https://wiki.teltonika-gps.com/view/Teltonika_Data_Sending_Protocols

Teltonika Data Sending Protocols

☐

Contents

Introduction

A codec is a device or computer program for encoding or decoding a digital data stream or signal. Codec is a portmanteau of coder-decoder. A codec encodes a data stream or a signal for transmission and storage, possibly in encrypted form, and the decoder function reverses the encoding for playback or editing.

Below you will see a table of all Codec types with IDs:

Codec 8	Codec 8 Extended	Codec 16	Codec 12	Codec 13	Codec 14
0x08	0x8E	0x10	0x0C	0x0D	0x0E

Also, there are using two data transport protocols: TCP and UDP. But it is not important which one will be used in Codec.

Codec for device data sending

In this chapter, you will find information about every Codec protocol which are used for device data sending and the differences between them.

Codec 8

- **Protocol Overview**

Codec8 – a main FM device protocol that is used for sending data to the server.

- **Codec 8 protocol sending over TCP**

TCP is a connection-oriented protocol that is used for communication between devices. The workings of this type of protocol is described below in the **communication with server** section.

- **AVL Data Packet**

The below table represents the AVL Data Packet structure:

0x00000000 (Preamble)	Data Field Length	Codec ID	Number of Data 1	AVL Data	Number of Data 2	CRC-16
4 bytes	4 bytes	1 byte	1 byte	X bytes	1 byte	4 bytes

Preamble – the packet starts with four zero bytes.
Data Field Length – size is calculated starting from Codec ID to Number of Data 2.
Codec ID – in Codec8 it is always `0x08`.
Number of Data 1 – a number that defines how many records are in the packet.
AVL Data – actual data in the packet (more information below).
Number of Data 2 – a number that defines how many records are in the packet. This number must be the same as “Number of Data 1”.
CRC-16 – calculated from Codec ID to the Second Number of Data. CRC (Cyclic Redundancy Check) is an error-detecting code used to detect accidental changes to RAW data. For calculation we are using CRC-16/IBM.

Note: for FMB640, FMB641, FMC640, and FMM640, minimum AVL record size is 45 bytes (all IO elements disabled). The maximum AVL record size is 255 bytes. Maximum AVL packet size is 512 bytes. For other devices, the minimum AVL record size is 45 bytes (all IO elements disabled). Maximum AVL packet size is 1280 bytes.

- AVL Data

The below table represents the AVL Data structure.

Timestamp	Priority	GPS Element	IO Element
8 bytes	1 byte	15 bytes	X bytes

Timestamp – a difference, in milliseconds, between the current time and midnight, January 1970 UTC (UNIX time).
Priority – a field that defines AVL data priority (more information below).
GPS Element – location information of the AVL data (more information below).
IO Element – additional configurable information from the device (more information below).

- Priority

The below table represents Priority values. Packet priority depends on device configuration and

records sent.

Priority	
0	Low
1	High
2	Panic

- GPS element

The below table represents the GPS Element structure:

Longitude	Latitude	Altitude	Angle	Satellites	Speed
4 bytes	4 bytes	2 bytes	2 bytes	1 byte	2 bytes

Longitude – east-west position.
Latitude – north-south position.
Altitude – meters above sea level.
Angle – degrees from north pole.
Satellites – number of satellites in use.
Speed – speed calculated from satellites.

Note: Speed will be `0x0000` if GPS data is invalid.

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

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

Where:
d – Degrees; m – Minutes; s – Seconds; ms – Milliseconds; p – Precision (10000000)
If the longitude is in the west or latitude in the south, multiply the result by –1.

Note:
To determine if the coordinate is negative, convert it to binary format and check the very first bit. If it is 0, the coordinate is positive. If it is 1, the 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 complement arithmetic.

