FM3400 Protocols

V0.02

Contents

1. FM34 DATA SENDING OVER TCP/IP PROTOCOL

1.1 AVL data packet

The table below shows the AVL structure, which is used to send data over TCP/IP to server:

<table>
<tr><th>4 zeros</th><td>4 bytes</td></tr>
<tr><th>Data length</th><td>4 bytes</td></tr>
<tr><th>Codec ID</th><td>1 byte</td></tr>
<tr><th>Number of data</th><td>1 byte</td></tr>
<tr><th rowspan="5">Data</th><td>1st byte</td></tr>
<tr><td>2nd byte</td></tr>
<tr><td>…</td></tr>
<tr><td>…</td></tr>
<tr><td>nth byte</td></tr>
<tr><th>Number of data</th><td>1 byte</td></tr>
<tr><th>CRC16</th><td>4 bytes</td></tr>
</table>

Byte fields description:	
Four zeros	For bytes with value 0x00, indicates new AVL packet.
Data length	Number of bytes calculated from Codec ID byte to second No. of data byte.
Codec ID	Constant value - 0x08.
Number of data	Number of AVL data arrays (up to 25 (0x19)).
Data	AVL data arrays.
Number of data	Number of AVL data arrays (up to 25(0x19)).
CRC	16bit CRC value of data calculated from Codec ID byte to second No. of data byte.

1.2 AVL Data

AVL data structure is shown in table below:

Timestamp	8 bytes
Priority	1 byte
GPS Element	15 bytes
IO element	n bytes

Byte fields description:

1) Timestamp

Timestamp – difference, in milliseconds, between the current time and midnight, January 1, 1970 UTC.

2) Priority

0	Low
1	High
2	Panic

3) GPS Element

GPS element consists of 15 bytes. Bytes meaning is given below:

Longitude	X coordinate	4 bytes
Latitude	Y coordinate	4 bytes
Altitude	In meters above sea level	2 bytes
Angle	In degrees (0° is north) increasing clock-wise	2 bytes
Satellites	Number of visible satellites	1 byte
Speed	Speed in km/h. 0x0000 if GPS data is invalid	2 bytes

If record is without valid coordinates – (there were no GPS fix in the moment of data acquisition) – Longitude, Latitude and Altitude values are last valid fix, and Angle, Satellites and Speed are 0.

Longitude and latitude are integer values built from degrees, minutes, seconds and milliseconds by formula:

$$\left(d + \frac{m}{60} + \frac{s}{3600} + \frac{ms}{3600000}\right) \cdot p,$$

where d – degrees, m – minutes, s – seconds, ms – milliseconds, p – precision (10000000)

If longitude is in west or latitude in south, multiply result by –1. To determine if the coordinate is negative, convert it to binary format and check the very first bit. If it is 0 then coordinate is positive, if it is 1 then coordinate is negative. Example:

Received value: **20 9C CA 80**
Converted to BIN: **00100000 10011100 11001010 10000000** first bit is **0**, which means coordinate is positive
Converted to DEC: **547146368**
For more information see two's compliment arithmetics.

4) IO element

Event IO ID	1 byte
Total number of IO	1 byte
N1 of One Byte IO	1 byte
1st IO ID	1 byte
1st IO Value	1 byte
…	…
N1th IO ID	1 byte
N1th IO value	1 byte
N2 of Two Byte IO	1 byte
1st IO ID	1 byte
1st IO Value	2 byte
…	…
N2th IO ID	1 byte
N2th IO value	2 byte
N4 of Four Bytes IO	1 byte
1st IO ID	1 byte
1st IO Value	4 byte
…	…
N4th IO ID	1 byte
N4th IO value	4 byte
N8 of Eight Bytes IO	1 byte
1st IO ID	1 byte
1st IO Value	8 byte
…	…
N8th IO ID	1 byte
N8th IO value	8 byte

Event IO ID – if data is acquired on event – this field defines which IO property has changed and generated an event. If data cause is not event – the value is 0.

N	total number of properties coming with record (N=N1+N2+N4+N8)
N1	number of properties, which length is 1 byte
N2	number of properties, which length is 2 bytes
N4	number of properties, which length is 4 bytes
N8	number of properties, which length is 8 bytes

1.3 Example

Received data:

```
000000000000009308030000014 0B49AA7EA000F0EB60023C346800000000000000000060
```

00000000000000930803**00000140B49AA7EA**000F0EB60023C346800000000000000000060
401004501F00050040002C700000000F10000601A00**00000140B499E49A**000F0EB60023C3
4680004D012005000000060401004501F00050040002C700000000F10000601A00**0000014**
0B4992154000F0EB60023C34680004D012005000000060401004501F00050040002C70000
0000F10000601A000300000D75

