# TT8750PLUS Message Decodification

Revision 1.1
04/02/2012
SkyPatrol LLC

# Index

## General Message Structure:

The message we received from the TT8750Plus device can be separated into two parts: the API Header, and the Device Data. Below is a diagram of this structure:

**Message Data**

| API Header | Device Data |
|---|---|

### *API Header (UDP messages):*

The base API message header is 4 bytes long for commands sent via UDP/IP.

Messages sent via UDP/IP has the following API header

Bytes 0 - 1: 16–bit API number (**refer to Table 1 below**)

Bytes 2: 8–bit Command Type information. This value determines the type of message being sent or received by the host (**refer to Table 1 below**)

Bytes 3: **Message type**: bit7, bit6, bit5, bit4 inform the message type. Please refer to the parameter <message type> in the command AT$TTMSGMASK.
**Need Ack**: bit3, bit2, bit1, bit0 inform whether the unit waits for the acknowledgement of the message.
0: do not wait for acknowledgement
1: wait for the acknowledgement

Bytes 4: API Optional Header Size. This field defines the size of the API Optional Header in Bytes. API Optional Header Size is up to the value of bit 27 of message mask.

Bytes 5 thru (5+m): API Optional Header. If it is 1, API optional header size will be 4 and the message mask will follow API optional header size. If it is 0, API optional header size will be 0. And this is only for the report from the unit to the server

**API Number (bytes 0-1):**
Two byte value that indicates the nature of the message, e.g. an Unsolicited Msg or ACK.

**Command Type (byte 2):**
One byte value that indicates that the command is requesting, e.g. Read or Write.

**MSG type (byte 3):**
Indicates the kind of message that is being generated. It is broken into two parts, the ***Message Type*** indicator, and the ***Need Ack*** indicator.

***Message Type indicator***: Bit 7, bit 6, bit 5, and bit 4. Possible decimal values are from 0-4.

**Message Type Indicator**
0: Keep alive message
1: Position report messages
2: Counter messages
3: Variables messages
4: Geofence messages

***Need Ack indicator***: Bit 3, bit 2, bit 1, and bit 0. Possible decimal values are 0 or 1.

**Need Ack Indicator**
0: Do not wait for acknowledgement
1: Wait for the acknowledgement

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|

|←---------**Message Type Indicator**-------→|←------------**Need Ack Indicator**----------→|

**API Optional Header Size (byte 4)**
One byte value that indicates the size of the optional header. May have a maximum value of 255.

**API Optional Header (byte 5 – m)**
The size of the optional header is taken from byte 4 above.

Below is the API Header table showing the combinations of the API Number, Command Type, Message Type Indicator, and API Optional Header Size.

| **API number** (values given below are decimal) | | **Command Type** | MSG type / Need ACK | API Optional Header Size | |
|---|---|---|---|---|---|
| Byte – 0 | Byte – 1 | Byte – 2 | Byte – 3 | Byte – 4 | Direction |
| 0 – 4 Reserved<br>5 – GPS Binary Read*<br>6– 65535 Reserved | | 0<br>(Read Request) | | 0 | Modem ← OTA |
| 0 – Unsolicited Msg Request<br>1 – 9 Reserved<br>10 – ACK<br>11 – Password<br>12 – 65535 Reserved | | 1<br>(Write Request) | | 0 | Modem ← OTA |
| **API number** (values given below are decimal) | | **Command Type** | MSG type / Need ACK | API Optional Header Size | |
| Byte – 0 | Byte – 1 | Byte – 2 | | Byte – 3 | Direction |
| 0 – Unsolicited Msg<br>1 – 3 Reserved<br>4 – ASCII Event Data (Param2>=256)*<br>5 – Binary Event Data*<br>6 – Reserved<br>7 – ASCII TAIP Data*<br>8 – $MSGSND Data<br>9 – Reserved<br>10 – ASCII Event Data (Param2<256) | | 2<br>(General Status Information) | | (Size of API Optional Header) | Modem → OTA |
| Echo first 2 bytes of an incoming data | | 3<br>(Error) | | (Size of API Optional Header) | Modem → OTA |
| 0 – 65535 | | 4<br>(AT Command) | | 0 | Modem ← OTA |
| Echo first 2 bytes of an incoming AT command request | | 5<br>(AT Command Response) | | (Size of API Optional Header) | Modem → OTA |
| First 2 bytes of AT$SNDMG command in ASCII format | | 6 | | (Size of API Optional Header) | Modem ←→ OTA |
| First 2 bytes of AT$SNDMG command in Binary format | | 7 | | (Size of API Optional Header) | Modem ←→ OTA |
| 10 | | 8 | | 0 | Modem ID / Modem → OTA (TCP only) |

