1.Commonly used contents in this protocol

1.1 Terms used in the protocol

machine	Attendance machine that is connected to internet and communicate with HTTP server.
operator	The man who submit command to specific machine. The operator sit at PC and send command to HTTP server and wait for result of that command through WEB.
operator command	The command which operator submit to machine。 For example, setting time or changing the user name registered on specific machine are all operator command.
enroll data	The data used in identifying user registered on machine. For example, the fingerprint data, face data, ID card number, password, etc.
transaction id	The operator command is received by machine and executed on the machine and the result is sent by the machine. To track each command for a specified machine, introduce transaction identifier. When an operator submit new command to a specific machine, first generate globally unique identifier to track that command. When

1.2 Communication between machine and HTTP server based on HTTP

All requests that a machine send to HTTP server are POST mode request.
Using binary data transfer method of POST mode, the machine send data to server.
The response corresponding this POST request also contains binary data in its body part.

1.3 The format of binary data used in request and response

The format of binary data used in HTTP request and response is as following.
The JSON string is placed first and arbitrary number of binary data is placed after the string data.

Length of JSON string (4 byte)	JSON string (len_str)	Length of first binary data (4 byte)	First binary data (len_bin_1)	…	Length of k th binary data (4 byte)	K th binary data (len_bin_n)

The string is in JSON format and encoded in UTF-8 format.
It is true that it’s possible to represent any data using JSON format. But using JSON format it’s somewhat space wasteful to represent binary data and increased data could influence transfer speed.
So in this protocol we use JSON string as possible as we can but in special cases we use binary data directly.

When binary data is required we place special mark in JSON string that inform where corresponding binary data is placed after the string.
For example, if the field value of JSON string is “BIN_k”, this explain that corresponding binary data is placed at k th position after string.
“k” is the position of binary data. The position value is start from 1.

Example :
{“log_array”:”BIN_1”}
This JSON string show that binary data containing log information is placed first position after the string.

1.4 The communication flow between machine and server

The communication flow between machine and server is largely classified by 2 mode.
One mode is that machine receives the command submitted by operator and send result to server.
The other is that machine notifies server that special event has occurred in machine(e.g, new log created or new enroll data created).

2. Processing operator command

When required, operator submit a command to a selected machine. That command is saved in database on HTTP server.
After successful submit of command, operator poll the result of his command repeatedly.
When command is completed successfully the status of command is changed as ‘RESULT’.
The detailed step is as following.

1) Operator select machine to submit command and get device id. (device_id)
   This time the following information are saved in database on WEB server.
   Transaction ID (trans_id),
   Machine ID (device_id),
   Command code (cmd_code),
   Parameter data required to execute command (cmd_param),
   Command status (trans_status),
   Last update time of transaction (trans_status_update_time)

2) In each specified interval, machine poll the database of WEB server to see if there is a command submitted to itself.
   If exist a command, fetch that command and execute and send result information to WEB server.

3) Operator also poll the database repeatedly. If the status of command show that command has been ended, operator can see the result of that command.

The trans_id is the transaction ID. Using this ID, machine and operator are able to track the command that is processing now.

The flow of processing is as figure 2-1.