Parsing data:

	No.	No. bytes	Bytes name	Value in HEX
	1	4	Four zeros	00000000
	2	4	Data lenght	00000093
	3	1	Codec ID	08
	4	1	Number of Data	03
First record	5	8	Timestamp (ms)	00000140B49AA7EA
	6	1	Priority	00
	7	4	N Longitude	0F0EB600 (252622336 -> 25,2622336 N Longitude)
	8	4	E Latitude	23C34680 (600000128 -> 60,0000128 E Latitude)
	9	2	Altitude	0000
	11	2	Angle	0000
	12	1	Satellites	00
	13	2	Speed	0000
	14	1	Event ID	00
	15	1	Number of IO elements	06
	16	1	Number of 1B IO elements	04
	17	1	1'st 1B IO element	01
	18	1	Value 1	00
	19	1	2'nd 1B IO element	45
	20	1	Value 2	01
	21	1	3'rd 1B IO element	F0
	22	1	Value 3	00
	23	1	4'th 1B IO element	50
	24	1	Value 4	04
	25	1	Number of 2B IO elements	00
	26	1	Number of 4B IO elements	02
	27	1	1'st 4B IO element	C7
	28	4	Value 1	00000000

	29	1	2'nd 4B IO element	F1
	30	4	Value 2	0000601A
	31	1	Number of 8B IO elements	00
Second record	32	8	Timestamp (ms)	00000140B499E49A
	…	…	…	…
Third record	58	8	Timestamp (ms)	00000140B4992154
	…	…	…	…
	84	1	Number of Data	03
	85	4	CRC16	00000D75

1.4 Communication with server

First when module connects to server, module sends its IMEI. First comes short identifying number of bytes written and then goes IMEI as text (bytes). For example IMEI 356307041613802 would be sent as **`000F333536333037303431363133383032`**
First two bytes denote IMEI length. In this case 000F means, that imei is 15 bytes long. After receiving IMEI, server should determine if it would accept data from this module. If yes server will reply to module **`01`** if not **`00`**. Note that confirmation should be sent as binary packet. I.e. 1 byte 0x01 or 0x00. Then module starts to send first AVL data packet. After server receives packet and parses it, server must report to module number of data received as integer (four bytes). If sent data number and reported by server doesn't match module resends sent data.

Example:

Module connects to server and sends IMEI: **`000F333536333037303431363133383032`**

Server accepts the module: **`01`**

Module sends data packet:

AVL data packet header	Four zero bytes, 'AVL data array' length – 254	**`00000000000000FE`**
AVL data array	Codec Id – 08, Number Of Data – 2. (Encoded using continuous bit stream. Last byte padded to align to byte boundary)	**`0802<data elements>02`**
CRC	CRC of 'AVL data array'	**`00008612`**

Server acknowledges data reception (2 data elements): **`00000002`**

2. FM3400 DATA SENDING OVER UDP/IP PROTOCOL

2.1 UDP channel protocol

UDP channel is a transport layer protocol above UDP/IP to add reliability to plain UDP/IP using acknowledgment packets. The packet structure is as follows:

UDP datagram			
UDP channel packet x N			
Packet length	Packet Id	Packet Type	Packet payload
2 bytes	2 bytes	1 byte	m bytes
Packet length (excluding this field) in big endian byte order	Packet id unique for this channel	1 - Data packet requiring acknowledgment	Data payload

Acknowledgment packet should have the same *packet ID* as acknowledged data packet and empty data payload. Acknowledgement should be sent in binary format.

UDP datagram			
Packet length	Packet Id	Packet Type	Packet payload
2 bytes	2 bytes	1 byte	0 bytes
0x0003	Same as in acknowledged packet	0x1	Empty

2.2 Sending AVL data using UDP channel

AVL data are sent encapsulated in UDP channel packets (*AVL Data Array*).

AVL data encapsulated in UDP channel packet			
AVL packet ID (1 byte)	Module IMEI	AVL data array	
ID identifying this AVL Data Array	IMEI of a sending module encoded the same as with TCP	Number of encoded data packets	Array of encoded AVL data

Server response to AVL data packet	
AVL packet ID (1 byte)	Number of accepted AVL elements (1 byte)

Server response to AVL data packet	
id of received AVL data packet	number of AVL data array entries from the beginning of array, which were accepted by the server

Scenario: Module sends UDP channel packet with encapsulated AVL data packet (*Packet* type=1).
Server sends UDP channel packet with encapsulated response (*Packet type*=1)
Module validates *AVL packet id* and *Number of accepted AVL elements*. If server response with valid *AVL packet id* is not received within configured timeout, module can retry sending.

