwd/gQQyic9oAMTWClA/wMQwO06/eERPDc9/xfVaUZ

```
Swf4+wP3AOPv1GzRzwWwaEKNLpNPBPv7A/P4/+v0f/MD+wMDAOv/95hFRVwmHA9XPW5/BAAUAwe/xwA==

 USER PIN=2 Name= Pri=0 Passwd=0 Card=89776433 Grp=1 TZ=0001000100000000 Verify=-1 ViceCard=123456789

 USER PIN=3 Name= Pri=0 Passwd=0 Card=89776433 Grp=1 TZ=0001000100000000 Verify=-1 ViceCard=223456789

 USER PIN=4 Name= Pri=0 Passwd=0 Card=89776433 Grp=1 TZ=0001000100000000 Verify=-1 ViceCard=323456789

 USER PIN=5 Name= Pri=0 Passwd=0 Card=89776433 Grp=1 TZ=0001000100000000 Verify=-1 ViceCard=423456789
```

Normal response message from the server

```
HTTP/1.1 200 OK

Content-Length: ${XXX}

……


OK:${XXX}
```

Note:

HTTP Status Line: The standard HTTP definition is used.

HTTP Response Header Field:

Content-Length Header field: According to HTTP 1.1, the data length of the specified response entity in the header field is usually used. If the length of the response entity is uncertain, Transfer-Encoding: chunked, Content-Length and Transfer-Encoding are also supported, whose header fields are all in compliance with the standard HTTP definition and require no elaboration here.

Response Entity: When data is received normally and processed successfully by the server, OK:${XXX} is returned, with ${XXX} representing the number of successfully processed record entries. When an error occurs, error description is simply returned.

11.15 Uploading Comparison Photo

The configuration PushProtVer parameter sent by the server for initialization information exchange is greater than or equal to version 2.2.14.

A request message from the client:

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

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

Content-Length: ${XXX}

……

${DataRecord}

Annotation:

HTTP request method: POST method

URI: /iclock/cdata

HTTP protocol version: 1.1

Client configuration information:

SN: ${Required} Serial number of the client

table=OPERLOG: ${Required}

Stamp: ${Optional} Latest timestamp at which the comparison photo is uploaded to the server. (For details, see the "OPERLOGStamp" parameter in "Initializing Information Exchange".)

Host header field: ${Required}

Content-Length header field: ${Required}

Other header fields: ${Optional}

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

BIOPHOTO${SP}PIN=${XXX}${HT}FileName=${XXX}${HT}Type=${XXX}${HT}Size=${XXX}${HT}Content=${XXX}
Note:

FileName=${XXX}: Filename of the biometric image, with only the jpg format supported currently.

Type=${XXX} : Biometric identification type

Value	Meaning
0	common
1	fingerprint
2	face
3	vocal print
4	iris
5	retina
6	palm print
7	finger vein

8	palm
9	visible light face

Size=${XXX}: Length of the biometric photo after base64 coding.

Content=${XXX}: When the biometric photo is transmitted, base64 coding needs to be conducted for th e original binary biometric photo.

A normal response message from the server:

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

OK
```

Annotation:

HTTP status line: Defined with standard HTTP protocol

HTTP response header field:

Content-Length header field: Based on the HTTP 1.1, this header field is usually used to specify the data l ength of the response entity. If the response entity size is uncertain, head fields of Transfer-Encoding: ch unked, Content-Length and Transfer-Encoding are supported, all of which are standard definitions of HT TP and are not described in details here.

Response entity: When the server normally receives data and successfully processes data, OK issent. Whe n an error occurs, the error description is replied.

Example:

Request from the client:

```
POST /iclock/cdata?SN=0316144680030&table=OPERLOG&Stamp=9999 HTTP/1.1
Host: 58.250.50.81:8011
User-Agent: iClock Proxy/1.09
Connection: close
Accept: */*
Content-Length: 1684

BIOPHOTO PIN=123    FileName=123.jpg    Type=2  Size=95040  Content=AAAA.....
```

A 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

OK
```

11.16 Uploading Error Log

The configuration PushProtVer parameter sent by the server for initialization information exchange is greater than or equal to version 2.4.1.

A request message from the client:

```
POST /iclock/cdata?SN=${SerialNumber}&table=ERRORLOG&Stamp=${XXX} HTTP/1.1
Host: ${ServerIP}: ${ServerPort}
Content-Length: ${XXX}
……

${DataRecord}
```

Annotation:

HTTP request method: POST method

URI: /iclock/cdata

HTTP protocol version: 1.1

Client configuration information:

SN: ${Required} Serial number of the client

table=ERRORLOG: ${Required}

Stamp: ${Optional} Latest timestamp at which the error log is uploaded to the server. (For details, see the "ERRORLOGStamp" parameter in "Initializing Information Exchange".)

Host header field: ${Required}

Content-Length header field: ${Required}

Other header fields: ${Optional}

Request entity: ${DataRecord}, error log data, in the following format:

ERRORLOG ErrCode=${XXX}$(HT)ErrMsg=${XXX}$(HT)DataOrigin=${XXX}$(HT)CmdId=${XXX}$(HT)Additi

onal=${XXX}
Note:

ErrCode=${XXX} : Error code. See appendix 9 for coding instructions.

ErrMsg=${XXX}: Error message

DataOrigin=${XXX} : Data source, dev means device source data, cmd means software sent data.

CmdId=${XXX} : Command number issued by the software

Additional=${XXX} : Additional information (base64 data), the native data format is json.

A normal response message from the server:

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

OK
```

Annotation:

HTTP status line: Defined with standard HTTP protocol

HTTP response header field:

Content-Length header field: Based on the HTTP 1.1, this header field is usually used to specify the data l ength of the response entity. If the response entity size is uncertain, head fields of Transfer-Encoding: ch unked, Content-Length and Transfer-Encoding are supported, all of which are standard definitions of HT TP and are not described in details here.

Response entity: When the server normally receives data and successfully processes data, OK issent. Whe n an error occurs, the error description is replied.

Example:

Request from the client:

```
POST /iclock/cdata?SN=0316144680030&table=ERRORLOG&Stamp=9999 HTTP/1.1

Host: 58.250.50.81:8011

User-Agent: iClock Proxy/1.09

Connection: close

Accept: */*

Content-Length: 71
```

ERRORLOG ErrCode=D01E0001 ErrMsg= DataOrigin=cmd CmdId=123 Additional=

A 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

OK

12 Get Command

If the server needs to operate the equipment, the server generates a command format, waits till the equipment initiates a request, and then sends a command to the equipment. For the result of command execution, see Reply Command.

A request message from the client:

```
Get /iclock/getrequest?SN=${SerialNumber}
Host: ${ServerIP}: ${ServerPort}
……
```

Annotation:

HTTP request method: GET method

URI: /iclock/getrequest

HTTP protocol version: 1.1

Client configuration information:

SN: ${Required} Serial number of the client

Host head field: ${Required}

Other header fields: ${Optional}

A normal response message from the server:

When no commands are sent, the reply is as follows:

```
HTTP/1.1 200 OK
Date: ${XXX}
Content-Length: 2
……

OK
```

When a command is sent, the reply is as follows:

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

${CmdRecord}

Annotation:

HTTP status line: Defined with standard HTTP protocol HTTP response header field: Date header field: ${Required} This header field is used for synchronization with the server time, in GMT format. For example, Date: Fri, 03 Jul 2015 06: 53: 01 GMT Content-Length header field: Based on HTTP 1.1, this header field is usually used to specify the data length of the response entity. If the response entity size is uncertain, head fields of Transfer-Encoding: chunked, Content-Length and Transfer-Encoding are supported, all of which are standard definitions of HTTP and are not described in details here. Response entity: ${CmdRecord}, issued command record, in the following data format: C: ${CmdID}: ${CmdDesc}${SP}${XXX} Note: ${CmdID}: This command ID is generated by the server randomly, supporting numbers and letters and with a length not over 16 digits. The client needs to reply to the command with this command ID. For details, see the "Reply Command" function as follows. ${CmdDesc}: Command description falls into data commands and control commands. The data command is unified as the "DATA" description and detailed in the following "Data Command" function, and all kinds of control commands are different descriptions. ${LF} is used to connect multiple records.

12.1 DATA Command

When ${CmdDesc} in a command issued by the server is “DATA”, this command is deemed as a data command. The client data can be added, deleted, modified, or queried, but different service data supports different operations. For details, see the following.

12.1.1 UPDATE Subcommand

Adding or modifying data: Whether adding or modifying depends on whether corresponding data exists on the client, and this operation has nothing to do with the server. The following shows the command format:

C: ${CmdID}: DATA${SP}UPDATE${SP}${TableName}${SP}${DataRecord} Note: UPDATE: This description is used to represent the operation of adding or modifying data. ${TableName}: Different names of service data tables, for example, the user information USERINFO. The following describes specific supported data.

${DataRecord}: Service data records in the form of key=value. Different service data has different key descriptions. The following describes the specifics.

12.1.1.1 User Information

The command format is:

C:${CmdID}:DATA${SP}UPDATE${SP}USERINFO${SP}PIN=${XXX}${HT}Name=${XXX}${HT}Pri=${XXX}${HT}Passwd=${XXX}${HT}Card=${XXX}${HT}Grp=${XXX}${HT}TZ=${XXX}${HT}Verify=${XXX}${HT}ViceCard=${XXX}

Note:

PIN=${XXX}: User ID

Name=${XXX}: User name. When the equipment is in Chinese, the GB2312 code is used. When the equipment is in another language, the UTF-8 code is used.

Pri=${XXX}: User privilege value, with the meaning described as below

Value	Description
0	Normal user
2	Registrar
6	Administrator
10	User-defined
14	Super administrator

Passwd=${XXX}: Password

Card=${XXX}: Card number, supporting two formats.

a. hexadecimal data, in the format of [%02x%02x%02x%02x], representing the first, second, third or fourth digit from left to right. For example, if the card number is 123456789, this is: Card=[15CD5B07]

b. string data. If the card number is 123456789, this is: Card=123456789

Grp=${XXX}: Group to which the user belongs, group 1 by default.

TZ=${XXX}: Information on number of the time period used by the user, in the format of XXXXXXXXXXXXXXXX. Digit 1-4 describe whether the group time period is used, digit 5-8 describe using personal time period 1, digit 9-12 describe using personal time period 2, and digit 13-16 describe using personal time period 3.

For example: 0000000000000000 represents use of the group time period.

0001000200000000 represents use of personal time period, with personal time period 1 using the time information of number 2 time period.

0001000200010000 represents using personal time period, with personal time period 1 using the time information of number 2 time period and personal time period 2 using the time information of number 1 time period.

Verify=${XXX} : User verification mode, does not contain the field, is null, or is set to -1(use group verification, if there is no access group, group verification is 0), otherwise see (appendix 7)

ViceCard=${XXX} : User card number (secondary card), string data. If the card number is 123456789,ViceCard=123456789

${LF} is used to connect multiple records.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.2 Identity Card Information

The command format is:

```
C:${CmdID}:DATA$(SP)UPDATE$(SP)IDCARD$(SP)PIN=${XXX}${HT}SNNum=${XXX}${HT}IDNum=${XXX}${HT}DNNum=${XXX}${HT}Name=${XXX}${HT}Gender=${XXX}${HT}Nation=${XXX}${HT}Birthday=${XXX}${HT}ValidInfo=${XXX}${HT}Address=${XXX}${HT}AdditionalInfo=${XXX}${HT}Issuer=${XXX}${HT}Photo=${XXX}${HT}FPTemplate1=${XXX}${HT}FPTemplate2=${XXX}${HT}Reserve=${XXX}${HT}Notice=${XXX}
```

Note:

PIN=${XXX} : User ID. If the user's information is not bound to the identity card, then the value of PIN is 0.

SNNum=${XXX} : Physical card number of identity card

IDNum=${XXX} : Citizen id number

DNNum=${XXX} : Identity card serial number (card body management number)

Name=${XXX} : Id Name, using utf-8 encoding

Gender=${XXX} : Gender code

1," male "

2," female"

Nation=${XXX}: Ethnic code

0,"Decoding error"

1," Han"

2," Mongol"

3,"Hui"

4," Tibetan"

5," Uighur"

6,"Miao"

7,"Yi"

8,"Zhuang"

9,"Buyi"

10,"Korean"

11,"Manchu"

12,"Dong"

13,"Yao"

14,"Bai"

15,"Tujia"

16,"Hani"

17,"Kazakh"

18,"Dai"

19,"Li"

20,"Lisu"

21,"Wa"

22,"She"

23,"Gaoshan"

24,"Lahu"

25,"Shui"

26,"Dongxiang"

27,"Naxi"

28,"Jingpo"

29,"Kirghiz"

30,"Du"

31,"Daur"

32,"Mulam"

33,"Qiang"

34,"Blang"

35,"Salar"

36,"Maonan"

37,"Gelao"

38,"Xibe"

39,"Achang"

40,"Pumi"

41,"Tajik"

42,"Nu"

43,"Uzbek"

44,"Russian"

45,"Evenki"

46,"De'ang"

47,"Bonan"

48,"Yugur"

49,"Gin"

50,"Tatar"

51,"Drung"

52,"Oroqin"

53,"Hezhen"

54,"Menba"

55,"Lhoba"

56,"Jino"

57,"Coding error"

97,"Other"

98," Foreign origin"

Birthday=${XXX} : Date of birth (format: yyyyMMdd)

ValidInfo=${XXX} : Period of validity, start date and end date (format: yyyyMMddyyyyMMdd)

Address=${XXX}: Address, encoded in UTF-8

AdditionalInfo=${XXX}: Machine read appends address, encoded in UTF-8

Issuer = ${XXX}: Issuing authority, use UTF-8 encoding.

Photo=${XXX} : Photo data stored by identity card, which is encrypted and converted into base64 data c

ontent for transmission.

FPTemplate1=${XXX} : Fingerprint 1_ fingerprint characteristic data, and converted into base64 data con tent for transmission.

FPTemplate2=${XXX} : Fingerprint 2_ fingerprint characteristic data, and converted into base64 data con tent for transmission.

Reserve=${XXX}: Reserve field

Notice=${XXX} : Note information, encoded in UTF-8.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.3 Fingerprint Template

The command format is:

C: ${CmdID}: DATA${SP}UPDATE${SP}FINGERTMP${SP}PIN=${XXX}${HT}FID=${XXX}${HT}Size=${XXX}${H T}Valid=${XXX}${HT}TMP=${XXX}

Note:

PIN=${XXX}: User ID

FID=${XXX}: Finger number, valued from 0 – 9.

Size=${XXX}: Length of binary data of the finger template after base64 coding

Valid=${XXX}: to describe the template validity and duress mark, with the following values and meaning s:

Value and description

0 Invalid template

1 Normal template

3 Duress template

TMP=${XXX}: When the fingerprint template is transmitted, base64 coding needs to be conducted for th e original binary fingerprint template.

${LF} is used to connect multiple records.

Note: The fingerprint algorithm version supported by this command is less than or equal to 10.0.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.4 Face Template

The command format is:

C: ${CmdID}: DATA${SP}UPDATE${SP}FACE${SP}PIN=${XXX}${HT}FID=${XXX}${HT}Valid=${XXX}${HT}Size=${XXX}${HT}TMP=${XXX}

Note:

PIN=${XXX}: User ID

FID=${XXX}: Face template number, valued from 0.

Size=${XXX}: Length of binary data of the face template after base64 coding

Valid=${XXX}: Face template validity mark, with the following values and meanings: Value and description

0 Invalid template

1 Normal template

TMP=${XXX}: When the face template is transmitted, base64 coding needs to be conducted for the original binary face template.

${LF} is used to connect multiple records.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.5 Finger Vein Template

The command format is:

C:${CmdID}:DATA${SP}UPDATE${SP}FVEIN${SP}Pin=${XXX}${HT}FID=${XXX}${HT}Index=${XXX}${HT}Valid=${XXX}${HT}Size=${XXX}${HT}Tmp=${XXX}

Note:

Pin=${XXX} : User ID

FID=${XXX} : Finger number, (0~9)

Index=${XXX} : One finger has multiple finger vein templates, and Index is the number of finger vein template (0~2).

SIZE=${XXX}: Length after base64 coding of the finger vein template binary data

Valid=${XXX} : Valid identification of the finger vein template, the values are as follows:

Value	Description
0	invalid template
1	normal template

Tmp=${XXX}: Base64 encoding of the original binary finger vein template is needed when transferring th e finger vein template.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.6 Unified Templates

The following new biometric template will be uploaded and downloaded in a unified format, using Type in the data to distinguish what Type of biometric template is, using the integrated format: palm template, etc.

The command format is:

```
C:${CmdID}:DATA${SP}UPDATE${SP}BIODATA${SP}Pin=${XXX}${HT}No=${XXX}${HT}Index=${XXX}${HT}V
alid=${XXX}${HT}Duress=${XXX}${HT}Type=${XXX}${HT}MajorVer=${XXX}${HT}MinorVer=${XXX}${HT}Fo
rmat=${XXX}${HT}Tmp=${XXX}
```

Note:

Each field explains see uploading Unified Templates.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.7 User Photo

The command format is:

```
C: ${CmdID}: DATA${SP}UPDATE${SP}USERPIC${SP}PIN=${XXX}${HT}Size=${XXX}${HT}Content=${XXX}
```

Note:

PIN=${XXX}: User ID

Size=${XXX}: Length of binary data of the user photo after base64 coding

Content=${XXX}: When the user photo is transmitted, base64 coding needs to be conducted for the original binary user photo.

${LF} is used to connect multiple records.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.8 Comparison Photo

The command format is:

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=${XXX}
Note:
PIN=${XXX} : User ID
Type=${XXX} : Biometric identification type

Value	Meaning
0	common
1	fingerprint
2	face (near-infrared)
3	vocal print
4	iris
5	retina
6	palm print
7	finger vein
8	palm
9	visible light face

Size=${XXX}: Length of the biometric photo after base64 coding.
Content=${XXX}: When the biometric photo is transmitted, base64 coding needs to be conducted for the original binary biometric photo.
Url=${XXX} : Server file storage address, currently only supports JPG format.
Format=${XXX} : Send mode, 0: base64 mode, 1: url mode
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.
${LF} is used to connect multiple records.
Note:
Url is the relative path of the occasion, directly send relative path.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.9 Short Message

The command format is:

C: ${CmdID}: DATA${SP}UPDATE${SP}SMS${SP}MSG=${XXX}${HT}TAG=${XXX}${HT}UID=${XXX}${HT}MIN=${XXX}${HT}StartTime=${XXX}
Note:
MSG=${XXX}: Content of the short message, supporting up to 320 bytes. When the equipment is in Chinese, the GB2312 code is used. When the equipment is in another language, the UTF-8 code is used.
TAG=${XXX}: Type of the short message, with the following values and meanings:
Value and description
253 Public short message
254 User short message
255 Reserved short message
UID=${XXX}: Number of the short message, supporting only integer.
MIN=${XXX}: Valid duration of the short message, in minute.
StartTime=${XXX}: Starting time for the short message to take effect, in the format of XXXX-XX-XX XX: XX: XX. For example, 2015-07-29 00: 00: 00
${LF} is used to connect multiple records.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.10 Personal Short Message User List

The command format is:

C: ${CmdID}: DATA${SP}UPDATE${SP}USER_SMS${SP}PIN=${XXX}${HT}UID=${XXX}
Note:
PIN=${XXX}: User ID
UID=${XXX}: Number of the short message, supporting only integer.
${LF} is used to connect multiple records.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.11 Publicity Picture

The command format is:

C:${CmdID}:DATA${SP}UPDATE${SP}ADPIC${SP}Index=${XXX}${HT}Size=${XXX}${HT}Extension=${XXX}${HT}Content=${XXX}
Note:
Index=${XXX}: Image index
Size=${XXX} : Image size
Extension=${XXX}: Image extension
Content=${XXX}: Image Base64 encoding
${LF} is used to connect multiple records.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.12 Work Code

The command format is:

```
C:${CmdID}:DATA${SP}UPDATE${SP}WORKCODE${SP}PIN=${XXX}${HT}CODE=${XXX}${HT}NAME=${XXX}
Note:
PIN=${XXX} : Working code index
CODE=${XXX} : Working code
NAME=${XXX} : Working code name
${LF} is used to connect multiple records.
```

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.13 Shortcut Key

The command format is:

```
C:${CmdID}:DATA${SP}UPDATE${SP}ShortcutKey${SP}KeyID=${XXX}${HT}KeyFun=${XXX}${HT}StatusCod
e==${XXX}${HT}ShowName=${XXX}${HT}AutoState=${XXX}${HT}AutoTime=${XXX}${HT}Sun=${XXX}${H
T}Mon=${XXX}${HT}Tue=${XXX}${HT}Wed=${XXX}${HT}Thu=${XXX}{HT}Fri=${XXX}${HT}Sat=${XXX}
Note:
KeyID: Shortcut key ID
Value       Corresponding key
  1              F1
  2              F2
  3              F3
  4              F4
  5              F5
  6              F6
  7              F7
  8              F8
KeyFun: Shortcut key function
Value       Corresponding function
  0              Undefined
  1              State key
  2              Work code
  3              Short message
  4              Key for help
  5              Check the attendance record
  6              Check the final attendance record
StatusCode: Attendance status
```

ShowName: Status name
AutoState: Auto switch
AutoTime: Automatic switching time from Monday to Sunday, 08:00; 09:00; 10:00; 11:00; 12:00; 13:00; 14:00
Sun: Whether to switch on Sunday
Mon: Whether to switch on Monday
Tue: Whether to switch on Tuesday
Wed: Whether to switch on Wednesday
Thu: Whether to switch on Thursday
Fri: Whether to switch on Friday
Sat: Whether to switch on Saturday

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.14 Access Group

The command format is:

```
C:${CmdID}:DATA${SP}UPDATE${SP}AccGroup${SP}ID=${XXX}${HT}Verify=${XXX}${HT}ValidHoliday=${XXX}${HT}TZ=${XXX}
```

Note:
ID: Number of access group
Verify: Group verification mode, with the default value of 0, as shown in (appendix 7)
Validholiday: Valid for holidays: value range 0-1
TZ format: For example: TZ=1; 1; 0; 0: the first number represents whether to use group time period, the second number represents time period 1, the third parameter represents time period 2, and the fourth parameter represents time period 3

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.15 Access Time Periods

The command format is:

```
C:${CmdID}:DATA${SP}UPDATE${SP}AccTimeZone${SP}UID=${XXX}${HT}SunStart=${XXX}${HT}SunEnd=${XXX}${HT}MonStart=${XXX}${HT}MonEnd=${XXX}${HT}TuesStart=${XXX}${HT}TuesEnd=${XXX}${HT}WedStart=${XXX}${HT}WedEnd=${XXX}${HT}ThursStart=${XXX}${HT}ThursEnd=${XXX}${HT}FriStart=${XXX}${HT}FriEnd=${XXX}${HT}SatStart=${XXX}${HT}SatEnd=${XXX}
```

Note:
UID: Time period number
SunStart: Sunday start time, 1159 means 11:59

SunEnd: Sunday end time, 2359 means 23:59
MonStart: Monday start time
MonEnd: Monday end time
TueStart: Tuesday start time
TuesEnd: Tuesday end time
WedStart: Wednesday start time
WedEnd: Wednesday end time
ThurStart: Thursday start time
ThursEnd: Thursday end time
FriStart: Friday start time
FriEnd: Friday end time
SatStart: Saturday start time
SatEnd: Saturday end time

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.16 Access Holiday

The command format is:

```
C:${CmdID}:DATA${SP}UPDATE${SP}AccHoliday${SP}UID=${XXX}${HT}HolidayName=${XXX}${HT}StartDa
te=${XXX}${HT}EndDate=${XXX}${HT}TimeZone=${XXX}
Note:
UID: Holiday number
HolidayName: Holiday name
StartDate: 1123 means November 23rd
EndDate: 1125 means November 25
TimeZone: Time period number
```

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.17 Access Combined Verification

The command format is:

```
C:${CmdID}:DATA${SP}UPDATE${SP}AccUnLockComb${SP}UID=${XXX}${HT}Group1=${XXX}${HT}Group2
=${XXX}${HT}Group3=${XXX}${HT}Group4=${XXX}${HT}Group5=${XXX}
Note:
UID: Group verification number
Group1: Group number of people. The group number in the person information
Group2: Group number of people. The group number in the person information
Group3: Group number of people. The group number in the person information
Group4: Group number of people. The group number in the person information
```

Group5: Group number of people. The group number in the person information

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.1.18 Blacklist of Identity Card Issued

The command format is:

C: ${CmdID}:DATA${SP}UPDATE${SP}Blacklist${SP}IDNum=${XXX}
Note:
IDNum: ID number

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.2 DELETE Subcommand

To delete data. The command format is:

C: ${CmdID}: DATA${SP}DELETE${SP}${TableName}${SP}${DataRecord}
Note:
DELETE: This description is used to represent the operation of deleting data.
${TableName}: Different service data table names. For example, the user information is USERINFO, and th e following describes specific supported data.
${DataRecord}: Condition for deleting data. Different service data supports different conditions. The follo wing describes the specifics.

12.1.2.1 User Information

The command format is:

C: ${CmdID}: DATA${SP}DELETE${SP}USERINFO${SP}PIN=${XXX}
Note:
PIN=${XXX}: User ID
To delete specified user information, including fingerprint template, face template and user photo.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.2.2 Fingerprint Template

The command format is:

C:${CmdID}:DATA${SP}DELETE${SP}FINGERTMP${SP}PIN=${XXX}
C:${CmdID}:DATA${SP}DELETE${SP}FINGERTMP${SP}PIN=${XXX}${HT}FID=${XXX}
Note:
PIN=${XXX}: User ID
FID=${XXX}: Finger number, valued from 0-9.
To delete specified fingerprint template. When only PIN information is transmitted, all fingerprints of the user are deleted.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.2.3 Face Template

The command format is:

C: ${CmdID}: DATA${SP}DELETE${SP}FACE${SP}PIN=${XXX}
Note:
PIN=${XXX}:User ID
To delete specified face template of the user

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.2.4 Finger Vein Template

The command format is:

C:${CmdID}:DATA${SP}DELETE${SP}FVEIN${SP}Pin=${XXX}
C:${CmdID}:DATA${SP}DELETE${SP}FVEIN${SP}Pin=${XXX}${HT}FID=${XXX}
Note:
PIN=${XXX}:User ID
FID=${XXX} : Finger number, (0~9)
To delete specified finger vein template of the user

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.2.5 Unified Templates

The command format is:

```
C:${CmdID}:DATA${SP}DELETE${SP}BIODATA${SP}Pin=${XXX}
C:${CmdID}:DATA${SP}DELETE${SP}BIODATA${SP}Pin=${XXX}{HT}Type=${XXX}
C:${CmdID}:DATA${SP}DELETE${SP}BIODATA${SP}Pin=${XXX}{HT}Type=${XXX}{HT}No=${XXX}
Note:
See upload unified template function for field description
To delete specified unified template of the user
```

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.2.6 User Photo

The command format is:

```
C: ${CmdID}: DATA${SP}DELETE${SP}USERPIC${SP}PIN=${XXX}
Note:
PIN=${XXX}: User ID
To delete specified user photo of the user
```

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.2.7 Comparison Photo

The command format is:

```
C:${CmdID}:DATA${SP}DELETE${SP}BIOPHOTO${SP}PIN=${XXX}
Note:
PIN=${XXX}: User ID
To delete specified comparison photo of the user
```

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

12.1.2.8 Short Message

The command format is:

```
C:${CmdID}:DATA${SP}DELETE${SP}SMS${SP}UID=${XXX}
Note:
UID=${XXX}: short message number, supporting only integers.
```

For how the result of command execution is replied, see the Reply Command function.For the Return value, see Appendix 1. The format of returned content is:

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

12.1.2.9 Work Code

The command format is:

```
C:${CmdID}:DATA${SP}DELETE${SP}WORKCODE${SP}CODE=${XXX}
Note:
CODE=${XXX} : Working code
```

For how the result of command execution is replied, see the Reply Command function.For the Return value, see Appendix 1. The format of returned content is:

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

12.1.2.10 Publicity Picture

The command format is:

```
C:${CmdID}:DATA${SP}DELETE${SP}ADPIC${SP}Index=${XXX}Note: Index=${XXX}: Image index
```

For how the result of command execution is replied, see the Reply Command function.For the Return value, see Appendix 1. The format of returned content is:

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

12.1.3 QUERY Subcommand

To query data, the command format is:

```
C: ${CmdID}: DATA${SP}QUERY${SP}${TableName}${SP}${DataRecord}
Note:
QUERY: This description is used to represent the operation of querying data.
${TableName}: Different service data table names. For example, the user information is USERINFO, and th
e following describes specific supported data.
${DataRecord}: Condition for querying data. Different service data supports different conditions. The foll
owing describes the specifics.
```

- **Attendance Record**

The command format is:

```
C: ${CmdID}: DATA${SP}QUERY${SP}ATTLOG${SP}StartTime=${XXX}${HT}EndTime=${XXX}
Note:
StartTime=${XXX}: Query starting time, in the format of XXXX-XX-XX XX: XX: XX. For example, 2015-07-29
 00: 00: 00
EndTime=${XXX}: Query ending time, in the format of XXXX-XX-XX XX: XX: XX. For example, 2015-07-29
23: 59: 59
```

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

To query the attendance record within specified time period. For how to upload, see "Uploading Attendance Record".

- **Attendance Photo**

The command format is:

```
C: ${CmdID}: DATA${SP}QUERY${SP}ATTPHOTO${SP}StartTime=${XXX}${HT}EndTime=${XXX}
Note:
StartTime=${XXX}: Query starting time, in the format of XXXX-XX-XX XX: XX: XX. For example, 2015-07-29
 00: 00: 00
EndTime=${XXX}: Query ending time, in the format of XXXX-XX-XX XX: XX: XX. For example, 2015-07-29
23: 59: 59
```

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

To query the attendance photo within specified time period. For how to upload, see "Uploading Attendance Photo".

- **User Information**

The command format is:

```
C: ${CmdID}: DATA${SP}QUERY${SP}USERINFO${SP}PIN=${XXX}
Note:
PIN=${XXX}: User ID
```

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

To query the basic information of specified user. For how to upload, see "Uploading User Information".

- **Fingerprint Template**

The command format is:

```
C: ${CmdID}: DATA${SP}QUERY${SP}FINGERTMP${SP}PIN=${XXX}${HT}FingerID=${XXX}
Note:
PIN=${XXX}: User ID
FingerID=${XXX}: Finger number, valued from 0 – 9.
```

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

To query the fingerprint template information of the user. When only the PIN information is transmitted, the information about all fingerprint templates of the user is queried. For how to upload, see "Uploading Fingerprint Template".

- **Unified Template**

The command format is:

```
C:${CmdID}:DATA${SP}QUERY${SP}BIODATA${SP}Type=${XXX}
```

C:${CmdID}:DATA${SP}QUERY${SP}BIODATA${SP}Type=${XXX}${HT}PIN=${XXX}
C:${CmdID}:DATA${SP}QUERY${SP}BIODATA${SP}Type=${XXX}${HT}PIN=${XXX}${HT} No=${XXX}
Note:
Type=${XXX} : Biometric Type

Value	Meaning
0	Common
1	Fingerprint
2	Face
3	Voiceprint
4	Iris
5	Retina
6	Palmprint
7	Finger vein
8	Palm
9	Visible light face

PIN=${XXX} : User ID
No=${XXX} : Biometric specific number, default value is 0.
[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 finger.
[Finger vein]: the same as fingerprints
[Face]: All is 0
[Iris]: 0 for left eye, 1 for right eye
[Palm]: 0 is left hand, 1 is right hand

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

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

To query the unified template information of the specified type. When only the Type information is transmitted, all unified template information of the specified type is queried. For how to upload, see "Uploading Unified Template".

12.2 CLEAR Command

12.2.1 Clearing Attendance Record

To clear the client attendance record, the command format is:

C: ${CmdID}: CLEAR${SP}LOG
Note:
CLEAR${SP}LOG is used to describe this command.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

```
ID=${XXX}&Return=${XXX}&CMD=CLEAR_LOG
Note:
CMD=CLEAR_LOG: CLEAR_LOG is used to describe this command.
```

12.2.2 Clearing Attendance Photo

To clear the client attendance photo, the command format is:

```
C: ${CmdID}: CLEAR${SP}PHOTO
Note:
CLEAR${SP}PHOTO is used to describe this command.
```

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

```
ID=${XXX}&Return=${XXX}&CMD=CLEAR_PHOTO
Note:
CMD=CLEAR_PHOTO: CLEAR_PHOTO is used to describe this command.
```

12.2.3 Clearing All Data

To clear all client data, the command format is:

```
C: ${CmdID}: CLEAR${SP}DATA
Note:
CLEAR${SP}DATA is used to describe this command.
```

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

```
ID=${XXX}&Return=${XXX}&CMD=CLEAR_DATA
Note:
CMD=CLEAR_DATA: CLEAR_DATA is used to describe this command.
```

12.2.4 Clearing Unified Template

To clear client unified template data of the specified type, the command format is:

```
C:${CmdID}:CLEAR${SP}BIODATA
Note:
```

CLEAR${SP}BIODATA is used to describe this command.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

I ID=${XXX}&Return=${XXX}&CMD=CLEAR_BIODATA
Note:
CMD= CLEAR_BIODATA: CLEAR_BIODATA is used to describe this command.

12.3 Check Command

12.3.1 Checking Data Update

The client is required to read configuration information from the server and re-upload corresponding data to the server based on the timestamp. For details, see "Initializing Information Exchange". Currently, only the server resetting the timestamp to 0 is supported. For example, set parameter Stamp to 0. After reading configuration parameters, the client conducts Uploading Attendance Record again, and the command format is:

C: ${CmdID}: CHECK
Note:
CHECK is used to describe this command.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

ID=${XXX}&Return=${XXX}&CMD=CHECK

12.3.2 Checking and Transmitting New Data

The client immediately checks whether new data exists and transmits the new data to the server. The command format is:

C: ${CmdID}: LOG
Note:
LOG is used to describe this command.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

ID=${XXX}&Return=${XXX}&CMD=LOG

12.3.3 Automatically Verifying Attendance Data

The server issues the verification for attendance records within a time period, start and end time of uploading by the attendance equipment, as well as total number of records. The verification is achieved by the server, and the command format is:

C: ${CmdID}: VERIFY${SP}SUM${SP}ATTLOG${SP}StartTime=${XXX}${HT}EndTime=${XXX}
Note:
VERIFY${SP}SUM is used to describe this command
StartTime=${XXX}: Starting time of issuing by the server, in the format of XXXX-XX-XX XX: XX: XX. For example, 2015-07-29 00: 00: 00
EndTime=${XXX}: Ending time of issuing by the server, in the format of XXXX-XX-XX XX: XX: XX. For example, 2015-07-29 00: 00: 00

For how the result of command execution is replied, see the Reply Command function(#replycmd). For the Return value, see Appendix 1(#appendix1). The format of returned content is:

ID=${XXX}&Return=${XXX}&CMD=VERIFY${SP}SUM&StartTime=${XXX}&EndTime=${XXX}&AttlogSum=${XXX}
Note:
AttlogSum=${XXX}: Total number of attendance records within the period from starting to ending time

12.4 Configuring Option Command

12.4.1 Option for Setting the Client

To set the client configuration information, the command format is:

C: ${CmdID}: SET${SP}OPTION${SP}${Key}=${Value}
Note:
SET${SP}OPTION is used to describe this command.
The configuration information is set in the form of key-value, and this command supports only the configuration of single configuration information.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

ID=${XXX}&Return=${XXX}&CMD=SET${SP}OPTION

12.4.2 Option for Refreshing the Client

The client reloads the configuration information. The command format is:

C: ${CmdID}: RELOAD${SP}OPTIONS

Note:
RELOAD${SP}OPTIONS is used to describe this command.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

ID=${XXX}&Return=${XXX}&CMD=RELOAD${SP}OPTIONS

12.4.3 Sending Client Information to the Server

The server gets information such as client configuration. The command format is:

C: ${CmdID}: INFO
Note:
INFO is used to describe this command.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

ID=${XXX}&Return=${XXX}&CMD=INFO${LF}${Key}=${Value}${LF}${Key}=${Value}${LF}${Key}=${Value}
${LF}${Key}=${Value}……
Note:
CMD=INFO is followed by specific customer configuration information, in the form of key-value.

12.5 File Command

12.5.1 Getting File in the Client

The client sends a server-specified file to the server. The command format is:

C: ${CmdID}: GetFile${SP}${FilePath}
Note:
GetFile is used to describe this command.
${FilePath}: File in the client system

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

ID=${XXX}${LF}SN=${SerialNumber}${LF}FILENAME=${XXX}${LF}CMD=GetFile${LF}Return=${XXX}${LF}C
ontent=${BinaryData}
Note:
Return=${XXX}: Size of returned file.
Content=${BinaryData}: Binary data flow of the transmitted file

12.5.2 Sending File to the Client

Function 1

The equipment is required to download a file from the server and saves the file in a specified folder. (After being downloaded, a .tgz file is automatically decompressed to the specified directory of FilePath or /mnt/mtdblock if no directory is specified. For a file in another format, the file save path and filename need to be specified.) This file must be provided by the server by HTTP, as well as the URL for obtaining this file. If the URL starts with "http://", the equipment deems the URL as a complete URL address, otherwise, the equipment appends the server's /iclock/ address to specified URL. The command format is:

C: ${CmdID}: PutFile${SP}${URL}${HT}${FilePath}
Note:
GetFile is used to describe this command.
${URL}: Address of the file to be downloaded from the server
${FilePath}: Destination path for the file to be saved on the client
Example 1: PutFile file/fw/X938/main.tgz main.tgz or PutFile file/fw/X938/main.tgz requires the equipment to download http: //server/iclock/file/fw/X938/main.tgz, and decompress main.tgz into the folder of /mnt/mtdblock.
Example 2: PutFile file/fw/X938/main.tgz /mnt/ requires the equipment to download http: //server/iclock/file/fw/X938/main.tgz, and decompress main.tgz into the folder of /mnt/.
Example 3: PutFile file/fw/X938/ssruser.dat /mnt/mtdblock/ssruser.dat requires the equipment to download http: //server/iclock/file/fw/X938/ssruser.dat, and remain the file to be /mnt/mtdblock/ssruser.dat.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

ID=${XXX}${LF}Return=${XXX}${LF}CMD=PutFile
Note:
Return=${XXX}: Size of the returned file

Function 2

C:${CmdID}:PutFile${SP}${URL}${HT}${FilePath}${HT}Action=${Value}
Note:
Use PutFile to describe the command.
${URL}：Address of the files to be downloaded from the server.
${FilePath}：Destination path where the files are stored in the client.
Action: describes what action to take after the file downloading is complete, supporting the following at present:
Action=SyncData: represents that the device is required to synchronize the data of the same data type with those in the downloaded files, that is, overwriting the old data in the device. The action requires two additional parameters, TableName and RecordCount. The complete command is as follows:
C:${CmdID}:PutFile${SP}${URL}${HT}${FilePath}${HT}Action=${Value}${HT}TableName=${Value}${HT}RecordCount=${Value}
TableName: represents the data type, supporting the following:
${Value} Data type

USERINFO User data
FINGERTMP Fingerprint data
FACE Face data
RecordCount: Number of records in the data packet.
Action=AppendData: represents that the data in the downloaded files should be appended to the device. The complete command is as follows:
C:${CmdID}:PutFile${SP}${URL}${HT}${FilePath}${HT}Action=AppendData
The format of the content in the compressed package is the same as that of distributed data commands, such as:
C:123:DATA UPDATE USERINFO PIN=1 Name=1 Pri=0 Passwd=1 Grp=1
C:124:DATA UPDATE FINGERTMP PIN=1 FID=11 SIZE=28 VALID=1 TMP=c2FmZHNhd3Jyd3JlcmVyZXJlcnc=
C:125:DATA UPDATE FACE PIN=1 FID=0 SIZE=28 VALID=1 TMP=c2FmZHNhd3Jyd3JlcmVyZXJlcnc=
C:126:DATA UPDATE FACE PIN=1 FID=1 SIZE=28 VALID=1 TMP=c2FmZHNhd3Jyd3JlcmVyZXJlcnc=
C:127:DATA UPDATE FACE PIN=1 FID=2 SIZE=28 VALID=1 TMP=c2FmZHNhd3Jyd3JlcmVyZXJlcnc=
C:128:DATA UPDATE FACE PIN=1 FID=3 SIZE=28 VALID=1 TMP=c2FmZHNhd3Jyd3JlcmVyZXJlcnc=
C:129:DATA UPDATE FACE PIN=1 FID=4 SIZE=28 VALID=1 TMP=c2FmZHNhd3Jyd3JlcmVyZXJlcnc=
C:130:DATA UPDATE FACE PIN=1 FID=5 SIZE=28 VALID=1 TMP=c2FmZHNhd3Jyd3JlcmVyZXJlcnc=
C:131:DATA UPDATE FACE PIN=1 FID=6 SIZE=28 VALID=1 TMP=c2FmZHNhd3Jyd3JlcmVyZXJlcnc=
C:132:DATA UPDATE FACE PIN=1 FID=7 SIZE=28 VALID=1 TMP=c2FmZHNhd3Jyd3JlcmVyZXJlcnc=
C:133:DATA UPDATE FACE PIN=1 FID=8 SIZE=28 VALID=1 TMP=c2FmZHNhd3Jyd3JlcmVyZXJlcnc=
C:134:DATA UPDATE FACE PIN=1 FID=9 SIZE=28 VALID=1 TMP=c2FmZHNhd3Jyd3JlcmVyZXJlcnc=
C:135:DATA UPDATE FACE PIN=1 FID=10 SIZE=28 VALID=1 TMP=c2FmZHNhd3Jyd3JlcmVyZXJlcnc=
C:136:DATA UPDATE FACE PIN=1 FID=11 SIZE=28 VALID=1 TMP=c2FmZHNhd3Jyd3JlcmVyZXJlcnc=
……

See the format of returned content as follows:

ID=${XXX}${LF}Return=${XXX}${LF}CMD=PutFile
Note:
Return=${XXX}: Size of the returned file.

12.6 Remote Enrollment Command

12.6.1 Enrolling User Fingerprint

The fingerprint enrollment is initiated by the server and conducted on the client. The command format is:

C: ${CmdID}: ENROLL_FP${SP}PIN=${XXX}${HT}FID=${XXX}${HT}RETRY=${XXX}${HT}OVERWRITE=${XXX}
Note:
ENROLL_FP is used to describe this command.
PIN=${XXX}: Enrolled user ID
FID=${XXX}: Enrolled fingerprint number
RETRY=${XXX}: Number of retries required if enrollment fails
OVERWRITE=${XXX}: Whether to overwrite the fingerprint. 0 means the fingerprint of corresponding use r exists and will not be overwritten and error information is returned. 1 means the fingerprint of correspo nding user exists and will be overwritten.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

ID=${XXX}&Return=${XXX}&CMD=ENROLL_FP

12.6.2 Enrolling Card Number

The card number enrollment is initiated by the server and conducted on the client. The command format is:

C:XXX:ENROLL_MF PIN=%s\tRETRY=%d
Example: C:123:ENROLL_MF PIN=408\tRETRY=3
Note:
PIN - User ID
RETRY - Number of retries
Returned value:
0 Command executed successfully
-1 Parameter error
-3 Access error
4 Register failed retries
5 Log out over time
6 Click Esc to exit the registration screen

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

ID=${XXX}&Return=${XXX}&CMD=ENROLL_MF

12.6.3 Enrolling Face, Palm Print (Unified Templates)

The fingerprint enrollment is initiated by the server and conducted on the client. The command format is:

ENROLL_BIO TYPE=%?\tPIN=%?tCardNo=%?\tRETRY=%?\tOVERWRITE=%?
TYPE:
0 /**< General template */
1 /**< Fingerprint */
2 /**< Face */
3 /**< Voice */
4 /**< Iris */
5 /**< Retina */
6 /**< Palm vein */
7 /**< Finger vein */
8 /**< Palm print */
9 /**< Visible light face */
PIN: Enrolled user ID
CardNo: Enrolled card number
RETRY: Number of retries required if enrollment fails
OVERWRITE: Whether to overwrite the face. 0 means the face of corresponding user exists and will not be overwritten and error information is returned. 1 means the face of corresponding user exists and will be overwritten.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

ID=${XXX}&Return=${XXX}&CMD=ENROLL_BIO

12.7 Control Command

12.7.1 Rebooting the Client

To reboot the client, the command format is:

C: ${CmdID}: REBOOT
Note:
REBOOT is used to describe this command.

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

ID=${XXX}&Return=${XXX}&CMD=REBOOT

12.7.2 Outputting the Door Unlocking Signal

The access equipment outputs the door unlocking signal. The command format is:

```
C: ${CmdID}: AC_UNLOCK
Note:
AC_UNLOCK is used to describe this command.
```

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

```
ID=${XXX}&Return=${XXX}&CMD=AC_UNLOCK
```

12.7.3 Canceling the Alarm Signal Output

The access equipment cancels the alarm signal output. The command format is:

```
C: ${CmdID}: AC_UNALARM
Note:
AC_UNALARM is used to describe this command.
```

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

```
ID=${XXX}&Return=${XXX}&CMD=AC_UNALARM
```

12.8 Other Commands

12.8.1 Executing the System Command

The server issues operating system commands which are supported by the client which send execution results to the server. The command format is:

```
C: ${CmdID}: SHELL${SP}${SystemCmd}
Note:
SHELL is used to describe this command
${SystemCmd}: Operating system command. For example, when the client is linux system, 1s is
supported.
```

For how the result of command execution is replied, see the Reply Command function. For the Return value, see Appendix 1. The format of returned content is:

```
ID=${XXX}${LF}SN=${SerialNumber}${LF}Return=${XXX}${LF}CMD=Shell${LF}FILENAME=shellout.txt${LF}
```

Content=${XXX}
Note:
Return=${XXX}: The value is the returned value for the system command.
Content=${XXX}: The value is the output content of the system command.

12.8.2 Online Update

Application scenario: Firmware used to remotely upgrade client devices from server software.

Method 1: Remotely upgrade the client's firmware, compatible controller and new architecture all-in-one machine. Upgrade files need to be converted by the server and then sent to the client.

Format:

The server issues the command:
C:${CmdID}:UPGRADE$(SP)checksum=${XXX},url=$(URL),size=${XXX}
The client downloads the upgrade package from the URL that comes with the command:
GET /iclock/file?SN=$(SerialNumber)&url=$(URL) HTTP/1.1
……
The client uploads the execution results:
ID=${CmdID}&Return=${XXX}&CMD=UPGRADE

Annotation:

Checksum: Represents md5 checksum
Url: Represents the download resource address of the upgrade file, and the upgrade file name is emfw.cfg
Size represents the original file size.
Note:
In this method, the firmware update file is converted to base64 format data by the server when it is issued. The file that the client receives needs to be converted to binary format and named emfw.cfg

Example:

The server issues the firmware upgrade command:
C:384:UPGRADE checksum=a5bf4dcd6020f408589224274aab132d,url=http*//localhost*8088\fireware\F20\admin\emfw.cfg,size=2312
The client requests 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 successful execution results:
ID=384&Return=0&CMD=UPGRADE

Method 2: Remote upgrade client firmware, directly obtain files, no need to transfer format, the client directly obtain files.

Format:

The server issues the command:
C:${CmdID}:UPGRADE$(SP)type=1,checksum=${XXX},size=${XXX},url=$(URL)
The client requests to download the upgrade package:
GET /iclock/file?SN=$(SerialNumber)&url=$(URL) HTTP/1.1
Cookie: token=af65a75608cf5b80fbb3b48f0b4df95a
Host: 192.168.213.17:8088
……
The client uploads the execution results:

ID=${CmdID}&Return=${XXX}&CMD=UPGRADE

Annotation:

Type: 1 means to get the upgrade file from the url. For the time being, only 1 is supported.
Checksum: Represents md5 checksum
Url: Represents the download resource address of the upgrade file, and the upgrade file name is emfw.cfg
Size represents the upgrade package size
Note:
In this method, what the client gets directly is the firmware update file, which does notneed to be converted to another format.

Example:

The server issues the firmware upgrade command:
C:123:UPGRADE
type=1,checksum=oqoier9883kjankdefi894eu,size=6558,url=http://192.168.0.13:89/data/emfw.cfg
The client requests 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 successful execution results:
ID=384&Return=0&CMD=UPGRADE

12.8.3 Background verification

Application scenario: After the fingerprint/face verification on the attendance device is successful, the personnel number will be uploaded to the back-end system by push, and the back-end system will return a result (whether the verification is allowed or not) to the attendance device after receiving the personnel number for logical judgment.

Format:

```
Client data sending
POST /iclock/cdata?SN=${SerialNumber}&type=PostVerifyData HTTP/1.1
Host: ${ServerIP}:${ServerPort}
 ……
${PostData}    // Uploaded data
```

Annotation:

```
HTTP request method: GET method
URI: /iclock/cdata
HTTP protocol version: 1.1
Client configuration information:
SN: ${Required} Serial number of the client
Type =PostRecordData means to upload recorded data
Host header field: ${Required}
Other header fields: ${Optional}

Normal server response
HTTP/1.1 200 OK
Date: ${XXX}
Content-Length: ${XXX}
 ……


  OK
```

Annotation:

```
HTTP status line: Defined with standard HTTP protocol
HTTP response header field:
Date header field: ${Required} uses this header field to synchronize server time, and the time format uses
 GMT format, such as Date: Fri, 03 Jul 2015 06:53:01 GMT
Content-Length header field: According to the HTTP 1.1, this header field is generally used to specify the
data length of the response entity. If the response entity size is uncertain, head fields of Transfer-
Encoding: chunked, Content-Length and Transfer-Encoding are supported, all of which are standard
definitions of HTTP and are not described in details here.
Response entity: When the server normally receives data and successfully processes data, OK is replied.
When an error occurs, the error description is replied.

Parameter configuration: PostSelfDefineDataType=PostVerifyData
```

13 Command Reply

After Getting Command Issued by the Server, the client needs to reply corresponding command.

A request message from the client:

```
POST /iclock/devicecmd?SN=${SerialNumber}
Host: ${ServerIP}: ${ServerPort}
 Content-Length: ${XXX}
……

${CmdRecord}
```

Annotation:

```
HTTP request method: GET method
URI: /iclock/devicecmd
HTTP protocol version: 1.1
Client configuration information:
SN: ${Required} Serial number of the client
Host head field: ${Required}
Content-Length header field: ${Required}
Other header fields: ${Optional}
Response entity: ${CmdRecord}, record of replied commands. The reply content all contains the ID\Retur
n\CMD information, with the following meanings:
ID: Number of the command issued by the client
Return: Returned result after the client executes the command
CMD: Description of the command issued by the server
A small number of replies contain other information. For specific reply content format, see the descriptio
n of each command.
${LF} is used to connect multiple command reply records.
```

A normal response message from the server:

```
HTTP/1.1 200 OK
Date: ${XXX}
Content-Length: 2
……

OK
```

Annotation:

```
HTTP status line: Defined with standard HTTP protocol
HTTP response header field:
Date header field: ${Required} This header field is used for synchronization with the server time, in GMT f
ormat. For example, Date: Fri, 03 Jul 2015 06: 53: 01 GMT
```

Content-Length header field: Based on HTTP 1.1, this header field is usually used to specify the data length of the response entity. If the response entity size is uncertain, head fields of Transfer-Encoding: chunked, Content-Length and Transfer-Encoding are supported, all of which are standard definitions of HTTP and are not described in details here.

Example:

A request from the client:
POST /iclock/devicecmd?SN=0316144680030 HTTP/1.1
Host: 58.250.50.81: 8011
User-Agent: iClock Proxy/1.09
Connection: close
Accept: */*
Content-Length: 143

ID=info8487&Return=0&CMD=DATA
ID=info8488&Return=0&CMD=DATA
ID=info8489&Return=0&CMD=DATA
ID=info7464&Return=0&CMD=DATA
ID=fp7464&Return=0&CMD=DATA

A response from the server:
HTTP/1.1 200 OK
Server: nginx/1.6.0
Date: Tue, 30 Jun 2015 01: 24: 48 GMT
Content-Type: text/plain
Content-Length: 2
Connection: close
Pragma: no-cache
Cache-Control: no-store

OK

14 Remote Attendance

When attendance is required for a user on a business trip and no information about this user is stored in the attendance machine, the user can check on attendance remotely. Current application scenario: The user uses the attendance machine keypad to directly enter ID and press OK, and then the attendance machine requests the server to issue all information about this user (basic information and fingerprint information). After that, the user checks on attendance. After being downloaded, the user information is stored in the attendance machine for a period of time. The saving time is set via a parameter. After this period of time, the user information will be deleted.

A request message from the client:

```
GET /iclock/cdata?SN=${SerialNumber}&table=RemoteAtt&PIN=${XXX} HTTP/1.1
Host: ${ServerIP}: ${ServerPort}
……
```

Annotation:

```
HTTP request method: GET method
URI: /iclock/cdata
HTTP protocol version: 1.1
Client configuration information:
   SN: ${Required} Serial number of the client
   table=RemoteAtt: Acquiring user information for remote attendance
   PIN=${XXX}: ID information to be required
   Host head field: ${Required}
   Other header fields: ${Optional}
```

A normal response message from the server:

```
When user information exists, the reply information is:
HTTP/1.1 200 OK
Date: ${XXX}
Content-Length: ${XXX}
……

