EELINK DEVICE PROTOCOL V2.0

本文档的版权归 深圳市移联通信技术有限责任公司 所有，保留所有权利。未经授权擅自复制、修改、传播本文档的部分或全部，将承担一切法律责任。

This document is protected by Copyright and the information contained herein is confidential. The document may not be copied and the information contained herein may not be used or disclosed except with the written permission of EELINK Co., Ltd. (C) 2018

REVISION HISTORY

Revision	Date	Description
V2.0.3	2019/04/29	Amend the description of Relay command
V2.0.2	2019/02/12	Change datum type of temperature
V2.0.1	2018/09/25	1. Add sensor support 2. Revise speed warning
V2.0.0	2018/02/08	Initial Release

TABLE OF CONTENTS

- REVISION HISTORY.
- TABLE OF CONTENTS.
- 1 SUMMARY.
  - 1.1 SCOPE.
  - 1.2 TERMS.
  - 1.3 DESCRIPTION.
- 2 PRINCIPLE.
  - 2.1 INTERFACE.
  - 2.2 SMS.
  - 2.3 IP.
  - 2.4 UART.
  - 2.5 USER QUERY.
- 3 DATA TYPES.
  - 3.1 INTEGER.
  - 3.2 STRING.
  - 3.3 RAW DATA.
  - 3.4 TIME.
  - 3.5 STATUS.
  - 3.6 POSITION.
- 4 TCP/IP.
  - 4.1 USING TCP.
  - 4.2 USING UDP.
- 5 PACKAGES.
  - 5.1 GENERAL.
  - 5.2 LOGIN PACKAGE — 0x01.
  - 5.3 HEARTBEAT PACKAGE — 0x03.
  - 5.4 LOCATION PACKAGE — 0x12.
  - 5.5 WARNING PACKAGE — 0x14.
  - 5.6 REPORT PACKAGE — 0x15.
  - 5.7 MESSAGE PACKAGE — 0x16.
  - 5.8 OBD DATA PACKAGE — 0x17.
  - 5.9 OBD BODY PACKAGE — 0x18.
  - 5.10 OBD FAULT PACKAGE — 0x19.
  - 5.11 PEDOMETER PACKAGE — 0x1A.
  - 5.12 PARAM-SET PACKAGE — 0x1B.
  - 5.13 INSTRUCTION PACKAGE — 0x80.
  - 5.14 BROADCAST PACKAGE — 0x81.
- 6 COMMANDS.
  - 6.1 SECURITY COMMANDS.

        - 6.1.1 LOGIN.
        - 6.1.2 LOGOUT.
        - 6.1.3 PASSWORD.
    - 6.2 ACTION COMMANDS.
        - 6.2.1 UPGRADE.
        - 6.2.2 RESET.
        - 6.2.3 SHUTDOWN.
        - 6.2.4 FACTORY.
        - 6.2.5 PAGING.
        - 6.2.6 CLEAR.
        - 6.2.7 LISTEN.
        - 6.2.8 RELAY.
        - 6.2.9 FWD.
    - 6.3 SETTING COMMANDS.
        - 6.3.1 IMEI.
        - 6.3.2 LANG.
        - 6.3.3 GMT.
        - 6.3.4 HBT.
        - 6.3.5 DELAY.
        - 6.3.6 APN.
        - 6.3.7 SERVER.
        - 6.3.8 COLLECT.
        - 6.3.9 MANAGER.
        - 6.3.10 AGPS.
        - 6.3.11 GSM.
        - 6.3.12 GPS.
        - 6.3.13 ALARM.
        - 6.3.14 MILEAGE.
        - 6.3.15 MOTION.
        - 6.3.16 SPEED.
        - 6.3.17 FENCE.
        - 6.3.18 SHIFT.
        - 6.3.19 SENSOR.
    - 6.4 QUERY COMMANDS.
        - 6.4.1 VERSION.
        - 6.4.2 PARAM.
        - 6.4.3 STATUS.
        - 6.4.4 STAT.
        - 6.4.5 POSITION / 123.
        - 6.4.6 WHERE.
        - 6.4.7 URL.
    - 6.5 PORT COMMANDS.
        - 6.5.1 GPIO.
        - 6.5.2 PORT.

- 6.6 PEDOMETER COMMANDS.
  - 6.6.1 BODY.
  - 6.6.2 PDM.
- 6.7 OBD COMMANDS.
  - 6.7.1 MONITOR.
  - 6.7.2 OBD.
  - 6.7.3 OBD,01.
  - 6.7.4 OBD,02.
  - 6.7.5 OBD,03.
  - 6.7.6 OBD,08.
  - 6.7.7 OBD,10.
  - 6.7.8 OBD,11.
  - 6.7.9 OBD,12.
  - 6.7.10 OBD,13.
  - 6.7.11 OBD,14.
  - 6.7.12 OBD,15.
  - 6.7.13 OBD,18.
  - 6.7.14 OBD,19.
- APPENDIX.
  - A.1 CHECKSUM ALGORITHM.
  - A.2 DEFLATE ALGORITHM.
  - A.3 PARAM-SET.
  - A.4 Extended OBD-II PIDs.

1 SUMMARY

1.1 SCOPE

This document describes the communication protocol between server and device, and the commands supported in device.

The protocol allows device to transmit vehicle location, status and other information to server. On the other hand, it also allows server to send commands to device.

The commands allow user or server to control and configure device, and retrieve data from device.

Normally, the interface between server and device is GPRS/3G/4G, and the interface between user and device is SMS.

1.2 TERMS

Terms	Explanation
A-GPS	Assisted GPS
ACC	Accessory (on car switches)
APN	Access Point Name
CAN	Controller Area Network
CID	Cell Tower Identifier
CRC	Cyclic Redundancy Check
GMT	Greenwich Mean Time
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communication
ICCID	Integrated Circuit Card Identifier

Terms	Explanation
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
LAC	Location Area Code
LBS	Location Based Services
MCC	Mobile Country Code
MNC	Mobile Network Code
NITZ	Network Identity and Time Zone
NTP	Network Time Protocol
RNC	Radio Network Controller
SMS	Short Message Service
OBD	On-Board Diagnostics
OTA	Over The Air
TCP	Transport Control Protocol
TFTP	Trivial File Transfer Protocol
UDP	User Datagram Protocol
UTC	Universal Coordinated Time
VIN	Vehicle Identification Number

1.3 DESCRIPTION

This document is a full rewritten version on the last version (V1.9) which is described in Chinese language.

The device, herein, is the shorter form of all equipment manufactured by EELINK. It is primarily a location reporting device that responds to a user or server query or is triggered by an event such as a change in the value of one of its functional elements. It supports Over The Air (OTA) upgrade of its firmware using a TFTP connection with a

specified server.

At startup, the device automatically turns “on” and registers with the GSM network. After that, it will attempt to create an IP network connection. If such a connection is unavailable, it will still allow connection through SMS or the physical UART. Configuration parameters are stored to flash memory and are automatically used on the next power up event.

The commands can be executed on any available connection as these connections are not exclusive. Commands and responses have identical syntax regardless of the channel they are transacted over.

Robust lockup protection is provided by use of a dedicated hardware watchdog that cycles power and resets the system if a lockup is detected.

The device has more features as follows:

1. Multiple location services (GPS, LBS, WiFi);
2. Supporting A-GPS;
3. Low power consumption;
4. Using 3-axis accelerometer;
5. Automatic time sync (GPS, NITZ, AGPS, NTP);
6. Managing a lot of third party components (e.g. OBD, Thermometer, Light Sensor, etc);

2 PRINCIPLE

2.1 INTERFACE

The device has three external interfaces:

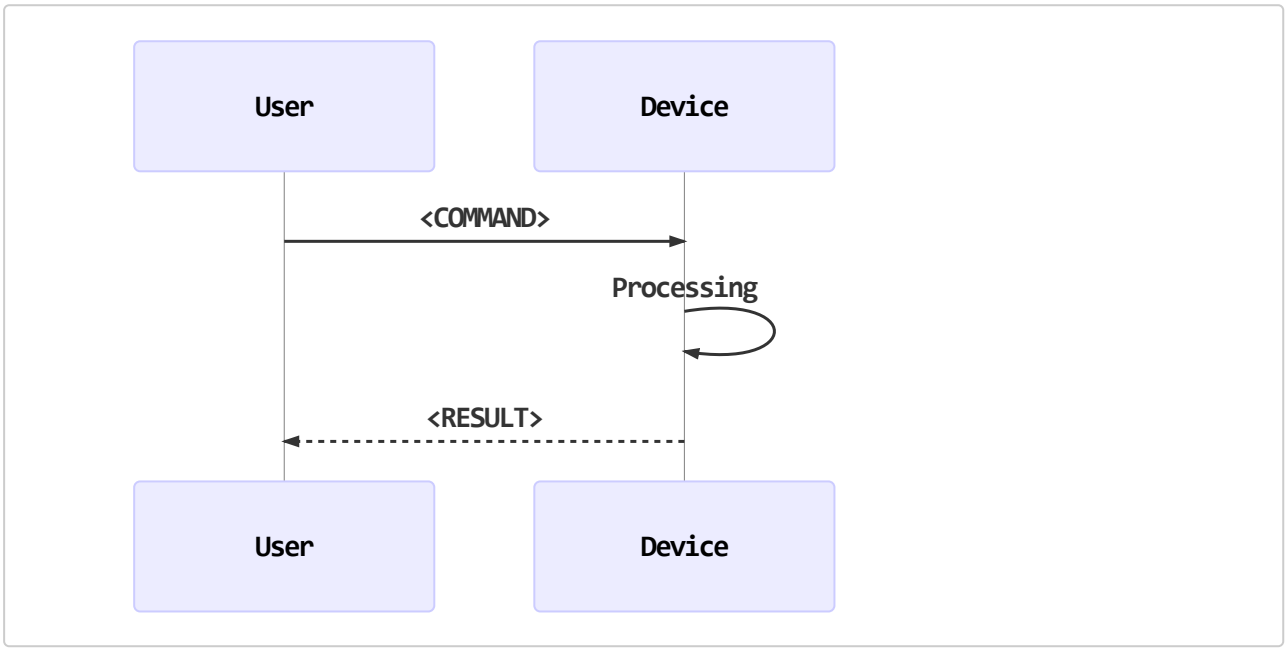
- Message over GSM (SMS)
- Data connection (GPRS/3G/4G)
- Serial connection (UART)

All of them can be used to communicate with device.

2.2 SMS

The commands, described in this document, are available in text format. They can be sent in their raw format from user to the device. After that, the results will be returned to user also in their raw format.

The workflow is as follows:



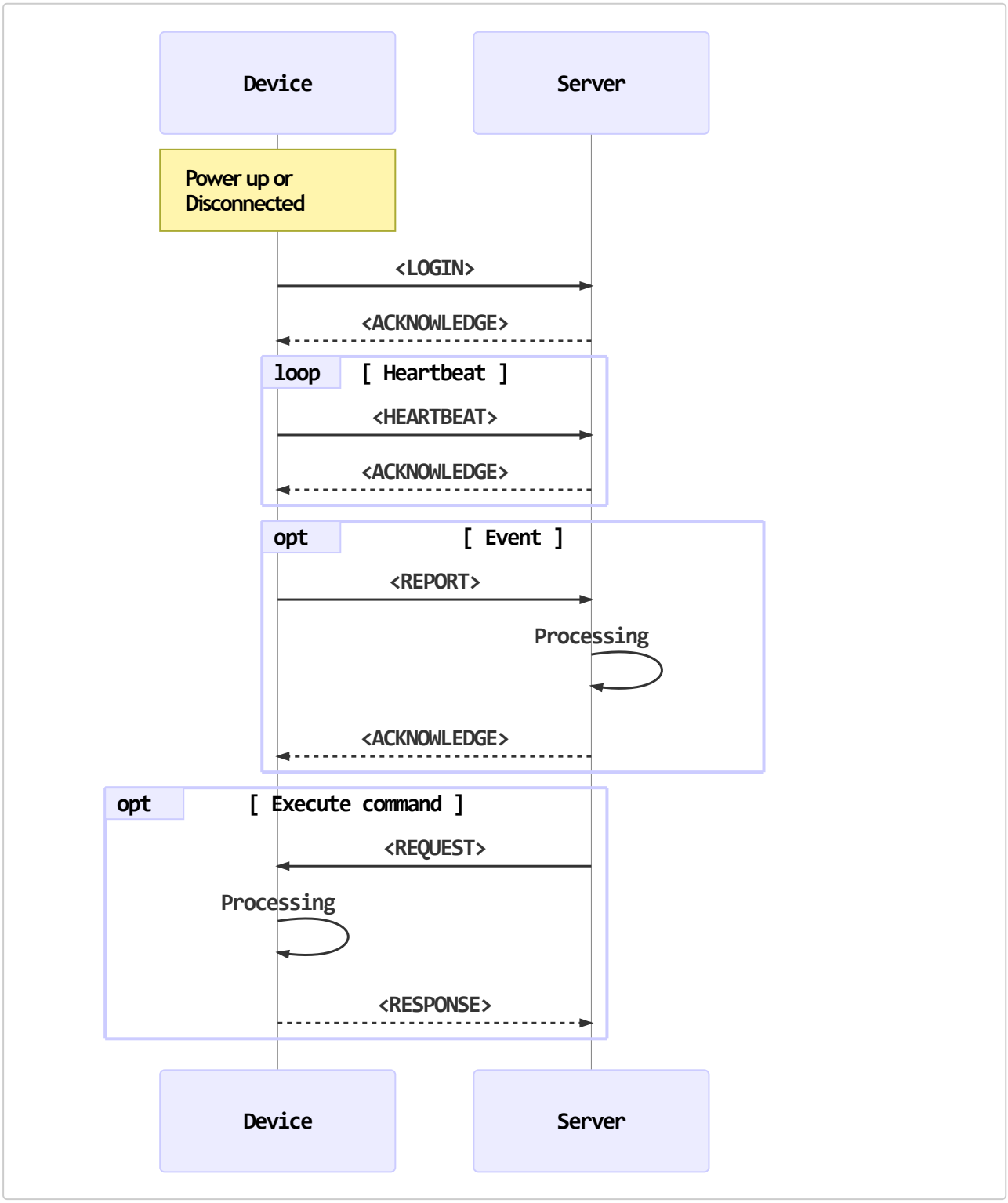
2.3 IP

The protocol, described in this document, defines the data packages between server and device. It is based on IP connection over the cellular. After a IP connection is established between server and device, the data packages are allowed to be exchanged between them.