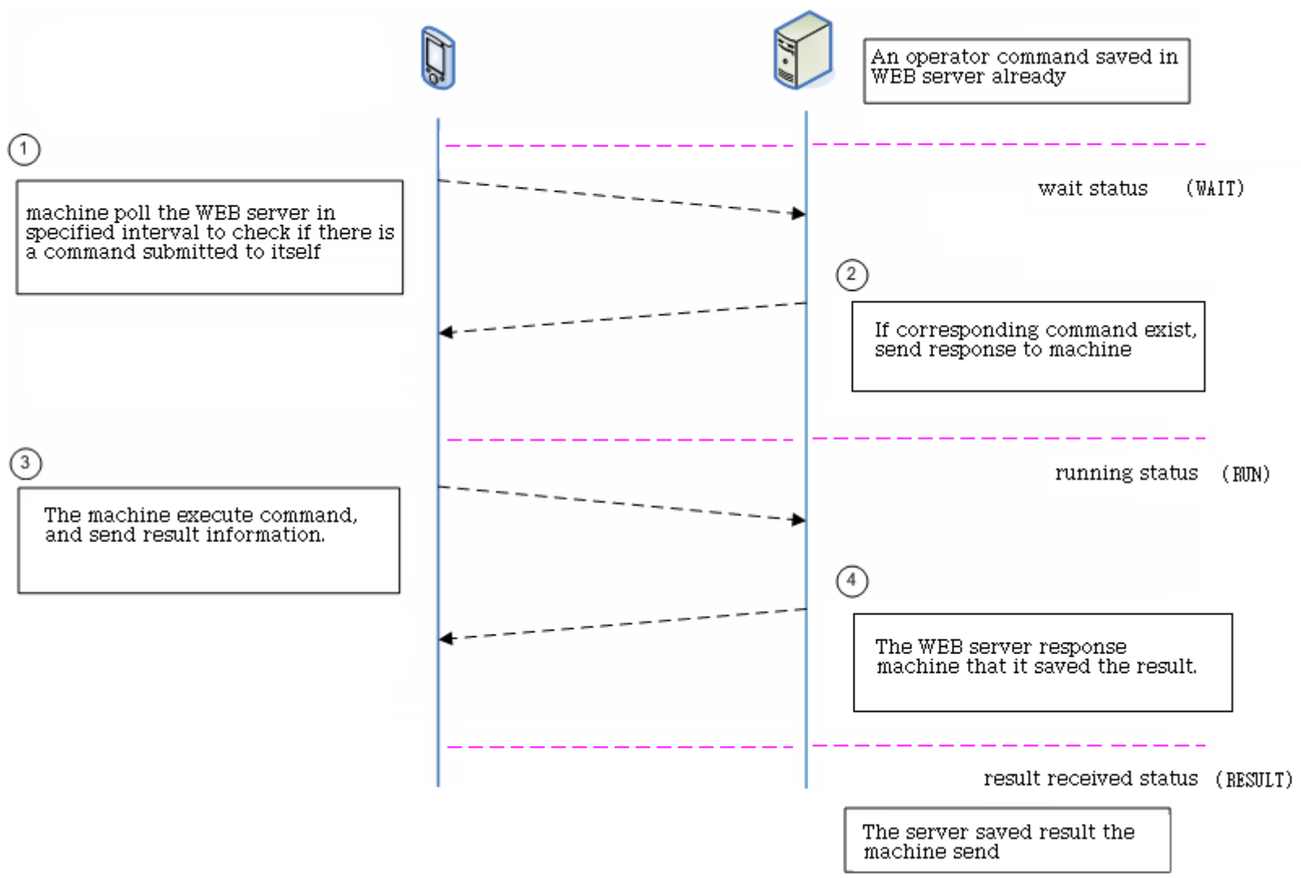


Figure 2-1. The flow of command processing.

When the machine send command result to server, if result data is very large(more than 10KB), it can send result data in several resuests.
This time the machine divide result data in several blocks and send each block in one HTTP resuest and the server receive each block, save it in temporary buffer until receives last block.
If the server receives last block(block sequence number is 0) it combines blocks received before and create result data identical to original data on the machine.
The flow of this process is as figure 2-2.

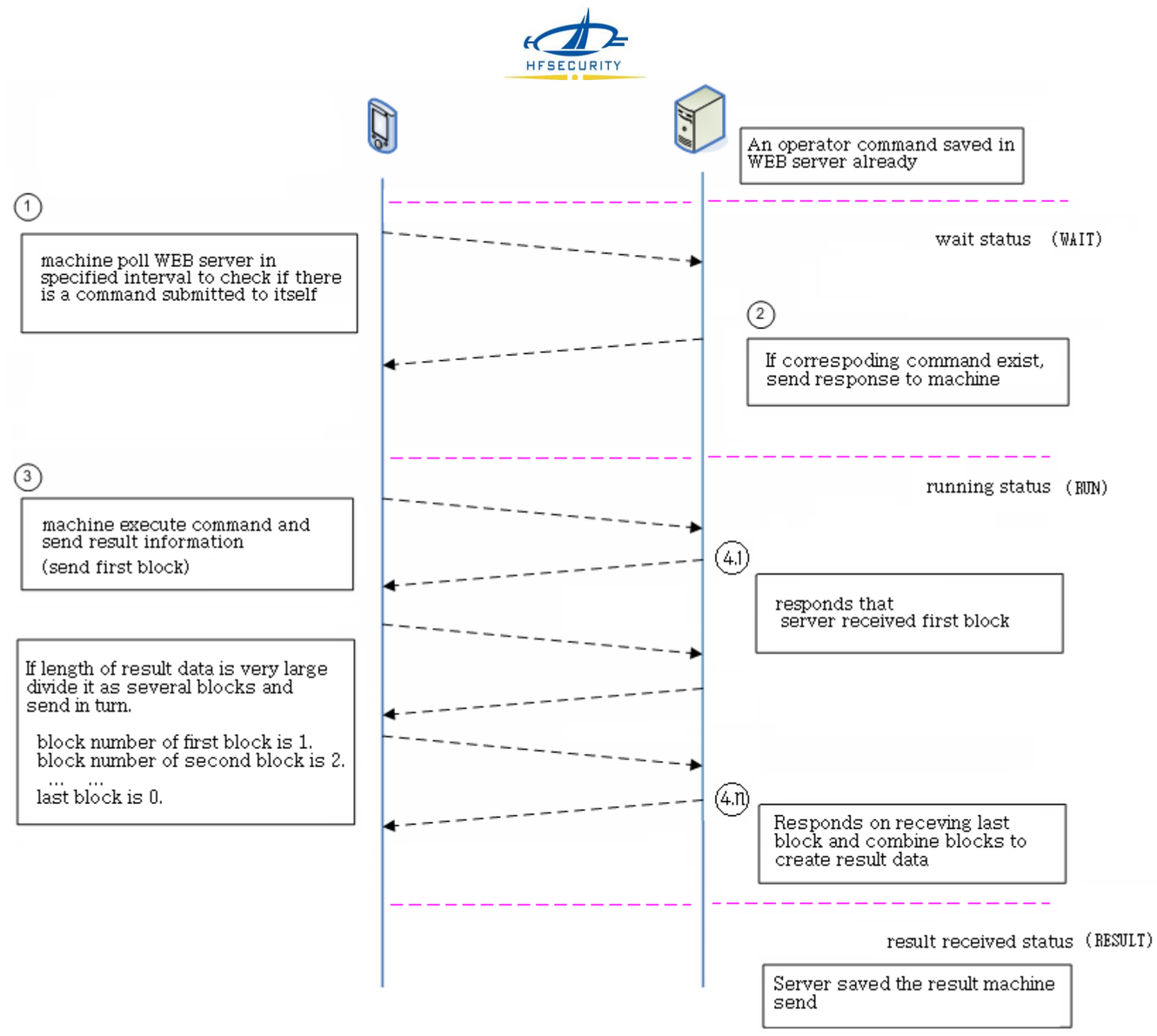


Figure2-2. If length of result data is very large, divide it as several block and send in turn.

