Programming Guide

WISE-PaaS/RMM 3.1

Wireless IoT Sensing Embedded Agent
WISE-Agent Programming Guide

ADVANTECH

Enabling an Intelligent Planet

Change Log:

Date	Version	Description / Major change
2015/02/21	V0.1	Scott Chang, create draft document
2015/03/08	V1.0	Scott Chang, first formal release
2015/11/9	V1.1	Scott Chang, revise content
2015/12/17	V1.2	Scott Chang, revise content
2015/12/22	V1.3	Rison Yeh, review and revise format

Table of Content

1 Introduction

WISE Agnet – a software development framework to communicate between device and RMM Server

Advantech provide a software development framework to communicate and exchange information between a device and RMM Server, called WISE Agent framework.
WISE Agent framework provides a rich set of user-friendly, intelligent and integrated interfaces, which speeds development, enhances security and makes agent application easier and simpler to communicate with RMM Server.

1.1 Benefits

- ✓ **Standardization**

  The communication protocol is based on MQTT protocol to communicate and exchange data with RMM Server. The IoT sensor data report format is followed the IPSO Spec. in JSON format.

- ✓ **Portability**

  Whole framework is written in C language and follow the ANSI C Standard, that C compilers are available for most systems and are often the first compiler provided for a new system.

- ✓ **Scalability**

  The WISE Agent Framework is functional partitioning into discrete scalable, reusable modules, and plug & playable.

1.2 Environment Requirements

1.2.1 Operating Systems

Windows XP Embedded

Windows XP Pro or Home Edition 32-bit

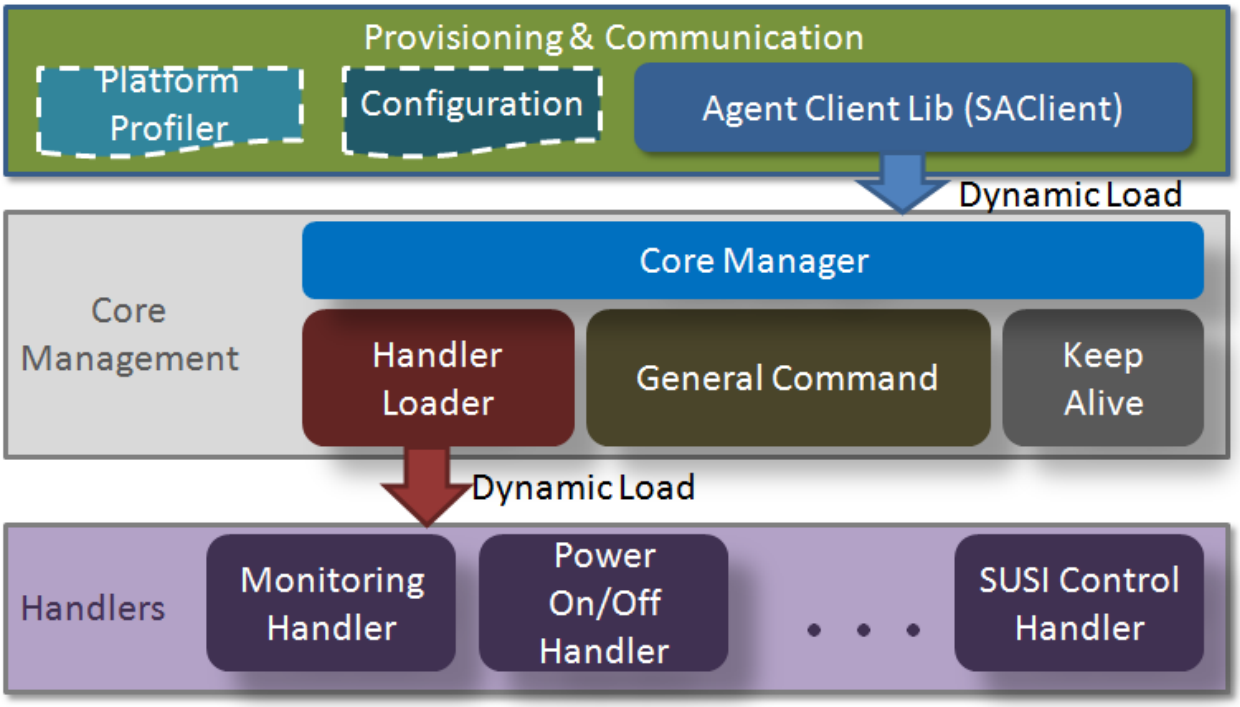
Windows 7 (x86 / x64)

WES7 (x86 / x64)

Windows 8 Desktop (x86 / x64)

2 WISE Agent Organization

2.1 Framework Architecture



WISE-Agent framework includes 3 different layers: Provisioning & Communication, Core Management and Handlers.

2.1.1 Provisioning & Communication

The Provisioning & Communication layer is the most important layer in this framework. The Provisioning & Communication layer includes one library to connect to RMM Server and two structures defining the device information and server configuration.

Agent Client Library:

The library, named 'SAClient', is the main library user needs to integrate into application, and it provides several simple API to communicate and exchange data with RMM Server. The SAClient API is described at 6 SAClient API.

Platform Profiler:

The Structure defines the Agent Profile including Agent, Platform and Custom Information. Agent Information carries the basic information about device ID, MAC Address, serial number, etc. Platform

Information carries the OS version, BIOS Version, CPU name, etc. Custom Information carries the product name, manufacturer name and device type. The structure is defined at 4.4 Profile Structure.

Configuration:

The Structure defines the Agent and Server Configuration. Agent Configuration configures the agent executing mode. Server Configuration configures the server IP, listen port, login ID and password. The structure is defined at 4.3 Configuration Structure.

2.1.2 Core Management

The Core Management layer is in charge of loading and managing the Handlers, bridging between SAClient and Handlers, and handling the commands of agent control.

Core Manager:

The core module, dynamic loaded by SAClient, takes charge of functional modules (Handler Loader, General Command and Keep Alive) integration and interact with SAClient to communicate with RMM Server.

Handler Loader:

This module will read an XML file (module_config.xml) to get the information of handlers, including the name and path of handlers and how many handlers to load. This module also manage those loaded handlers.

General Command:

This module handles all the commands to control the Agent, such as agent update, rename host name. This module also handles the commands that need to be delivered to all handlers, such as “GetCapability” to collect the capability of each handler, “AtuoReportStart” and “AutoReportStop” to control the sensor data report for every handler.