There are two categories of data packages. One is that device report something to server and server acknowledges it. Another is that server request something and device responses it. The format of data packages will be discussed in the following chapters.

The commands, described in this document, can be embedded in a specific package and be sent to device. The result embedded in another specific package will be returned to server.

Whole workflow is as follows:



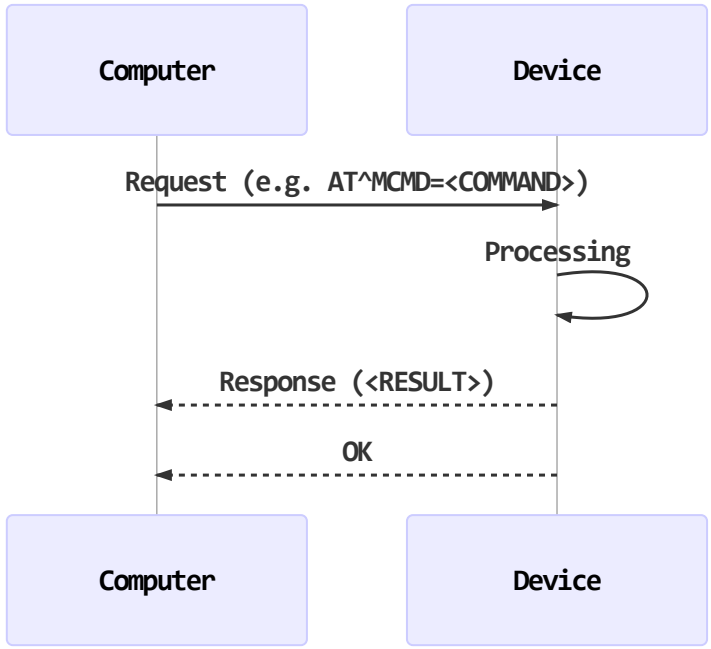
When device is powered up or session is disconnected, it attempts to establish a connection to server. After being connected, device sends a login package to server at first. All other packages will not be sent until device received the acknowledge from server.

If using TCP, after the connection is established successfully, the heartbeat package will be sent periodically in a specific interval. The reason is to keep the connection and to detect the availability of connection. If all acknowledges of 3 heartbeat packages are not received, current session will be disconnected and a new one will be established. This mechanism exists only in the protocol based on TCP, and it is unavailable in the protocol based on UDP.

While the connection is valid, device will send relevant reports according to different events. The most primary report is the location report which describes the location of device. Server can plot the track of device by collecting all location reports.

2.4 UART

Normally, AT Command Set is running on the serial port. It is a wired connection. Its workflow is as follows:



The standard AT Command Set is described in another document. The commands, described in this document, can be sent by an extension AT command in the following format:

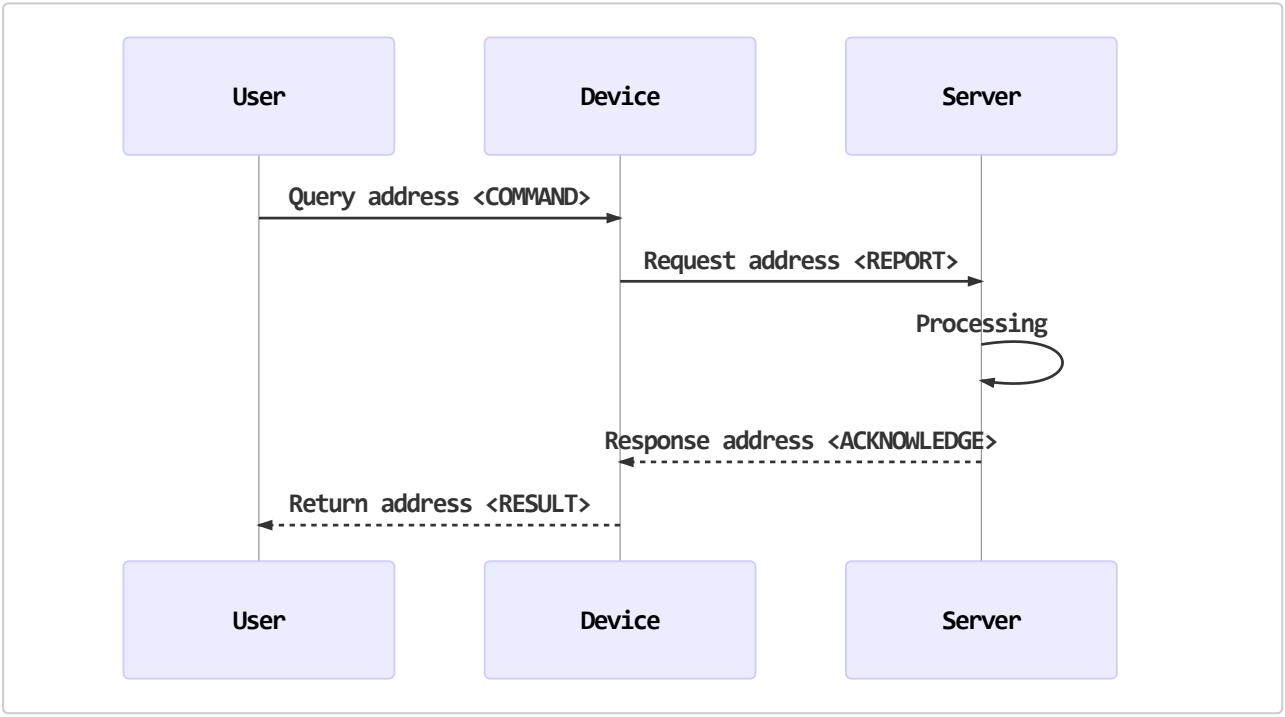
```
AT^MCMD=<COMMAND>
```

```
<RESULT>
OK
```

2.5 USER QUERY

There is a special workflow. When user wants to retrieve the location of device, he can send a query-address command to device. But device do not know its address except coordinates, so it sents its coordinates to server. Then server returns the address to device and device forwards it to user.

The work flow is as follows:



3 DATA TYPES

We will see a lot of data later in the following chapters. All of them belong to a few types. In this chapter, we discuss common data types used in protocol.

3.1 INTEGER

Integer is the most important data type in protocol. Most data are represented in an integer, e.g. the data length, the package type, the satellites number, etc.

The positive integers are represented in their binary value in unsigned format. And the negative integers are represented in the complement system in signed format.

Moreover, all float point numbers are transformed to integers. For example: temperature is written in normal format as “-6.5” Celsius degree. However, it can also be represented in “-65” in “0.1 Celsius degree”. In this form, we convert a float point number to an integer. The detail of conversion will be found later in Chapter 5 PACKAGES.

There are 6 integer types:

- unsigned 8 bits integer, from 0 to 255
- unsigned 16 bits integer, from 0 to 65535
- unsigned 32 bits integer, from 0 to 4294967295
- signed 8 bits integer, from -128 to 127
- signed 16 bits integer, from -32768 to 32767
- signed 32 bits integer, from -2147483648 to 2147483647

In the communication protocol, the MSB of integer is transferred prior to the LSB. As a result, the byte order of an integer in all packages is from MSB to LSB.

3.2 STRING

All strings are coded in UTF-8.

Most strings in package have a limited length, e.g. password, name, etc. We use a fixed size space to contain them. If the size of space is more than the length of string, rest bytes will be zero. The length of string is never more than the size of space.

Only few strings have a variable length. When they appear in package, their length must be able to be calculated based on other data field. They will be discussed in detail later in Chapter 5 PACKAGES.

The byte order of a string is always from the first byte to the last byte.

3.3 RAW DATA

Some data are provided by the third party, e.g. OBD data, picture data, etc. Normally, they are written into package in their raw format without any modification.

3.4 TIME

All time are coded as one unsigned 32 bits integer. All of them are represented in UTC (GMT) time. In another word, they are the time in time zone 0.

The value of integer is the seconds from 1970/01/01 00:00:00. For example, a decimal 1480209825 (hexadecimal 0x583A35A1) is 2016-11-27 01:23:45.

3.5 STATUS

A unsigned 16 bits integer is used to represent the status of device. The definition of each bits are described as listed below:

Bit	Description
0	1: GPS is fixed 0: GPS is not fixed
1	1: Device is designed for car 0: Device is not designed for car
2	1: Car engine is fired (only valid when bit 1 is 1) 0: Car engine is not fired
3	1: Accelerometer is supported 0: No accelerometer
4	1: The motion-warning is activated (only valid when bit 3 is 1)

Bit	Description
	0: The motion-warning is deactivated
5	1: Relay control is supported 0: No relay control
6	1: The relay control is triggered (only valid when bit 5 is 1) 0: The relay control is not triggered
7	1: External charging is supported 0: No external charging
8	1: Device is charging (only valid when bit 7 is 1) 0: Device is not charging
9	1: Device is active (only valid when bit 3 is 1) 0: Device is stationary
10	1: GPS module is running 0: GPS module is not running
11	1: OBD module is running (only valid when OBD is supported) 0: OBD module is not running
12	1: DIN0 is high level (only valid when DIN0 is supported) 0: DIN0 is low level
13	1: DIN1 is high level (only valid when DIN1 is supported) 0: DIN1 is low level
14	1: DIN2 is high level (only valid when DIN2 is supported) 0: DIN2 is low level
15	1: DIN3 is high level (only valid when DIN3 is supported) 0: DIN3 is low level

> Note:
>
> 1. *DIN* is the abbreviation of digital input port.

3.6 POSITION

Position is a compound data type. It includes all data related to location, e.g. latitude,

longitude, altitude, GSM BSID, WLAN BSSID, etc. In order to get minimum length, it is defined in variable size. It includes only valid data and eliminates invalid one. A *mask* is used to indicate which data are valid.

The structure of a position is described as below:

Name	Bytes	Description
Time	4	The event time (UTC) when position data is collected
Mask	1	The mask to indicate which data are valid (BIT0 ~ BIT6: GPS, BSID0, BSID1, BSID2, BSS0, BSS1, BSS2)
GPS Data		**The following data are related to GPS and valid only if BIT0 of *mask* is 1**
Latitude	4	-90.0 ~ 90.0 degree — Signed 32 bits integer from -162000000 to 162000000 (in 1/500”)
Longitude	4	-180.0 ~ 180.0 degree — Signed 32 bits integer from -324000000 to 324000000 (in 1/500”)
Altitude	2	Signed 16 bits integer from -32768 to 32767 (in meters)
Speed	2	Unsigned 16 bits integer (in km/h)
Course	2	Unsigned 16 bits integer from 0 to 360 (in degrees)
Satellites	1	The number of satellites
BSID0		**The following data are related to home base station and valid only if BIT1 of *mask* is 1**
MCC	2	Mobile Country Code — Unsigned 16 bits integer
MNC	2	Mobile Network Code — Unsigned 16 bits integer
LAC	2	Location Area Code — Unsigned 16 bits integer
CID	4	Cell ID with RNC — Unsigned 32 bits integer
RxLev	1	Cell signal level — Unsigned 8 bits integer (0: -110dB 1:-109dB 2:-108dB … 110: 0dB)
BSID1		**The following data are related to the 1st neighbor base station and valid only if BIT2 of *mask* is 1**

Name	Bytes	Description
LAC	2	Same as definition in *BSID0*
CI	4	Same as definition in *BSID0*
RxLev	1	Same as definition in *BSID0*
BSID2		**The following data are related to the 2nd neighbor base station and valid only if BIT3 of *mask* is 1**
LAC	2	Same as definition in *BSID0*
CI	4	Same as definition in *BSID0*
RxLev	1	Same as definition in *BSID0*
BSS0		**The following data are related to the 1st WiFi hot-spot and valid only if BIT4 of *mask* is 1**
BSSID	6	WiFi MAC address
RSSI	1	WiFi signal level — Signed 8 bits integer (in dB)
BSS1		**The following data are related to the 2nd WiFi hot-spot and valid only if BIT5 of *mask* is 1**
BSSID	6	Same as definition in *BSS0*
RSSI	1	Same as definition in *BSS0*
BSS2		**The following data are related to the 3rd WiFi hot-spot and valid only if BIT6 of *mask* is 1**
BSSID	6	Same as definition in *BSS0*
RSSI	1	Same as definition in *BSS0*

4 TCP/IP

The protocol between device and server is based on IP network. It can be based on either TCP or UDP. This chapter describes the difference between using TCP and using UDP.