Table 1

***Device Data:***

This section describes the structure of the data that the device will transmit to the server. As indicated in the section API Header, the device can transmit 5 different Message Type Indicators. These indicators are as follows:

**Message Type Indicator**
0: Keep alive message
1: Position report messages
2: Counter messages (we are not going to implement)
3: Variables messages (we are not going to implement)
4: Geofence messages (we are not going to implement)

Based on the Message Type Indicator that the device generated, the message needs to be decoded accordingly. Below are the steps to decode the various Message Type Indicators available on the TT8750Plus device.

All messages that the device will generate in our application will be in binary.

## Message Type Indicator 0 – Keep Alive Message:

The message mask value is obtained as a result of selecting individual bit-fields from the table below.

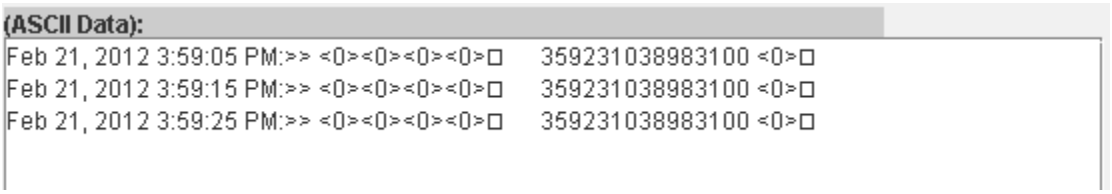
| **Message Mask** | **Description** |
|---|---|
| Bits 0 | 1 = send all data generated as a result of this table in Binary format<br>0 = send all data generated as a result of this table in ASCII format |
| Bits 1 | 1 = add parm1 data to UDP message (4 – bytes in Binary format, 11 – bytes of data in ASCII format)<br>0 = do not add parm1 data to outbound UDP message |

| | |
|---|---|
| Bits 2 | 1 = add $MDMID value (22 – bytes of ASCII data – irrespective of Bit– 0 setting)<br>0 = do not add $MDMID value |
| Bits 3-26 | Reserved |
| Bits 27 | 1 = Add Position report mask<br>0 = do not add Position report mask |
| Bits 28 | 1 = Add sequence number<br>0 = Do not add sequence number |
| Bits 29-31 | Reserved |

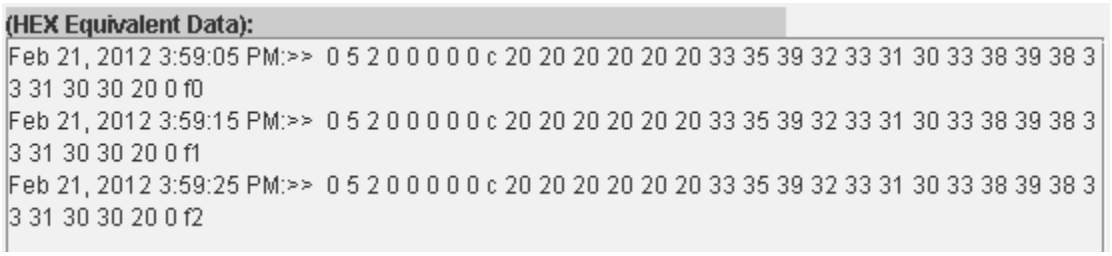
Format: 10000000000000000000000000111
Decimal: 268435463