2.3 Examples

Module sends the data:
`00FDCAFE01DD000F3335363333037303431363133383032080 2<data elements>02`

UDP channel header	Length	**0x00FD**
	ID	**0xCAFE**
	Packet type	**0x01**
AVL packet header	AVL packet ID	**0xDD**
	IMEI	**0x000F333536333303730343136313338303 2**
AVL data array	Codec ID	**0x08**
	Number of data	**0x02<data elements>0x02**

Server must respond with acknowledgment:
`0005CAFE01DD02`

UDP channel header	Length	**0x0005**
	ID	**0xCAFE**
	Packet type	**0x01**
AVL packet acknowledgment	AVL packet ID	**0xDD**
	Number of accepted data	**0x02**

Packet example:
00A1CAFE01AA333536333037303431363133383032080100000013FEBDD19C8000F0E9FF02 09A71800069000012000000F0EB60023C34680004D0120050000000604010045 01F00050 040002C700000000F10000601A0001

No.	Name	Value in HEX
1	Data length	00A1
2	Packet identification	CAFE
3	Packet type	01
4	Packet ID	AA
5	IMEI length	000F
6	IMEI	333536333037303431363133383032
7	Codec ID	08
8	Number of data	01
9	Timestamp	0000013FEBDD19C8
10	Priority	00
11	GPS	0F0E9FF0209A71800069000012000 0
12	Event ID	00
13	Number of IO elements	06
14	Number of 1B IO elements	04
15	1'st 1B IO element	01
16	Value 1	00
17	2'nd 1B IO element	45
18	Value 2	01
19	3'rd 1B IO element	F0
20	Value 3	00
21	4'th 1B IO element	50
22	Value 4	04
23	Number of 2B IO elements	00
24	Number of 4B IO elements	02
25	1'st 4B IO element	C7
26	Value 1	00000000
27	2'nd 4B IO element	F1
28	Value 2	0000601A
29	Number of 8B IO elements	00
30	Number of data	01

3. SENDING DATA USING SMS

AVL data or events can be sent encapsulated in binary SMS. TP-DCS field of these SMS should indicate that message contains 8-bit data (for example: TP-DCS can be 0x04).

SM data (TP-UD)	
AVL data array	IMEI: 8 bytes
array of encoded AVL data	IMEI of sending module encoded as a big endian 8-byte long number

4. IO ELEMENTS

Permanent I/O elements (are always sent to server if enabled)			
IO ID	Property Name	Bytes	Description
1	Digital Input Status 1	1	Logic: 0 / 1
2	Digital Input Status 2	1	Logic: 0 / 1
179	Digital output status 1	1	Logic: 0 / 1
180	Digital output status 2	1	Logic: 0 / 1
21	GSM signal	1	GSM signal level value in scale 1 – 5
24	Speed	2	Value in km/h, 0 – xxx km/h
	External Voltage	2	Voltage: mV, 0 – 30 V
69	GPS/GNSS Power	2	States: 0 – short circ., 1 – connected.
80	Data Mode	1	0 – home on stop, 1 – home on move, 2 – roaming on stop, 3 – roaming on move, 4 – unknown on stop, 5 – unknown on move
181	PDOP	2	Probability * 10; 0-500
182	HDOP	2	Probability * 10; 0-500
199	Odometer Value (Virtual Odometer)	4	Distance between two records: m
200	Deep Sleep	1	0 – not deep sleep mode, 1 – deep sleep mode
205	Cell ID	2	GSM base station ID
206	Area Code	2	Location Area code (LAC), it depends on GSM operator. It provides unique number which assigned to a set of base GSM stations. Max value: 65536
240	Accelerometer	1	0 – not moving, 1 – moving.
241	GSM operator	4	Currently used GSM Operator code
67	Battery voltage	2	Battery voltage, mV
68	Battery current	2	Battery current, mA
70	PCB temperature	4	PCB/battery temperature
Eventual I/O elements (generate and send record to server only if appropriate conditions are met)			
Element ID	Property Name	Bytes	Description
155	Geofence zone 01	1	Event: 0 – target left zone, 1 – target entered zone
156	Geofence zone 02	1	Event: 0 – target left zone, 1 – target entered zone
157	Geofence zone 03	1	Event: 0 – target left zone, 1 – target entered zone
158	Geofence zone 04	1	Event: 0 – target left zone, 1 – target entered zone
159	Geofence zone 05	1	Event: 0 – target left zone, 1 – target entered zone
175	Auto Geofence	1	Event: 0 – target left zone, 1 – target entered zone

250	Trip	1	1 – trip start, 0 – trip stop
71	Jamming detection	1	Event: 0 – end of jamming, 1 – jamming detection
253	Eco driving type	1	1 – harsh acceleration, 2 – harsh braking, 3 - harsh cornering
254	Eco driving value	1	Depending on green driving type: if harsh acceleration or braking – g*100 (value 123 -> 1.23g), if harsh cornering – degrees (value in radians)
255	Over Speeding	2	At over speeding start km/h, at over speeding end km/h

5. 24 POSITION SMS DATA PROTOCOL

24-hour SMS is usually sent once every day and contains GPS data of last 24 hours. TP-DCS field of this SMS should indicate that message contains 8-bit data (i.e. TP-DCS can be 0x04).