- IO Element

Event IO ID	1 byte
N of Total IO	1 byte
N1 of One Byte IO	1 byte
1’st IO ID	1 byte
1’st IO Value	1 byte
…	
N1’th IO ID	1 byte
N1’th IO Value	1 byte
N2 of Two Bytes	1 byte
1’st IO ID	1 byte
1’st IO Value	2 bytes
…	
N2’th IO ID	1 byte
N2’th IO Value	2 bytes
N4 of Four Bytes	1 byte
1’st IO ID	1 byte
1’st IO Value	4 bytes
…	
N4’th IO ID	1 byte
N4’th IO Value	4 byte
N8 of Eight Bytes	1 byte
1’st IO ID	1 byte
1’st IO Value	8 byte
…	
N8’IO ID	1 byte
N8’IO Value	8 bytes

Event IO ID – if data is acquired on the event – this field defines which IO property has changed and generated an event. For example, when if the Ignition state changes and it generates an event, the Event IO ID will be `0xEF` (AVL ID: 239). If it’s not an eventual record – the value is 0.
N – a 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.
N’th IO ID - AVL ID.
N’th IO Value - AVL ID value.

- **Communication with server**

First, when the module connects to the server, the module sends its IMEI. First comes a short identifying the number of bytes written and then goes IMEI as text (bytes).
For example, IMEI `356307042441013` would be sent as `000F333536333037303432343431303133`.
The first two bytes denote IMEI length. In this case `0x000F` means, that IMEI is 15 bytes long.
After receiving IMEI, the 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 a binary packet. I.e. 1 byte `0x01` or `0x00`.
Then the module starts to send the first AVL data packet. After the server receives a packet and parses it, the server must report to the module number of data received as an integer (four bytes). If the sent data number and the reported by the server don’t match module resends the sent data.

- Example:

The module connects to the server and sends IMEI:
`000F333536333037303432343431303133`
The server accepts the module:
01
The module sends data packet:

AVL Data Packet Header	AVL Data Array	CRC-16
Four Zero Bytes – 0x00000000, "AVL Data Array" length – 0x000000FE	Codec ID – 0x08, Number of Data – **0x02** (Encoded using continuous bit stream. The last byte is padded to align to the byte boundary)	CRC of "AVL Data Array"
00000000000000FE	08**02**...(data elements)...**02**	00008612

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

- **Examples**

The hexadecimal stream of AVL Data Packet receiving and response in these examples are given in the hexadecimal form. The different fields of packets are separated into different table columns for better readability and some of them are converted to ASCII values for better understanding.

1'st example
Receiving one data record with each element property (1 byte, 2 bytes, 4 bytes, and 8 bytes).

Received data in the hexadecimal stream:
`000000000000003608010000016B40D8EA30010000000000000000000000000000000105021503010101425E0F01F10000601A014E0000000000000000010000C7CF`

Parsed:

AVL Data Packet	
AVL Data Packet Part	**HEX Code Part**
Zero Bytes	00 00 00 00
Data Field Length	00 00 00 36
Codec ID	08
Number of Data 1 (Records)	01

AVL Data	Timestamp	00 00 01 6B 40 D8 EA 30 (GMT: Monday, June 10, 2019, 10:04:46 AM)
	Priority	01
	Longitude	00 00 00 00
	Latitude	00 00 00 00
	Altitude	00 00
	Angle	00 00
	Satellites	00
	Speed	00 00
	Event IO ID	01
	N of Total ID	05
	N1 of One Byte IO	02
	1’st IO ID	15 (AVL ID: 21, Name: GSM Signal)
	1’st IO Value	03
	2’nd IO ID	01 (AVL ID: 1, Name: DIN1)
	2’nd IO Value	01
	N2 of Two Bytes IO	01
	1’st IO ID	42 (AVL ID: 66, Name: External Voltage)
	1’st IO Value	5E 0F
	N4 of Four Bytes IO	01
	1’st IO ID	F1 (AVL ID: 241, Name: Active GSM Operator)
	1’st IO Value	00 00 60 1A
	N8 of Eight Bytes IO	01
	1’st IO ID	4E (AVL ID: 78, Name: iButton)
	1’st IO Value	00 00 00 00 00 00 00 00
	Number of Data 2 (Number of Total Records)	01
	CRC-16	00 00 C7 CF

Server response: `00000001`

2'nd example
Receiving one data record with one or two different element properties (1 byte, 2 bytes).

