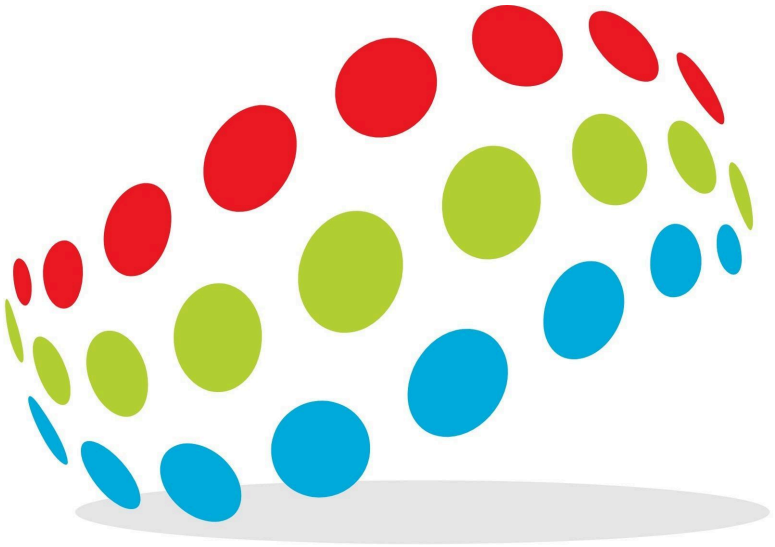
FJ1000 Integration Guide



Positioning Universal Inc.

Version 1.3

Document Revisions

Date	Version Number	Document Changes
2017.08.18	1.0	First Official Release
2017.10.17	1.1	Minor text improvements
2018.02.23	1.2	Added pulse command for outputs, io query for inputs and outputs
2018.03.16	1.3	Added debug section and commands

Table of Contents

1 Introduction

The FJ1000 is a vehicle tracking device that uses a GPS satellite receiver to determine location information and an LTE transceiver to communicate information to and from a land based server. In addition, the FJ1000 has several Inputs that may be configured to detect states of sensors and Outputs that can control accessories. Optional hardware features include a 3-axis accelerometer, Bluetooth module, Piezo buzzer, and backup battery with charging circuit.

The FJ1000 is powered by standard vehicle power (12V or 24V). It accepts several harnesses: a 4 wire power harness, a 4 wire UART harness, and a 14 pin IO harness for various different IO functions on some versions of the device.

Communications between the FJ1000 and the server are carried out using the UDP protocol with sequenced acknowledgment messages to confirm receipt of messages by the server. Configuration of the device can be conducted using serial communications over the UART port or UDP messages via the cellular modem.

2 Hardware

2.1 Physical and Electrical Specifications

Dimensions: 103mm * 53mm * 18.4mm

Weight: 63g (without battery)

Input Voltage: 6-36VDC

Power consumption:

- Active mode: 70mA @12VDC
- Low power mode: 15mA @ 12VDC
- Sleep mode: 6mA@ 12VDC
- Deep sleep: 2 mA@ 12VDC
- Deep Sleep on backup battery: 1.3mA @3.7VDC

Operating temperature: -30℃ to 70℃

Storage temperature: -40℃ to 85℃

2.2 Input/Output

Digital Inputs: 4

Input 0 is biased low and used as Ignition Sense

Inputs 1,2, and 3 are biased high and used as general purpose inputs

Relay driver Outputs: 4

Output 0 is typically used for a starter inhibit device

Outputs 1,2,and 3 are used as general purpose outputs

All outputs are open collector type outputs which provide a ground when set, and can handle 250mA of current to drive a relay.

Function LEDs: 2

GPS Status GREEN

Cellular Status AMBER

Other inputs:

ADC: One Analog to Digital Converter input

1-Wire Bus

Bluetooth

I2C serial bus

2.3 3.3V UART

A 4 wire connector provides Ground, RX, TX, and a reference 3.3V. The UART is typically used for debug, programming, and configuration. Standard debug operates at 115200 baud, 8N1.