Raw Data Example:

Binary Data:




HEX Equivalent Data:




HEX Data Breakup Example:

**00 05 02 00 00 00 00 00 0c 20 20 20 20 20 20 33 35 39 32 33 31 30 33 38 39 38 33 31 30 30 20 00 f0**

| **Description:** | **Size** | **Value (HEX):** | **Meaning:** |
|---|---|---|---|
| API Header (5 bytes) | 5 bytes | **00 05 02 00 00** | (From Table 1)<br>**API Number**: (**00 05)h** -> Binary Event Data<br><br>**Command Type**: (**02)h** -> General Status Information<br><br>**MSG Type**: **00h** -> Broken into two parts: (00)h = $(00000\ 0000)_b$ (1 byte)<br><br>*Message Type Indicator (bit 7,6,5 & 4)* -> (0000)b = 0d => Keep Alive Message<br><br>*Need Ack Indicator (bit 3,2,1 & 0)* -> (0000)b = 0d = No Ack needed.<br><br>**API Optional Header Size**: (**00)h** -> No Optional Header |
| Device Data | | | |
| Param1 (Status) | 4 bytes | **00 00 00 0c** | This is the Status of the message. Convert the entire string into decimal.<br><br>**(00 00 00 0c)h** = 12d |
| Callsign ID | 22 bytes | **20 20 20 20 20 20 33 35 39 32 33 31 30 33 38 39 38 33** | This is the callsign ID of the device. First and Last bytes are spaces.<br>Convert each byte individually to ASCII.<br><br>**(20 20 20 20 20 20 33 35 39 32 33 31 30 33 38 39 38 33 31 30 30 20)h** = |

| | | | |
|---|---|---|---|
| | | **31 30 30 20** | 359231038983100 |
| Sequence Number | 2 bytes | **00 f0** | This is the sequence number of the message. Convert the entire string to decimal.<br><br>**(00 f0)h** = 240d |

## Message Type Indicator 1 – Position Report Message:

| Message Mask | Description |
|---|---|
| Bits 0 | 1 = send all data generated as a result of this table in Binary format<br><br>0 = send all data generated as a result of this table in ASCII format |
| Bits 1 | 1 = add parm1 data to UDP message (4 – bytes in Binary format, 11 – bytes of data in ASCII format)<br><br>0 = do not add parm1 data to outbound UDP message |
| Bits 2 | 1 = add $MDMID value (22 – bytes of ASCII data – irrespective of Bit– 0 setting)<br><br>0 = do not add $MDMID value |
| Bits 3 | 1 = add $IOGPA (GPIO data) in ASCII-HEX format (2 – bytes in Binary format, 6 – bytes in ASCII format)<br><br>0 = do not add GPIO direction and data value. |
| Bits 4 | 1 = add Analog input 1 to UDP message (2 – bytes in Binary format, 5 – bytes of data in ASCII format)<br><br>0 = do not add Analog input 1 data to outbound UDP message |
| Bits 5 | 1 = add Analog input 2 to UDP message (2 – bytes in Binary format, 5 – bytes of data in ASCII format)<br><br>0 = do not add Analog input 1 data to outbound UDP message |
| Bits 6 | 1 = Message is stored in non-volatile memory until it can be sent, regardless of network status.<br><br>0 = Code checks network status before storing message in non-volatile memory. If it appears that the message can be sent out immediately (network status is clear and message queue has few or no messages pending), the message is stored in the non-volatile message queue until it can be sent. Otherwise, the message is deleted. |