Received data in the hexadecimal stream:
`000000000000002808010000016B40D9AD80010000000000000000000000000000000010302150301010425E100000010000F22A`

Parsed:

AVL Data Packet

AVL Data Packet Part	HEX Code Part
Zero Bytes	00 00 00 00
Data Field Length	00 00 00 28
Codec ID	08
Number of Data 1 (Records)	01

AVL Data	Timestamp	00 00 01 6B 40 D9 AD 80 (GMT: Monday, June 10, 2019, 10:05:36 AM)
	Priority	01
	Longitude	00 00 00 00
	Latitude	00 00 00 00
	Altitude	00 00
	Angle	00 00
	Satellites	00
	Speed	00 00
	Event IO ID	01
	N of Total ID	03
	N1 of One Byte IO	02
	1’st IO ID	15 (AVL ID: 21, Name: GSM Signal)
	1’st IO Value	03
	2’nd IO ID	01 (AVL ID: 1, Name: DIN1)
	2’nd IO Value	01
	N2 of Two Bytes IO	01
	1’st IO ID	42 (AVL ID: 66, Name: External Voltage)
	1’st IO Value	5E 0F
	N4 of Four Bytes IO	00
	N8 of Eight Bytes IO	00
	Number of Data 2 (Number of Total Records)	01
	CRC-16	00 00 F2 2A

Server response: `00000001`

3'rd example
Receiving two or more data records with one or more different element properties.

Received data in the hexadecimal stream:
```
000000000000004308020000016B40D57B480100000000000000000000000000000000010101010
00000000000016B40D5C198010000000000000000000000000000000
10101010100000020000252C
```

Parsed:

AVL Data Packet

AVL Data Packet Part	HEX Code Part
Zero Bytes	00 00 00 00
Data Field Length	00 00 00 43
Codec ID	08
Number of Data 1 (Records)	02

AVL Data (1'st record)	Timestamp	00 00 01 6B 40 D5 7B 48 (GMT: Monday, June 10, 2019, 10:01:01 AM)
	Priority	01
	Longitude	00 00 00 00
	Latitude	00 00 00 00
	Altitude	00 00
	Angle	00 00
	Satellites	00
	Speed	00 00
	Event IO ID	01
	N of Total ID	01
	N1 of One Byte IO	01
	1’st IO ID	01 (AVL ID: 1, Name: DIN1)
	1’st IO Value	00
	N2 of Two Bytes IO	00
	N4 of Four Bytes IO	00
	N8 of Eight Bytes IO	00
AVL Data (2'nd record)	Timestamp	00 00 01 6B 40 D5 C1 98 (GMT: Monday, June 10, 2019 10:01:19 AM)
	Priority	01
	Longitude	00 00 00 00
	Latitude	00 00 00 00
	Altitude	00 00
	Angle	00 00
	Satellites	00
	Speed	00 00
	Event IO ID	01
	N of Total ID	01
	N1 of One Byte IO	01
	1’st IO ID	01 (AVL ID: 1, Name: DIN1)
	1’st IO Value	01
	N2 of Two Bytes IO	00
	N4 of Four Bytes IO	00
	N8 of Eight Bytes IO	00
Number of Data 2 (Number of Total Records)		02
CRC-16		00 00 25 2C

Server response: `00000002`

- ## Codec8 protocol sending over UDP

Codec8 protocol [over UDP] is a transport layer protocol above UDP/IP to add reliability to plain UDP/IP using acknowledgment packets.

- **AVL Data Packet**

The packet structure is as follows:

UDP Datagram

Example	2 bytes
Packet ID	2 bytes
Not Usable Byte	1 byte
Packet Payload	Variable

Example – packet length (excluding this field) in big ending byte order.
Packet ID – packet ID unique for this channel.
Not Usable Byte – not usable byte.
Packet payload – data payload.

- Acknowledgment packet

The acknowledgment packet should have the same Packet ID as an acknowledged data packet and empty Data Payload. Acknowledgment should be sent in binary format.

Acknowledgment Packet

Packet Length	Packet ID	Not Usable Byte
2 bytes	2 bytes	1 byte

Packet Length – packet length by sending/response data.
Packet ID – same as in acknowledgment packet.
Not Usable Byte – always will be `0x01`.

- Sending AVL Packet Payload using UDP channel

The below table represents the Sending Packet Payload structure.