Keep Alive:

This module is the software watchdog to keep the threads of Handlers alive by calling Handler_Get_Status.

2.1.3 Handlers

The Handler layer includes all the handlers we designed for industrial personal computer (IPC).

Hardware Monitor:

This Handler collects and reports the information of main board and Hard-disk. This Handler supports SUSI 4 and SUSI 3.02 to access the main board information. It also supports the S.M.A.R.T. (Self-Monitoring, Analysis and Reporting Technology) to access the information of Hard-disk or Advantech SQFlash. But this Handler is designed for RMM server and the report data does not follow the IPSO spec.

SUSI Control:

This Handler supports SUSI IoT interface, it can get and set sensor data, such as GPIO, m2Talk, etc., and the report data follow the IPSO spec.

Hard-disk Monitor:

This Handler supports Hard-disk and Advantech SQFlash status report and follow the IPSO spec.

Software Monitor:

This Handler will monitor the CPU and Memory usage and collect the process information the process status. But this Handler is designed for RMM server and the report data does not follow the IPSO spec.

Process Monitor:

This Handler is same with Software Monitor but the report data follows the IPSO spec.

Network Monitor:

This Handler monitors all the Ethernet or Wireless network status and report sensor data which follows the IPSO spec.

Power On/Off:

This Handler supports remote power control including power on, power off, reboot and hibernate. This Handler also integrates the Intel® Active Management Technology (Intel® AMT) to improve the function of remote power control.

Protection:

This Handler integrates the McAfee tool to protect the device system. The Handler will send alert message while any invalid application is executed.

Recovery:

This Handler integrates the Acronis Backup & Recovery tool to schedule backup the system and recovery the system remotely.

Remote KVM:

This Handler integrates the ultra VNC tool to support remote access. This Handler also integrates the Intel® Active Management Technology (Intel® AMT) to improve the function of remote access.

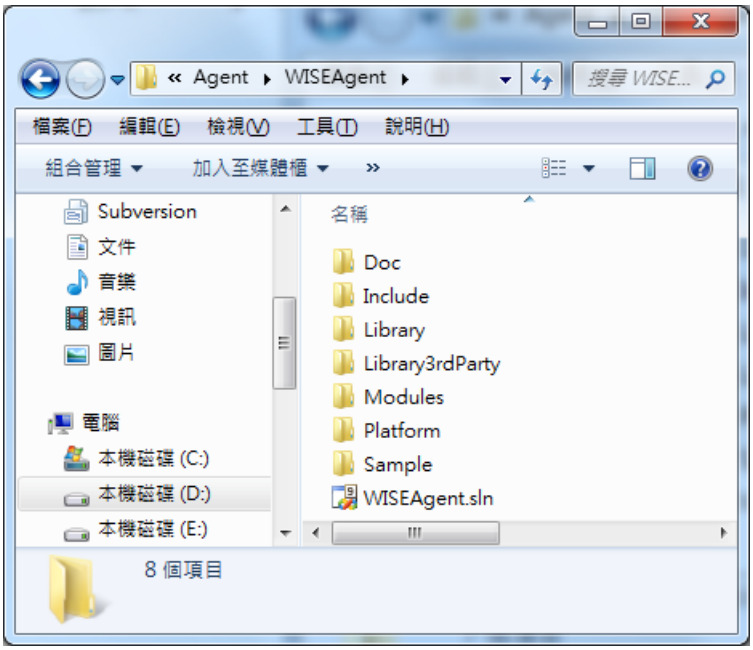
Terminal:

This Handler supports terminal command to control remote device.

Screenshot:

This Handler can take a desktop snapshot and upload to RMM Server to let user monitor current desktop status.

2.2 Directory Structure



WISE Agent Framework is released with these directory structure.

Doc:

In Doc folder, include all documents for WISE-Agent

Include:

In Include folder, there are two header file: susiaccess_def.h, to define the configuration and profile structures. And susiaccess_handler_api.h, to define the handler api.

Library:

In Library folder, include all libraries we implement for WISE-Agent, such as: SAClient, the main library of WISE-Agent, Core Manager, Handler Loader, and so on.

Library3rdParty:

In Library3rdParty folder, we provide the 3rd party libraries we used. Such as: mosquitto, openssl, etc.

Modules:

In Modules folder, there are Handlers we used in WISE Agent.

Platform:

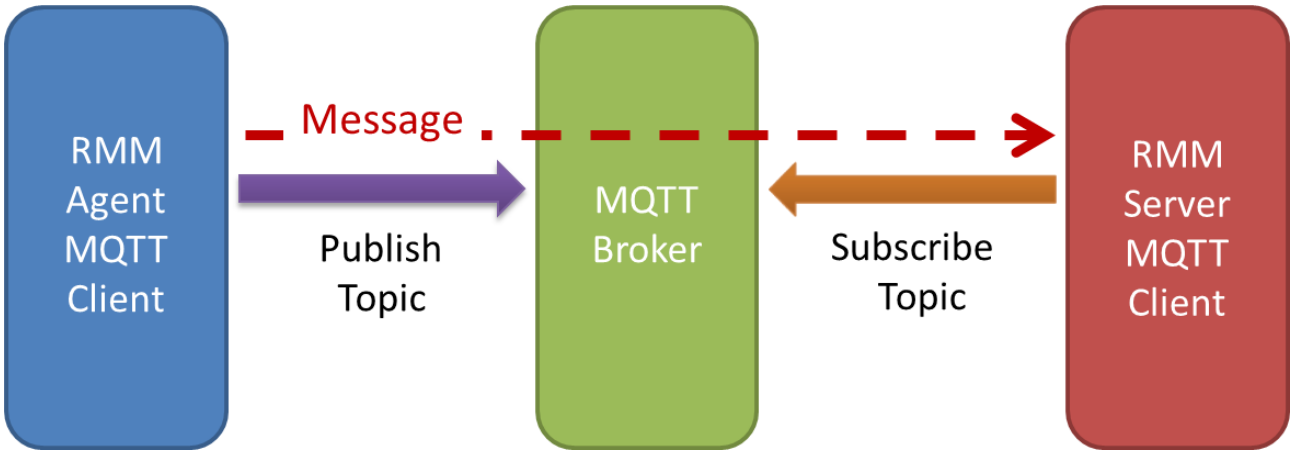
In Platform folder, we put the Cross-compiled library for windows and Linux in there.

Sample:

In Sample folder, we provide several application and handler sample projects.