4.1 USING TCP

TCP is a transmission protocol based on IP network. It establishes a stream-like tube between device and server, and provides reliable, ordered, and error-checked delivery of a stream of octets. Any data enter the tube, and they will arrived their destination with correct content in correct order. As a result, the packages, described in the following chapters, are injected into TCP session without any extra encapsulation.

Its disadvantage is also its stream-like feature. The delivered data may be split and recombined during transmitting. So, in order to recover original package, the destination must detect the package head, then get the package body. As a result, the destination must save all partial packages. Only when a whole package is recognized, the destination can process it.

A TCP tube is as below:

...	Package 1	Package 2	...	Package N	...

4.2 USING UDP

With UDP, device can send messages, in this case referred to as datagrams, to other hosts on an IP network. Prior communications are not required in order to set up transmission channels or data paths. It has no handshaking dialogues, and thus exposes the device to any unreliability of the underlying network: there is no guarantee of delivery, ordering, or duplicate protection. As a result, an extra encapsulation must be applied on all transmitting packages.

A UDP header will be added to the transmitting data. It includes datagram size, datagram checksum and device IMEI. The destination must check the integrity of datagram using them. Each datagram is allowed to contain multiple packages, but its

total size is limited up to 1200 bytes.

A UDP datagram is as below:

UDP Header	Package 1	Package 2	...	Package N

The structure of a UDP header is described as below:

Name	Bytes	Description
Mark	2	'EL'
Size	2	Datagram size from next byte to end
Checksum	2	Datagram checksum (see note 1) from next byte to end
IMEI	8	Device IMEI

> Note:
>
> 1. The checksum algorithm is described in Appendix CHECKSUM ALGORITHM.

5 PACKAGES

5.1 GENERAL

In general, the structure of a package is described as below:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
Content	N	Package content

> Note:
>
> 1. *Package size* is the total size of *Sequence* and *Content*.
> 2. *Package sequence number* starts from 1 after device is rebooted, and is increased by 1 for every package.
> 3. All response packages must have same *Package identifier* and *Package sequence number* as the request package has.

The *package identifier* are listed as below:

PID	Direction	Respond?	Type
0x01	UP	Y	Login package
0x03	UP	Y	Heartbeat package
0x12	UP	Y/N	Location package
0x14	UP	Y	Warning package
0x15	UP	Y	Report package
0x16	UP	Y	Message package

PID	Direction	Respond?	Type
0x17	UP	Y	OBD data package
0x18	UP	Y	OBD body package
0x19	UP	Y	OBD fault package
0x1A	UP	Y	Pedometer package
0x1B	UP	Y	Param-set package
0x80	DOWN	Y	Instruction package
0x81	DOWN	Y	Broadcast package

> Note:
>
> 1. *UP* means that this package is originated by device. It will be sent to server and server should process and respond it.
> 2. *DOWN* means that this package is originated by server. It will be sent to device and device should process and respond it.
> 3. The PID of an *UP* package is less than 0x7F, and the PID of a *DOWN* package is equal to or greater than 0x80.
> 4. If using TCP, server do not need to respond the location package (0x12).

5.2 LOGIN PACKAGE — 0x01

If using TCP, login package will be sent to server immediately after every session is established. If using UDP, it will be sent only once after the device is rebooted. The server must respond it, or all other packages will not be sent.

Its structure is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x01
Size	2	Package size from next byte to end — Unsigned 16 bits integer

Name	Bytes	Description
Sequence	2	Package sequence number — Unsigned 16 bits integer
IMEI	8	Device IMEI
Language	1	Device language: 0x00 — Chinese; 0x01 — English; Other — Undefined
Timezone	1	Device timezone — Signed 8 Bits integer (in 15 mins)
Sys Ver	2	System version — Unsigned 16 bits integer (e.g. 0x0205: V2.0.5)
App Ver	2	Application version — Unsigned 16 bits integer (e.g. 0x0205: V2.0.5)
PS Ver	2	Param-set (see note 1) version — Unsigned 16 bits integer (e.g. 0x0001: V1)
PS OSize	2	Param-set original size — Unsigned 16 bits integer
PS CSize	2	Param-set compressed size (see note 2) — Unsigned 16 bits integer
PS Sum16	2	Param-set checksum (see note 3) — Unsigned 16 bits integer

> Note:
>
> 1. *Param-set* is the set of all parameters, which is described in Appendix A.3 PARAM-SET.
> 2. The deflate algorithm is described in Appendix A.2 DEFLATE ALGORITHM.
> 3. The checksum algorithm is described in Appendix A.1 CHECKSUM ALGORITHM.

The response from server is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x01

Name	Bytes	Description
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
Time	4	Current time (UTC) in the server
Version	2	Protocol version (see note 1) — 0x01: default
PS Action	1	Param-set action mask (see note 2)

> Note:
>
> 1. *Protocol version" is the version of the protocol supported by server. If it is different from the protocol version in device, device will generate and transmit only the compatible packages to server.
> 2. When server receives the login package from device, it can check the information of Param-set to determine how to operates the Param-set. Then 2 optional actions may be taken and both of them can be taken at the same time:
> Bit0: 1 — Tell device to upload the Param-set immediately. 0 — Do not upload it now.
> Bit1: 1 — Tell device to upload the Param-set if changed in the future. 0 — Do not upload it in the future.

An example of login package and the response:

```
U: 67670100180005035254407167747100200205020500010432000088BD
D: 676701000900005590BD477000103
```

5.3 HEARTBEAT PACKAGE — 0x03

Heartbeat package only appears in TCP. It is used to keep the session active. In common situation, if nothing are sent via the pathway (GPRS) in a few minutes, the pathway may be recycled by network provider. So heartbeat package must be sent to keep the session active when it is nearly due.

If device has sent some heartbeat packages and not received any response, it will cut the corrupt connection and attempt to establish a new one.

UDP is not a stream-like protocol, so heartbeat package is not necessary for it.

Its structure is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x03
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
Status	2	Device status, see Section 3.5 STATUS

The response from server is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x03
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer

An example of heartbeat package and the response:

```
U: 676703000400070188
D: 67670300020007
```

5.4 LOCATION PACKAGE — 0x12

Location package is the most important package. It transfers the position and other information of device to server. If using TCP, the server need not respond it. If using UDP, the server must respond it.

Its structure is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x12
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
Location	N	Device position, see Section 3.6 POSITION
Status	2	Device status, see Section 3.5 STATUS
Battery	2	Battery voltage (in mV) — Unsigned 16 bits integer
AIN0	2	AIN0 value (in mV) — Unsigned 16 bits integer
AIN1	2	AIN1 value (in mV) — Unsigned 16 bits integer
Mileage	4	Device mileage (in m) — Unsigned 32 bits integer
GSM Cntr	2	GSM counter from last *GSM* command (in min) — Unsigned 16 bits integer
GPS Cntr	2	GPS counter from last *GPS* command (in min) — Unsigned 16 bits integer
PDM Step	2	Accumulated steps today — Unsigned 16 bits integer
PDM Time	2	Accumulated walking time today (in sec) — Unsigned 16 bits integer
Temperature	2	Internal temperature (in (1/256)C) — Signed 16 bits integer
Humidity	2	Humidity (in (1/10)%) — Unsigned 16 bits integer
Illuminance	4	Illuminance (in (1/256)lx) — Unsigned 32 bits integer
CO2	4	CO2 concentration (in ppm) — Unsigned 32 bits integer
Probe	2	Probe temperature (in (1/16)C) — Signed 16 bits integer

> Note:

> 1. *AIN* is the abbreviation of analog input port.
> 2. *Mileage* is accumulated only when GPS is fixed.
> 3. *GSM Counter* is used to recognize which phase GSM module is in. The phases are described in Section 6.3.11 GSM.
> 4. *GPS Counter* is used to recognize which phase GPS module is in. The phases are described in Section 6.3.12 GPS.
> 5. Not all fields are valid in a device, so please ignore non-existent or unnecessary data. For example: sensor data are valid only if relevant sensors exist. If a datum is invalid, its value will be zero.

The response from server is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x12
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer

An example of location package and the response:

```
U:
6767120041002259OBD94203026B940D0C3952AD00210000000000001CC0001A53F0170F0AB1301890F08
000000000000C2D0001C00160000000000000000000000000000000000
D: 67671200020022
```

5.5 WARNING PACKAGE — 0x14

A warning package will be sent to server when a specific warning occurs.

Its structure is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x14

Name	Bytes	Description
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
Location	N	Device position, see Section 3.6 POSITION
Warning	1	Warning type — Unsigned 8 bits integer
Status	2	Device status, see Section 3.5 STATUS

The *warning type* is listed as below:

Value	Description
0x02	SOS
0x01	External power cut-off
0x03	Battery low (only for the device which uses a battery as main power)
0x08	GPS antenna open-circuit (only for the device with external GPS antenna)
0x09	GPS antenna short-circuit (only for the device with external GPS antenna)
0x04	Activity warning (only for the device with an accelerometer)
0x85	Crash warning (only for the device with an accelerometer)
0x86	Free-fall warning (only for the device with an accelerometer)
0x81	Under-speed warning
0x82	Over-speed warning
0x83	In-to-fence warning
0x84	Out-of-fence warning
0x05	Shift warning
0x20	Out of internal temperature range
0x21	Out of humidity range

Value	Description
0x22	Out of illuminance range
0x23	Out of CO2 concentration range
0x24	Out of probe temperature range

The response from server is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x14
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
Content	N	Warning content — String

> Note:
>
> 1. *Warning content* has variable length. Its length is the size of the body.
> 2. If *warning content* is not empty, it will be sent as a message to all managers registered in device.

An example of warning package and the response:

```
U:
6767140024000A590BD54903026B940D0C3952AD00210004000000501CC0001A53F0170F0AB19020789
D:
676714004A000A534F53E68AA5E8ADA621E5B9BFE4B89CE79C81E6B7B1E59CB3E5B882E58D97E5B1B1E5
8CBAE9BD90E6B091E9819333EFBC88E8B79DE5AE87E998B3E5A4A7E58EA630E7B1B3EFBC89
```

5.6 REPORT PACKAGE — 0x15

A report package will be sent to server when a specific event occurs.

Its structure is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x15
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
Location	N	Device position, see Section 3.6 POSITION
Report	1	Report type — Unsigned 8 bits integer
Status	2	Device status, see Section 3.5 STATUS

The *report type* is listed as below:

Value	Description
0x01	ACC on
0x02	ACC off
0x03	DINx changed

The response from server is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x15
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer

An example of report package and the response:

```
U:
6767150024000B590BD57103026B940D0C3952AD00210000000000501CC0001A53F0170F0AB18020789
```

```
D: 67671500020008
```

5.7 MESSAGE PACKAGE — 0x16

If the command from user needs a result from server or it is a server-managed command, it will be encapsulated into a message package, and then the package will be transmitted to server. After that, server should return a meaningful result if it is a valid command, or returns a simple acknowledge with empty content. Finally, device will forward the valid response to user.

If the command is a device-managed command, it will be processed and responded directly by device.

Its structure is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x16
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
Location	N	Device position, see Section 3.6 POSITION
Number	21	Phone number — String
Content	N	Message content — String

> Note:
>
> 1. All query-address commands will be sent to server.
> 2. The message which is single line and ends with '?' and '#' will be sent to server.

The response from server is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x16
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
Number	21	Phone number — String
Result	N	Result content — String

> Note:
>
> 1. *Phone number* must be the same as the *phone number* in the request package.
> 2. If *result content* is not empty, it will be sent as a message to *phone number*.

An example of message package and the response:

```
U:
6767160039000D590BD5AF03026B940D0C3952AD00210000000000501CC0001A53F0170F0AB1732303138
353636323231323530000000000000000000313233
D:
6767160056000D32303138353636323231323530000000000000000000E5B9BFE4B89CE79C81E6B7B1E59C
B3E5B882E58D97E5B1B1E58CBAE9BD90E6B091E9819333EFBC88E8B79DE5AE87E998B3E5A4A7E58EA631
39E7B1B3EFBC89
```

5.8 OBD DATA PACKAGE — 0x17

If device has an OBD module, it will attempt to read the specific PID data from vehicle. If specific condition is met, an OBD data package will be generated and transmitted to server.

Its structure is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x17
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
Location	N	Device position, see Section 3.6 POSITION
PID Data	N	PID data

The *PID data* contains many groups. Each group has 5 bytes. The first byte in a group is the PID code, and following 4 bytes are its value. For example, 00FFFFFFFF represents that the value of PID00 is 0xFFFFFFFF, 0233445566 represents the value of PID02 if 0x33445566, etc. All standard PIDs are described in Wikipedia OBD-II PIDs, and all extended PIDs are described in Appendix A.4 Extended OBD-II PIDs.

> Note:
>
> 1. The PIDs to be monitored are defined by command “MONITOR”.