| | |
|---|---|
| Bits 7 | 1 = add input <function category> number (1 – byte in binary format, 3 – bytes in ASCII format)<br><br>0 = do not add input <function category> number |
| Bits 8 | 1 = add GPS data (3 – bytes of Date information in Binary format or up to 80 – bytes of $GPGGA NMEA message if Bit-0 is set to 0)<br><br>0 = do not add this particular field of GPS data |
| Bits 9 | 1 = add 1-byte of STATUS information in Binary<br><br>0 = do not add this particular field of GPS data |
| Bits 10 | 1 = add GPS data (4 – bytes of Latitude information in Binary format or up to 80 – bytes of $GPGSA NMEA message if Bit-0 is set to 0)<br><br>0 = do not add this particular field of GPS data |
| Bits 11 | 1 = add GPS data (4 – bytes of Longitude information in Binary format or up to two 80 – bytes of $GPGSV NMEA message if Bit-0 is set to 0)<br><br>0 = do not add this particular field of GPS data |
| Bits 12 | 1 = add GPS data (2 – bytes of Velocity information in Binary format or up to 80 – bytes of $GPRMC NMEA message if Bit-0 is set to 0)<br><br>0 = do not add this particular field of GPS data |
| Bits 13 | 1 = add 2-bytes of HEADING information in Binary<br><br>0 = do not add this particular field of GPS data |
| Bits 14 | 1 = add GPS data (3 – bytes of Time information in Binary format or 0 bytes if Bit-0 is set to 0)<br><br>0 = do not add this particular field of GPS data |
| Bits 15 | 1 = add GPS data (3 – bytes of Altitude information in Binary format or 0 bytes if Bit-0 is set to 0)<br><br>0 = do not add this particular field of GPS data |
| Bits 16 | 1 = add GPS data (1 – byte of Number Of Satellites In View |

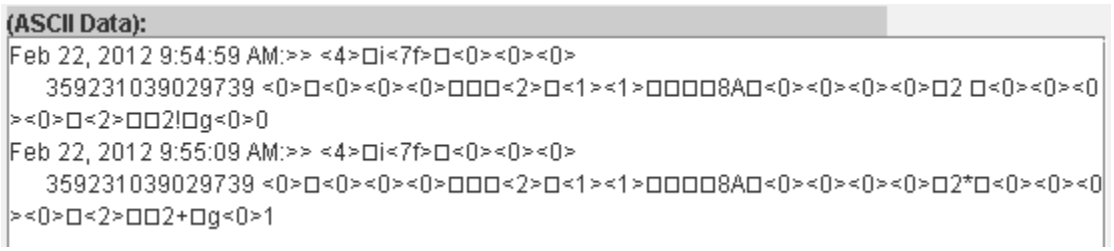
| | |
|---|---|
| | information in Binary format or 0 bytes if Bit-0 is set to 0)<br><br>0 = do not add this particular field of GPS data |
| Bits 17 | 1 = add battery level percentage (2 - bytes of in Binary format or 3 - bytes if Bit-0 is set to 0)<br><br>0 = do not add this particular field |
| Bits 18 | 1 = send this OTA message via SMS when GPRS services is not available<br><br>0 = send this OTA message via GPRS only |
| Bits 19 | 1 = send Last Valid GPS data if current data is invalid<br><br>0 = send current GPS data – valid or invalid |
| Bits 20 | 1 = add Trip Odometer reading (4 - bytes of Trip Odometer information in Binary format or 11 - bytes if Bit-0 is set to 0)<br><br>0 = do not add this particular field of GPS data<br><br>NOTE: The Trip Odometer is associated with the AT$TTTRODOM command. |
| Bits 21 | 1 = add Odometer reading (4 - bytes of Trip Odometer information in Binary format or 11 - bytes if Bit-0 is set to 0)<br><br>0 = do not add this particular field of GPS data<br><br>NOTE: The Trip Odometer is associated with the AT$TTODOM command. |
| Bits 22 | 1 = add RTC time (6 – bytes of RTC time in Binary format or 13 – bytes if Bit-0 is set to 0)<br><br>0 = do not add RTC time with GPS data |
| Bits 23 | 1 = Replace/append device id field with 10-byte device id (including one leading and one ending space character) if bit-0 is set to 0.<br><br>Replace/append it with 8-bytes long device id value if bit-0 is set to 1 (no leading or ending space characters in binary mode.)<br><br>(NOTE: bit-22 setting overrides bit-2 setting) |