AVL data encapsulated in UDP channel packet

AVL Packet ID	IMEI Length	Module IMEI	AVL Data Array
1 byte	2 bytes	15 bytes	X bytes

AVL Packet ID – ID identifying this AVL packet.
IMEI Length – always will be `0x000F`.
Module IMEI – IMEI of a sending module encoded the same as with TCP.
AVL Data Array – an array of encoded AVL data (same as TCP AVL Data Array).

- Server response Packet Payload using UDP channel

The below table represents the Server Response Packet Payload structure.

Server Response to AVL Data Packet

AVL Packet ID	Number of Accepted AVL Elements
1 byte	1 byte

- **Communication with server**

The module sends the UDP channel packet with an encapsulated AVL data packet. The server sends the UDP channel packet with an encapsulated response module that validates the AVL Packet ID and the Number of accepted AVL elements. If the server response is not received with a valid AVL Packet ID within configured timeout, the module can retry sending.

- Example:

The module sends the data:

UDP Channel Header	AVL Packet Header	AVL Data Array
Length – 0x00FE, Packet ID – 0xCAFE Not Usable Byte – 0x01	AVL Packet ID – 0xDD, IMEI Length – 0x000F IMEI – 0x313233343536373839303132333435 (Encoded using continuous bit stream. The last byte is padded to align to the byte boundary)	Codec ID – 0x08, Number of Data – 0x02 (Encoded using continuous bit stream)
00FECAFE01	DD000F313334353637383903132333435	0802…(data elements)…02

The server must respond with an acknowledgment:

UDP Channel Header	AVL Packet Acknowledgment
Length – 0x0005, Packet ID – 0xCAFE, Not Usable Byte – 0x01	AVL Packet ID – 0xDD, Number of Accepted Data – 0x02
0005CAFE01	DD02

- **Example**

The hexadecimal stream of AVL Data Packet receiving and response in this example is given in the hexadecimal form. The different fields of the packet are separated into different table columns for better readability and some of them are converted to ASCII values for better understanding.

Received data in the hexadecimal stream:
`003DCAFE0105000F333532303933303836343033363535080100000016B4F815B3001000000000000000000000000000000000000000000103021503010101425DBC000001`

Parsed:

AVL Data Packet		
AVL Data Packet Part		**HEX Code Part**
UDP Channel Header	Length	00 3D
	Packet ID	CA FE
	Not usable byte	01
AVL Packet Header	AVL packet ID	05
	IMEI Length	00 0F
	IMEI	33 35 32 30 39 33 30 38 36 34 30 33 36 35 35

AVL Data Array	Codec ID	08
	Number of Data 1 (Records)	01
	Timestamp	00 00 01 6B 4F 81 5B 30 (GMT: Thursday, June 13, 2019, 6:23:26 AM)
	Priority	01
	Longitude	00 00 00 00
	Latitude	00 00 00 00
	Altitude	00 00
	Angle	00 00
	Satellites	00
	Speed	00 00
	Event IO ID	01
	N of Total ID	03
	N1 of One Byte IO	02
	1’st IO ID	15 (AVL ID: 21, Name: GSM Signal)
	1’st IO Value	03
	2’nd IO ID	01 (AVL ID: 1, Name: DIN1)
	2’nd IO Value	01
	N2 of Two Bytes IO	01
	1’st IO ID	42 (AVL ID: 66, Name: External Voltage)
	1’st IO Value	5D BC
	N4 of Four Bytes IO	00
	N8 of EightBytes IO	00
	Number of Data 2 (Number of Total Records)	01

The server response in the hexadecimal stream: `0005CAFE010501`

Parsed:

Server Response to AVL Data Packet

Server Response Part		HEX Code Part
UDP Channel Header	Length	00 05
	Packet ID	CA FE
AVL Packet Acknowledgment	Not usable byte	01
	AVL packet ID	05
	Number of Accepted Data	01

Codec 8 Extended

- **Protocols overview**

Codec8 Extended is used for FMBXXX family devices. This protocol looks familiar to Codec8 but they have some differences. The main differences between them are shown in below table:

	Codec8	Codec8 Extended
Codec ID	0x08	0x8E
AVL Data IO element length	1 byte	2 bytes
AVL Data IO element total IO count length	1 byte	2 bytes
AVL Data IO element IO count length	1 byte	2 bytes
AVL Data IO element AVL ID length	1 byte	2 bytes
Variable size IO elements	Does not include	Includes variable size elements