The flow of receiving and executing command and the format of HTTP request and response are almost same but the detailed contents are different on commands.

2.1 The HTTP request and response to receive operator command for a machine

A machine send HTTP request to WEB server and receive response in specified interval to receive command submitted for itself. Detailed format are as following.

2.1.1 Request that the machine send to receive command

A machine send HTTP POST request to WEB server in every specified interval.
This time the following fields are placed in the header part of POST request.

field name	meaning	necessity	value range	detailed description
request_code	request code	must exist	The value must be following string. “receive_cmd”	Represent that this request is to receive command submitted to this machine.
dev_id	device identifier	must exist	Composed of english or digit character. Max length is 24.	The identifier of a machine. Each machine connected to the same HTTP server must have a unique ID.
Content-type	MIME type	must exist	The value must be following string. “application/octet-stream”	The MIME type of this content.
Content-Length	length of message part	must exist	Integer number。	Content-Length must be equal to the length of message in byte.

For example the following HTTP header is sent to WEB server.

```
POST / HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Connection: close
Content-Type: application/octet-stream
request_code: receive_cmd
dev_id: 001
Content-Length: 201
```

Please note above contents displayed in bold font.

In HTTP body part, datas that has format described in 1.3 are placed.
The content of body part are as following.

```
{
    "fk_name": <1>,
    "fk_time": <2>,
```

```
    "fk_info": {
        "supported_enroll_data": <3.1>,
        "fk_bin_data_lib": <3.2>,
        "firmware": <3.3>
       "firmware_filename":<3.4>,
        "fp_data_ver":<3.5>
    }
}
```

"fk_name" field : Name of a machine

"fk_time" field : Time on the machine when send this HTTP request. The format of time field is YYMMDDhhmmss.

"fk_info" field : Information that describe the features of the machine.

"supported_enroll_data" field : Represent enroll data types used to identify a user。For example, if fingerprint is used to identify a user, set as ["FP"].

"fk_bin_data_lib" field: The name of shared library used to analyze some binary data sent by machine on HTTP server.
For example, if the value of this field is "FKDataHS001", the name of library to analyze bin data that this machine send is "FKDataHS001.dll".

"firmware" field: The version string of firmware of this machine.

"firmware_filename" field: The prefix of firmware file name corresponding this machine. This information is used when download firmware file to machine.
If the prefix of file that contains new firmware is not equal with this field, that file is considered to be not valid and not downloaded.

"fp_data_ver" field: The version of the fingerprint data used in this machine. There are many versions of fingerprint data used in machines.
If fingerprint data of incorrect version is saved in machine the incorrect identification come out.
So other version of fingerprint data must be converted to correct version. This field is used when converting fingerprint data.

For example:

```
{
        "supported_enroll_data":["FP","PASSWORD","IDCARD","FACE"],
        "fk_bin_data_lib":"FKDataHS001",
        "firmware":"FK725HS001"
        "firmware_filename":"zlif105hs02_bs_um"
        "fp_data_ver":128
}
```

When the machine send HTTP request to receive operator command there is no additional binary data placed after above JSON string.

2.1.2 Response that the HTTP server send when receive operator command

If the HTTP server receive request described above, server send response as following.

Response is composed of header and body part.
The header of response contains following fields.

field name	meaning	necessity	value range	detailed description
response_code	Code showing the result.	must exist	String of eng chars or digits. Max length is 64. All the eng char is big capital.	Represent the result of getting operator command. OK : success ERROR : fail to get operator command.
trans_id	transaction identifier	optional	String of eng chars or digits. Max length is 16.	Identifier to track the process of a command.
cmd_code	command code	optional	String of eng chars or digits. Max length is 32. All the eng char is big capital.	Code to identy various commands. ex: GET_ENROLL_DATA
Content-type	MIME type	must exist	Must be following string. “application/octet-stream”	The MIME type of this content.
Content-Length	length of message part	must exist	Integer number.	Content-Length must be equal to the length of message in byte.

For example） The following text may be send to client from server.

```
HTTP/1.1 200 OK
Cache-Control: private
Server: Microsoft-IIS/7.5
Set-Cookie: ASP.NET_SessionId=453lmc45jaelft45glb2mdre; path=/; HttpOnly
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Wed, 10 Dec 2014 04:47:42 GMT
Connection: close
Content-Length: 39
Content-Type: application/octet-stream
response_code: OK
trans_id: 201
cmd_code: GET_ENROLL_DATA
```

The body part of server response would be different for various commands.

2.2 The HTTP request and response to send result of executing command on a machine

After executing a command, the machine send HTTP request and receive response to report the result of command to server. Detailed format are as following:

2.2.1 Request to send result of executing command

The machine send following HTTP POST request to WEB server.
This time the following fields are placed in header part.

field name	meaning	necessity	value range	detailed description
request_code	Type of http request	must	Must be following string. “send_cmd_result”	Represent that this request is to send result of executing command.
dev_id	Identifier of a machine	must	String of eng chars or digits. Max length is 24.	Refer to 2.1.1
trans_id	Identifier of a transaction	must	String of eng chars or digits. Max length is 16.	Refer to 2.1.1
cmd_return_code	Code that explain the result concisely.	must	String of eng chars or digits. Max length is 64.	Represent the result of a command. When command executed successfully value of this field is “OK”. When some error occurred, value is the string starting “ERROR”.
blk_no	Block number of big result data	option	Integer	This field value is the block number of big result data. If the result data is very big, divide it into some blocks and assign number to each block starting 1 for first block. The last block number is 0. The size of each block is equal to the value of Content_Length field.
Content-type	MIME type	must	Must be following string. “application/octet-stream”	The MIME type of this content.
Content-Length	length of message part	option	Integer	Content-Length must be equal to the length of message in byte.