Note, that 24 position data protocol is used only with subscribed SMS. Event SMS use standard AVL data protocol.

5.1 Encoding

To be able to compress 24 GPS data entries into one SMS (140 octets), the data is encoded extensively using bit fields. Data packet can be interpreted as a bit stream, where all bits are numbered as follows:

Byte 1	Bits 0-7
Byte 2	Bits 8-15
Byte 3	Bits 16-24
Bytes 4-…	Bits 25-…

Bits in a byte are numbered starting from least significant bit. A field of 25 bits would consist of bits 0 to 24 where 0 is the least significant bit and bit 24 – most significant bit.

5.2 Structure

SMS Data Structure			
	Size (bits)	Field	Description
	8	CodecId	CodecId = 4
	35	Timestamp	Time corresponding to the first (oldest) GPS data element, represented in seconds elapsed from 2000.01.01 00:00 EET.
	5	ElementCount	Number of GPS data elements.
ElementCount *		GPSDataElement	GPS data elements.
		Byte-aling padding	Padding bits to align to 8-bits boundary
	64	IMEI	IMEI of sending device as 8-byte long integer

The time of only the first GPS data element is specified in *Timestamp* field. Time corresponding to each further element can be computed as

elementTime = *Timestamp* + (1 hour * elementNumber).

<table>
<tr><th colspan="5">GPS Data Element</th></tr>
<tr><td>Size (bits)</td><td>Field</td><td colspan="3">Description</td></tr>
<tr><td>1</td><td>ValidElement</td><td colspan="3">ValidElement=1 – there is a valid GpdDataElement following,
ValidElement=0 – no element at this position.</td></tr>
<tr><td>1</td><td>DifferentialCoords</td><td colspan="2">Format of following data.</td><td rowspan="6">ValidElement = 1</td></tr>
<tr><td>14</td><td>LongitudeDiff</td><td>Difference from previous element‘s longitude.
LongitudeDiff = prevLongitude – Longitude + $2^{13} – 1$</td><td rowspan="2">DifferentialCoords = 1</td></tr>
<tr><td>14</td><td>LatitudeDiff</td><td>Difference from previous element‘s latitude
LatitudeDiff = prevLatitude – Latitude + $2^{13} – 1$</td></tr>
<tr><td>21</td><td>Longitude</td><td>Longitude= {(LongDegMult + 18 * 10^8) * (2^{21} – 1)} over {36*10^8}</td><td rowspan="2">DifferentialCoords = 0</td></tr>
<tr><td>20</td><td>Latitude</td><td>Latitude=(LatDegMult + 9*10^8) * (2^{20} – 1) over {18*10^8}</td></tr>
<tr><td>8</td><td>Speed</td><td colspan="2">Speed in km/h.</td></tr>
</table>

Longitude	longitude field value of *GPSDataElement*
Latitude	latitude field value of *GPSDataElement*
LongDegMult	longitude in degrees multiplied by 10^7 (integer part)
LatDegMult	latitude in degrees multiplied by 10^7 (integer part)
prevLongitude	longitude field value of previous *GPSDataElemen*
prevLatitude	latitude field value of previous *GPSDataElement*

5.3 Decoding GPS position

When decoding GPS data with *DifferentialCoords*=1, *Latitude* and *Longitude* values can be computed as follows:

$$Longitude = prevLongitude - LongitudeDiff + 2^{13} - 1,$$
$$\text{Latitude} = prevLatitude - LatitudeDiff + 2^{13} - 1.$$

If there were no previous non-differential positions, differential coordinates should be computed Assuming

prevLongitude=prevLatitude=0.

When *Longitude* and *Latitude* values are known, longitude and latitude representation in degrees can be computed as follows:

$$Long\quad Deg = \frac{Longitude \cdot 360}{2^{21} - 1} - 180\,;$$

$$Lat\quad Deg = \frac{Latitude \cdot 180}{2^{20} - 1} - 90\,;$$

6. CHANGE LOG

Nr.	Date	New version number	Comments
1	2014-01-13	0.01	Documentation created
2	2015-06-15	0.02	UDP protocol description correction.