The response from server is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x17
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer

5.9 OBD BODY PACKAGE — 0x18

If device has an OBD module, it will attempt to read the body status of vehicle. Whenever the body status is changed, an OBD body package will be generated and transmitted to server.

Its structure is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x18
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
Location	N	Device position, see Section 3.6 POSITION
Body	4	Body status

The *body status* has 4 bytes which is described as below:

- BYTE0: Door status
  - BIT0: Left-front door (0: Closed 1: Open)
  - BIT1: Right-front door (0: Closed 1: Open)
  - BIT2: Left-back door (0: Closed 1: Open)
  - BIT3: Right-back door (0: Closed 1: Open)
  - BIT4: Luggage door (0: Closed 1: Open)
- BYTE1: Gear status
  - 0x00: Not supported
  - 0x50: Park
  - 0x52: Reverse
  - 0x4E: Neutral
  - 0x44: Drive
- BYTE2: LED status
  - BIT0: Engine LED (0: Closed 1: Open)
  - BIT1: ABS LED (0: Closed 1: Open)
  - BIT2: SRS LED (0: Closed 1: Open)
  - BIT3: Brake LED (0: Normal/Released 1: Abnormal/Braking)
  - BIT4: Central locking (0: Unlocked 1: Locked)
- BYTE3: Misc
  - BIT0: ACC (0: Off 1: On)
  - BIT1: Tire Pressure Monitor System (0: Normal 1: Abnormal)
  - BIT2, BIT3: Remote key (00: No key is pressed 01: *Unlock* key is pressed 10: *Lock* key is pressed)

For example: 1F001000. The first byte 0x1F represents that all doors are open. The

second byte 0x00 represents that reading gear status is not supported. The third byte 0x10 represents that car has been locked. The forth byte 0x00 represents that ACC is off, tire pressure is normal and no remote key is pressed.

The response from server is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x18
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer

5.10 OBD FAULT PACKAGE — 0x19

If device has an OBD module, it will attempt to read the fault code of vehicle. Whenever some faults occur in vehicle, an OBD fault package will be generated and transmitted to server .

Its structure is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x19
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
Location	N	Device position, see Section 3.6 POSITION
Fault	N	Fault code

The first byte of *fault code* is the data type which is always 0x00 by now. The following bytes have many group. Each group has 3 bytes which represents a fault. The first 2

bytes in a group are the fault code which is described by car manufacturer. The third byte in a group is the fault status (0x01: Determined 0x02: Pending).

For example: 00020502009302. The first byte 0x00 is the data type. 0x020502 and 0x009302 represent there has been 2 faults which are P0205 and P0093. Both faults have a pending status.

> Note:
>
> 1. The description of all fault codes can be found on https://www.obd-codes.com/trouble_codes/.

The response from server is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x19
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer

5.11 PEDOMETER PACKAGE — 0x1A

If device has an accelerometer, it will attempt to analyze the acceleration data as a pedometer. The result will be sent to server per a day.

Its structure is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x1A
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer

Name	Bytes	Description
Today	4	UTC time of last day, see Section 3.4 TIME
All Step	4	Accumulated steps in total — Unsigned 32 bits integer
All Time	4	Accumulated walking time in total (in sec) — Unsigned 32 bits integer
All Distance	4	Accumulated walking distance in total (in mm) — Unsigned 32 bits integer
All Energy	4	Accumulated burnt energy in total (in cal) — Unsigned 32 bits integer
Day Step	4	Accumulated steps in last day — Unsigned 32 bits integer
Day Time	4	Accumulated walking time in last day (in sec) — Unsigned 32 bits integer
Day Distance	4	Accumulated walking distance in last day (in mm) — Unsigned 32 bits integer
Day Energy	4	Accumulated burnt energy in last day (in cal) — Unsigned 32 bits integer

The response from server is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x1A
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer

5.12 PARAM-SET PACKAGE — 0x1B

Param-set package will be sent to server when server requests param-set or param-set is changed. The data of param-set may be compressed and split. Server can not parse it

until all blocks are received and merged.

Its structure is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x1B
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
PS Ver	2	Param-set (see note 1) version — Unsigned 16 bits integer (e.g. 0x0001: V1)
PS OSize	2	Param-set original size — Unsigned 16 bits integer
PS CSize	2	Param-set compressed size (see note 2) — Unsigned 16 bits integer
PS Sum16	2	Param-set checksum (see note 3) — Unsigned 16 bits integer
Offset	2	Uploading offset — Unsigned 16 bits integer
Data	N	Uploading data

> Note:
>
> 1. *Param-set* is the set of all parameters, which is described in Appendix A.3 PARAM-SET.
> 2. The deflate algorithm is described in Appendix A.2 DEFLATE ALGORITHM.
> 3. The checksum algorithm is described in Appendix A.1 CHECKSUM ALGORITHM.

The response from server is:

Name	Bytes	Description
Mark	2	0x67 0x67

Name	Bytes	Description
PID	1	Package identifier — 0x1B
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
Next	1	Indicate whether next block is sent — 0: Do not send 1: Send

An example of param-set package and the response:

```
U:
67671B009E0005000104320092 66DF000008053FC0A20341303EFE8110D414404C0680185610CEF3A23C
8C18154005AB64300BD0AAA845755C0CE331CF0C1B036478B843D0EA288988320B42D068956405053C11
A4588FA38803FD599EC6EF4B7383D0FC3FB7333919EA637F3D8EFB1D79F9D27B8D7782191146AE344DC0
766F01599EE898BBE5ED3217444DBECA0AB4BADA4B08224A48F235D59759EDEB2A24EE9C20
D: 67671B0003000500
```

5.13 INSTRUCTION PACKAGE — 0x80

Sometimes, server need request device to change a setting or to do something else. In order to do that, an instruction package will be sent to device.

Its structure is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x80
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
Type	1	Instruction type (see note 1) — Unsigned 8 bits integer
UID	4	Instruction UID (see note 2) — Unsigned 32 bits integer

Name	Bytes	Description
Content	N	Instruction content — String

The *instruction type* is:

- 0x01: Indicate that *instruction content* is a device command
- Other: Reserved

The *instruction UID* is a unique number recognized by server. Device must respond same UID in response package.

The response from device is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x80
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
Type	1	Instruction type — Unsigned 8 bits integer
UID	4	Instruction UID — Unsigned 32 bits integer
Result	N	Instruction result — String

An example of instruction package and the response:

```
D: 676780000F5788014C754C7576657273696F6E23
U:
67678000905788014C754C75494D45493A33353235343430373136373734373 10A494D53493A393436303031373331393831373638340A49434349443A383938363031313438343132303231333033383 20A53595354454D3A4D373630315F56322E302E350A56455253494F4E3A4D584150505F56322E302E350A4255494C443A4D617920203520323031372030393A32313A3038
```

5.14 BROADCAST PACKAGE — 0x81

Sometimes, server need request device to send a message to specific phone number or to broadcast a message to all managers. In order to do that, a broadcast package will be sent to device.

Its structure is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x81
Size	2	Package size from next byte to end — Unsigned 16 bits integer
Sequence	2	Package sequence number — Unsigned 16 bits integer
Type	1	Broadcast type (see note 1) — Unsigned 8 bits integer
Number	21	Phone number (see note 2) — String
Content	N	Message content — String

The *broadcast type* is:

- 0x01: Indicate that *message content* should be sent to specific recipient
- Other: Reserved

If the *phone number* is not empty, it will be the specific recipient. If it is empty, all managers will be specific recipients.

The response from device is:

Name	Bytes	Description
Mark	2	0x67 0x67
PID	1	Package identifier — 0x81
Size	2	Package size from next byte to end — Unsigned 16 bits integer

Name	Bytes	Description
Sequence	2	Package sequence number — Unsigned 16 bits integer

6 COMMANDS

All command keywords are case insensitive, but their parameters should be case sensitive. For example, “STATUS#” is the same as “status#”, but “APN,CMNET#” is different from “APN,cmnet#”.

In general, end with ‘#’ indicates it is an executive command, and end with ‘?’ indicates it is a query command. For example, “APN,cmnet#” is used to modify APN setting, and “APN?” is used to query current APN setting.

6.1 SECURITY COMMANDS

6.1.1 LOGIN

If a password has been set in device, device is protected. If user wishes to send a command to a protected device via short message, it is necessary for him to log in device. After he logs in, he acquires the privilege to execute a command on device. At last, user can log out from device by specific command, or will log out automatically if idle more than 15 minutes.

If user has been registered as a manager, he need not log in to send a command.

```
LOGIN,[PASSWORD]#
> Login OK
> Login Error
```

6.1.2 LOGOUT

This command requests to log out from device immediately.

```
LOGOUT#
> Logout OK
> Logout Error
```

6.1.3 PASSWORD

This command requests to change the password in device.

```
PASSWORD,[OLD PASSWORD],[NEW PASSWORD]#
> SET PASSWORD OK
> SET PASSWORD Error
```

6.2 ACTION COMMANDS

6.2.1 UPGRADE

This command requests to upgrade the firmware in device.

```
UPGRADE,[REMOTE],[DOMAIN],[PORT]#
> Upgrade OK
> Upgrade Error
> Upgrade Error: Device busy
> Upgrade Error: Transfer fail
> Upgrade Error: Disk full
> Upgrade Error: File wrong

UPGRADE?
> TRANSFER:[STATE],[BLOCKS]
```

Name	Description
[REMOTE]	The remote file name
[DOMAIN]	The address of upgrade server (domain name or IP)
[PORT]	The port of upgrade server
[STATE]	The transferring state: IDLE, INITIAL, ACTIVE
[BLOCKS]	The transferred blocks

6.2.2 RESET

This command requests to reboot device.

```
RESET#
> Reset OK
> Reset Error
```

6.2.3 SHUTDOWN

This command requests to shutdown device. It is valid only if device does not have an automatic power control system.

```
SHUTDOWN#
> Shutdown OK
> Shutdown Error
```

6.2.4 FACTORY

This command requests to restore all settings to default in factory.

```
FACTORY#
> Factory OK
> Factory Error
```

6.2.5 PAGING

This command requests to upload a location package (0x12) now.

```
PAGING#
> Paging OK
> Paging Error
```

6.2.6 CLEAR

This command requests to clear all pending packages.

```
CLEAR#
> Clear OK
> Clear Error
```

6.2.7 LISTEN

This command requests to originate an outgoing call to specific phone number. The recipient may answer it in order to listen what is happening.

```
LISTEN,[NUMBER]#
> Listen OK
> Listen Error
```

Name	Description
[NUMBER]	The phone number to be called

6.2.8 RELAY

This command requests to set/unset relay. It is valid only if device supports a relay.

```
RELAY,[PATTERN]#
> Relay OK
> Relay Error

RELAY?
> RELAY:[STATE]
```

Name	Description
[PATTERN]	0: Disable relay 1: Enable relay immediately 2: Enable relay safely
[STATE]	ON/OFF

> Note:

> 1. When [PATTERN] is set to 1, the relay command will be executed immediately.
> 2. When [PATTERN] is set to 2, the relay command will be executed safely. The vehicle is safe only when the speed is lower than 20km/h if GPS is fixed, or it is stationary if GPS is not fixed.

6.2.9 FWD

This command requests to forward a message to specific phone number. After message is forwarded, all responses from the recipient in 5 minutes will be sent back to original owner. It can be used to query some information from service provider, e.g. remaining fee or network traffic.

```
FWD,[NUMBER],[CONTENT]#
> Forward OK
> Forward Error
```

Name	Description
[NUMBER]	The phone number
[CONTENT]	The message content

6.3 SETTING COMMANDS

6.3.1 IMEI

This command requests to change the IMEI of device.

```
IMEI,[IMEI]#
> SET IMEI OK
> SET IMEI Error

IMEI?
> IMEI:[IMEI]
```

Name	Description
[IMEI]	The IMEI of device

6.3.2 LANG

This command requests to change the language of device.

```
LANG,[LID]#
> SET LANG OK
> SET LANG Error

LANG?
> LANG:[LNAME]
```

Name	Description
[LID]	The language ID — 0: English 1: Chinese
[LNAME]	The language name — EN/CN

6.3.3 GMT

This command requests to change the time zone of device.

```
GMT,[E/W],[HOUR],[MINUTE],[DST]#
> SET GMT OK
> SET GMT Error

GMT?
> GMT:[E/W][TZ],[DST]
```

Name	Description
[E/W]	Which globe — E: East W: West
[HOUR]	The hour part of time difference — -12 ~ 12
[MINUTE]	The minute part of time difference — 0, 15, 30, 45

Name	Description
[DST]	The day saving time (in hours) — 0, 1, 2
[TZ]	The time difference

6.3.4 HBT

This command requests to change the heartbeat timer. This parameter defines the idle time before device originates a heartbeat package in TCP session.

```
HBT,[HBT]#
> SET HBT OK
> SET HBT Error

HBT?
> HBT:[HBT]
```

Name	Description
[HBT]	The heartbeat timer (in minutes) — Float point number

6.3.5 DELAY

This command requests to change the answer timer. This parameter defines the delay time before device answers an incoming call.

```
DELAY,[DELAY]#
> SET DELAY OK
> SET DELAY Error

DELAY?
> DELAY:[DELAY]
```

Name	Description
[DELAY]	The answer timer (in seconds) — Integer

6.3.6 APN

This command requests to change the APN of GPRS/3G/4G.

```
APN,[APN],[USERNAME],[PASSWORD]#
> SET APN OK
> SET APN Error

APN?
> APN:[APN],[USERNAME],[PASSWORD]
```

Name	Description
[APN]	The APN to GPRS/3G/4G service
[USERNAME]	The username for GPRS/3G/4G service
[PASSWORD]	The password for GPRS/3G/4G service

6.3.7 SERVER

This command requests to change the address of location server.

```
SERVER,[SERVER]#
> SET SERVER OK
> SET SERVER Error

SERVER?
> SERVER:[SERVER]([IP])
```

Name	Description
[SERVER]	The URI of location server
[IP]	The IP address of location server, e.g. 202.128.0.32

> Note:
>
> 1. If using TCP, the URI should be as “tcp://www.domain.com:12345”.