- **Codec 8 Extended protocol sending over TCP**
- **AVL data packet**

The below table represents the AVL data packet structure:

0x00000000 (Preamble)	Data Field Length	Codec ID	Number of Data 1	AVL Data	Number of Data 2	CRC-16
4 bytes	4 bytes	1 byte	1 byte	X bytes	1 byte	4 bytes

Preamble – the packet starts with four zero bytes.
Data Field Length – size is calculated starting from Codec ID to Number of Data 2.
Codec ID – in Codec8 Extended it is always `0x8E`.
Number of Data 1 – a number that defines how many records are in the packet.
AVL Data – actual data in the packet (more information below).
Number of Data 2 – a number that defines how many records are in the packet. This number must be the same as “Number of Data 1”.
CRC-16 – calculated from Codec ID to the Second Number of Data. CRC (Cyclic Redundancy Check) is an error-detecting code used to detect accidental changes to RAW data. For calculation we are using CRC-16/IBM.

Note: for FMB640, FMB641, FMC640, and FMM640, minimum AVL record size is 45 bytes (all IO elements disabled). The maximum AVL record size is 255 bytes. For other devices, the minimum AVL record size is 45 bytes (all IO elements disabled). Maximum AVL packet size is 1280 bytes.

- AVL Data

The below table represents the AVL Data structure:

Timestamp	Priority	GPS Element	IO Element
8 bytes	1 byte	15 bytes	X bytes

Timestamp – a difference, in milliseconds, between the current time and midnight, January 1970 UTC (UNIX time).
Priority – a field that defines AVL data priority (more information below).
GPS Element – locational information of the AVL data (more information below).
IO Element – additional configurable information from the device (more information below).

- Priority

The below table represents Priority values. Packet priority depends on device configuration and records sent.

Priority	
0	Low
1	High
2	Panic

- GPS element

The below table represents the GPS Element structure:

Longitude	Latitude	Altitude	Angle	Satellites	Speed
4 bytes	4 bytes	2 bytes	2 bytes	1 byte	2 bytes

Longitude – east-west position.
Latitude – north-south position.
Altitude – meters above sea level.
Angle – degrees from north pole.
Satellites – number of satellites in use.
Speed – speed calculated from satellites.

Note: Speed will be `0x0000` if GPS data is invalid.

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

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

Where:
d – Degrees; m – Minutes; s – Seconds; ms – Milliseconds; p – Precision (10000000)
If the longitude is in the west or latitude in the south, multiply the result by –1.

Note:

To determine if the coordinate is negative, convert it to binary format and check the very first bit. If it is `0`, the coordinate is positive, if it is `1`, the 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 complement arithmetic.

- IO Element

Event IO ID	2 bytes
N of Total IO	2 bytes
N1 of One Byte IO	2 bytes
1’st IO ID	2 bytes
1’st IO Value	1 byte
…	
N1’th IO ID	2 bytes
N1’th IO Value	1 byte
N2 of Two Bytes	2 bytes
1’st IO ID	2 bytes
1’st IO Value	2 bytes
…	
N2’th IO ID	2 bytes
N2’th IO Value	2 bytes
N4 of Four Bytes	2 bytes
1’st IO ID	2 bytes
1’st IO Value	4 bytes
…	
N4’th IO ID	2 bytes
N4’th IO Value	4 byte
N8 of Eight Bytes	2 bytes
1’st IO ID	2 bytes
1’st IO Value	8 byte
…	
N8’IO ID	2 bytes
N8’IO Value	8 bytes
NX of X Byte IO	2 bytes
1’st IO ID	2 bytes
1’st IO Length	2 bytes
1’st IO Value	Defined by length
…	
NX’th IO ID	2 bytes
NX’th Length	2 bytes
NX’th Value	Defined by length

Event IO ID – if data is acquired on the event – this field defines which IO property has changed and generated an event. For example, when if the Ignition state changes and it generates an event, the Event IO ID will be 0x00EF (AVL ID: 239). If it’s not an eventual record – the value is 0x0000.
N – a 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.
NX – a number of properties, which length is defined by the length element. **N’th IO ID** - AVL ID.
N'th Lenght - AVL ID value lenght.
N’th IO Value - AVL ID value.