3 WISE Agent Communication Protocol

The communication protocol used in WISE Agent is MQTT protocol. MQTT is publish/subscribe, extremely simple and lightweight, messaging protocol designed for constrained devices and low-bandwidth, high-latency or unreliable networks



3.1 WISE Agent Packet Format

The basic WISE Agent Packet in JSON format.

```
"susiCommData":{
   <Custom_Data, JSON format string>
   "agentID":<Agent_ID>,
   "commCmd":<Command_ID>,
   "requestID":<Request_ID, preserved for SA3.0>,
   "handlerName":<Handler_Name>,
   "sendTS":<Time_Tick, Time tick from 1977>
}
```

3.2 Agent Connection Protocol

3.2.1 Will Message

WISE Agent setup the Will Message to Broker, while connected to broker, by calling “mosquitto_will_set”. While Client lost connection or keep-alive timeout, the will message will send to the client that subscribe the topic.

Topic:

/cagent/admin/devId/willmessage

Payload:

```
"susiCommData":{
   "devID":"000014DAE996BE04",
   "hostname":"PC001104",
   "sn":"14DAE996BE04",
   "mac":"14DAE996BE04",
   "version":"1.0.0.0",
   "type":"IPC",
   "product":"",
   "manufacture":"",
   "account":"anonymous",
   "password":"",
   "status":0,
   "commCmd":1,
   "requestID":21,
   "agentID":"000014DAE996BE04",
   "handlerName":"general",
   "sendTS":1423536737}
}
```

3.2.2 Agent Information.

WISE Agent sends Agent information with “status:1” after connected to notify server the device is connected, and send “status:0” before disconnect to notify server the device will disconnect.

Topic:

/cagent/admin/<agentID>/agentinfoack

Payload:

```
"susiCommData":{
```

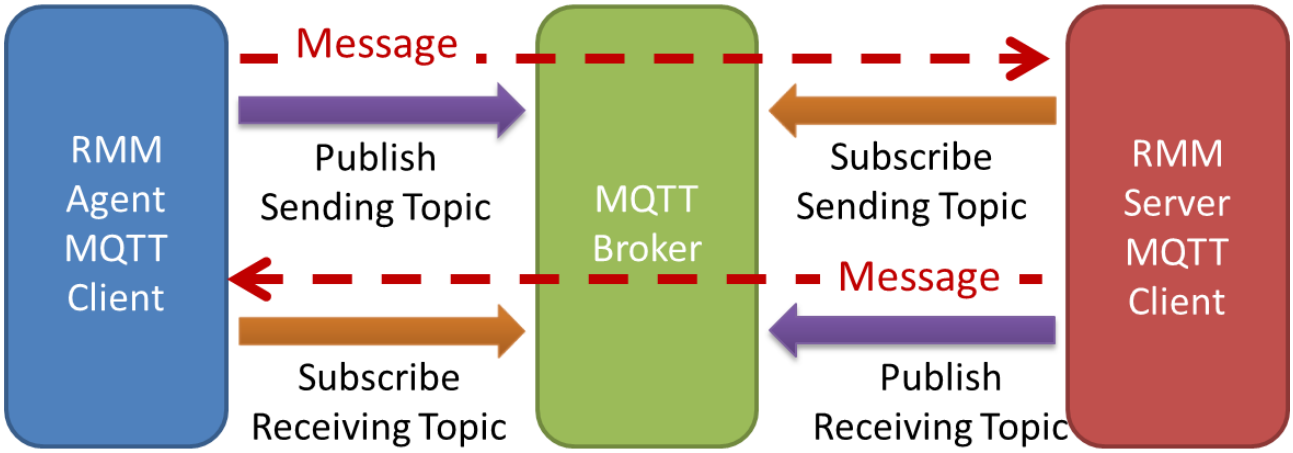
```
    "devID":"000014DAE996BE04",
    "hostname":"PC001104",
    "sn":"14DAE996BE04",
    "mac":"14DAE996BE04",
    "version":"1.0.0.0",
    "type":"IPC",
    "product":"",
    "manufacture":"",
    "account":"anonymous",
    "password":"",    //encrypt with DES and Base64
    "status":1,
    "commCmd":1,
    "requestID":21,
    "agentID":"000014DAE996BE04",
    "handlerName":"general",
    "sendTS":1423536737}
}
```

3.3 Agent Command

To communicate with RMM Server, WISE Agent defined two topics to receive command and send message.

Sending Topic: /cagent/admin/<Agent_ID>/agentactionreq

Receiving Topic: /cagent/admin/<Agent_ID>/agentcallbackreq



3.3.1 Send OS Information

WISE Agent sends the OS Information to RMM Server to record the agent executing environment.

Topic:

/cagent/admin/<agentID>/agentactionreq

Payload:

```
"susiCommData":{
  "osInfo":
  {
    "cagentVersion":"1.0.0",
    "osVersion":"Windows 7 Service Pack 1",
    "biosVersion":"",
    "platformName":"",
    "processorName":"",
    "osArch":"X64",
    "totalPhysMemKB":8244060,
    "macs":"<mac1>;<mac2>",
    "IP":"<local_IP>"
   },
  "commCmd":116,
  "requestID":16,
```

```
  "agentID":"000014DAE996BE04",
  "handlerName":"general",
  "sendTS":1424765401
}
```

3.3.2 Agent Update

RMM Server can send the Agent Update Command to ask WISE Agent to download and upgrade to newer version.

Topic:

/cagent/admin/<agentID>/agentcallbackreq

Payload:

```
"susiCommData":{
  "params":{
    "userName":"<account>", //ftp server login name
    "pwd":"<password>",     //ftp server login password
    "port":2121,            //ftp server listen port
    // download path
    "path":"/upgrade/SA30AgentSetupV3.0.999_for_V3.0.26.exe",
    // md5 checksum.
    "md5":"FC98315DEB2ACD72B1160BC7889CE29C"
  },
  "commCmd":111,
  "requestID":16,
  "agentID":"000014DAE996BE04",
  "handlerName":"general",
  "sendTS":1424765401
}
```

3.4 IoT Command

3.4.1 Get Handler Capability

RMM Server can send the Get Capability Command to get the capability of handlers in WISE Agent.

Topic:

/cagent/admin/<agentID>/agentcallbackreq

Payload:

```
"susiCommData":
{
    "requestID":1001,          //cagent_request_general
    "catalogID": 4,
    "commCmd":2051,   //general_info_spec_req
    “handlerName”:”general”
}
```

3.4.2 Send Capability

WISE Agent collects the capability of handlers and send back to RMM Server.