2. If using UDP, the URI should be as “udp://www.domain.com:54321”.
3. [IP] is valid only when device has connected to location server.

6.3.8 COLLECT

This command requests to change the parameters of location collection. All parameters define how to collect location package and how many location packages to be cached before they are sent to server. We have 3 strategies to collect location package.

The first strategy is based on time. The location packages are generated in specific interval. There are 2 intervals: [INTERVAL] and [ACTIVE]. [INTERVAL] defines the regular interval. [ACTIVE] defines the time interval when device is active/moving.

The second strategy is based on position. After device moves a specific distance, a location package will be generated. [DISTANCE] defines the gap.

The third strategy is based on course. When device turns more than a specific angle, a location package will be generated. [TURN] defines the angle.

Every strategy can be omitted if its parameter is set to 0. If multiply strategies are set, a location packages will be generated when any one is met.

```
COLLECT,[INTERVAL],[DISTANCE],[TURN],[ACTIVE],[QUANTITY]#
> SET COLLECT OK
> SET COLLECT Error

COLLECT?
> COLLECT:[INTERVAL],[DISTANCE],[TURN],[ACTIVE],[QUANTITY]
```

Name	Description
[INTERVAL]	The time interval (in seconds)
[DISTANCE]	The running distance (in meters)
[TURN]	The turning angle (in degrees)
[ACTIVE]	The time interval when device is moving/active (in seconds)
[QUANTITY]	The number of cached location packages before they are sent

6.3.9 MANAGER

This command requests to add/remove/modify one or more managers in device. Up to 8 managers can be added into device.

```
MANAGER,[INDEX],[NUMBER],[ALIAS]#
> SET MANAGER OK
> SET MANAGER Error

MANAGER,[INDEX]?
> MANAGER[INDEX]:[NUMBER],[ALIAS]
> GET MANAGER Error
```

Name	Description
[INDEX]	The index of manager — Integer, 0 - 8
[NUMBER]	The phone number of manager
[ALIAS]	The alias of manager

> Note:
>
> 1. When [INDEX] is 0, it means a command to all managers.
> 2. [ALIAS] can be displayed on screen if it is supported.

Example	Description
MANAGER,1,13012345678#	Add/change the first manager to 13012345678 without alias
MANAGER,1,13012345678,MUM#	Add/change the first manager to 13012345678 with an alias
MANAGER,1#	Remove the first manager
MANAGER,0#	Remove all managers
MANAGER,1?	Return the first manager
MANAGER,0?	Return all managers

6.3.10 AGPS

This command requests to change the address of AGPS server in device. Now, three AGPS schemes are supported: SUPL, MediaTek EPO and uBlox AssistNow. Most products support one or two of them.

```
AGPS,[SERVER]#
> SET AGPS OK
> SET AGPS Error

AGPS?
> AGPS:[AGPS]([IP])
```

Name	Description
[SERVER]	The URI of AGPS server
[IP]	The IP address of AGPS server, e.g. 202.128.0.32

> Note:
>
> 1. The URI format for SUPL is “supl://supl.google.com:7276”.
> 2. The URI format for MediaTek EPO is “mtk://www.domain.com:12345”.
> 3. The URI format for uBlox AssistNow is “ubx://agps.u-blox.com:46434”.
> 4. [IP] is valid only when device has connected to server.

6.3.11 GSM

GSM module has 2 states: ON and OFF. OFF is also called flight mode. When GSM module is OFF, GSM and GPRS will be disconnected. Normally, we can use this command to save a lot of power consumption.

This command requests to change the work mode and parameters of GSM module. It defines how and when GSM module switches in 2 states.

```
GSM,[MODE],[T0],[T1_TOTAL],[T1_WAKING],[T2_PERIODIC],[T2_WAKING]#
> SET GSM OK
```

```
> SET GSM Error

GSM?
> GSM:[MODE],[T0],[T1_TOTAL],[T1_WAKING],[T2_PERIODIC],[T2_WAKING],[GSM_RUN]
```

Name	Description
[MODE]	The work mode — 0: ALWAYS ON 1: AUTOMATIC 2: ON TIMERS 3: ALWAYS OFF
[T0]	The work time after GSM module is awaken (in seconds)
[T1_TOTAL]	The total time of phase 1 (in minutes)
[T1_WAKING]	The work time in phase 1 (in minutes)
[T2_PERIODIC]	The periodic time of phase 2 (in minutes)
[T2_WAKING]	The work time in phase 2 (in minutes)
[GSM_RUN]	The running time from last GSM command (in minutes)

If [MODE] is 0, GSM module will be always ON and other parameters are omitted.

If [MODE] is 3, GSM module will be always OFF and other parameters are omitted.

If [MODE] is 1 or 2, GSM module will be ON based on the timer defined with other parameters. There are 2 phases to be defined. At first, GSM module enters phase 1 when the command is executed. Then it enters phase 2 after phase 1 is finished. Phase 1 will run about [T1_TOTAL] minutes. In phase 1, GSM module will be ON about [T1_WAKING] minutes and then be OFF in remaining time. Phase 2 is periodic in [T2_PERIODIC] minutes. In each period of phase 2, GSM module will be ON about [T2_WAKING] minutes and then be OFF in remaining time.

If [MODE] is 1, GSM module will be ON if it is active besides the above timers. The activity is detected by accelerometer.

Example	Description
GSM,0#	GSM module is always ON.
GSM,3#	GSM module is always OFF.

Example	Description
GSM,1#	GSM module is ON when device is active.
GSM,1,120,0,0,60,5#	GSM module is ON about 5 minutes per 60 minutes or when device is active.
GSM,2,120,0,0,60,5#	GSM module is ON about 5 minutes per 60 minutes.
GSM,2,120,120,120,60,5#	GSM module is ON 120 minutes at first. Then it is ON about 5 minutes per 60 minutes.

6.3.12 GPS

GPS module has 2 states: ON and OFF. When GPS module is OFF, GPS chip is closed. Normally, we can use this command to save a lot of power consumption.

This command requests to change the work mode and parameters of GPS module. It defines how and when GPS module switches in 2 states.

```
GPS,[MODE],[T0],[T1_TOTAL],[T1_WAKING],[T2_PERIODIC],[T2_WAKING]#
> SET GPS OK
> SET GPS Error

GPS?
> GPS:[MODE],[T0],[T1_TOTAL],[T1_WAKING],[T2_PERIODIC],[T2_WAKING],[GPS_RUN]
```

Name	Description
[MODE]	The work mode — 0: ALWAYS ON 1: AUTOMATIC 2: ON TIMERS 3: ALWAYS OFF
[T0]	The work time after GPS module is awaken (in seconds)
[T1_TOTAL]	The total time of phase 1 (in minutes)
[T1_WAKING]	The work time in phase 1 (in minutes)
[T2_PERIODIC]	The periodic time of phase 2 (in minutes)
[T2_WAKING]	The work time in phase 2 (in minutes)
[GPS_RUN]	The running time from last GPS command (in minutes)

If [MODE] is 0, GPS module will be always ON and other parameters are omitted.

If [MODE] is 3, GPS module will be always OFF and other parameters are omitted.

If [MODE] is 1 or 2, GPS module will be ON based on the timer defined with other parameters. There are 2 phases to be defined. At first, GPS module enters phase 1 when the command is executed. Then it enters phase 2 after phase 1 is finished. Phase 1 will run about [T1_TOTAL] minutes. In phase 1, GPS module will be ON about [T1_WAKING] minutes and then be OFF in remaining time. Phase 2 is periodic in [T2_PERIODIC] minutes. In each period of phase 2, GPS module will be ON about [T2_WAKING] minutes and then be OFF in remaining time.

If [MODE] is 1, GPS module will be ON if it is active besides the above timers. The activity is detected by accelerometer.

Example	Description
GPS,0#	GPS module is always ON.
GPS,3#	GPS module is always OFF.
GPS,1#	GPS module is ON when device is active.
GPS,1,120,0,0,60,5#	GPS module is ON about 5 minutes per 60 minutes or when device is active.
GPS,2,120,0,0,60,5#	GPS module is ON about 5 minutes per 60 minutes.
GPS,2,120,120,120,60,5#	GPS module is ON 120 minutes at first. Then it is ON about 5 minutes per 60 minutes.

6.3.13 ALARM

This command requests to add/remove/modify one or more alarm clocks in device. Up to 8 alarm clocks can be added into device. They can be used to do something in specific time.

```
ALARM,[INDEX],[ACTION],[TIME],[WEEK],[NOTE]#
> SET ALARM OK
> SET ALARM Error
```

```
ALARM,[INDEX]?
> ALARM[INDEX]:[ACTION],[TIME],[WEEK],[NOTE]
> GET ALARM Error
```

Name	Description
[INDEX]	The index of alarm clock — Integer, 0 - 8
[ACTION]	The action when alarm clock is expired — Integer
[TIME]	The expire time of alarm clock
[WEEK]	The week days of alarm clock — Hexadecimal
[NOTE]	The note of alarm clock

> Note:
>
> 1. When [INDEX] is 0, it means a command to all alarm clocks.

[ACTION]:

- 0: Alarm clock is removed
- 1: Power on automatically
- 2: Power off automatically
- 3: Display note on screen and play the preset tone 1
- 4: Display note on screen and play the preset tone 2
- 5: Display note on screen and play the preset tone 3
- 6: Display note on screen and play the preset tone 4
- 7: Display note on screen and play the preset tone 5
- 8: Display note on screen and play the preset tone 6
- 9: Display note on screen and play the preset tone 7
- 10: Display note on screen and play the preset tone 8

[WEEK]:

- BIT0: Sunday
- BIT1: Monday
- BIT2: Tuesday
- BIT3: Wednesday
- BIT4: Thursday
- BIT5: Friday
- BIT6: Saturday

Example	Description
ALARM,1,3,18:00,7F,DINNER#	Add/change the first alarm clock at 18:00 everyday
ALARM,2,4,22:00,3E,BEDTIME#	Add/change the second alarm clock at 22:00 on weekday
ALARM,1#	Remove the first alarm clock
ALARM,0#	Remove all alarm clocks
ALARM,1?	Return the first alarm clock
ALARM,0?	Return all alarm clocks

6.3.14 MILEAGE

This command requests to initialize the mileage in device. After the mileage is initialized, it will be increased automatically when GPS is fixed.

```
MILEAGE,[MILEAGE]#
> SET MILEAGE OK
> SET MILEAGE Error

MILEAGE?
> MILEAGE:[MILEAGE](km)
```

Name	Description
[MILEAGE]	The mileage (in km)

6.3.15 MOTION

This command requests to enable/disable motion warning and set its parameter. After motion warning is enabled, any vibration will trigger a warning.

```
MOTION,[SENSE],[DELAY]#
> SET MOTION OK
> SET MOTION Error
```

```
MOTION?
> MOTION:[SENSE],[DELAY]
```

Name	Description
[SENSE]	The sensitivity
[DELAY]	The delay time before a warning is emitted (in seconds)

[SENSE]:

- 0: Disable warning.
- 1 ~ 9: Enable warning. 1 is the most sensitive, and 9 is the least sensitive.

Example	Description
MOTION#	Disable motion warning
MOTION,2,5#	Trigger motion warning when an enough vibration continues 5 seconds

6.3.16 SPEED

This command requests to enable/disable speed warning and set its parameter. After speed warning is enabled, any speed not in range will trigger a warning.

```
SPEED,[LOW],[HIGH],[OVER]#
> SET SPEED OK
> SET SPEED Error

SPEED?
> SPEED:[LOW],[HIGH],[OVER]
```

Name	Description
[LOW]	The low limit of the speed (in km/h)
[HIGH]	The high limit of the speed (in km/h)
[OVER]	The speed threshold (in km/h) over which the device will drive the relay

> Note:
>
> 1. When [LOW] is 0, the under-speed warning is disabled.
> 2. When [HIGH] is 0, the over-speed warning is disabled.
> 3. When [OVER] is 0, the speed-relay feature is disabled.