2.4 GPS Specification

-163 dBm Tracking Sensitivity

Location Technology (56 channel GPS)

Location Accuracy (2 meter CEP50)

2.5 Cellular Communication

Verizon LTE Cat. 1

UDP/SMS

2.6 Harnesses

There are 3 harnesses that may be used with the FJ1000:

1. 4 wire power harness with one input and one output
2. 4 wire UART harness (includes optional Vdd)
3. 14 pin I/O harness

Power Harness 4 wires

1. Red V+ is connected to Positive , +12VDC or +24VDC
2. Black V- is connected to Negative or Ground
3. White Input 0 - optionally connected to ignition sense

4. Green Output 0 – optionally used for Starter Interrupt Device

UART Harness 4 wires

1. Black Ground
2. Red Vdd (3.3V)
3. Blue RX
4. Green TX

IO Harness 14 wires

This harness contains wires for 1 Wire Bus, Power (both 3.3V and 12V to power accessories), I2C, Inputs, ADC, Outputs, and Ground. The table below shows signals and the relevant connector pin number and the color of the wire in the IO harness.

SIGNAL	PIN	COLOR
1 WIRE BUS	1	BROWN/WHITE
VDD	2	RED/WHITE
I2C SCL	3	PINK GREEN
I2C SDA	4	POWDER BLUE
INPUT 1	5	BLUE
INPUT 2	6	ORANGE
INPUT 3	7	VIOLET
BOOT 0	8	BLACK/WHITE
ADC 0	9	PINK
OUTPUT 1	10	BROWN
OUTPUT 2	11	YELLOW
OUTPUT 3	12	GREY
VCC	13	RED
GROUND	14	BLACK

3 Programming and Configuration

FJ1000 can be programmed and configured either via the serial UART connection or via an Over the Air (OTA) process on the cellular data network.

3.1 Firmware, Application, and Settings Files

There are multiple levels of software and settings files contained on the device:

1. Modem software is flashed onto the cellular modem at the factory. This firmware controls communications between the modem and the cellular carrier. In the very rare event that this firmware needs to be upgraded, it will be performed by Positioning Universal.
2. Device operating system firmware manages the control of modules on the device and provides the operating environment for Applications. In the very rare event that this operating system firmware needs to be upgraded, it will be performed by Positioning Universal.
3. Applications are written in JavaScript and can be loaded to the device via the serial UART or OTA. Applications can be updated to add functionality or improve performance when opportunities for improvement are found.
4. Settings files are loaded to the device via the serial UART or OTA. Settings files contain the parameters used by the application to control tracking and other behaviors.

3.2 DMAN Server

A Device Manager Server on the Positioning Universal infrastructure automatically updates Applications and Settings files for groups of devices. When an update of the application or new settings are released, they are loaded into the DMAN server and assigned to the Groups of devices that are to receive the update. The DMAN server automatically updates devices to the latest assigned versions. In normal operation, devices “check in” to the DMAN server regularly to report their health, and to check whether they are due to get updates.

Device will take the updates either immediately if commanded via the “update” command, or during the next quiescent period, before entering Sleeping state.

3.3 UART Programming, Trace, and Commands

In the factory and on a test bench, the serial UART connection is used to flash an image containing the root firmware, application, and default settings files using special factory software. Specific queries, commands, and debug tracing messages are also available

over the UART. In normal operation, the UART is only used for debug, command, and tracing purposes. Custom applications may use this UART to communicate with accessories.

The serial UART consists of RX, TX, and Ground signals on the 4 pin UART connector. Communications uses a TTL level of 3.3V. A supply of 3.3V is available on this connector to power a low power accessory (like a logger). NOTE: supplying 3.3V to the FJ1000 via this interface from a USB-TTL serial adapter will power the processor, but will not provide enough power for the cellular modem to work.

To communicate between a computer and the device using the UART, it is necessary to have a serial terminal program installed on the computer. Several examples are Hyperterm and Realterm on Windows, and CoolTerm on either Windows or Mac computers. Communications is at 115200 baud, 8N1.