Topic:

/cagent/admin/<agentID>/agentactionreq

Payload:

```
"susiCommData":
{
    "infoSpec":
    {
        "HWM":    // handler name
        {
        e:
        [
            { "n": "v01", "n": "cpu1_volt", type:"volt", "u": "V", "max":20, "min":10},
            { "n": "v02", "n": "cpu2_volt", type:"volt", "u": "V", "max":20, "min":10},
            { "n": "t01", "n": "cpu1_temp", type:"temp", "u": "Cel", "max":100, "min":10},
            { "n": "V143360", "n": "OEM0", type:"current", "u": "A", "max":100, "min":10}
        ]
        “bn”:”HWM”
    }
    "requestID": 2001,   //cagent_general
```

```
    "commCmd": 2052,        //general_info_spec_rep
    "catalogID": 4,
    "agentID": "000014DAE996BE04",
    "sendTS": 1417000000004,      // time tick from 1970
    “handlerName”:”general”
}
```

3.4.3 Start Auto Report

RMM Server can send the Start Auto Report Command to ask WISE Agent report sensor data repeatedly.

Topic:

/cagent/admin/<agentID>/agentcallbackreq

Payload:

```
"susiCommData":
{
    "requestID":1001,
    "catalogID": 4,
    "commCmd": 2053,        //general_start_auto_upload_req
    "handlerName”:”general",
    "requestItems":{
      "SUSIControl":{
        "e":[
          {"n":" SUSIControl/GPIO"},
          {"n":" SUSIControl/Voltage/v1"}
        ]
      },
      "HDDMonitor":{
        "e":[
          {"n":" HDDMonitor /hddInfoList"},
          {"n":" HDDMonitor /hddSmartInfoList/PowerOnHoursPOH"}
        ]
      },
    //Report the sensor data of voltage v1 and whole GPIO in SUSI Control Handler
    //"requestItems":{"All":{}},
    // If requestItems contain string "All":{}, Whole handler should report all sensor data.
    "autoUploadIntervalSec": 30
}
```

3.4.4 Stop Auto Report

RMM Server can send the Stop Auto Report Command to ask WISE Agent stop reporting.

Topic:

/cagent/admin/<agentID>/agentcallbackreq

Payload:

```
"susiCommData":
{
    "requestID":1001,
    "catalogID": 4,
    "commCmd": 2056,        //general_stop_auto_upload_req
    “handlerName”:”general”
    "requestItems":{"All":{}}
}
```

3.4.5 Stop/Stop Auto Report Response

WISE Agent send Start/Stop AutoReport command response back to RMM Server.

Topic:

/cagent/admin/<agentID>/agentactionreq

Payload:

```
    "susiCommData":
    {
    "requestID":1001,
    "catalogID": 4,
    "commCmd": 2054,        // general_start_auto_upload_rep
    “handlerName”:”general”,
    “result”:”SUCCESS”
    }
```

3.4.6 Send Sensor Data Report

WISE Agent can send the sensor data report to RMM Server repeatedly after receive the Start Auto Report Command.

Topic:

/cagent/admin/<agentID>/deviceinfo

Payload:

```
"susiCommData":
{
    "requestID":2001,
    "catalogID": 4,
    "commCmd": 2055,        //general_info_upload_rep
    "agentID": "000014DAE996BE04",
    "sendTS": 1417000000004,    // time tick from 1970
    “handlerName”:”general”,
    “Data”:
    {
        "HWM":
        {
            "e":
            [
                { "n": "v01", "v": 34.5 },
                { "n": "v02", "v": 69.8 }
            ],
            "bn": "HWM"
        }
    }
}
```

4 Global Definitioin

susiaccess_def.h file includes the constants and flags that are required for programming.

4.1 Status Codes

The global defined status code is used in WISE Agent Framework and Handlers.

```
#define AGENT_STATUS_OFFLINE 0
```

Description

The Agent have not connect to server.

```
#define AGENT_STATUS_ONLINE 1
```

Description

The Agent is connected to server.

4.2 Variable Buffer Length

The global defines buffer length is used in WISE Agent Framework and Handlers.

```
#define DEF_FILENAME_LENGTH 32
```

Description

The buffer length of file name.

```
#define DEF_DEVID_LENGTH 32
```

Description

The buffer length of Device ID.

```
#define DEF_HOSTNAME_LENGTH 32
```

Description

The buffer length of Host Name.

```
#define DEF_SN_LENGTH 32
```

Description

The buffer length of Serial Number.

```
#define DEF_MAC_LENGTH 16
```

Description

The buffer length of MAC Address.

#define DEF_LAL_LENGTH 20

Description

The buffer length of GPS Location.

#define DEF_VERSION_LENGTH 16

Description

The buffer length of Version.

#define DEF_MAX_STRING_LENGTH 128

Description

The maximum buffer length of String.

#define DEF_RUN_MODE_LENGTH 32

Description

The buffer length of Run Mode.

#define DEF_ENABLE_LENGTH 8

Description

The buffer length of Enable Flag string.

#define DEF_USER_PASS_LENGTH 128

Description

The buffer length of User Name and Password encode string.

#define DEF_PORT_LENGTH 8

Description

The buffer length of Port string.

#define DEF_KVM_MODE_LENGTH 8

Description

The buffer length of KVM Mode.

#define MAX_TOPIC_LEN 32

Description

The maximum buffer length of MQTT Topic.

#define DEF_OSVERSION_LEN 64

Description

The buffer length of OS Version.

```
#define DEF_MAX_PATH 260
```

Description

The maximum buffer length of Path.

4.3 Configuration Structure

The Structure defines the Agent Configuration including the Server and Executing Mode.

4.3.1 Server Setting

char serverIP[DEF_MAX_STRING_LENGTH]

Description

The buffer of Server IP.

char serverPort[DEF_PORT_LENGTH]

Description

The buffer of Server Listen Port.

char loginID[DEF_USER_PASS_LENGTH]

Description

The buffer of Server Login ID.

char loginPwd[DEF_USER_PASS_LENGTH]

Description

The buffer of Server Login Password.

4.3.2 Executing Mode

char runMode[DEF_RUN_MODE_LENGTH]

Description

The buffer of Agent Running Mode. The Mode includes: ‘remote’ and ‘standalone’.

‘remote’ means the agent is controlled by remote server.

‘standalone’ means the agent is controlled by standalone UI.

char autoStart[DEF_ENABLE_LENGTH]

Description

The buffer of Auto Start. autoStart is ‘True’ means the agent will connect to server automatically while agent service starts.