| | |
|---|---|
| | 0 = Sent the device id as defined by Bit-2 |
| Bits 24 | 1 = add battery level percentage (2 - bytes of in Binary format or 3 - bytes if Bit-0 is set to 0)<br><br>0 = do not add this particular field |
| Bits 25 | 1 = add GPS overspeed data (6 – bytes of Odometer information in Binary format or 6 to 18 – bytes if Bit-0 is set to 0).<br><br>Binary format: xxyyzz:<br><br>xx is speed specified by AT$TTGPSOSI (unit: knots);<br><br>yy is the maximum speed incurred during the interval<br><br>(unit: knots, 1/10 knot accuracy);<br><br>zz is the interval duration (unit: seconds);<br><br>ASCII format: " x y z": space delineated, length of each field varies with its value<br><br>0 = do not add this particular field of GPS data |
| Bits 26 | 1 = Add cell information as follows:<br><br>If Binary format (Bit0=1) is selected, please refer to the "Bit 25 Binary Format" table in section (Bit 26 Binary Format Table)<br><br>If ASCII format (Bit0=0) is selected please refer to the "Bit 25 ASCII Format" table in section (Bit 26 ASCII Format Table)<br><br>0 = Do not add cell information |
| Bit 27 | 1 = Add Position report mask<br><br>0 = do not add Position report mask |
| Bits 28 | 1 = Add sequence number<br><br>0 = Do not add sequence number |
| Bits 29 - 31 | Reserved |

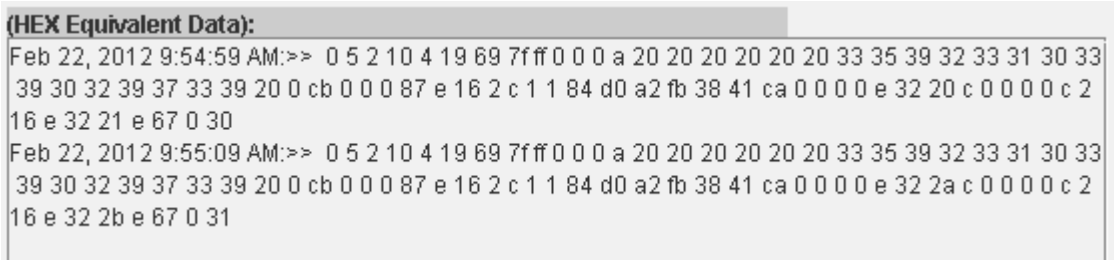
Format: 11001011010010111111111111111
Decimal: 426344447

Raw Data Example:

Binary Data:




HEX Data:




HEX Data Breakup Example:

**00 05 02 10 04 19 69 7F FF 00 00 00 0A 20 20 20 20 20 20 33 35 39 32 33 31 30 33 39 30 32 39 37 33 39 20 00 CB 00 00 00 87 0E 16** 02 0C 01 **01 84 D0 A2 FB 38 41 CA 00 00 00 00 0E 32 20 0C 00 00 00 00 0C 02 16 0E 32 21 0E 67 00 30**

| **Description:** | **Size** | **Value (HEX):** | **Meaning:** |
|---|---|---|---|
| API Header (5 bytes) | 5 bytes | **00 05 02 10 04** | (From Table 1)<br>**API Number**: $(\mathbf{00\ 05})_H$ -> Binary Event Data (2 bytes)<br><br>**Command Type**: $(\mathbf{02})_H$ -> General Status Information (1 byte)<br><br>**MSG Type**: $(\mathbf{10})_H$ -> Broken into two parts: (10)h = $(00010000)_b$ (1 byte)<br><br>*Message Type Indicator (bit 7,6,5 & 4)* -> (0001)b = 0d => Position Message<br><br>*Need Ack Indicator (bit 3,2,1 & 0)* -> (0000)b = 0d = No Ack needed.<br><br>**API Optional Header Size**: $(\mathbf{04})_H$ -> Optional Header includes 4 bytes of information (Refer to next line below) |
| Optional Header | 4 bytes variable (depending on **API Optional Header Size**) | **19 69 7F FF** | This is the message format that is being used. Convert the entire string to HEX.<br><br>$(19\ 69\ 7F\ FF)_H$ = 426344447 d |
| Device Data | | | |
| Param1 (Status code) | 4 bytes | **00 00 00 0A** | This is the Status code of the message. Convert the entire string into decimal. |