The most common way to physically connect a computer to the device is to attach a USB-TTL serial adaptor to the 4 pin serial harness. Note that the optional Vdd wire (red) is NOT usually connected for basic communications. Examples of suitable USB-TTL serial adapters include those that incorporate the MAX3232 integrated circuit, or the CH340 chipset. Note that you may need to download and install drivers for these devices on a computer in order to use them. There are many different versions of these adapters available. Typically, the device that incorporate a crystal perform better. One version is available on Amazon from this link: Serial Converter adapter

Once connected, trace information will be sent as ASCII data over the UART interface. This allows the user to monitor operation of the device and see how it responds to various occurrences, as well as see a report of GPS status every 10 seconds. Queries and Commands shown in the next chapter can be sent to the device from the computer and responses seen in the terminal window.

3.4 SMS Queries and Commands

The queries and commands shown in the following chapter of this guide can be sent by SMS to the phone number of the device, and responses will be sent to the phone number of the device used to send the query or command.

3.5 UDP Queries and Commands

The queries and commands shown in the following chapter of this guide can be sent by UDP to the IP and port of the device. Due to cellular carrier security features, these queries and commands are usually queued on the sending server, and sent immediately after the server receives a message from the device. In order to indicate that the UDP message is a command, it is necessary to prepend the equals sign = to the front of the command. For example, to send the **agps** command to a device via UDP, it is necessary

to send it as **=agps** so that the device sees the = first and recognizes the following data as a command.

4 Queries and Commands

Queries and commands can be sent to a device via the serial UART, an SMS message, or via a UDP message over cellular. All queries and most commands will generate a response if successful that begins with the word PASS. If a command is not recognized by the device, it will respond with a message beginning with the word FAIL.

4.1 Queries

Queries request status information from the device, and the device responds accordingly.

Version

The device responds with the version numbers of the application and settings file and other information. An example response to the **version** query is:

```
PASS::version;app=3023 sett=3023 firm=3124 cell=4.3.1.2 [29492]
imei=354196070019771 imsi=311480142180885
iccid=89148000001404515503 msidn=15889776934
```

Results in device reporting its:

- current application version
- settings version
- Firmware version
- Cellular modem firmware version
- IMEI for device
- IMSI for device
- ICCID
- MSISDN (phone number)

Getgps

The device responds with current GPS status and location information. An example response to the **getgps** query is:

```
PASS::getgps;17-8-17 21:30:47 49.26447 -123.125595 s11/0.9 0.0g
0.0k 309.8* 41.1V 4.2B 11101110 -115dbm
good=0.1,49.26447,-123.125595 reset=0
```

Where the message can be broken into:

- Fix time
- Latitude
- Longitude
- # satellites, sX where X is number
- Horizontal Accuracy, /X where X is number
- Absolute G force, Xg where X is number
- Speed in kph, Xk
- Heading in degrees, X*
- VIN in Volts, XV
- Battery Voltage in Volts, XB
- Outputs/Inputs 0 = off, 1 = on, Output 3,2,1,0 then Input 3,2,1,0
- Rssi, Xdbm
- Time in seconds since Last good GPS fix, good=X
- Last good GPS lat, ,X
- Last good GPS lng, ,X
- Number of GPS resets since boot, reset=X

Getcomm

The device responds with current status of the cellular connection and details of the connection. An example response to the **getcomm** query is:

```
 PASS::getcomm;avail=true change=1136.6 wake=51.1 rssi=-51
lastAT=1.9 lastSent=17.0 lastRcvd=14.6 woAck=65536.4
cell=13911830 lac=13826 reset=0
```

Which consists of

- com available true or false
- time since last comm state change in seconds
- time from last comm module wake in seconds
- current rssi value
- time in seconds since last AT command was received
- time in seconds since last UDP message was sent
- time in seconds since last UDP message was received
- age in seconds of device when last comm was sent without an ack
- cell tower id
- local area code

- number of reset of comm module since device boot.