The body part of this request would be different for various commands.

2.2.2 Response that the HTTP server send when send the result of executing command

After the web server receive the result data the machine send, it send response as following.
The header part of this response is as following.

field name	meaning	necessity	value range	detailed description
response_code	Code that explain the response.	must	String of eng chars or digits. Max length is 64. The English characters must be big characters.	Represent whether the WEB server received the result data the machine send successfully. OK : success ERROR : fail
trans_id	transaction identifier	must	String of eng chars or digits. Max length is 16.	Refer 2.1.2

3. Detailed contents of request and response according to each operator command

By this time, the supported operator commands are as following.

Description of command	Code of command
Get one enroll data of registered user from machine	GET_ENROLL_DATA
Set one enroll data of registered user to machine	SET_ENROLL_DATA
Synchronize time with server	SET_TIME
Reset machine	RESET_FK
Delete a user	DELETE_USER
Set the name of a user	SET_USER_NAME
Set the privilege of a user	SET_USER_PRIVILEGE
Enable/Disable enroll user function on machine	ENABLE_ENROLL_FUNC
Get the list of enroll number registered on machine	GET_USER_ID_LIST
Get the log list existing on machine	GET_LOG_DATA
Enable user to be identified on a machine	ENABLE_USER
Set the name of a machine	SET_FK_NAME
Clear all log data	CLEAR_LOG_DATA
Clear all enroll data	CLEAR_ENROLL_DATA
Get the status information of a machine	GET_DEVICE_STATUS
Set all information of a user to machine	SET_USER_INFO
Get all information of a user from machine	GET_USER_INFO
Change the WEB server information	SET_WEB_SERVER_INFO
Update firmware of a machine	UPDATE_FIRMWARE

3.1. Get one enroll data of registered user from machine (GET_ENROLL_DATA)

Get a kind of enroll data of registered user on a machine.

Request the machine send		Response the WEB server send

-- HTTP header -- request_code: receive_cmd dev_id: <2> -- HTTP body -- JSON string representing the device information.		
		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: GET_ENROLL_DATA -- HTTP body -- {<4>} format of <4> {"user_id":"<4.1>","backup_number":<4.2>} user_id : Identifier of registered user. backup_number : Number to identify the kind of enroll data. May be one of the following value. 0 ~ 9 : Ten fingerprint data of a user 10 : Password of a user 11 : ID card number of a user 12 : Face data of a user

-- HTTP header - request_code: send_cmd_result dev_id:<2> trans_id:<3> cmd_return_code:<4> blk_no:<5> The value of <4> could be as following. OK : Success to get specified enroll data registered on the machine. EEROR_NOT_EXIST : Not exist specified enroll data. This time there are no data in body part. The value of <5> is the block number of split result data. After success of this command, the size of result data would be different according to the kind of enroll data(fingerprint, password, face). Sometimes the result data is very large. e,g, the size of face data is about 20KB. So this time divide the result data into some blocks and transmit separately. -- HTTP body -- The following data is split into several blocks. <6> + bin_1 In <6> there are placed following JSON string. {"enroll_data":"BIN_1"}		
		-- HTTP header -- response_code: <1> trans_id: <2>

3.2. Set one enroll data of registered user to machine (SET_ENROLL_DATA)

Set a kind of enroll data saved in business database to a machine.

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2> -- HTTP body -- JSON string representing the device information.		

		-- HTTP header --
		response_code: <1>
		trans_id: <2>
		cmd_code: SET_ENROLL_DATA
		
		-- HTTP body --
		{<4>} + bin_1
		
		The format of <4>
		{
		"user_id":"<4.1>",
		"backup_number":<4.2>,
		"<4.3>":"BIN_1"
		}
		
		About meaning of each field refer 3.1。

-- HTTP header -- request_code: send_cmd_result dev_id: <2> trans_id:<3> cmd_return_code:<4> blk_no:<5> -- HTTP body -- After executing this command there are no result data, so body part is empty.		
		-- HTTP header -- response_code: <1> trans_id: <2>

3.3. Synchronize time with server (SET_TIME)

Synchronize the time of a machine with WEB server.

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2>		

-- HTTP body -- JSON string representing the device information.		
		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: SET_TIME -- HTTP body -- {<4>} The format of <4> {"time":"<4.1>"} In <4.1> place the string representing the time. The format of time string is YYYYMMDDhhmmss.
-- HTTP header -- request_code: send_cmd_result dev_id: <2> trans_id:<3> cmd_return_code:<4> blk_no:<5> -- HTTP body -- After executing this command there are no result data, so body part is empty.		
		-- HTTP header -- response_code: <1> trans_id: <2>

3.4. Reset machine (RESET_FK)

Sometimes there are needs to reset(reboot) the machine. This time submit this command.
For example the status of a command is “RUN” and never change.
If it’s impossible to suspect the reason of this status, it may be the best option to reboot the machine.
After reboot, the machine ignore all commands executing before and try to receive new command.