Example	Description
SPEED#	Disable speed warning
SPEED,30,0#	Enable under-speed warning when speed is less than 30km/h
SPEED,0,100#	Enable over-speed warning when speed is more than 100km/h
SPEED,30,100#	Enable both under-speed warning and over-speed warning
SPEED,0,0,80#	Drive the relay when the speed is over 80 km/h

6.3.17 FENCE

This command requests to add/remove/modify one or more fences in device. Up to 8 fences can be added into device. Each fence can be round or rectangle. And it can also be out-type, in-type or bidirectional. If it is a out-type, the outside of fence is banned. If device leaves it, an out-of-fence warning will be triggered. If it is a in-type, the inside of fence is banned. If device enters it, an in-to-fence warning will be triggered. If it is bidirectional, any action to cross the border will trigger a warning.

```
FENCE,[INDEX],[FLAG],[LNG0],[LAT0],[RADIUS]#
FENCE,[INDEX],[FLAG],[LNG1],[LAT1],[LNG2],[LAT2]#
> SET FENCE OK
> SET FENCE Error

FENCE,[INDEX]?
> FENCE[INDEX]:[FLAG],[LNG0],[LAT0],[RADIUS]
> FENCE[INDEX]:[FLAG],[LNG1],[LAT1],[LNG2],[LAT2]
> GET FENCE Error
```

Name	Description
[INDEX]	The index of fence — Integer, 0 - 8

Name	Description
[FLAG]	The type and shape of fence — String, each char represents an attribution
[LNG0],[LAT0]	The longitude and latitude of the center of round fence
[RADIUS]	The radius of round fence (in meters)
[LNG1],[LAT1]	The longitude and latitude of the left-top corner of rectangle fence
[LNG2],[LAT2]	The longitude and latitude of the right-bottom corner of rectangle fence

> Note:
>
> 1. When [INDEX] is 0, it means a command to all fences.
> 2. When [LNG0] and [LAT0] are empty, we use last fixed position.

[FLAG]:

- N/A: Fence is disabled
- O: Out-type fence
- I: In-type fence
- C: Bidirectional fence
- R: Round fence
- S: Rectangle fence

Example	Description
FENCE,1,OR,,,500#	Setup fence 1 (out-type, round) round last fixed position
FENCE,1,IR,,,500#	Setup fence 1 (in-type, round) round last fixed position
FENCE,1,CR,,,500#	Setup fence 1 (bidirectional, round) round last fixed position
FENCE,1,OR,113.5,22.5,500#	Setup fence 1 (out-type, round) round specific position
FENCE,1,IR,113.5,22.5,500#	Setup fence 1 (in-type, round) round specific position

Example	Description
FENCE,1,CR,113.5,22.5,500#	Setup fence 1 (bidirectional, round) round specific position
FENCE,1,OS,113.2,22.2,113.8,22.8#	Setup fence 1 (out-type, rectangle)
FENCE,1,IS,113.2,22.2,113.8,22.8#	Setup fence 1 (in-type, rectangle)
FENCE,1,CS,113.2,22.2,113.8,22.8#	Setup fence 1 (bidirectional, rectangle)
FENCE,1#	Remove the first fence
FENCE,0#	Remove all fences
FENCE,1?	Return the first fence
FENCE,0?	Return all fences

6.3.18 SHIFT

This command requests to enable/disable a shift fence in device. Shift fence is an automatic fence. It becomes valid whenever ACC is OFF, and returns invalid when ACC is ON. When ACC is OFF and car moves out of it, a shift warning will be triggered.

In order to make it to work, ACC line must be connected correctly.

```
SHIFT,[RADIUS]#
> SET SHIFT OK
> SET SHIFT Error

SHIFT?
> SHIFT:[RADIUS]
```

Name	Description
[RADIUS]	The radius of shift fence (in meters)

[RADIUS]:

- 0: Shift fence is disabled
- Other: Shift fence is enabled

> Note:
>
> 1. Normally, the radius should be more than 50m, or a wrong warning may be triggered because of random drifting position.

6.3.19 SENSOR

This command requests to enable/disable sensor warnings and set their parameter. If a sensor warning is enabled and the actual value of the sensor is beyond the limits, a relevant warning will be triggered.

```
SENSOR,[NAME],[LOW],[HIGH],[DELAY]
> SET SENSOR OK
> SET SENSOR ERROR

SENSOR?
Temperature:[LOW],[HIGH],[DELAY],[VALUE]
Light:[LOW],[HIGH],[DELAY],[VALUE]
TS:[CHIP]
LS:[CHIP]
```

Name	Description
[NAME]	The sensor name to be set
[LOW]	The low limit of the sensor
[HIGH]	The high limit of the sensor
[DELAY]	The delay time before a warning is emitted (in seconds)
[VALUE]	The actual value of the sensor
[CHIP]	The chip model of the sensor

[NAME]:

- T: Internal Temperature
- H: Humidity
- L: Light
- C: CO2
- P: Probe Temperature

> Note:
>
> 1. When [LOW] is equal to [HIGH], the relevant warning is disabled.

6.4 QUERY COMMANDS

6.4.1 VERSION

This command requests to return the firmware version in device.

```
VERSION#
VERSION?
> IMEI:[IMEI]
  IMSI:[IMSI]
  ICCID:[ICCID]
  SYSTEM:[SYS VERSION]
  VERSION:[APP VERSION]
  BUILD:[BUILT TIME]
```

Name	Description
[IMEI]	The IMEI of device
[IMSI]	The IMSI of SIM
[ICCID]	The ICCID of SIM
[SYS VERSION]	The operating system version
[APP VERSION]	The application version
[BUILT TIME]	The built time of firmware

> Note:
>
> 1. The firmware in device contains two parts. One is the operating system, and another is application. The application is upgradable over the air.

Example	Result
VERSION#	IMEI:354188046036385

Example	Result
	IMSI:9460016668297433 ICCID:89860116851009444751 SYSTEM:M6130_V2.0.5 VERSION:MXAPP_V2.0.5 BUILD:Apr 26 2017 17:35:13

6.4.2 PARAM

This command requests to return major parameters in device.

```
PARAM#
PARAM?
> IMEI:[IMEI]
  APN:[APN]
  SERVER:[SERVER]
  COLLECT:[TIMERS]
  LANG:[LNAME]
  GMT:[GMT]
  SAVING:[SAVING]
```

Name	Description
[IMEI]	The IMEI of device
[APN]	The APN (see Section 6.3.6 APN)
[SERVER]	The URI of location server (see Section 6.3.7 SERVER)
[TIMERS]	The timers of location collection (see Section 6.3.8 COLLECT)
[LNAME]	The language name (see Section 6.3.2 LANG)
[GMT]	The time zone (see Section 6.3.3 GMT)
[SAVING]	The work mode of GPS (see Section 6.3.12 GPS)

Example	Result
PARAM#	IMEI:354188046036385

Example	Result
	APN:cmnet SERVER:” tcp://hzgps.sky200.com:32001” COLLECT:60,0,0,0,1 LANG:CN GMT:E12.00 SAVING:0

6.4.3 STATUS

This command requests to return current status in device.

```
STATUS#
STATUS?
> BATTERY:[BATTERY]
  GPRS:[GPRS]
  GSM:[GSM],[SIGNAL]
  GPS:[GPS],[SATELLITE]
  ACC:[ACC]
  RELAY:[RELAY]
  POWER:[POWER]
  MS:[MS]
```

Name	Description
[BATTERY]	The battery percentage — 0% ~ 100%
[GPRS]	The GPRS status — CLOSED, FAILED, SUCCESS
[GSM]	The GSM status — CLOSED, NONE, HIGH, MED, LOW
[SIGNAL]	The signal level — 0: -110dB 1: -109dB 2: -108dB 50: -60dB 70: -40dB
[GPS]	The GPS status — CLOSED, FIXED, UNFIXED
[SATELLITE]	The satellites number
[ACC]	The ACC status — ON, OFF
[RELAY]	The relay status — ON, OFF

Name	Description
[POWER]	The external power status — OK, NC
[MS]	The motion sensor model

> Note:
>
> 1. Not all data will be shown in result. For example, [ACC] will be lacked if device is not designed for vehicle.

Example	Result
STATUS#	BATTERY:100% GPRS:CLOSED GSM:MED,22 GPS:FIXED,6 ACC:ON RELAY:OFF MS:LIS3DH

6.4.4 STAT

This command requests to return current statistics in device.

```
STAT#
STAT?
> MILEAGE:[MILEAGE]
  BOOTUP:[BOOTUP COUNT]
  UPLOAD:[UPLOAD AMOUNT]
  DOWNLOAD:[DOWNLOAD AMOUNT]
  POWER:[POWER TIME]
  ACC:[ACC TIME]
  GPS:[GPS TIME]
```

Name	Description
[MILEAGE]	The mileage (in km)
[BOOTUP COUNT]	The boot-up count

Name	Description
[UPLOAD AMOUNT]	The total amount of upload data
[DOWNLOAD AMOUNT]	The total amount of download data
[POWER TIME]	The accumulative time when power is on
[ACC TIME]	The accumulative time when ACC is on
[GPS TIME]	The accumulative time when GPS is on

> Note:
>
> 1. The amount of upload and download data is calculated only with valid data. They do not include the excess amount used in communication. So they may be different with the amount from network provider. They are only for reference.

Example	Result
STAT#	MILEAGE:0.36(km) BOOTUP:13 UPLOAD:0KB DOWNLOAD:0KB POWER:0D01:25 ACC:0D02:28 GPS:0D01:25

6.4.5 POSITION / 123

This command requests to return the recent address of device.

```
POSITION#
POSITION?
123
> [ADDRESS]
> [URL]
```

Name	Description
[ADDRESS]	The address
[URL]	The google URL and other information

> Note:
>
> 1. After user requests the recent address, device will send current coordinates to server and then server will return address. If server is not accessible or server does not return it, device will return google URL to user.

Example	Result
POSITION#	1027 Flatbush Ave, Brooklyn, NY 11226, USA
POSITION#	http://maps.google.com/?q=22.555525,113.940147 <0.9km/h 0.0> <2017-05-02 22:13:03> IMEI:354188046036385
123	1027 Flatbush Ave, Brooklyn, NY 11226, USA
123	http://maps.google.com/?q=22.555525,113.940147 <0.5km/h 0.0> <2017-05-02 22:13:10> IMEI:354188046036385

6.4.6 WHERE

This command requests to return the recent coordinate of device.

```
WHERE#
WHERE?
> Lat:[LATITUDE]
  Lon:[LONGITUDE]
  Course:[COURSE]
  Speed:[SPEED]
  DateTime:[DATETIME]
```

Name	Description
[LATITUDE]	The latitude (in degrees)
[LONGITUDE]	The longitude (in degrees)
[COURSE]	The moving course
[SPEED]	The moving speed (in km/h)
[DATETIME]	The date and time

Example	Result
WHERE#	Lat:N22.55552 Lon:E113.94014 Course:0.0 Speed:0.2km/h DateTime:2017-05-02 22:19:14

6.4.7 URL

This command requests to return the google URL of device.

```
URL#
URL?
> [URL]
```

Name	Description
[URL]	The google URL and other information

Example	Result
URL#	http://maps.google.com/?q=22.555525,113.940147 <0.0km/h 0.0> <2017-05-02 22:20:34> IMEI:354188046036385

6.5 PORT COMMANDS

6.5.1 GPIO

This command requests to setup external GPIOs.

```
GPIO,[TIN],[TOUT],[PULL]#
> SET GPIO OK
> SET GPIO Error

GPIO?
> GPIO:[TIN],[TOUT],[PULL]
```

Name	Description
[TIN]	Indicates which GPIOs are input port (see note for format)
[TOUT]	Indicates which GPIOs are output port (see note for format)
[PULL]	Indicates which GPIOs are pulled up (see note for format)

> Note:
>
> 1. The format of each parameter is ‘xxxx’. Each ‘x’ represents from GPIO0 to GPIO3, and its value is 1 or 0. Normally, 1 is valid and 0 is invalid. For example, 1100 for [TIN] means that GPIO0 and GPIO1 are input type and other GPIOs are not. 0100 for [PULL] means that GPIO1 is pulled up and others are pulled down.
> 2. If a GPIO is not defined as either input type or output type, it will be kept as the default type defined by manufacturer.
> 3. [PULL] is valid only for GPIOs which are input type.
> 4. When a GPIO is configured as output type, its level is defined by command **PORT,[DOUT]#** which is described in Section 6.5.2 PORT
> 5. Not all products have 4 GPIOs. Please refer to its product description.

Example	Description
GPIO,1000,0100,1000#	GPIO0 is input type and pulled up. GPIO1 is output type. Other GPIOs are undefined.

Example	Description
GPIO,1111,0000,1010#	All 4 GPIOs are input type. GPIO0 and GPIO2 are pulled up. GPIO1 and GPIO3 are pulled down.
GPIO,0000,1111,0000#	All 4 GPIOs are output type.

6.5.2 PORT

This command requests to write external output GPIOs or read external input GPIOs and ADCs.

```
PORT,[DOUT],[MASK]#
> Port OK
> Port Error

PORT?
> DIN:[DIN]
  DOUT:[DOUT]
  AIN1:[AIN1]
  AIN2:[AIN2]
```

Name	Description
[DIN]	Indicates the level of all input GPIOs (see note 1)
[DOUT]	Indicates the level of all output GPIOs (see note 1)
[AIN1]	Indicates the level of ADC1 (in mV)
[AIN2]	Indicates the level of ADC2 (in mV)
[MASK]	Indicates which output GPIOs should be configured (see note 2)

> Note:
>
> 1. The format of [DIN] and [DOUT] is ‘xxxx’. Each ‘x’ represents from GPIO0 to GPIO3, and its value is 1 or 0. Normally, 1 is high level and 0 is low level.
> 2. The format of [MASK] is ‘xxxx’. Each ‘x’ represents from GPIO0 to GPIO3, and its value is 1 or 0. Normally, 1 means the relevant output

> GPIOs should be configured.
> 3. Not all products have 4 GPIOs and 2 ADCs. Please refer to its product description.