Clock

The device responds with the time as reported by the cellular modem, GPS, and CPU. An example response to the **clock** query is:

```
PASS::clock;age=66161.10435009765 mcu=17-8-15 22:54:52
gps=17-8-15 22:54:32 cell=17-8-15 22:54:27
```

Which gives:

- the age in seconds device since boot,
- the current mcu time
- the last reported good gps time
- the last good cell time
- Note that GPS and Cellular Timestamps are not updated while in low power or deep sleep mode.

State

The device responds with the states the device is currently in. An example response to the **state** query is:

```
PASS::state;Moving heart=1762.9 ignOn=none vibr=131.3
ColdBoot=135.2 lastVolt=42.1 pwrOn=true mem=1408 sid=false
```

Which consists of:

- current state (Moving, Stopped, Sleeping)
- the time in seconds to next heartbeat
- whether ignition is on
- time since last vibration event in seconds
- the time since last Cold Boot in seconds
- the last reported vehicle voltage (Vin)
- whether power is connected to the device
- SID (starter interrupt device) is active or not

where

Returns a google map url using the last good lat lng reported by device. Used through SMS to quickly map the the device's current location.

Example:
`PASS::where;http://maps.google.com/?49.26445910000,-123.12561510000`

settings,json

The **settings,json** command will respond with all values of the settings file in json format.

4.2 Commands

Commands force the device to take specified actions. They can be sent via SMS, a serial UART, or via the cellular data connection.

report,<event code>
The **report** command forces the device to send a regular event message with the event code specified in the command.
Eg. **report,9** would force a report with event code 9.

reboot
The **reboot** command forces a warm boot of the main processor and sub modules of the device. The reboot will occur in 5 seconds to give the device a chance to send a result in command ack based upon sms or udp delivery. If comm fails then no ack may be received even though the device does actually reboot. The device reports a WARM_BOOT message when it powers back up.

agps
The **agps** command initiates an assisted GPS update based upon current cell location.

clearmsg
The **clearmsg** command clears all queued messages from the message send buffer.

sms,<to phone number>,<payload>
Send an SMS message with specified payload to specified phone number.

comm,reset
Causes comm module to be reset.

update

Checks latest versions of application and settings for a device on DMAN and if different than existing application and/or setting then start update.

bootin
Sends a boot checkin message to DMAN, which updates status and statistics for the device.

4.3 Inputs and Outputs

A series of commands are used to query input status and to create output activity.

enableinputs,<on/off>
The **enableinputs,on** command enables messages to be created and sent when the state of the inputs changes.
The **enableinputs,off** command disables reporting messages for input state changes. The io status byte will still be updated for debug and event reports.
The value of this setting survives a power cycle.

Input,<input>
The **input** read query responds with the state of the specified digital input, where 0 represents a low value (0 volts) and 1 indicates a high value (> 3.3V). An example response to the query **input,2** is:

```
input,2;1
```

Which indicates that input 2 is high. (Since input 2 is biased high, it may actually be disconnected or floating.)

Inputs 1-3 are biased high, and can be triggered by pulling the voltage to ground. Input transitions are reported by messages sent to the server, and the current state of inputs and outputs are reflected by the individual bits of a byte in all location messages. Input reports are filtered using debounce parameters that can be individually set for each input and for each transition direction (High or Low). These parameters are in the settings file managed on DMAN.

Sid,<on/off>
The **sid,on** command activates the Starter Interrupt Device (SID) feature, which sets output 0, provided cellular coverage is available. The SID feature will clear output 0 when cellular service is not available, to avoid standing vehicles out of coverage.
The **sid,off** command cancels the SID feature and clears output 0.
The value of the SID setting survives a power cycle.