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2> -- HTTP body --		

表示机器信息的字符串		
		-- HTTP header -- response_code: RESET_FK trans_id: <2>

3.5. Delete a user (DELETE_USER)

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2> -- HTTP body -- JSON string representing the device information.		
		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: DELETE_USER -- HTTP body -- {<4>} The format of <4> {"user_id":"<4.1>"} user_id : user identifier to delete
-- HTTP header -- request_code: send_cmd_result dev_id: <2> trans_id:<3> cmd_return_code:<4> blk_no:<5> -- HTTP body -- After executing this command there are no result data, so body part is empty.		
		-- HTTP header -- response_code: <1>

		trans_id: <2>

3.6. Set the name of a user (SET_USER_NAME)

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2> -- HTTP body -- JSON string representing the device information.		

		-- HTTP header --
		response_code: <1>
		trans_id: <2>
		cmd_code: SET_USER_NAME
		
		-- HTTP body --
		{<4>}
		
		The format of <4>
		
		{
		"user_id":"<4.1>",
		"user_name":<4.2>
		}
		user_name ： Name data of a user encoded in UTF-8.

-- HTTP header -- request_code: send_cmd_result dev_id: <2> trans_id:<3> cmd_return_code:<4> blk_no:<5> -- HTTP body -- After executing this command there are no result data, so body part is empty.		
		-- HTTP header -- response_code: <1> trans_id: <2>

3.7. Set the privilege of a user (SET_USER_PRIVILEGE)

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2> -- HTTP body -- JSON string representing the device information.		
		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: SET_USER_PRIVILEGE -- HTTP body -- {<4>} The format of <4> {"user_id":"<1>","user_privilege":"<2>"} user_privilege : Represent the privilege to operate machine. Can be one of the following. MANAGER : Super mananger REGISTER : A person responsible to enroll other user. OPERATOR : A person to carry out some administrative operation. USER : Normal user
-- HTTP header -- request_code: send_cmd_result dev_id: <2> trans_id:<3> cmd_return_code:<4> blk_no:<5> -- HTTP body -- After executing this command there are no result data, so body part is empty.		
		-- HTTP header -- response_code: <1> trans_id: <2>

3.8. Enable/Disable enroll user function on machine (ENABLE_ENROLL_FUNC)

Request the machine send		Response the WEB server send

-- HTTP header -- request_code: receive_cmd dev_id: <2> -- HTTP body -- JSON string representing the device information.		
		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: ENABLE_ENROLL_FUNC -- HTTP body -- {<4>} The format of <4> {"enable_flag":<5>} enable_flag : Flag to enable or disable enroll function on a machine. ON : Enable enroll function (big capital) OFF : Disable enroll function
-- HTTP header -- request_code: send_cmd_result dev_id: <2> trans_id:<3> cmd_return_code:<4> blk_no:<5> -- HTTP body -- After executing this command there are no result data, so body part is empty.		
		-- HTTP header -- response_code: <1> trans_id: <2>

3.9. Get the list of enroll number registered on machine (GET_USER_ID_LIST)

Get the list of enroll data numbers registered on the machine. A enroll data number is consisted of user id and backup number.

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2>		

-- HTTP body -- JSON string representing the device information.		
		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: GET_USER_ID_LIST -- HTTP body -- This command does not contain parameters. So there are no data in body part.
-- HTTP header - request_code: send_cmd_result dev_id:<2> trans_id:<3> cmd_return_code:<4> blk_no:<5> After executing this command, the result data would contain id number list, and sometimes contain id list of 10 thousand users, and the size of list would be very large. This time the result data would be divided into small data blocks. -- HTTP body -- The data as following would be split into small data blocks. {<6>} + bin_1 In <6> there is placed as following JSON string. {“user_id_count”:<6.1>,”one_user_id_size”:<6.2>,”user_id_array”:”BIN_1”} user_id_count : Total number of enroll data ids contained in list. one_user_id_size : The size of one enroll data id structure. user_id_array : The value of this field must be “BIN_1”. The corresponding binary data after this JSON string contains actual enroll data id list. The format of each enroll data id structure placed in binary data is different on different machine type. To analyze correctly this structure, corresponding specific library must be used. The name of library is contained in device information sent by machine when the machine try to get command submitted to itself.		
		-- HTTP header -- response_code: <1> trans_id: <2>

3.10. Get the log list existing on machine (GET_LOG_DATA)

从机器获取某一个时间段之内的机器记录数据

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2> -- HTTP body -- JSON string representing the device information.		
		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: GET_LOG_DATA -- HTTP body -- {<4>} The format of <4> {"begin_time":" <1>","end_time":" <2>" } The format of time value is of YYYYMMDDhhmmss. "begin_time" field is empty or this field does not exist, get all log data before "end_time" field value. "end_time" field is empty or this field does not exist, get all log data after "begin_time" field value. If two field are all not present, get all log data recorded in machine.
-- HTTP header - request_code: send_cmd_result dev_id:<2> trans_id:<3> cmd_return_code:<4> blk_no:<5> After executing this command, the result data sometimes would contain all log data in machine and the size of result data would be very large. This time the result data would be divided into small data blocks. -- HTTP body -- The data as following would be split into small data blocks. {<6>} + bin_1 In <6> there is placed as following JSON string. {"log_count":<6.1>,"one_log_size":<6.2>,"log_array":"BIN_1"}		

log_count : Total log count obtained by executing this command. one_log_size : The size of one log structure. log_array : The value of this field must be “BIN_1”. The corresponding binary data after this JSON string contains actual log data array. The format of each log data structure placed in binary data is different on different machine type. To analyze correctly this structure, corresponding specific library must be used. The name of library is contained in device information sent by machine when the machine try to get command submitted to itself.		
		-- HTTP header -- response_code: <1> trans_id: <2>