| | | | |
|---|---|---|---|
| | | | **(00 00 00 0A)**$_H$ = 10d |
| Callsign ID | 22 bytes | **20 20**<br>**20 20**<br>**20 20**<br>**33 35**<br>**39 32**<br>**33 31**<br>**30 33**<br>**39 30**<br>**32 39**<br>**37 33**<br>**39 20** | This is the callsign ID of the device. First and Last bytes are spaces.<br>Convert each byte individually to ASCII.<br><br>$(20)_H$ $(20)_H$ $(20)_H$ $(20)_H$ $(20)_H$<br>$(20)_H$ $(33)_H$ $(35)_H$ $(39)_H$ $(32)_H$<br>$(33)_H$ $(31)_H$ $(30)_H$ $(33)_H$ $(39)_H$<br>$(30)_H$ $(32)_H$ $(39)_H$ $(37)_H$ $(33)_H$<br>$(39)_H$ $(20)_H$ =<br>359231039029739 |
| IO Data | 2 bytes | **00 CB** | I/O Data from the device. Convert to binary to get the status of the I/Os.<br><br>(00 CB)H = (011001011)b<br>Bit 0 \|←--------→\| Bit 8<br><br>Bit 0 = 1 (Input 1)<br>Bit 1 = 1 (Input 2)<br>Bit 2 = 0<br>Bit 3 = 1 (Main Power State)<br>Bit 4 = 0<br>Bit 5 = 0<br>Bit 6 = 1<br>Bit 7 = 1<br>Bit 8 = 0 (Ignition State) |
| Analog Input 1 | 2 bytes | **00 00** | Analog Input 1 (Optional) |
| Analog Input 2 | 2 bytes | **00 87** | Analog Input 2 (Optional) |
| Function Category | 1 byte | **0E** | This is why the message was triggered. Convert entire string to decimal:<br>$(0E)_H$ = 14d (14 is the input function category for a timer) |
| GPS Date | 3 bytes | **16 02**<br>**0C** | GPS Data that the message was generated. Convert each |

| | | | |
|---|---|---|---|
| | | | byte individually to ASCII.<br><br>$(16)_H$ $(02)_H$ $(0C)_H$ =<br>22 (day) 2 (month) 12 (year) |
| GPS Status | 1 byte | **01** | GPS Status from the device. Convert to decimal.<br><br>1 = Valid Position<br>9 = Invalid Position |
| Latitude | 3 bytes | **01 84 D0 A2** | Refer to section "Lat/Long Conversion" for conversion instructions.<br><br>Latitude = 25.481378 |
| Longitude | 4 bytes | **FB 38 41 CA** | Refer to section "Lat/Long Conversion" for conversion instructions.<br><br>Longitude = -80.199222 |
| Velocity | 2 bytes | **00 00** | The velocity of the device. Convert to decimal, and divide by 10 to get the velocity in knots.<br><br>$(00\ 00)_H = 0_D$<br>$0_D/10$ = 0.0 knots |
| Heading | 2 bytes | **00 00** | Heading of the device. Convert to decimal, and divide by 10 to get the heading in degrees.<br><br>$(00\ 00)_H = 0_D$<br>$0_D/10$ = 0.0 degrees |
| GPS Time | 3 bytes | **0E 32 20** | The GPS UTC time that the message was generated. Convert each byte individually to decimal. |