DATA${SP}UPDATE${SP}USERINFO${SP}PIN=${XXX}${HT}Name=${XXX}${HT}Passwd=${XXX}${HT}Card=
${XXX}${HT}Grp=${XXX}${HT}TZ=${XXX}${HT}Pri=${XXX}
DATA${SP}UPDATE${SP}FINGERTMP${SP}PIN=${XXX}${HT}FID=${XXX}${HT}Size=${XXX}${HT}Valid=${XX
X}${HT}TMP=${XXX}
```

Annotation: ${LF} is used to connect multiple data records of the response entity. For specific data format, see Issuing User Information and Issuing Fingerprint Template.

Appendix 1

Error Code	Description
0	Successful
-1	The parameter is incorrect.
-2	The transmitted user photo data does not match the given size.
-3	Reading or writing is incorrect.
-9	The transmitted template data does not match the given size.
-10	The user specified by PIN does not exist in the equipment.
-11	The fingerprint template format is illegal.
-12	The fingerprint template is illegal.
-1001	Limited capacity
-1002	Not supported by the equipment
-1003	Command execution timeout
-1004	The data and equipment configuration are inconsistent.
-1005	The equipment is busy.
-1006	The data is too long.
-1007	Memory error
-1008	Failed to get server data

Enroll_FP/Enroll_BIO Error Code	Description
2	Enroll Fingerprint: Fingerprints of the user already exist.
4	Enroll Fingerprint: Registration fails, usually caused by the inferior quality of fingerprints or the inconsistency of the three fingerprints.
5	Enroll Fingerprint: Registered fingerprints already exist in the fingerprint database.
6	Enroll Fingerprint: Registration is cancelled.
7	Enroll Fingerprint: Registration cannot proceed due to the busy device.

PutFile（Action=SyncData）Error Code	Description
n > 0	Data is synchronized, with n commands successfully processed.

Appendix 2

Language number	Meaning
83	Simplified Chinese
69	English
97	Spanish
70	French
66	Arabic
80	Portuguese
82	Russian
71	German
65	Persian
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 3

Operation code	Meaning
0	Startup
1	Shutdown
2	Authentication fails
3	Alarm
4	Access menu
5	Change settings
6	Enroll fingerprint
7	Enroll password
8	Enroll HID card
9	Delete user
10	Delete fingerprint
11	Delete password
12	Delete RF card
13	Clear data
14	Create MF card
15	Enroll MF card
16	Register MF card
17	Delete MF card registration
18	Clear MF card content
19	Move enrolled data into the card
20	Copy data in the card to the machine
21	Set time
22	Delivery configuration
23	Delete entry and exit records
24	Clear administrator privilege
25	Modify access group settings
26	Modify user access settings
27	Modify access time period

28	Modify unlocking combination settings
29	Unlock
30	Enroll a new user
31	Change fingerprint attribute
32	Duress alarm

Appendix 4

Operation code	Operation object 1	Operation object 2	Operation object 3	Operation object 4
2	If 1:1 authentication is used, this is user ID.			
3	Alarm	For alarm causes, see Appendix 5.		
5	Sequence number of modified setting item	Value after modification		
6	User ID	Sequence number of the fingerprint	Length of the fingerprint template	
9	User ID			
10	User ID			
11	User ID			
12	User ID			

Appendix 5

Alarm reason	Meaning
50	Door Close Detected
51	Door Open Detected
53	Out Door Button
54	Door Broken Accidentally
55	Machine Been Broken
58	Try Invalid Verification

65535	Alarm Cancelled

Appendix 6

Protocol version rules

- **Released version of the protocol:**

2.2.14
2.3.0
2.4.0
2.4.1
Encryption protocol version: 2.4.0 and above

- **Device end:**

The device pushes the protocol version currently used by push to the server through the following protocol
GET /iclock/cdata?SN=${SerialNumber}&options=all&pushver=${XXX}&language=${XXX}&pushcommkey=${XXX}
The server returns the release protocol version used by the server for this request and returns the protocol version to the device.
PushProtVer= XXX. If this parameter is not returned, the default protocol version used by the server is 2.2.14.
The device interacts with the lower version based on the version of the protocol used by the current push and that returned by the server.

- **Server-side:**

The server side obtains the protocol version used by push on the device side according to the following request. If there is no pushver field, then the default device USES the 2.2.14 protocol version.
GET /iclock/cdata?SN=${SerialNumber}&options=all&pushver=${XXX}&language=${XXX}&pushcommkey=${XXX}
The service side need to return which released software use protocol version:
PushProtVer = XXX
The server interacts with the lower version based on the protocol version used by the software and the one uploaded by the device.

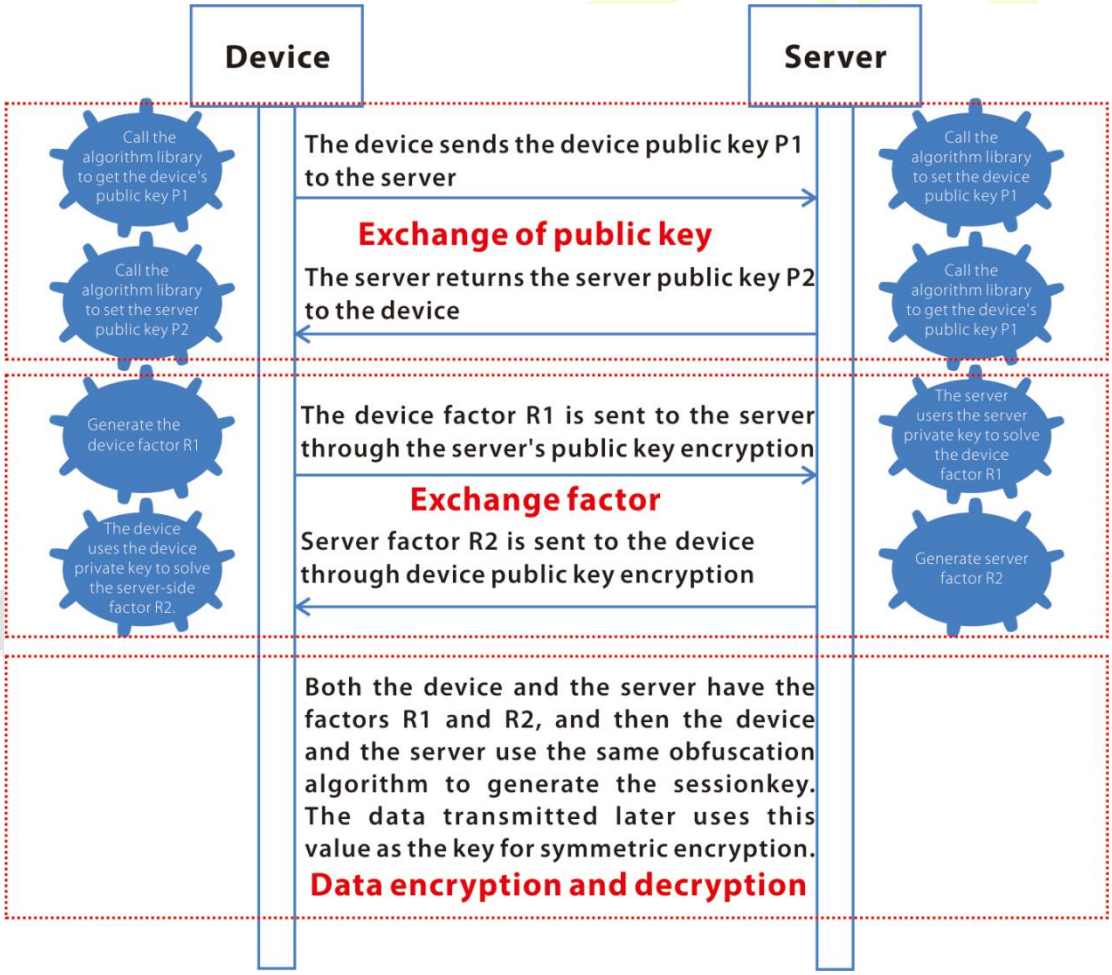
Appendix 7

Verification	Description
0	Finger vein or face or fingerprint or card or password (automatic identification)
1	Only fingerprint
2	User ID verification
3	Only password
4	Only card
5	Fingerprint or password
6	Fingerprints or card
7	Card or password
8	User ID + fingerprint
9	Fingerprint + password
10	Card + fingerprint
11	Card + password
12	Fingerprint + password + card
13	User ID + fingerprint + password
14	User ID + fingerprint or Card + fingerprint
15	Face
16	Face + fingerprint
17	Face + password
18	Face + card
19	Face + fingerprint + card
20	Face + fingerprint + password
21	Finger vein
22	Finger vein + password