3.10. Get the log list existing on machine (GET_LOG_DATA_EXT)

1. GET_LOG_COUNT

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2> -- HTTP body -- JSON string representing the device information.		
		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: GET_LOG_DATA_EXT -- HTTP body -- {<4>} The format of <4> {"begin_time":”<1>”,"end_time":”<2>”,” command” : GET_COUNT } The format of time value is of YYYYMMDDhhmmss. "begin_time" field is empty or this field does not exist, get all log data before "end_time" field

		value. "end_time" field is empty or this field does not exist, get all log data after "begin_time" field value. If two field are all not present, get all log data recorded in machine. 1: GET_COUNT 2: READ_LOG

-- HTTP header -
request_code: send_cmd_result
dev_id:<2>
trans_id:<3>
cmd_return_code:<4>
blk_no:<5>
 	
After executing this command, the result data sometimes would contain all log data in machine and the size of result data would be very large.
This time the result data would be divided into small data blocks.
 	
-- HTTP body --
The data as following would be split into small data blocks.
{<6>}
 	
In <6> there is placed as following JSON string.
{
"command":GET_COUNT,
"total_log_count":<6.1>,
"log_frame_count":<6.2>,
"one_frame_size":<6.3>
}		

		-- HTTP header -- response_code: <1> trans_id: <2>

2. READ_LOG

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2> -- HTTP body -- JSON string representing the device information.		

		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: GET_LOG_DATA_EXT -- HTTP body -- {<4>} The format of <4> {"begin_time":"<1>","end_time":"<2>","command":READ_LOG, "log_frame_index":<3>} The format of time value is of YYYYMMDDhhmmss. "begin_time" field is empty or this field does not exist, get all log data before "end_time" field value. "end_time" field is empty or this field does not exist, get all log data after "begin_time" field value. If two field are all not present, get all log data recorded in machine.
-- HTTP header - request_code: send_cmd_result dev_id:<2> trans_id:<3> cmd_return_code:<4> blk_no:<5> After executing this command, the result data sometimes would contain all log data in machine and the size of result data would be very large. This time the result data would be divided into small data blocks. -- HTTP body -- The data as following would be split into small data blocks. {<6>} + bin_1 In <6> there is placed as following JSON string. { "command":READ_LOG, "log_frame_index":<6.1>, "log_count":<6.2>, "log_array": [{"user_id":"<6.3.1.1>","verify_mode":<6.3.1.2>,"io_mode":<6.3.1.3>,"io_time":"<6.3.1.4>","log_image":"BIN_1"}, {"user_id":"<6.3.2.1>","verify_mode":<6.3.2.2>,"io_mode":<6.3.2.3>,"io_time":"<6.3.2.4>","log_image":"BIN_2"}, … {"user_id":"<6.3.k.1>","verify_mode":<6.3.k.2>,"io_mode":<6.3.k.3>,"io_time":"<6.3.k.4>","log_		

image":" BIN_k" },] "log_next_frame_index":<6.4>, } log_array : The value of this field must be "BIN_1". The corresponding binary data after this JSON string contains actual log data array. The format of each log data structure placed in binary data is different on different machine type. To analyze correctly this structure, corresponding specific library must be used. The name of library is contained in device information sent by machine when the machine try to get command submitted to itself. "log_next_frame_index": if this value is > 0 , what cmd is GET_LOG_DATA_EXT, will be send with new trans_id and this value.		
		-- HTTP header -- response_code: <1> trans_id: <2>

3.11. Enable user to be identified on a machine (ENABLE_USER)

Enable/Disable already registered user to be identified using a machine.
When disabled user try to identify using a machine, the matching process successes but no log is recorded in machine and no log is sent to server.

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2> -- HTTP body -- JSON string representing the device information.		
		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: ENABLE_USER -- HTTP body -- {<4>}

		The format of <4> {"user_id":<1>,"enable_flag":<2>} enable_flag field can have following value. ON : Enable specific user OFF : Disable specific user
-- HTTP header -- request_code: send_cmd_result dev_id: <2> trans_id:<3> cmd_return_code:<4> blk_no:<5> -- HTTP body -- After executing this command there are no result data, so body part is empty.		
		-- HTTP header -- response_code: <1> trans_id: <2>

3.12. Set the name of a machine (SET_FK_NAME)

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2> -- HTTP body -- JSON string representing the device information.		
		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: SET_FK_NAME -- HTTP body -- {<4>} The format of <4> {"fk_name":"<4.1>"} <4.1> must be English character.

-- HTTP header -- request_code: send_cmd_result dev_id: <2> trans_id:<3> cmd_return_code:<4> blk_no:<5> -- HTTP body -- After executing this command there are no result data, so body part is empty.		
		-- HTTP header -- response_code: <1> trans_id: <2>

3.13. Clear all log data (CLEAR_LOG_DATA)

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2> -- HTTP body -- JSON string representing the device information.		
		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: CLEAR_LOG_DATA -- HTTP body -- This command has no parameter, so the body part is empty.
-- HTTP header -- request_code: send_cmd_result dev_id: <2> trans_id:<3> cmd_return_code:<4> blk_no:<5> -- HTTP body -- After executing this command there are no result data, so body part is empty.		
		-- HTTP header -- response_code: <1> trans_id: <2>

3.14. Clear all enroll data (CLEAR_ENROLL_DATA)

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2> -- HTTP body -- JSON string representing the device information.		
		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: CLEAR_ENROLL_DATA -- HTTP body -- This command has no parameter, so the body part is empty.
-- HTTP header -- request_code: send_cmd_result dev_id: <2> trans_id:<3> cmd_return_code:<4> blk_no:<5> -- HTTP body -- After executing this command there are no result data, so body part is empty.		
		-- HTTP header -- response_code: <1> trans_id: <2>

3.15. Get the status information of a machine (GET_DEVICE_STATUS)

Get the status information of a machine. For example, the number of all registered users, the number of managers or normal users, the number of enrolled fingerprints, faces, passwords or ID cards, the number of logs, etc.