| | | | |
|---|---|---|---|
| | | | $(0E)_H$ $(32)_H$ $(20)_H$ = 14 (hours) 50 (minutes) 32 (seconds) |
| No. of Satellites | 1 byte | **0C** | The number of satellites seen by the device. Convert to decimal.<br><br>$(0C)_H$ = 12 satellites |
| Odometer | 4 bytes | **00 00 00 00** | The odometer reading of the device. Convert to decimal to obtain the odometer in meters.<br><br>$(00\ 00\ 00\ 00)_H = 0_D$ meters |
| RTC Time | 6 bytes | **0C 02 16 0E 32 21** | The RTC time of the device when the message was generated. Convert each byte individually.<br><br>$(0C)_H$ $(02)_H$ $(16)_H$ $(0E)_H$ $(32)_H$ $(21)_H$ = 12 (year) 2 (month) 22 (day) 14 (hour) 50 (minutes) 33 (seconds) |
| Battery Voltage Level | 2 bytes | **0E 67** | Battery voltage value of the main power supply. Convert to decimal and divide by 1000.<br><br>$(0E\ 67)_H = 3687_D$<br>$3687_D/1000 = 3.687$ V |
| Sequence No. | 2 bytes | **00 30** | Sequence number of the message |

## Lat/Long Conversion

If Lat or Long is > 7F FF FF FF, then it has a negative degree

**Part 1: Positive Coordinate (Latitude Example)**

Lat: (**01 84 D0 A2**)$_H$

Since Lat is <= 7F FF FF FF, the latitude is positive

Step 1 – Convert to decimal:
Lat_decimal = 25481378

Step 2 – Divide by 1000000
Lat_decimal = +25.481378

**Part 2: Negative Coordiante (Longitude Example)**

Long: **(FB 38 41 CA)$_H$**

Since Long is > 7F FF FF FF, the longitude is negative

Step 1 – 2's complement
Long_comp = (FF FF FF FF)$_H$ – Long
Long_comp = (FF FF FF FF)$_H$ - (FB 38 41 CA)$_H$ = (04 C7 BE 35)$_H$

Step 2 – Convert to decimal
Long_decimal = 80199221

Step 3 – Divide by 1000000
Long_decimal = 80.199221

Step 4 – Add the negative
Long_decimal = -80.199221

# Geofence:

## *Setup geofence*

The command to setup one geofence is:

AT$TTGEOFNC=<index> <radius>,<latitude>,<longitude>

Where :

| | |
|---|---|
| <index> | 1-25. Defines the circular geofence index. |
| <radius> | 0-1000000. Defines radius of the circle from given Latitude and Longitude coordinates (in meters) |

<latitude> Defines the latitude for the center point of a circle (degrades)
<longitude> Defines the longitude for the center point of a circle (degrades)

The response from the device is:

OK

***Setup dbouce geofence***

The command to setup the dbounce geofence is:

AT$TTGFDB=<out_cnt>,<in_cnt>

Where :
<out_cny> 0 – 250. Consecutive GPS position reports outside a geofence
<in_cnt> 0 – 250. Consecutive GPS position reports inside a geofence

The response from the device is

OK

## ACK Messages

To disable sending of a message that requires acknowledgement, the server should send the following data, indicating an ACK, to stop sending of the messages.

| Bytes | Data Description | Comments |
|---|---|---|
| 0 | 0x00 | Parameter Number |
| 1 | 0x0A | |
| 2 | 0x01 | Write Request |
| 3 | 0x00 | MSG type / Need ACK |
| 4 | 0x00 | Response status |

## AT Commands format.

## ***Write AT Command Request***

AT commands can be sent to the modem by OTA. The host-to-modem message structure for reading/writing an AT command is as follows:

| Bytes | Data Description | Comments |
|---|---|---|
| 0 | 0x00 | Sequence Number |
| 1 | 0x01 | |
| 2 | 0x04 | AT Command Read/Write |
| 3 | 0x00 | MSG type/ Need ACK |
| 4 | 0x00 | Reserved |
| 5 | …… | AT Command |

## ***Read AT Command Request***

The modem will respond with the following message:

| Bytes | Data Description | Comments |
|---|---|---|
| 0 | 0x00 | Sequence Number |
| 1 | 0x01 | |
| 2 | 0x05 | AT Command Response |
| 3 | 0x00 | MSG type/ Need ACK |
| 4 | 0x00 | Reserved |
| 5 | …… | AT Command Response |