4.4 Profile Structure

The Structure defines the Agent Profile including the Agent, Custom and Platform Information.

4.4.1 Agent Information

char version[DEF_MAX_STRING_LENGTH]

Description

The buffer of Agent Version.

char hostname[DEF_HOSTNAME_LENGTH]

Description

The buffer of Host Name.

char devId[DEF_DEVID_LENGTH]

Description

The buffer of Device ID.

char sn[DEF_SN_LENGTH]

Description

The buffer of Serial Number.

char mac[DEF_MAC_LENGTH]

Description

The buffer of MAC Address.

char lal[DEF_LAL_LENGTH]

Description

The buffer of GPS Location.

char workdir[DEF_MAX_PATH]

Description

The buffer of Agent Working Directory.

4.4.2 Custom Information

char type[DEF_MAX_STRING_LENGTH]

Description

The buffer of Device Type. Such as: IPC, Gateway or SensorNode

char product[DEF_MAX_STRING_LENGTH]

Description

The buffer of Product name.

char manufacture[DEF_MAX_STRING_LENGTH]

Description

The buffer of Manufacturer name.

4.4.3 Platform Information

char osversion[DEF_OSVERSION_LEN]

Description

The buffer of OS Version.

char biosversion[DEF_VERSION_LENGTH]

Description

The buffer of BIOS Version.

char platformname[DEF_FILENAME_LENGTH]

Description

The buffer of Device Platform Name.

char processorname[DEF_FILENAME_LENGTH]

Description

The buffer of Processor Name.

char osarchitect[DEF_FILENAME_LENGTH]

Description

The buffer of OS Architecture. Such as: X86 or X64.

long totalmemsize

Description

The device total memory size. The unit is megabyte.

char maclist[DEF_MAC_LENGTH*16]

Description

The buffer of MAC Address List. Maxiumum support 16 MAC Address.

char localip[DEF_MAX_STRING_LENGTH]

Description

The buffer of Local IP Address.

4.5 Packet Structure

The Structure defines the Packet for communicating and exchanging data with RMM Server.

int cmd

Description

The Command ID defines in Handler is used to identify the handler action.

int requestID

Description

The Request ID is used to identify the packet which belongs to specified handler. Preserved for old RMM Server, now is replaced by Handler Name.

char devId[DEF_DEVID_LENGTH]

Description

The buffer of Device ID.

char handlerName[MAX_TOPIC_LEN]

Description

The buffer of Handler Name.

char* content

Description

The Pointer of Packet Content buffer.

5 SAClient Definition

SAClient.h file includes the API declaration, constants and flags that are required for programming.

5.1 Status Codes

All SAClient API functions return a status code from a common list of possible errors immediately. Any function may return any of the defined status codes.

saclient_false -1

Description

The SAClient API library is encountered an undefined error.

Actions

None.

saclient_success 0

Description

No error.

Actions

None.

saclient_config_error 1

Description

Configuration is invalid.

Actions

Varify Initialize Parameters.

saclient_profile_error 2

Description

Profile is invalid.

Actions

Varify Initialize Parameters.

saclient_no_init 3

The SAClient API library is not yet or unsuccessfully initialized. saclient_initialize needs to be called prior to the first access of any other SAClient API functions.

Actions

Call saclient_initialize.

saclient_callback_null 4

Description

The callback function is not assigned yet.

Actions

Assigned the callback function.

saclient_callback_error 5

Description

The callback function is encountered an error.

Actions

Varify the callback function parameters.

saclient_no_connnect 6

Description

The SAClient is not yet or lost connect to server.

Actions

Call saclient_connect.

saclient_connect_error 7

Description

The SAClient cannot cannect to server.

Actions

Varify configuration of Initialize Parameters.

saclient_init_error 8

Description

SAClient encountered an error while calling saclient_initialize.

Actions

Varify Initialize Parameters..

saclient_network_sock_timeout 16

Description

SAClient connection timeout.

Actions

Check server status.

saclient_network_sock_error 17

Description

SAClient cannect failed.

Actions

Check server status.

saclient_send_data_error 18

Description

SAClient encountered an error while sending data.

Actions

Retry.

saclient_report_agentinfo_error 19

Description

SAClient encountered an error while sending report data.

Actions

Retry.

saclient_send_willmsg_error 20

Description

SAClient encountered an error while sending will message.

Actions

Retry.

5.2 Callback Function

5.2.1 SACLIENT_CONNECTED_CALLBACK

```
typedef void (*SACLIENT_CONNECTED_CALLBACK)()
```

Description:

Callback function to handle on connected event.

Parameters:

None.

Return Status Code:

None.

5.2.2 SACLIENT_LOSTCONNECT_CALLBACK

```
typedef void (*SACLIENT_LOSTCONNECT_CALLBACK)()
```

Description:

Callback function to handle lost connect event.

Parameters:

None.

Return Status Code:

None.

5.2.3 SACLIENT_DISCONNECT_CALLBACK

```
typedef void (*SACLIENT_DISCONNECT_CALLBACK)()
```

Description:

Callback function to handle disconnect event.

Parameters:

None.

Return Status Code:

None.

5.2.4 SACLIENT_MESSAGE_RECV_CALLBACK

```
typedef void (*SACLIENT_MESSAGE_RECV_CALLBACK)(char* topic,
                                               susiaccess_packet_body_t *pkt, void *pRev1, void* pRev2)
```

Description:

Callback function to receive messages sent from server.

Parameters:

topic

Pointer to a buffer of received MQTT Topic.

pkt

Pointer to a buffer of received packet data.

pRev1

Reserved Pointer.

pRev2

Reserved Pointer.

Return Status Code:

None.

6 SAClient API

The SAClient APIs provide functions to communicate and exchange data between device and RMM Server. SAClient API can be implemented in various operation systems.

6.1 Initialization Functions

6.1.1 saclient_initialize

```
int saclient_initialize(susiaccess_agent_conf_body_t * config,
                        susiaccess_agent_profile_body_t * profile,
                        void * loghandle)
```

Description:

General initialization of the SAClient API, prior to calling any SAClient API functions, the library needs to be initialized by calling this function. The status code for all SAClient API function will be **saclient_no_init** unless this function is called.

Parameters:

config

Pointer to a buffer of Configuration data.

profile

Pointer to a buffer of Profile data.

loghandle

Pointer to the file handles for logging.