Output,<output>,<state(0/1)>

The **output** write command sets or clears the specified output. Outputs act low, so the state value 0 sets the output and state value 1 clears the output. For example, **output,2,0** will set output 2 and **output,2,1** will clear output 2.

buzz,<on time>,<off time>,<frequency>,<count>
The **buzz** command creates a series of pulsed beeps on the Piezo buzzer. The command specifies in ms the duration of the buzz pulse (on time), the separation of the pulses (off time), the frequency of signal in Hz (2800 Hz is a typical value), and the number (count) of beeps to be generated. For example, **buzz,500,1000,2800,3** would create 3 beeps lasting 0.5 seconds, with a 1 second quiet time between them, at a frequency of 2800 Hz. The piezo buzzer does not have a wide frequency response, so low and high frequencies do not generate a very loud beep.

pulse,<output>, <count>,<on time>,<off time>
The **pulse** command creates a series of pulses on the specified output. On means the open collector output is pulled to ground (set) permitting the flow of current. Off means the output is cleared and current does not flow. For example, **pulse,2,3,1000,500** would create 3 pulses on output 2 that last 1 second each with a half second space between them.

io
The **io** command reports the current status of outputs 0-3 and inputs 0-3. The value of the pins is reported even for models that lack the IO control and biasing circuits.

mems,<frequency>,<scale>,<threshold>,<duration>
The **mems** command provides a simple way to change the settings related to vibration wake on the device, for testing and calibration purposes. Along with the **buzzvibe** feature, it is intended not for production use, but for testing and selecting the appropriate settings for a targeted applications. The values of the options match those seen on DMAN, in the Settings file for setting up the MEMs device. For example, **mems,5,0,4,20** would set up the following requirement for alerting the device:
frequency index = 5 (200Hz)
scale = 0 (2G)
threshold = 4/127*scale = 0.063G
duration = 20/200hz = 0.100s

4.4 AT Commands to Modem Module

AT commands can be sent directly to the cellular modem module on the device using the command **sendAT,<ATcmd>**

For example, **sendAT,CEREG?** will send the command "AT+CEREG?" to the cellular modem.

NOTE: Sending AT commands to the cellular modem can change critical settings on the cellular modem and may interfere with its operation and block communications. This command is only to be used by developers familiar with the use of AT commands on cellular modems.

4.5 Trace and other Debug Tools

Console trace can be controlled using the trace command. **trace,off** disables the trace messages. You can set the period of GPS trace messages in seconds; for example, **trace,5** will produce a GPS status trace every 5 seconds.

The information provided by the console trace can be made more verbose. The extended verbosity level is enabled using the command **verbose,on** and disabled using **verbose,off**.

When debugging a remote device, it may be helpful to generate extra reports to the server on certain events. This can be enabled using **trace_reports,on** and disabled using **trace_reports,off**.

To assist in selecting appropriate values for the vibration sensor, a function is available to beep the buzzer each time a vibration interrupt is generated by the accelerometer. The function is enabled using **buzz_vibe,on** and disabled using **buzz_vibe,off**.
When debugging messaging on a test device, it may be helpful to see from the LEDs when a message is generated or sent. Using **message_led,on** will result in a green LED flutter when a message is generated on a device, and a red LED flutter when a message is sent by the modem. **message_led,off** will disable the feature.

These settings are volatile and will not survive a cold boot. Also, loading a Settings file via DMAN will return these features to their default values. The extra features default to OFF, and trace is enabled with a default of 10 seconds.

4.6 Use of a Passcode for Commands

If a Passcode is specified for the device settings, this passcode must be included as the first argument of the command. For example, if the passcode “security” is specified, then some examples of commands would include:
sid,security,on
getcomm,security
buzz,security,500,1000,2800,3

5 LED Indicators

There are 2 indicator LEDs on the device, which are used to indicate the status of cellular communications and GPS, and to indicate when messages are created and transmitted.