23	Finger vein + card
24	Finger vein + password + card

25	Palm print
26	Palm print + card
27	Palm print + face
28	Palm print + fingerprint
29	Palm print + fingerprint + face
200	Other

Appendix 8

Data encryption key exchange scheme



- Algorithm: Encryption algorithm library will be unified packaging, the device used for the static library.
- Scheme:

  a) The asymmetric encrypted public-private key is initialized when the device and server reconnect.

b) The device and server exchange public keys:

- The device sends the device public key P1 to the server.
- The server returns the server public key P2 to the device.
- Complete the public key exchange. Both the device and the server have public keys P1 and P2.

c) Device and server exchange factors:

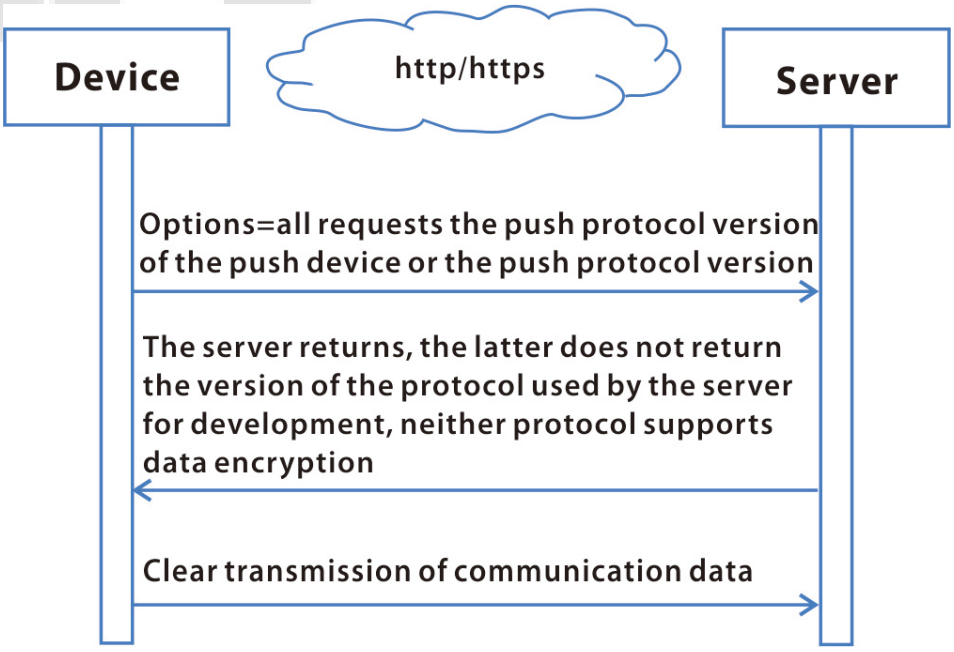
- The device generates the factor R1 and sends it to the server via the server's public key encryption.
- The server uses the server private key to solve the device factor R1.
- The server generates factor R2 and sends it to the device through the device's public key encryption.
- The device uses the device private key to solve the server factor R2.
- Complete the factor exchange. Both the device and the server have factors R1 and R2.

d) Device and server at the same time have factor R1, R2, and then confused device and a server using the same algorithm was born into a session key (sessionKey), after the transfer of data to value as the symmetric encryption keys.

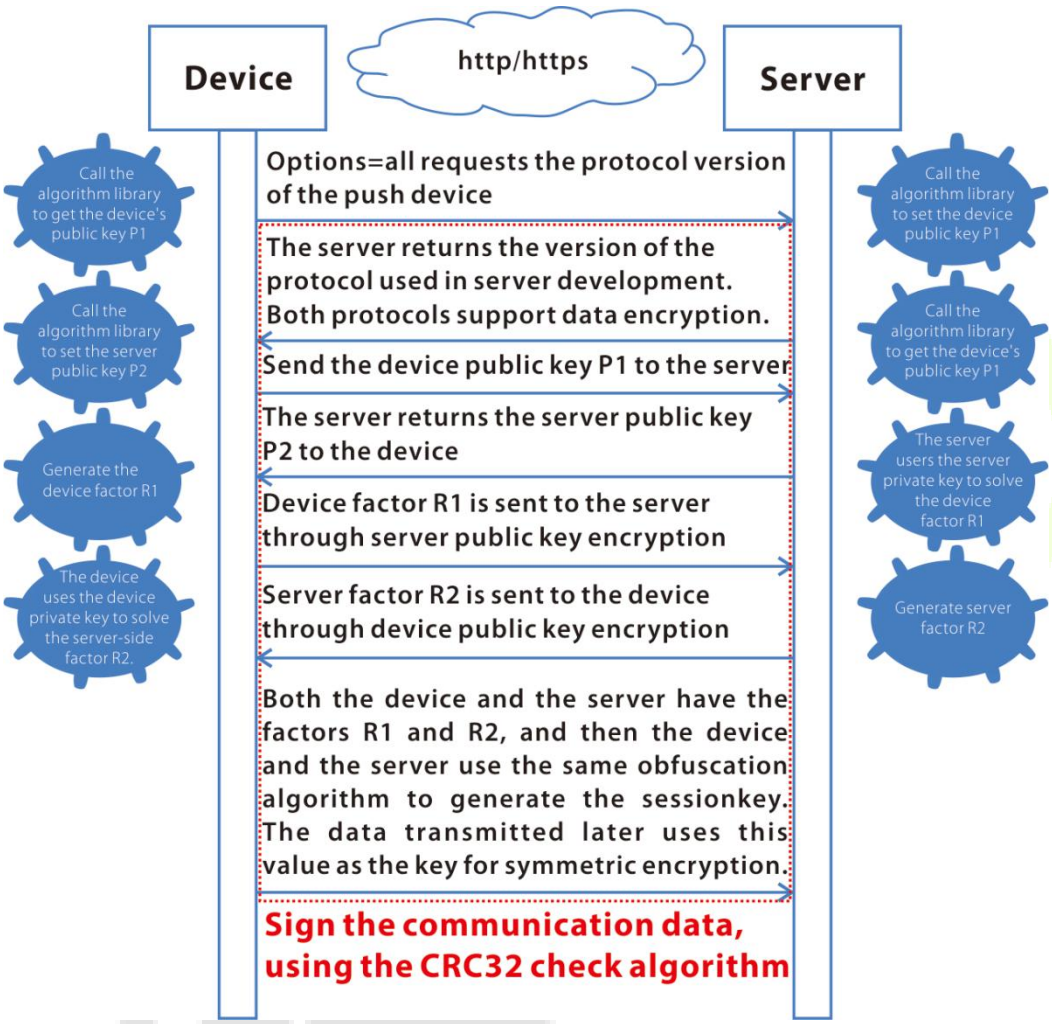
Compatibility scheme

Compatibility is achieved according to the protocol version used by the device and server, as follows:

- Case 1



- Case 2



Annotation:

a) The device determines whether to use HTTPS or HTTP based on the server address set.

b) In the first request protocol header of the existing device, pushver field is added to the current communication protocol version number of the device, and PushProtVer is added to the data returned by the software to indicate which protocol version the software was developed on. The device and server take the lowest protocol version and communicate according to the lowest protocol version.

Case 1: When protocol versions of both the server and the device are not supported, explicit transmission of data communication is used.
Case 2: Set a protocol version that supports data encryption. When both the server and the device support the protocol version, use the data encryption scheme.

The order of interaction is as follows:

➢ The new protocol exchanges the public keys P1 and P2 of the device and server.

- The new protocol exchanges the factors R1 and R2 of the device and server.
- Crc32 verification is carried out for the signature of communication data. Both the device and the server have factors R1 and R2 at the same time. Then, the device and the server use the same obconfusion algorithm to generate sessionKey (sessionKey).

Appendix 9

Error code	Description
00000000	Succeed
D01E0001	Face detection failed
D01E0002	Face occlusion
D01E0003	Lack of clarity
D01E0004	Face angle is too big
D01E0005	Live detection failed
D01E0006	Extraction template failed

According to the error code generation end + module + type + error value definition

Error generator (first)

D: error code returned by the device

S: error code returned by the software

Module (2nd ~ 3rd)

Device-end:

01: PUSH communication module

02: Template processing module

03: Hardware interaction module

04: PULL communication module

05: Offline communication module

06: Data transfer module

07: Licensing service module

Software-side:

Undetermined

Type (fourth)

E: ERROR

Error value (5th ~ 8th)

Integer data

Appendix 10 Biometric Type Index Definition

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	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 template and face template.
MultiBioVersion	Supported algorithms	The type is defined bit by bit. Different types are separated by colons, 0 means not supported, non-0 means supported version number. Such as: 0: 10: 0: 7: 0: 0: 0: 0: 0: 0, indicating support for fingerprint algorithm10.0 and near-infrared face algorithm7.0.
MaxMultiBioDataCount	Supports maximum number of bio-templates.	The type is defined bit by bit. Different types are separated by colons, 0 means not supported, non-0 means supported maximum capacity. 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.
MaxMultiBioPhotoCount	Supports maximum number of biometric photos.	The type is defined bit by bit. Different types are separated by colons, 0 means not supported, non-0 means supported maximum capacity. 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.
MultiBioDataCount	The current capacity of bio-templates	The type is defined bit by bit. Different types are separated by colons. 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.
MultiBioPhotoCount	The current capacity of biometric photos	The type is defined bit by bit. Different types are separated by colons. 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.

ZKTeco Industrial Park, No. 26, 188 Industrial Road,

Tangxia Town, Dongguan, China.

Phone : +86 769 - 82109991

Fax : +86 755 - 89602394

www.zkteco.com