Return Status Code:

Condition	Return Value
Library initialized	saclient_success
Fail	saclient_false
Success	saclient_success
Invalid Configuration	saclient_config_error
Invalid Profile	saclient_profile_error

6.1.2 saclient_uninitialize

```
void saclient_uninitialize()
```

Description:

General function to un-initialize the SAClient API library. This should be called before program exit.

Parameters:

None

Return Status Code:

None

6.2 Connection Functions

6.2.1 saclient_connect

int saclient_connect()

Description:

Connect to server that defined in Configuration data of **saclient_initialize** parameters.

Parameters:

None

Return Status Code:

Condition	Return Value
Library uninitialized	saclient_no_init
Connect timeout	saclient_network_sock_timeout
Connect fail	saclient_network_sock_error
Send AgentInfo fail	saclient_report_agentinfo_error
Send Will Message fail	saclient_send_willmsg_error
Connect fail	saclient_false
Success	saclient_success

6.2.2 saclient_disconnect

void saclient_disconnect()

Description:

Disconnect from server.

Parameters:

None

Return Status Code:

None

6.2.3 saclient_server_connect

int saclient_server_connect (char const * ip, int port, char const * mqttauth)

Description:

Connect to specific server.

Parameters:

ip

Specific server IP.

port

Specific server listen port.

mqttauth

Specific server connection authentication string encrypt with DES and Base64.

Return Status Code:

Condition	Return Value
Library uninitialized	saclient_no_init
Connect timeout	saclient_network_sock_timeout
Connect fail	saclient_network_sock_error
Send AgentInfo fail	saclient_report_agentinfo_error
Send Will Message fail	saclient_send_willmsg_error
Connect fail	saclient_false
Success	saclient_success

6.2.4 saclient_connection_callback_set

```
void saclient_connection_callback_set(
                SACLIENT_CONNECTED_CALLBACK on_connect,
                SACLIENT_LOSTCONNECT_CALLBACK on_lost_connect,
                SACLIENT_DISCONNECT_CALLBACK on_disconnect)
```

Description:

Register the connection callback function to handle the connection event.

Parameters:

on_connect

Function Pointer to handle connect success event.

on_lost_connect

Function Pointer to handle lost connect event. The SAClient will reconnect automatically, if left as NULL.

on_disconnect

Function Pointer to handle disconnect event.

Return Status Code:

None

6.2.5 saclient_getsocketaddress

int saclient_getsocketaddress(char* clientip, int size)

Description:

Get local IP address that connects to server.

Parameters:

client

Pointer to a buffer for received client IP.

size

the buffer size of client IP.

Return Status Code:

Condition	Return Value
Library uninitialized	saclient_no_init
Library not connected	saclient_no_connnect
Failed	saclient_false
Success	saclient_success

6.3 Send/Receive Function

6.3.1 saclient_publish

```
int saclient_publish(char const * topic, susiaccess_packet_body_t const * pkt)
```

Description:

Send message wrapped in packet structure to server on specific MQTT topic.

Parameters:

topic

Pointer to a buffer of MQTT Topic.

pkt

Pointer to a buffer of packet data.

Return Status Code:

Condition	Return Value
Library uninitialized	saclient_no_init
Library not connected or lost connect	saclient_no_connnect
Fail	saclient_false
Success	saclient_success

6.3.2 saclient_subscribe

```
int saclient_subscribe(char const * topic,
                       SACLIENT_MESSAGE_RECV_CALLBACK msg_recv_callback)
```

Description:

Register a callback function to receive message from server on specific MQTT topic.

Parameters:

topic

Pointer to a buffer of MQTT Topic.

msg_recv_callback

Pointer to a message receive callback function.

Return Status Code:

Condition	Return Value
Library uninitialized	saclient_no_init
Library not connected or lost connect	saclient_no_connnect

Fail	saclient_false
Success	saclient_success

7 Handler API Definition

susiaccess_handler_api.h file includes the API declaration, constants and flags that are required for programming.

7.1 Error Codes

All Handler API functions return an error code from a common list of possible errors immediately. Any function may return any of the defined error codes.

handler_fail -1

Description

The Handler library is encountered an undefined error.

Actions

None.

handler_success 0

Description

No error.

Actions

None.

handler_no_init 1

The Handler library is not yet or unsuccessfully initialized. handler_initialize is called right after the Handler loaded by Core Management.

Actions

Handler initialize fail.

handler_callback_null 2

Description

The callback function is not assigned yet.

Actions

Assigned the callback function.

handler_callback_error 3

Description

The callback function is encountered an error.

Actions

Varify the callback function parameters.

handler_no_connnect 4

Description

The SAClient is not yet or unsuccessfully connected to server.

Actions

Wait Handler_OnStatusChange to switch the status to AGENT_STATUS_ONLINE.

handler_init_error 5

Description

Handler encountered an error while calling handler_initialize.

Actions

Varify Initialize Parameters.

7.2 Handler Status Codes

Handler need to monitor the internal threads status with the status code. Core Management monitors the handler status by calling Handler_Get_Status function to get the status code.

handler_status_no_init	-1

Description

The Handler not initialize yet.

handler_status_init	0

Description

The Handler is initialized.

handler_status_start	1

Description

The Handler thread is executing.

handler_status_stop	2

Description

The Handler thread is terminated.

handler_status_busy	3

Description

The Handler thread is blocked.

7.3 Callback Function Error Codes

All Handler API Callback functions return an error code from a common list of possible errors immediately.

cagent_success 0

Description

No error.

Actions

None.

cagent_no_init 1

Description

The Core Management is not yet or unsuccessfully initialized.

Actions

Retry after Core Management Initialized.

cagent_callback_null 2

Description

The callback function is not assigned yet.

Actions

Check the callback function pointer.

cagent_callback_error 3

Description

The callback function is encountered an error.

Actions

Varify the callback function parameters.

cagent_no_connnect 4

Description

The SAClient is not yet or lost connect to server.

Actions

Wait connect and retry.

cagent_connect_error 5

Description

The SAClient cannot cannect to server.

Actions

Check the server status.

cagent_init_error 6

Description

Core Management encountered an error while initialize.

Actions

Check system environment.

cagent_network_sock_timeout 16

Description

SAClient connection timeout.

Actions

Check server status.

cagent_network_sock_error 17

Description

SAClient cannect failed.

Actions

Check server status.

cagent_send_data_error 18

Description

SAClient encountered an error while sending data.

Actions

Retry.

7.4 Notify Severity Codes

The severities codes are used to identify the event notify severity level, sending to RMM Server by calling HandlerSendEventNotify callback function.