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2>		

-- HTTP body -- JSON string representing the device information.		
		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: GET_DEVICE_STATUS -- HTTP body -- This command has no parameter, so the body part is empty.

-- HTTP header --
request_code: send_cmd_result
dev_id: <2>
trans_id:<3>
cmd_return_code:<4>
blk_no:<5>
 	
-- HTTP body --
{<6>}
 	
In <6>, there is placed as following JSON string.
{
“total_user_count”:<6.1>,
“user_count”:<6.2>,
“manager_count”:<6.3>,
“fp_count”:<6.4>,
“face_count”:<6.5>,
“password_count”:<6.6>,
“idcard_count”:<6.7>,
“total_log_count”:<6.8>
}
<6.1> ~ <6.8> are all integers.		

		-- HTTP header -- response_code: <1> trans_id: <2>

3.16. Set all information of a user to machine (SET_USER_INFO)

Set to a machine all information of a user such as fingerprint, face, password, ID card, name, privilege etc, saved in business database.

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd		

dev_id: <2> -- HTTP body -- JSON string representing the device information.		

		-- HTTP header --
		response_code: <1>
		trans_id: <2>
		cmd_code: SET_USER_INFO
		
		-- HTTP body --
		{<4>} + bin_1 + bin_2 + … + bin_k
		
		The format of <4>
		{
		"user_id":<4.1>,
		"user_name":<4.2>,
		"user_privilege":<4.3>,
		"enroll_data_array":
		[
		{"backup_number":<4.4.1>,"enroll_data":"BIN_1"},
		{"backup_number":<4.4.2>,"enroll_data":"BIN_2"},
		…,
		{"backup_number":<4.4.k>,"enroll_data":"BIN_k"},
		],
		"user_photo":"BIN_(k+1)"
		}
		
		user_id : user identifier.
		
		user_name : user name encoded in UTF-8.
		
		user_privilege : user privilege to operate a machine.
		
		enroll_data_array : The JSON array string containing enroll data as element such as fingerprint, face, password, idcard, etc.
		Each element of this JSOn array is JSON object having two field "backup_number", "enroll_data".
		Value of "enroll_data" field shows where the corresponding binary data is placed after JSON string.
		The binary data contains actual enroll data.
		
		user_photo : On some machine, save the photo image of a user.
		Value of "user_photo" field shows where the photo image is placed after JSON string.

-- HTTP header -- request_code: send_cmd_result dev_id: <2> trans_id:<3> cmd_return_code:<4> blk_no:<5> -- HTTP body -- After executing this command there are no result data, so body part is empty.		
		-- HTTP header -- response_code: <1> trans_id: <2>

3.17. Get all information of a user from machine (GET_USER_INFO)

Get from a machine all information of a user such as fingerprint, face, password, ID card, name, privilege etc.

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2> -- HTTP body -- JSON string representing the device information.		
		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: GET_USER_INFO -- HTTP body -- {<4>} The format of <4> {"user_id":"<1>"} user_id : user identifier to get information from a machine.

-- HTTP header --
request_code: send_cmd_result
dev_id: <2>
trans_id:<3>
cmd_return_code:<4>
blk_no:<5>
 	
-- HTTP body --
{<6>} + bin_1 + bin_2 + … + bin_k
 	
The format of <6> JSON string
{
"user_id":<6.1>,
"user_name":<6.2>,
"user_privilege":<6.3>,
"enroll_data_array":
[
{"backup_number":<6.4.1>,"enroll_data":"BIN_1"},
{"backup_number":<6.4.2>,"enroll_data":"BIN_2"},
…,
{"backup_number":<6.4.k>,"enroll_data":"BIN_k"},
],
"user_photo":"BIN_(k+1)"
}
The format of this result data is equal to the format of parameter data of SET_USER_INFO command.		

		-- HTTP header -- response_code: <1> trans_id: <2>

3.18. Change the WEB server information (SET_WEB_SERVER_INFO)

Change the IP address and port number of WEB server of this machine.
Execute this command to connect a machine to another WEB server.
Once a machine connected to another WEB server, that machine can no longer communicate with original server but communicate with new server.
After executing this command and rebooting, new settings would be in effect.