The **orange LED** indicates communications status:

- Blinking 1Hz indicates no PDP session is active – searching for cell signal
- Solid indicates an active PDP session
- A quick flash every 5 seconds indicates that the cellular modem is in low power mode

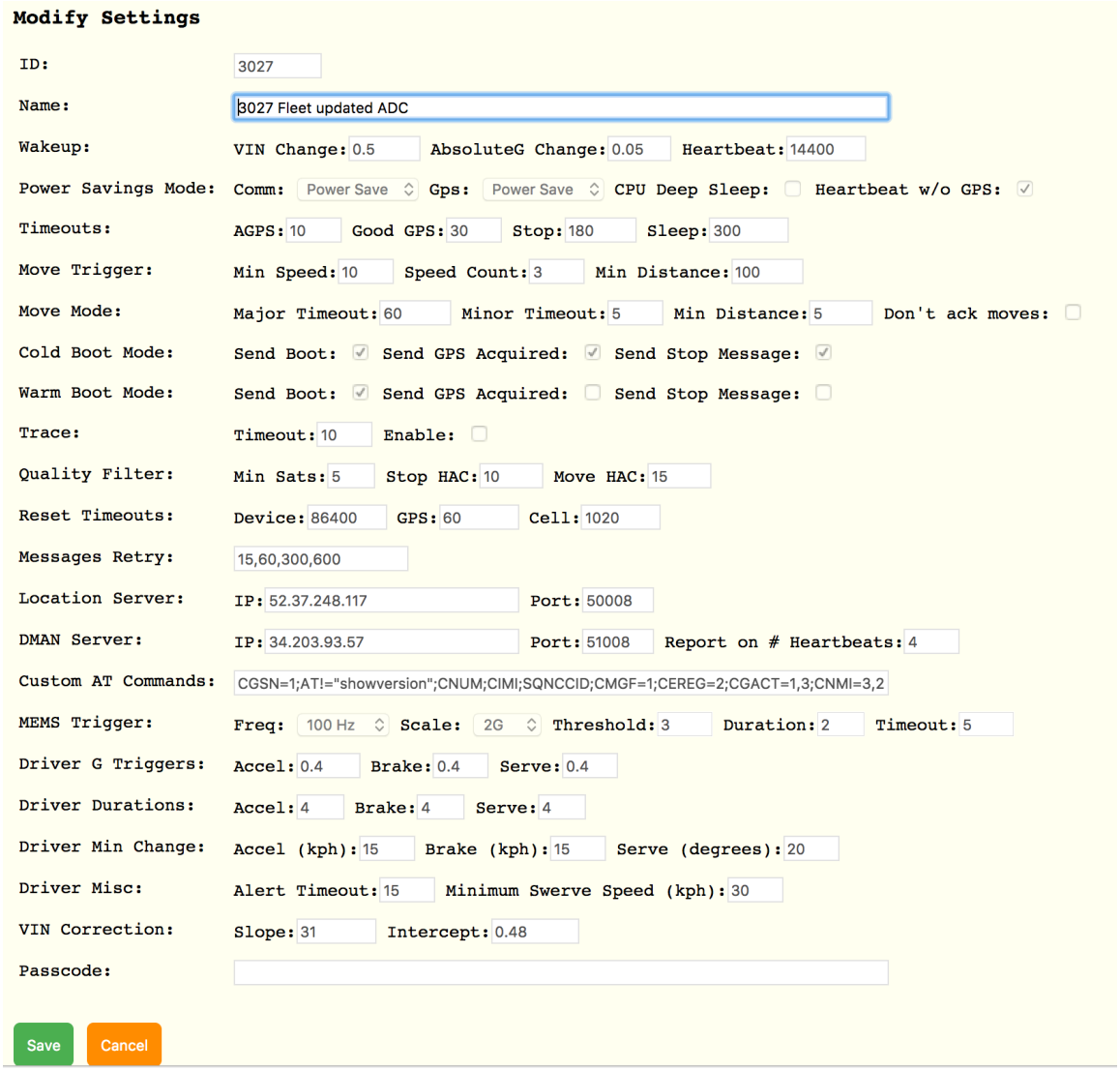
The **green LED** indicates GPS status:

- Blinking 1Hz indicates no reliable GPS fix – searching for GPS signal
- Solid indicates a good quality GPS fix
- A quick flash every 5 seconds indicates that the GPS module is in low power mode

6 Settings File

The parameters used by the application to control tracking and other functions are contained in a settings file on the DMAN server and loaded to devices over the air. A simple interface allows users to specify the values of relevant parameters for devices or groups of devices. Below is an example of the settings file creation interface.