Severity_Emergency	0

Description

In emergency severity level, the system is unusable.

Severity_Alert	1

Description

In alert severity level, the action must be taken immediately.

Severity_Critical	2

Description

System trigger the crtical conditions.

Severity_Error	3

Description

System trigger the error conditions.

Severity_Warning	4

Description

System trigger the warning conditions.

Severity_Informational	5

Description

System send Informational messages.

Severity_Debug	6

Description

System send debug level messages.

7.5 Callback Function

7.5.1 HandlerSendCbf

```
typedef AGENT_SEND_STATUS    (*HandlerSendCbf) ( HANDLE const handler,
        int enum_act, void const * const requestData, unsigned int const requestLen,
        void *pRev1, void* pRev2 );
```

Description:

Callback function to send the response message. Handler_Recv received the command from RMM Server and send the response message with this callback function.

Parameters:

handler

Pointer to a buffer of Handler Info structure.

enum_act

The response message command ID.

requestData

Pointer to a buffer of message string.

requestLen

The buffer length of message string.

pRev1

Reserved Pointer.

pRev2

Reserved Pointer.

Return Status Code:

Condition	Return Value
Library uninitialized	cagent_no_init
Library unconnected	cagent_no_connnect
No response	cagent_network_sock_timeout
Send Fail	cagent_send_data_error
Success	cagent_success

7.5.2 HandlerSendEventCbf

```
typedef AGENT_SEND_STATUS    (*HandlerSendEventCbf) ( HANDLE const handler,
        HANDLER_NOTIFY_SEVERITY severity, void const * const requestData,
        unsigned int const requestLen, void *pRev1, void* pRev2 );
```

Description:

Callback function to send the event notify message. Handler can send event notify message by calling this callback function if the notify condition is triggered.

Parameters:

handler

Pointer to a buffer of Handler Info structure.

severity

The event severity level.

requestData

Pointer to a buffer of message string.

requestLen

The buffer length of message string.

pRev1

Reserved Pointer.

pRev2

Reserved Pointer.

Return Status Code:

Condition	Return Value
Library uninitialized	cagent_no_init
Library unconnected	cagent_no_connnect
No response	cagent_network_sock_timeout
Send Fail	cagent_send_data_error
Success	cagent_success

7.5.3 HandlerSendCapabilityCbf

```
typedef AGENT_SEND_STATUS    (*HandlerSendCapabilityCbf) (
        HANDLE const handler,
        void const * const requestData, unsigned int const requestLen,
        void *pRev1, void* pRev2 );
```

Description:

Callback function for IoT to send the handler capability message. Handler can update the capability actively by calling this callback function.

Parameters:

handler

Pointer to a buffer of Handler Info structure.

requestData

Pointer to a buffer of capability string that follow IPSO spec in JSON format.

requestLen

The buffer length of capability string.

pRev1

Reserved Pointer.

pRev2

Reserved Pointer.

Return Status Code:

Condition	Return Value
Library uninitialized	cagent_no_init
Library unconnected	cagent_no_connnect
No response	cagent_network_sock_timeout
Send Fail	cagent_send_data_error
Success	cagent_success

7.5.4 HandlerAutoReportCbf

```
typedef AGENT_SEND_STATUS    (*HandlerAutoReportCbf) ( HANDLE const handler,
    void const * const requestData, unsigned int const requestLen,
    void *pRev1, void* pRev2 );
```

Description:

Callback function for IoT to send the sensor data report. Handler can update the sensor data value by calling this callback function.

Parameters:

handler

Pointer to a buffer of Handler Info structure.

requestData

Pointer to a buffer of sensor data string that follow IPSO spec in JSON format.

requestLen

The buffer length of sensor data string.

pRev1

Reserved Pointer.

pRev2

Reserved Pointer.

Return Status Code:

Condition	Return Value

Library uninitialized	cagent_no_init
Library unconnected	cagent_no_connnect
No response	cagent_network_sock_timeout
Send Fail	cagent_send_data_error
Success	cagent_success

7.5.5 HandlerSendCustCbf

```
typedef AGENT_SEND_STATUS    (*HandlerSendCustCbf) ( HANDLE const handler,
        int enum_act, void const * const topic,
        void const * const requestData, unsigned int const requestLen,
        void *pRev1, void* pRev2 );
```

Description:

Callback function to send a message on custom MQTT topic. Handler can send message to a specific MQTT topic by calling this callback function.

Parameters:

handler

Pointer to a buffer of Handler Info structure.

enum_act

The custom message command ID.

topic

Pointer to a buffer of MQTT topic string.

requestData

Pointer to a buffer of message string.

requestLen

The buffer length of message string.

pRev1

Reserved Pointer.

pRev2

Reserved Pointer.

Return Status Code:

Condition	Return Value
Library uninitialized	cagent_no_init
Library unconnected	cagent_no_connnect
No response	cagent_network_sock_timeout
Send Fail	cagent_send_data_error
Success	cagent_success

7.5.6 HandlerSubscribeCustCbf

```
typedef AGENT_SEND_STATUS    (*HandlerSubscribeCustCbf) ( void const * const topic,
                                HandlerCustMessageRecvCbf recvCbf);
```

Description:

Callback function to subscribe a custom MQTT topic with a receive callback function to receive custom message. Handler can subscribe a specific topic with a message receive callback functo to receive the custom message or commands.

Parameters:

topic

Pointer to a buffer of custom MQTT topic.

recvCbf

Pointer of a receive callback function.

Return Status Code:

Condition	Return Value
Library uninitialized	cagent_no_init
No Callback function	cagent_callback_null
Internal error	cagent_callback_error
Success	cagent_success

7.5.7 HandlerCustMessageRecvCbf

```
typedef void (*HandlerCustMessageRecvCbf)(char * const topic, void* const data,
                                const size_t datalen, void *pRev1, void* pRev2);
```

Description:

Callback function to receive custom message. Handler can register the callback function to receive the custom message by calling HandlerSubscribeCustCbf callback function.

Parameters:

topic

Pointer to a buffer of MQTT topic string.

data

Pointer to a buffer of message string.

datalen

The buffer length of message string.

pRev1

Reserved Pointer.

pRev2

Reserved Pointer.