Example	Description
PORT,1000,1000#	Only GPIO0 is configured to high level.
PORT,0000,0100#	Only GPIO1 is configured to low level.
PORT,1000,1100#	GPIO0 is configured to high level. GPIO1 is configured to low level. Other GPIOs are not changed.
PORT?	DIN:0100 DOUT:1000 AIN1:1200 AIN2:0

6.6 PEDOMETER COMMANDS

6.6.1 BODY

This command requests to set the height and weight of user, which will be used in the calculation of pedometer.

```
BODY,[HEIGHT],[WEIGHT]#
> SET BODY OK
> SET BODY Error

BODY?
> HEIGHT:[HEIGHT](cm)
> WEIGHT:[WEIGHT](kg)
```

Name	Description
[HEIGHT]	The height of user (in cm)
[WEIGHT]	The wight of user (in kg)

Example	Description
BODY,160,70#	SET BODY OK
BODY?	HEIGHT:160(cm) WEIGHT:70(kg)

6.6.2 PDM

This command requests to clear or return all statistics in pedometer.

```
PDM#
> PDM OK
> PDM Error

PDM?
> TOTAL:
    STEPS:[A_STEPS]
    TIME:[A_TIME]
    DISTANCE:[A_DISTANCE](km)
    ENERGY:[A_ENERGY](kCal)
  TODAY:
    STEPS:[T_STEPS]
    TIME:[T_TIME]
    DISTANCE:[T_DISTANCE](km)
    ENERGY:[T_ENERGY](kCal)
```

> Note:
>
> 1. **PDM#** is used to clear all results of pedometer.

Name	Description
[A_STEPS]	The accumulated steps in total
[A_TIME]	The accumulated walking time in total
[A_DISTANCE]	The accumulated walking distance in total (in km)
[A_ENERGY]	The accumulated burnt energy in total (in kcal)

Name	Description
[T_STEPS]	The accumulated steps today
[T_TIME]	The accumulated walking time today
[T_DISTANCE]	The accumulated walking distance today (in km)
[T_ENERGY]	The accumulated burnt energy today (in kcal)

Example	Description
PDM#	All statistics are cleared.
PDM?	TOTAL: STEPS:126580 TIME:25:20 DISTANCE:34.73(km) ENERGY:2387.17(kCal) TODAY: STEPS:3618 TIME:0:49 DISTANCE:3.28(km) ENERGY:261.50(kCal)

6.7 OBD COMMANDS

6.7.1 MONITOR

This command requests to setup which PIDs should be monitored by OBD module.

```
MONITOR,[PIDS]#
> SET MONITOR OK
> SET MONITOR Error

MONITOR?
> MONITOR:[PIDS]
```

Name	Description
[PIDS]	Indicates which PIDS should be monitored

> Note:
>
> 1. [PIDS] is a group of hexadecimal number. Each hexadecimal number is a PID. For example: 0C0D898A8B means PID0C, PID0D, PID89, PID8A and PID8B. All standard PIDs are described in Wikipedia OBD-II PIDs, and all extended PIDs are described in Appendix A.4 Extended OBD-II PIDs.
> 2. Up to 50 PIDs can be monitored.
> 3. Some PIDs are monitored by system. They will be added into list automatically.

Example	Description
MONITOR,0C0D898A8B#	PID0C, PID0D, PID89, PID8A and PID8B should be monitored by OBD.

6.7.2 OBD

This command requests to communicate with OBD module. It contains a group of secondary commands. We will discuss general information in this section, and enumerate all secondary commands in following sections.

```
OBD,[CODE],[ARG]#
> OBD,[CODE],[RST],[DATA]
```

Name	Description
[CODE]	The command code passed to OBD module
[ARG]	The command arguments passed to OBD module
[RST]	The response result from OBD module
[DATA]	The response data from OBD module

[CODE]:

- 01 — Get the firmware version of OBD module

- 02 — Get the serial number of OBD module
- 03 — Get the VIN (Vehicle Identification Number)
- 08 — Set the model of vehicle
- 10 — Get supported PIDs in vehicle
- 11 — Read the value of specific PIDs from vehicle
- 12 — Get supported PIDs in the freeze frame of vehicle
- 13 — Read the value of specific PIDs from the freeze frame of vehicle
- 14 — Get the fault code
- 15 — Clear the fault code
- 18 — Get the vehicle body status
- 19 — Control some vehicle body parts

[ARG]:

- The command arguments is related with command code. They are described in following sections.

[RST]:

- 00 — Success
- 01 — Timeout
- 02 — Invalid command code
- 03 — Invalid command arguments
- 04 — Communication error with vehicle

[DATA]:

- The response data is related with command code. They are described in following sections.
- The response data is valid only if the response result is 00 which means that command is executed successfully.

6.7.3 OBD,01

This command requests to get the firmware version of OBD module.

```
OBD,01#
> OBD,01,00,5100003C0000
```

[ARG]: None
[DATA]: The firmware version — 0x5100003C0000.

6.7.4 OBD,02

This command requests to get the serial number of OBD module.

```
OBD,02#
> OBD,02,00,066CFF575454834987114547
```

[ARG]: None
[DATA]: The serial number — 0x066CFF575454834987114547.

6.7.5 OBD,03

This command requests to get the VIN.

```
OBD,03#
> OBD,03,00,313233343536373839414243444546474 8
```

[ARG]: None
[DATA]: The VIN — 123456789ABCDEFGH.

6.7.6 OBD,08

This command requests to set the model of vehicle. It is necessary to enable nonstandard OBD features, e.g. body status, fual consumption, etc.

```
OBD,08,0000#
> OBD,08,00,0000
```

[ARG]: The model of vehicle
[DATA]: The model of vehicle

The draft model list:

Code	Manufacturer / Model
0100	Volkswagen
0200	GM

Code	Manufacturer / Model
0300	Ford
0400	Toyota
0500	Honda
0600	Nissan
0700	Kia
0800	Hyundai
0900	BMW
0A00	Mercedes-Benz
0B00	Subaru
0C00	Mitsubishi
0D00	Renault
0E00	Peugeot
0F00	Land Rover
1000	Volvo
1100	Ssangyong

> Note:
>
> 1. More models have been supported in latest product. Please contact us to get full model list.

6.7.7 OBD,10

This command requests to get the PIDs supported in vehicle.

```
OBD,10#
> OBD,10,00,00FFFFFFFF20FFFFE000
```

[ARG]: None
[DATA]: The standard data that describes which PIDs are supported.

The standard data contains many groups. Each group has 5 bytes. The first byte in a group is the PID code, and following 4 bytes are its value. For example, 00FFFFFFFF represents that the value of PID00 is 0xFFFFFFFF, 0233445566 represents the value of PID02 is 0x33445566, etc. All standard PIDs are described in Wikipedia OBD-II PIDs, and all extended PIDs are described in Appendix A.4 Extended OBD-II PIDs.

More detail about the data format can be also found in Wikipedia OBD-II PIDs.

In above example:
PID00: FFFFFFFF
PID20: FFFFE000

After being parsed according to Wikipedia, we know PID00 ~ PID33 are supported in vehicle.

6.7.8 OBD,11

This command requests to read the value of specific PIDs from vehicle.

```
OBD,11,0020408A8B0C0D0E#
>
OBD,11,00,00FFFFFFFF20FFFFFFFF40FFFFFFFF8A00000000008B000000000C4E2000000D660000000E33
445566
```

[ARG]: The list of PIDs to be read. In above example, the value of PID00, PID20, PID40, PID8A, PID8B, PID0C, PID0D and PID0E will be read.
[DATA]: The standard data that are read from OBD module.

The standard data contains many groups. Each group has 5 bytes. The first byte in a group is the PID code, and following 4 bytes are its value. For example, 00FFFFFFFF represents that the value of PID00 is 0xFFFFFFFF, 0233445566 represents the value of PID02 is 0x33445566, etc. All standard PIDs are described in Wikipedia OBD-II PIDs, and all extended PIDs are described in Appendix A.4 Extended OBD-II PIDs.

More detail about the data format can be also found in Wikipedia OBD-II PIDs.

In above example:
PID00: FFFFFFFF
PID20: FFFFFFFF
PID40: FFFFFFFF

PID8A: 00000000
PID8B: 00000000
PID0C: 4E200000 — Engine RPM = (A * 256 + B) / 4 = ((0x4E * 256) + 0x20) / 4 = 5000rpm
PID0D: 66000000 — Vehicle speed = A = 0x66 = 102km/h
PID0E: 33445566

All values can be parsed according to Wikipedia and Appendix.

6.7.9 OBD,12

This command requests to get the PIDs supported in the freeze frame of vehicle.

```
OBD,12#
> OBD,12,00,00445566772044556677404455667 6
```

[ARG]: None
[DATA]: The standard data that describes which PIDs are supported.

The standard data contains many groups. Each group has 5 bytes. The first byte in a group is the PID code, and following 4 bytes are its value. For example, 00FFFFFFFF represents that the value of PID00 is 0xFFFFFFFF, 0233445566 represents the value of PID02 is 0x33445566, etc. All standard PIDs are described in Wikipedia OBD-II PIDs, and all extended PIDs are described in Appendix A.4 Extended OBD-II PIDs.

More detail about the data format can be also found in Wikipedia OBD-II PIDs.

In above example:
PID00: 44556677
PID20: 44556677
PID40: 44556676

All values can be parsed according to Wikipedia and Appendix.

6.7.10 OBD,13

This command requests to read the value of specific PIDs from the freeze frame of vehicle.

```
OBD,13,0C0E1C1E#
> OBD,13,00,0C445566770E445566771C445566771E44556677
```

[ARG]: The list of PIDs to be read. In above example, the value of PID0C, PID0E, PID1C and PID1E will be read.
[DATA]: The standard data that are read from OBD module.

The standard data contains many groups. Each group has 5 bytes. The first byte in a group is the PID code, and following 4 bytes are its value. For example, 00FFFFFFFF represents that the value of PID00 is 0xFFFFFFFF, 0233445566 represents the value of PID02 is 0x33445566, etc. All standard PIDs are described in Wikipedia OBD-II PIDs, and all extended PIDs are described in Appendix A.4 Extended OBD-II PIDs.

More detail about the data format can be also found in Wikipedia OBD-II PIDs.

In above example:
PID0C: 44556677
PID0E: 44556677
PID1C: 44556677
PID1E: 44556677

All values can be parsed according to Wikipedia and Appendix.

6.7.11 OBD,14

This command requests to get the fault code.

```
OBD,14#
> OBD,14,00,00
> OBD,14,00,00020502009302
```

[ARG]: None
[DATA]: The fault code (see Section 5.10 OBD FAULT PACKAGE — 0x19)

6.7.12 OBD,15

This command requests to clear the fault code.

```
OBD,15#
> OBD,15,00,01
> OBD,15,00,00
```

[ARG]: None
[DATA]: The result of action — 01: Success 00: Failed

6.7.13 OBD,18

This command requests to get the vehicle body status.

```
OBD,18#
> OBD,18,00,1F001000
```

[ARG]: None
[DATA]: The vehicle body status (see Section 5.9 OBD BODY PACKAGE — 0x18)

6.7.14 OBD,19

This command requests to control some vehicle body status.

```
OBD,19,0100#
> OBD,19,00,01
> OBD,19,00,00
```

[ARG]: The action

- BYTE1:
  - 0x01: Central locking
  - 0x02: Horn
  - 0x03: Emergency light
  - 0x04: Windows
- BYTE2:
  - 0x00: Turn off
  - 0x01: Turn on

[DATA]: The result of action — 01: Success 00: Failed

APPENDIX

A.1 CHECKSUM ALGORITHM

Please contact us to get source code.

sum16.h:

```
#ifndef __SUM16_H__
#define __SUM16_H__

#if defined(__cplusplus)
extern "C" {
#endif

typedef unsigned char           mx_uint8;
typedef unsigned short          mx_uint16;
typedef unsigned int            mx_uint32;

mx_uint16 mx_sum16(mx_uint16 sum16, const mx_uint8 * data, mx_uint32 size);

#if defined(__cplusplus)
}
#endif

#endif // __SUM16_H__
```

sum16.c:

```
#include "sum16.h"

mx_uint16 mx_sum16(mx_uint16 sum16, const mx_uint8 * data, mx_uint32 size)
{
    while (size != 0)
    {
```

```
        sum16 = ((sum16 << 1) | (sum16 >> 15)) + *data;
        data++;
        size--;
    }

    return(sum16);
}
```

A.2 DEFLATE ALGORITHM

Please contact us to get source code.