Please see the DMAN User Guide for details on how to create Settings Files.



7 Parsing Messages

Messages are transmitted in binary form and parsed as hex values. Transmission is by the UDP protocol and the device typically expects an ACK response for each message to confirm that the server has successfully received each message, unless "No-Ack" Moving Location messages are specified in the Settings file.

The message protocol is given by the tables below.

7.1 Location Message

Location messages are triggered by many different events. The Location Message is the main message type sent to the Location server.

Name	*Format*	*Description*
Checksum	us_short	Modified Fletcher checksum of payload (all data excepting the checksum itself). See section 7.6 for a code example.
IMEI	us_long	An 8 byte value representing the IMEI of the device
Msg Type	us_byte	1 for Ack 2 for no Ack
Sequence	us_byte	Value to be used in ack to uniquely identify message
Event Type	us_byte	Type of location that is being reported. See Event ID Codes.
Event Date	us_int	A 4 byte value representing the datetime of the event in seconds since 1970-01-01 00:00:00 UTC.
Fix Delay	us_byte	Squared difference from event Date (actual = event date - delay^2)
Lat	s_int	A 4 byte signed integer value where Latitude = value/10000000 in decimal degrees
Lng	s_int	A 4 byte signed integer value where Longitude = value/10000000 in decimal degrees
Speed	us_byte	A value representing speed where 1 = 1 kph up to 160 kph and then every extra 1 = 5 kph. So a value of 1 = 1 kph, 93 = 93 kph, 163 = 175 kph, 200 = 360 kph
Head	us_byte	Degrees = round(360/256 * value). So a value of 0 = 0 degree, 10 = 14 degrees, 180 = 253 degrees, 181 = 255 degrees.
Main Power	us_short	Main power supply, usually vehicle electrical system. Volts =

(VIN)		value/100. So a value of 1315 = 13.15 Volts
Battery Power	us_byte	Internal battery of unit, used when main power supply is no longer available (not present on all devices). Volts = value/10. So a value of 41 = 4.1 Volts
Sats	us_byte	Number of satellites used to calculate the Lat, Lng values
HAC	us_byte	The Horizontal Accuracy. A value that hints at the quality of the Lat, Lng fix. HAC = value/10. So a value of 36 = 3.6 HAC
Peak G	us_byte	The maximum differential G value (from rest) recorded during period since last location message was reported. G = value/100. So a value like 50 = 0.5 G
RSSI	s_byte	The strength of the Cellular data connection
IO	us_byte	An 8 bit value where for example 10101100: Bit 7: Output 3 = 1 Bit 6: Output 2 = 0 Bit 5: Output 1 = 1 Bit 4: Output 0 = 0 Bit 3: Input 3 = 1 Bit 2: Input 2 = 1 Bit 1: Input 1 = 0 Bit 0: Input 0 = 0
Delta	us_short	Distance device has travelled since last Location message.

7.2 Location Message with Minors

If the use of minor messages has been specified in settings then move messages will consist of a Regular Location Message + Minor Locations. The format of the Minor Locations is as follows and is appended to regular Location Message.

Minor Locations	6 bytes/loc	us_byte: date in delta seconds from previous s_short: lat in delta 1/1000000 from previous s_short: lng in delta 1/1000000 from previous us_byte: speed Minor locations are optimized for low data overhead. They can support devices moving up to 1300 kph at 10 second reporting given the lat/lng delta is in 1/1000000 of a degree which represents an accuracy of 0.11 meters given 111km per degree (worst case) so max distance allowed per signed short is 3635m per report

7.3 Location Ack Response

This is a message from the server back to the device confirming that it has received and successfully parsed an incoming Locations Message.