Return Status Code:

Condition	Return Value
Library uninitialized	cagent_no_init
Library unconnected	cagent_no_connnect
No response	cagent_network_sock_timeout
Send Fail	cagent_send_data_error
Success	cagent_success

7.6 Handler Info Structure

The Structure defines the handler and agent information and used to exchange the data between Core Management and the Handler

7.6.1 Handler Information

char Name[MAX_TOPIC_LEN]

Description

The buffer of handler name.

int RequestID

Description

The Handler Request ID is used to received packet with same request ID. Preserved for old RMM Server, now is replaced by Handler Name.

int ActionID

Description

The Handler send response packet with Action ID is to tell server the packet belone to which handler. Preserved for old RMM Server, now is replaced by Handler Name.

7.6.2 Agent Information

char ServerIP[MAX_PATH]

Description

The buffer of server IP.

int ServerPort

Description

The server listen port.

void* loghandle

Description

Pointer to the file handles for logging.

cagent_agent_info_body_t * agentInfo

Description

The Pointer to Agent_Info structure for more detail Information.

HandlerSendCbf sendcbf

Description

The function pointer to send general response message.

HandlerSendEventCbf sendeventcbf

Description

The function pointer to send event notify message.

HandlerSendCapabilityCbf sendcapabilitycbf

Description

The function pointer to send handler capability message.

HandlerAutoReportCbf sendreportcbf

Description

The function pointer to send sensor data message.

HandlerSendCustCbf sendcustcbf

Description

The function pointer to send message on custom MQTT topic.

HandlerSubscribeCustCbf subscribecustcbf

Description

The function pointer to subscribe the custom MQTT topic with a message receive callback function.

7.7 Agent Info Structure

The Structure defines the Agent information.

char hostname[DEF_HOSTNAME_LENGTH]

Description

The buffer of Host Name.

char devId[DEF_DEVID_LENGTH]

Description

The buffer of Device ID.

char sn[DEF_SN_LENGTH]

Description

The buffer of Serial Number.

char mac[DEF_MAC_LENGTH]

Description

The buffer of MAC Address.

char version[DEF_MAX_STRING_LENGTH]

Description

The buffer of Agent Version.

char type[DEF_MAX_STRING_LENGTH]

Description

The buffer of Device Type. Such as: IPC, Gateway or SensorNode

char product[DEF_MAX_STRING_LENGTH]

Description

The buffer of Product name.

char manufacture[DEF_MAX_STRING_LENGTH]

Description

The buffer of Manufacturer name.

int status

Description

The connection status. 0 (AGENT_STATUS_OFFLINE) is offline, 1(AGENT_STATUS_ONLINE) is online.

8 Handler API

The Handler implement the Handler API can be dynamic load and interact with Core Management.

8.1 Initialization Functions

8.1.1 Handler_Initialize

int Handler_Initialize(HANDLER_INFO *handler)

Description:

General initialization of the Handler API. This function will be called right after handler is loaded by Core Management. And exchange the handler and agent information with the Handler_Info structure.

Parameters:

handler

Pointer to a buffer of Handler Information.

Return Status Code:

Condition	Return Value
Library initialized	handler_success
Fail	handler_fail
Success	handler_success

8.2 Information Functions

8.2.1 Handler_Get_Status

int Handler_Get_Status(HANDLER_THREAD_STATUS * pOutStatus)

Description:

Get current Handler status. Core Management monitors the status, reference to 7.2 by calling Handler_Get_Status. If Handler returns status is handler_status_busy, the Core Management will restart the Handler by calling Handler_Stop and Handler_Start.

Parameters:

pOutStatus

Pointer to a buffer of handler status.

Return Status Code:

Condition	Return Value
Library uninitialized	handler_no_init
Fail	handler_fail
Success	handler_success

8.2.2 Handler_OnStatusChange

void Handler_OnStatusChange(HANDLER_INFO *handler)

Description:

Handle the Agent status change event. Core Management will calling this api while Agent status changed.

Parameters:

handler

Pointer to a buffer of Handler Information.

Return Status Code:

None

8.2.3 Handler_Recv

void Handler_Recv(char * const topic, void* const data, const size_t datalen,
void *pRev1, void* pRev2)

Description:

Receive message that sending from RMM Server. Core Management will calling Handler API while received the message that handler name or request ID is matched with this Handler.

Parameters:

topic

Pointer to a buffer of received MQTT Topic.

data

Pointer to a buffer of received data.

datalen

Pointer to a buffer of received data size.

pRev1

Reserved Pointer.

pRev2

Reserved Pointer.

Return Status Code:

None

8.3 Control Functions

8.3.1 Handler_Start

int Handler_Start(void)

Description:

Activate the Handler. Core Management activates the Handler by calling this API.

Parameters:

None

Return Status Code:

Condition	Return Value
Library uninitialized	handler_no_init
Fail	handler_fail
Success	handler_success

8.3.2 Handler_Stop

int Handler_Stop (void)

Description:

Stop the Handler. Core Management stop the Handler by calling this API.

Parameters:

None

Return Status Code:

Condition	Return Value
Library uninitialized	handler_no_init
Fail	handler_fail
Success	handler_success

8.4 IoT Functions

8.4.1 Handler_Get_Capability

int Handler_Get_Capability(char ** pOutReply)

Description:

Get Handler Capability. Core Management retrieves the Handler capability by calling this API and send to RMM Server.

Parameters:

pOutReply

Pointer to a buffer of capability string. Handler should allocate a memory to store the capability and pass the pointer

Return Status Code:

Condition	Return Value
Library uninitialized	handler_no_init
Fail	handler_fail
Success	handler_success

8.4.2 Handler_AutoReportStart

void Handler_AutoReportStart(char *pInQuery)

Description:

Start report sensor data. Handler start report sensor data repeatedly while Core Management received the Start Auto Upload Command and calling the Handler_AutoReportStart API.

Parameters:

pInQuery

Pointer to a buffer of auto report command. The command carried the auto report information, include: Upload Interval and request items.

Return Status Code:

None

8.4.3 Handler_AutoReportStop

void Handler_AutoReportStop (char *pInQuery)

Description:

Stop report sensor data. Handler stop report sensor data while Core Management received the Stop Auto Upload Command and calling the Handler_AutoReportStop API.

Parameters:

pInQuery

Pointer to a buffer of auto report command. The command carried the stop information, include: request items.

Return Status Code:

None

8.4.4 Handler_MemoryFree

void Handler_MemoryFree(char *pInData)

Description:

Free the Capability memory allocated by calling Handler_Get_Capability. After Core Management collect the Handler Capability by calling Handler_Get_Capability, the handler will allocate a memory to store the capability. Core Management need to calling Handler_MemoryFree to release these memory.

Parameters:

pInData

Pointer to a buffer that allocated by Handler.

Return Status Code:

None