deflate.h:

```
#ifndef __DEFLATE_H__
#define __DEFLATE_H__

#if defined(__cplusplus)
extern "C" {
#endif

typedef unsigned char           mx_uint8;
typedef unsigned short          mx_uint16;
typedef unsigned int            mx_uint32;

typedef struct
{
    /// Default is 130 (from 4 to 258). The larger value, the more memory
used in compression.
    unsigned short  maxLength;
    /// Default is 65535 (from 4 to 65535). The larger value, the slower
speed in compression.
    unsigned short  maxOffset;
    /// Memory alloc function
    void *          (*malloc)(unsigned int);
    /// Memory free function
```

```
    void            (*free)(void *);
} mx_ezconfig;

mx_uint32 mx_extractEz(mx_uint8 *dst, const mx_uint8 *src, mx_uint32 size,
const mx_ezconfig *cfg);

#if defined(__cplusplus)
}
#endif

#endif // __DEFLATE_H__
```

deflate.c:

```
#include <mem.h>
#include <stdlib.h>
#include "deflate.h"

#define EZ_END_SYMBOL           256
#define EZ_MIN_LENGTH           4

typedef struct
{
    int             symbol;
    unsigned int    count;
    unsigned short  child0;
    unsigned short  child1;
} ez_node;

typedef struct
{
    // I/O stream
    unsigned char   *ioPtr;
    unsigned int    ioPos;

    // Work buffer
    ez_node         *pNodes;
```

```
    unsigned int    nNodes;
} ez_context;

typedef mx_ezconfig ez_config;

static const ez_config ez_cfg = { 130, 65535, malloc, free };

static const unsigned char ez_mask[] = { 0x00, 0x01, 0x03, 0x07, 0x0F,
0x1F, 0x3F, 0x7F, 0xFF };

static unsigned int ez_readBits(ez_context *ctx, unsigned int bits)
{
    unsigned char *ptr;
    unsigned int  left;
    unsigned int  result;

    ptr  = ctx->ioPtr;
    left = ctx->ioPos + 1;

    result = 0;
    while (bits >= left)
    {
        result = (result << left) | (*ptr & ez_mask[left]);
        bits -= left;

        left = 8;
        ptr++;
    }

    if (bits > 0)
    {
        result = (result << bits) | ((*ptr >> (left - bits)) &
ez_mask[bits]);
        left -= bits;
    }

    ctx->ioPtr = ptr;
```

```
    ctx->ioPos = left - 1;
    return(result);
}

static unsigned int ez_readBit(ez_context *ctx)
{
    unsigned char *ptr;
    unsigned int  pos;
    unsigned int  result;

    ptr = ctx->ioPtr;
    pos = ctx->ioPos;

    result = (*ptr >> pos) & 0x01;
    if (pos != 0)
        pos--;
    else
        pos = 7, ptr++;

    ctx->ioPtr = ptr;
    ctx->ioPos = pos;
    return(result);
}

static ez_node * ez_readTree(ez_context *ctx)
{
    ez_node *node;

    node = &ctx->pNodes[ctx->nNodes++];

    if (ez_readBit(ctx))
    {
        // Read symbol for leaf node
        node->symbol = ez_readBits(ctx, 9);
        node->child0 = 0;
        node->child1 = 0;
    }
```

```
    else
    {
        // Read branch
        node->symbol = -1;
        node->child0 = ez_readTree(ctx) - ctx->pNodes;
        node->child1 = ez_readTree(ctx) - ctx->pNodes;
    }

    return(node);
}

static int ez_readSymbol(ez_context *ctx)
{
    ez_node *node;

    node = ctx->pNodes;
    while (node->symbol < 0)
    {
        if (ez_readBit(ctx))
            node = ctx->pNodes + node->child1;
        else
            node = ctx->pNodes + node->child0;
    }

    return(node->symbol);
}

static unsigned int ez_decode(ez_context *ctx, unsigned char *dst)
{
    int           symbol;
    unsigned int  length, offsetH, offsetL;
    unsigned char *ptr1, *ptr2;

    ptr1 = dst;
    while (1)
    {
        symbol = ez_readSymbol(ctx);
```

```
        if (symbol == EZ_END_SYMBOL)
            break;

        if (symbol < EZ_END_SYMBOL)
            *(ptr1++) = (unsigned char)(symbol);

        if (symbol > EZ_END_SYMBOL)
        {
            length  = symbol - (EZ_END_SYMBOL - EZ_MIN_LENGTH + 1);
            offsetH = ez_readSymbol(ctx);
            offsetL = ez_readSymbol(ctx);

            ptr2 = ptr1 - ((offsetH << 8) | offsetL);
            for (; length != 0; length--)
                *(ptr1++) = *(ptr2++);
        }
    }

    return(ptr1 - dst);
}

// The largest output buffer must contain all data
unsigned int ez_extract(unsigned char *dst, const unsigned char *src,
unsigned int size, const ez_config *cfg)
{
    unsigned int  result = 0;
    ez_context    ctx;

    if (cfg == NULL)
        cfg = &ez_cfg;

    memset(&ctx, 0, sizeof(ctx));

    ctx.ioPtr = (unsigned char *)src;
    ctx.ioPos = 7;

    ctx.pNodes = (*cfg->malloc)((2 * 512 - 1) * sizeof(ez_node));
```

```
    if (ctx.pNodes == NULL)
        goto END;

    memset(ctx.pNodes, 0, (2 * 512 - 1) * sizeof(ez_node));
    ez_readTree(&ctx);
    result = ez_decode(&ctx, dst);

END:
    if (ctx.pNodes != NULL)
        (*cfg->free)(ctx.pNodes);

    return(result);
}

mx_uint32 mx_extractEz(mx_uint8 *dst, const mx_uint8 *src, mx_uint32 size,
const mx_ezconfig *cfg)
{
    return(ez_extract(dst, src, size, cfg));
}
```

A.3 PARAM-SET

There are many parameters in device and all of them are stored in non-volatile memory. Server can read them by the above commands. However, the reading is slow because of the poor performance of IP connection over the cellular. Moreover, sometimes, there is no response because device is offline due to weak signal. As a result, user often has a bad experience to access the parameters he wishes to know.

In order to improve the user experience, server can keep a copy of all parameters. If so, server can show user the parameters immediately when user attempts to access them. Then, if user requests to change them, server saves the modifications and pushes them to device when it is online.

The set of all parameters is called *param-set*. When device is logged in, server can check the integrity of *param-set*, and requests device to upload it if it is different from the copy in server. In addition, when any change to *param-set* happens in device, it can also be synchronized up with server. All synchronization are optional and can be disabled if

server does not need the copy.

In order to reduce network traffic, *param-set* is compressed before being transmitted. Server must deflate it before use.

The structure of a *param-set* is described as below:

Name	Bytes	Description
PASSWORD	8	The password of device — String
FLAG	2	The flags of device (see note 1)
STYLE	2	The styles of device (see note 2)
HBT	2	The heartbeat timer (in seconds)
DELAY	2	The answer timer (in seconds)
COLLECT		**The following data are related to command "COLLECT?"**
INTERVAL	2	The time interval (in seconds)
DISTANCE	2	The running distance (in meters)
TURN	2	The turning angle (in degrees)
ACTIVE	2	The time interval when device is moving/active (in seconds)
QUANTITY	2	The number of cached location packages before they are sent
GSM		**The following data are related to command "GSM?"**
FLAG	2	The flags of GSM module (see note 3)
T0	2	The work time after GSM module is awaken (in seconds)
T1_TOTAL	2	The total time of phase 1 (in minutes)
T1_WAKING	2	The work time in phase 1 (in minutes)
T2_PERIODIC	2	The periodic time of phase 2 (in minutes)

Name	Bytes	Description
T2_WAKING	2	The work time in phase 2 (in minutes)
GPS		**The following data are related to command "GPS?"**
FLAG	2	The flags of GPS module (see note 4)
T0	2	The work time after GPS module is awaken (in seconds)
T1_TOTAL	2	The total time of phase 1 (in minutes)
T1_WAKING	2	The work time in phase 1 (in minutes)
T2_PERIODIC	2	The periodic time of phase 2 (in minutes)
T2_WAKING	2	The work time in phase 2 (in minutes)
XLIFE		**The following data are related to command "XLIFE?"**
T_INITIAL	2	Reserved
T_PERIODIC	2	Reserved
T_WAKING	2	Reserved
WARNING		**The following data are related to command "WARNING?"**
W_MASK	4	Reserved
W_MESSAGE	4	Reserved
W_SOCKET	4	Reserved
SPEED		**The following data are related to command "SPEED?"**
LOW	2	The low value of speed range (in 0.1km/h)
HIGH	2	The high value of speed range (in 0.1km/h)
SHIFT		**The following data are related to command "SHIFT?"**
RADIUS	2	The radius of shift fence (in meters)
FENCE,01		**The following data are related to command**

Name	Bytes	Description
		“FENCE,01?”
FLAG	2	The flags of fence (see note 5)
RANGE1	4	The longitude of the center of round fence (in 1/500”), or The longitude of the left-top corner of rectangle fence (in 1/500”)
RANGE2	4	The latitude of the center of round fence (in 1/500”), or The latitude of the left-top corner of rectangle fence (in 1/500”)
RANGE3	4	The radius of round fence (in meters), or The longitude of the right-bottom corner of rectangle fence (in 1/500”)
RANGE4	4	Unused for round fence, or The latitude of the right-bottom corner of rectangle fence (in 1/500”)
FENCE,02		Same as **FENCE,01**
FENCE,03		Same as **FENCE,01**
FENCE,04		Same as **FENCE,01**
FENCE,05		Same as **FENCE,01**
FENCE,06		Same as **FENCE,01**
FENCE,07		Same as **FENCE,01**
FENCE,08		Same as **FENCE,01**
MOTION		**The following data are related to command “MOTION?”**
SENSE	2	The sensitivity
DELAY	2	The delay time before a warning is emitted (in seconds)
POWER		**The following data are related to command**

Name	Bytes	Description
		"POWER?"
DOWN	2	Reserved
LACK	2	Reserved
ACC	2	Reserved
PORT		**The following data are related to command "PORT?"**
DOUT	2	Indicates the level of all output GPIOs
BODY		**The following data are related to command "BODY?"**
HEIGHT	2	The height of user (in cm)
WEIGHT	2	The wight of user (in kg)
MONITOR		**The following data are related to command "MONITOR?"**
MASK	32	Indicates which PIDS should be monitored (see note 6)
MANAGER,01		**The following data are related to command "MANAGER,01?"**
NUMBER	40	The phone number of manager — String
ALIAS	24	The alias of manager — String
MANAGER,02		Same as **MANAGER,01**
MANAGER,03		Same as **MANAGER,01**
MANAGER,04		Same as **MANAGER,01**
ALARM,01		**The following data are related to command "ALARM,01?"**
ACTION	1	The action when alarm clock is expired
WEEK	1	The week days of alarm clock
HOUR	1	The hour part of expire time of alarm clock

Name	Bytes	Description
MINUTE	1	The minute part of expire time of alarm clock
NOTE	60	The note of alarm clock
ALARM,02		Same as **ALARM,01**
ALARM,03		Same as **ALARM,01**
ALARM,04		Same as **ALARM,01**
ALARM,05		Same as **ALARM,01**
ALARM,06		Same as **ALARM,01**
ALARM,07		Same as **ALARM,01**
ALARM,08		Same as **ALARM,01**
SENSOR,T		**The following data are related to command "SENSOR,T?"**
LOW	4	Reserved
HIGH	4	Reserved
DELAY	2	Reserved
SENSOR,H		Same as **SENSOR,T**
SENSOR,L		Same as **SENSOR,T**
SENSOR,C		Same as **SENSOR,T**

> Note:
>
> 1. *Flag* of device:
>   BIT12: Fuel is cut (relay is triggered)
>   Other: Reserved
> 2. *Style* of device:
>   BIT00: LED always works (never sleep)
>   Other: Reserved
> 3. *Flag* of GSM module:
>   BIT00, BIT01: Work mode (see Section 6.3.11 GSM)
>   Other: Reserved
> 4. *Flag* of GPS module:

> BIT00, BIT01: Work mode (see Section 6.3.12 GPS)
> Other: Reserved
> 5. *Flag* of fence:
> BIT00: Fence mode: 0: Round 1: Rectangle
> BIT04: Outside type
> BIT05: Inside type
> BIT15: Active: 0: Inactive 2: Active
> 6. *Mask* of OBD:
> BYTE00 BIT00: Indicates whether PID00 is monitored
> ...
> BYTE00 BIT07: Indicates whether PID07 is monitored
> BYTE01 BIT00: Indicates whether PID08 is monitored
> ...
> BYTE32 BIT07: Indicates whether PIDFF is monitored

A.4 Extended OBD-II PIDs

Extended OBD-II PIDs are supported only for specific vehicle models. Please contact us to get more information.

```
X = (A << 24) + (B << 16) + (C << 8) + D
```

PID	Description	Units	Formula
88	Fuel consumption per hour	l/h	X
89	Fuel consumption per 100km	l/100km	X / 10 *Valid only if (speed ≥ 5km/h)*
8A	Odometer	km	X
8B	Remaining fuel (see note 1)	%	(X - 0x8000) / 10 *Valid only if (X ≥ 0x8000)*
8B	Remaining fuel (see note 2)	l	X / 10 *Valid only if (X < 0x8000)*

> Note:
>
> 1. The remaining fuel is returned in percentage for Buick, Chevrolet and Honda.
> 2. The remaining fuel is returned in liter for Volkswagen, Audi, Skoda,

Toyota, Ford and Nissan.

3. In order to get precise remaining fuel, vehicle must be horizontal and stationary. Or the result may be unstable and imprecise.