Name	*Format*	*Description*
Msg Type	us_byte	42d (0x2A)
Sequence	us_byte	Value specified in request

7.4 Command Request

Sent from server to device, e.g. reboot request

Name	*Format*	*Description*
Msg Type	us_byte	61d (0x3D)
Command	ASCII String	Command String. See Commands

7.5 Command Response

Response from device when it has processed command request.

Name	*Format*	*Description*
Checksum	us_short	Fletcher checksum of payload (all data excepting the checksum itself)
IMEI	us_long	An 8 byte value representing the IMEI of the device
Msg Type	us_byte	8
Result	us_byte	0 = Failure, 1 = PASS
Cmd_Length	us_byte	Length of original command
Command	ASCII String	Original command
Resp_Length	us_byte	Length of response, 0 if no optional response
Response	ASCII String	An optional response caused by command

7.6 Modified Fletcher Checksum Calculation

The following Java code snippet explains how the modified Fletcher checksum is calculated from the incoming payload:

```
byte[] data = // message payload
int dataLength = data.length;
// calc Fletcher for each message
int CK_A = 0;
int CK_B = 0;
for (int i = 2; i < dataLength; i++) {
  CK_A = (CK_A + data[i]) & 0xFF;
  CK_B = (CK_B + CK_A) & 0xFF;
}
// check Fletcher
if (data[0] != (byte) CK_A || data[1] != (byte) CK_B) {
  System.out.println(" Fletcher mismatch:: expected=" + (byte) CK_A + "," + (byte)
CK_B
    + " rcvd=" + data[0] + "," + data[1]
    + " " + Hex.encodeHexString(data));
}
```

Here are some examples of using the checksum:

Example 1:
Received Hex data string is:
5b2000014223b2ef2d5901021e59ed00a100000000000000000000000d9225000037cdfe0000
The first 2 bytes (5b20) are the checksum, which convert to 91,32 in decimal.
The payload used for calculation of the checksum and to be parsed is:
00014223b2ef2d5901021e59ed00a100000000000000000000000d9225000037cdfe0000

Example 2:
Received Hex data string is:
e1a300014195acb2480d01460559ed00a6001d608bd6b6a70cd82b7f05a92a0b1d1ecdff0252fb022
3000c2bfb0282000a2ffb0267ffd433fb0153013f26fbfff502eb21fbfff803512cfb0008027529fbfffa0
13b15fbfffc019e10fb0000025319fb0003026421
The first 2 bytes (e1a3) are the checksum, which convert to -31,-93 in decimal.
The payload used for calculation of the checksum and to be parsed is:
00014195acb2480d01460559ed00a6001d608bd6b6a70cd82b7f05a92a0b1d1ecdff0252fb0223000
c2bfb0282000a2ffb0267ffd433fb0153013f26fbfff502eb21fbfff803512cfb0008027529fbfffa013b1
5fbfffc019e10fb0000025319fb0003026421

8 Event ID Codes

Regular report messages from the tracker include a ID code byte that indicates the reason for the message. Below is a list of codes used (in decimal) and proposed to date. Some of these are only intended for troubleshooting and might not be generated by production applications.

2	Ignition OFF
3	Heartbeat
4	Ignition ON
5	Move Interval
6	In1 High
7	In1 Low
8	In2 High
9	In2 Low
10	In3 High
11	In3 Low
22	Power Connected
23	Power Disconnected
24	Starter Disabled
25	Starter Enabled
26	Stop
27	Minor Location
28	Virtual Ignition ON
30	GPS Acquired
33	Hard Acceleration
34	Hard Brake
35	Hard Left
36	Hard Right
39	Cold Boot
42	Warm Boot
44	Periodic Boot
104	Vibe Wake
105	Volt Wake
106	Ephem Wake
107	Sleep
108	Period Boot
110	Comm Reset for Session Unavailable
111	GPS Reset
112	Comm Reset for No Ack
113	Comm Reset for No AT response
114	Comm Reset by Command
116	Absolute G Wake