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2>		

-- HTTP body -- JSON string representing the device information.		
		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: SET_WEB_SERVER_INFO -- HTTP body -- {<4>} The format of <4> {"server_ip":"<1>","server_port":<2>} server_ip : IP4 address string of server. for example: 192.168.0.1 server_port : Port number of server.
-- HTTP header -- request_code: send_cmd_result dev_id: <2> trans_id:<3> cmd_return_code:<4> blk_no:<5> -- HTTP body -- After executing this command there are no result data, so body part is empty.		
		-- HTTP header -- response_code: <1> trans_id: <2>

3.19. Update firmware of a machine (UPDATE_FIRMWARE)

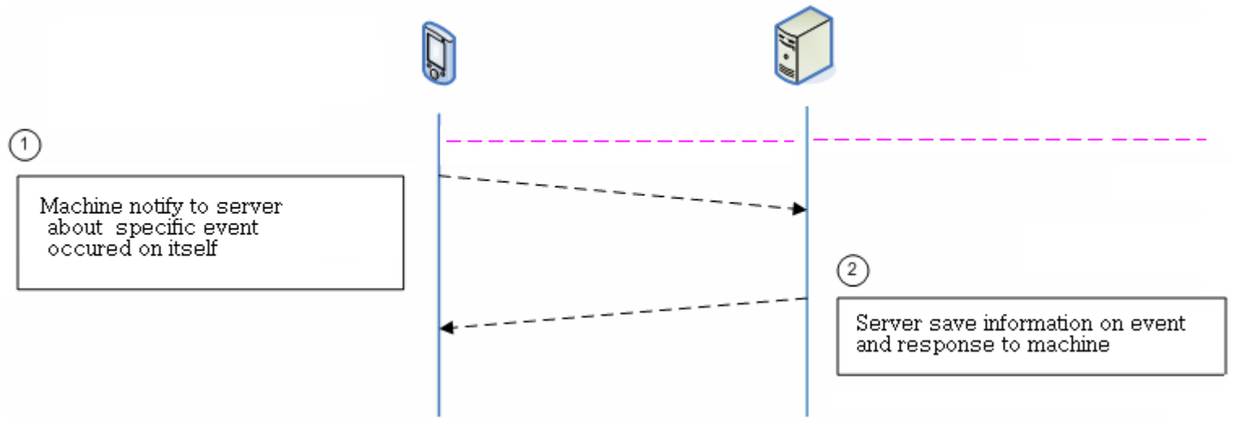
Update firmware of a machine.
After executing this command, the machine reboot automatically.

Request the machine send		Response the WEB server send
-- HTTP header -- request_code: receive_cmd dev_id: <2> -- HTTP body -- JSON string representing the device information.		

		-- HTTP header -- response_code: <1> trans_id: <2> cmd_code: UPDATE_FIRMWARE -- HTTP body -- {<4>} + bin_1 The format of <4> {"firmware_file_name":<1>,"firmware_bin_data":"BIN_1"} firmware_file_name : The filename of firmware to update. Each machine recognizes pre-specified firmware file name. If this value is not match, the firmware is not updated. firmware_bin_data : Shows that after the JSON string, corresponding binary data is placed.
-- HTTP header -- request_code: send_cmd_result dev_id: <2> trans_id:<3> cmd_return_code:<4> blk_no:<5> -- HTTP body -- After executing this command there are no result data, so body part is empty.		
		-- HTTP header -- response_code: <1> trans_id: <2>

4. Real-time transmission

When certain events occur on a machine, use real-time transmission to notify specific information to the WEB server.
Real-time transmission, not similar to the operator command process, but machine send HTTP request to WEB server in active and receive the response.
In real-time transmission, there are two types by this time.



4.1 Real-time log transmission

Each time a new log recorded on a machine(for example a user is identified using fingerprint), the new log information is transferred to WEB server.

Request the machine send		Response the WEB server send
-- HTTP header - request_code: realtime_glog dev_id:<2> -- HTTP body -- <3> + bin_1 In <3>, JSON string representing log information is placed. Some machines also contain log image captured on identification time, so the body part can contain binary data. The format of <3> is as following. { "user_id":"<3.1>", "verify_mode":<3.2>, "io_mode":<3.3>, "io_time":" <3.4>" ,		

"log_image":" BIN_1" } user_id : The id of user recorded log. verify_mode : The verify mode used when record log. Value of this field can be following JSON array. The order of element in this array represent the order of identification method. For example, [“FP”, “PASSWORD”] shows that first identified by using fingerprint and sfter by using password. io_mode : The purpose why a user passed this machine. (For attendance or leave) io_time : Time a user passed. Format of time value is of YYYYMMDDhhmmss. log_image: Shows that log image data is placed after this JSON string.		
		-- HTTP header -- response_code: <1> The value of <1> shows that the information a machine sent are saved successfully on server database. OK : success ERROR : fail If the machine receive response_code as ERROR, the machine consider that transmission is failed and retry this transmission until receive OK response_code.

4.2 Real-time enroll data transmission

Each time a new enroll data of a user is registered on a machine, the machine send that enrolled data to WEB server.

Request the machine send		Response the WEB server send
-- HTTP header - request_code: realtime_enroll_data dev_id:<2> -- HTTP body -- <3> + bin_1 + bin_2 + … + bin_k In <3>, JSON string representing enrolled data is placed. The format of <3> is as following. {		

"user_id":"<3.1>", "user_name":"<3.2>", "user_privilege":"<3.3>", "enroll_data_array": [{"backup_number":<3.4.1>,"enroll_data":"BIN_1"}, {"backup_number":<3.4.2>,"enroll_data":"BIN_2"}, …, {"backup_number":<3.4.k>,"enroll_data":"BIN_k"},], "user_photo":"BIN_(k+1)" } The format of <3> is equal to the format of parameter data of SET_USER_INFO command.		
		-- HTTP header -- response_code: <1> The value of <1> shows that the information a machine sent are saved successfully on server database. OK : success ERROR : fail If the machine receive response_code as ERROR, the machine consider that transmission is failed and retry this transmission until receive OK response_code.