- **Communication with server**

Communication with the server is the same as with the Codec8 protocol, except in Codec8 Extended protocol Codec ID is 0x8E.

- Example:

The module connects to the server and sends IMEI:
`000F333536333037303432343431303133`
The server accepts the module:
`01`
The module sends data packet:

AVL Data Packet Header	AVL Data Array	CRC-16
Four Zero Bytes – 0x00000000, "AVL Data Array" length – 0x000000FE	Codec ID – 0x8E, Number of Data – **0x02** (Encoded using continuous bit stream. The last byte is padded to align to the byte boundary)	CRC of "AVL Data Array"
00000000000000FE	8E**02**...(data elements)...**02**	00008612

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

- **Example**

The hexadecimal stream of AVL Data Packet receiving and response in this example is given in the hexadecimal form. The different fields of the packet are separated into different table columns for better readability and some of them are converted to ASCII values for better understanding.

Received data in the hexadecimal stream:
`000000000000004A8E010000016B412CEE0001000000000000000000000000000000000100050`
`0010001010001001100 1D00010010015E2C880002000B000000003544C87`
`A000E000000001DD7E06A00000100002994`

Parsed data:

AVL Data Packet	
AVL Data Packet Part	**HEX Code Part**
Zero Bytes	00 00 00 00
Data Field Length	00 00 00 4A
Codec ID	8E
Number of Data 1 (Records)	01

	Timestamp	00 00 01 6B 41 2C EE 00 (GMT: Monday, June 10, 2019, 11:36:32 AM)
	Priority	01
	Longitude	00 00 00 00
	Latitude	00 00 00 00
	Altitude	00 00
	Angle	00 00
	Satellites	00
	Speed	00 00
	Event IO ID	00 01
	N of Total ID	00 05
	N1 of One Byte IO	00 01
	1’st IO ID	00 01 (AVL ID: 1, Name: DIN1)
AVL Data	1’st IO Value	01
	N2 of Two Bytes IO	00 01
	1’st IO ID	00 11 (AVL ID: 17, Name: Axis X)
	1’st IO Value	00 1D
	N4 of Four Bytes IO	00 01
	1’st IO ID	00 10 (AVL ID: 16, Name: Total Odometer)
	1’st IO Value	01 5E 2C 88
	N8 of Eight Bytes IO	00 02
	1’st IO ID	00 0B (AVL ID: 11, Name: ICCID1)
	1’st IO Value	00 00 00 00 35 44 C8 7A
	2’nd IO ID	00 0E (AVL ID: 14, Name: ICCID2)
	2’nd IO Value	00 00 00 00 1D D7 E0 6A
	NX of X Byte IO	00 00
	Number of Data 2 (Number of Total Records)	01
	CRC-16	00 00 29 94

Server response: `00000001`

- **Codec8 Extended protocol sending over UDP**
- **UDP channel protocol**

AVL data packet is the same as with Codec8, except Codec ID is changed to `0x8E`. AVL Data encoding was performed according to Codec8 Extended protocol.

- **Communication with server**

The module sends the UDP channel packet with an encapsulated AVL data packet. The server sends the UDP channel packet with an encapsulated response module that validates the AVL Packet ID and the Number of accepted AVL elements. If the server response is not received with a valid AVL Packet ID within configured timeout, the module can retry sending.

- Example:

The module sends the data:

UDP Channel Header	AVL Packet Header	AVL Data Array
Length – 0x00FE, Packet ID – 0xCAFE Not Usable Byte – 0x01	AVL Packet ID – 0xDD, IMEI Length – 0x000F IMEI – 0x313233343536373839303132333435 (Encoded using continuous bit stream. The last byte is padded to align to the byte boundary)	Codec ID – 0x8E, Number of Data – 0x02 (Encoded using continuous bit stream)
00FECAFE01	DD000F313334353637383930313233435	8E02…(data elements)…02

The server must respond with an acknowledgment:

UDP Channel Header	AVL Packet Acknowledgment
Length – 0x0005, Packet ID – 0xCAFE, Not Usable Byte – 0x01	AVL Packet ID – 0xDD, Number of Accepted Data – 0x02
0005CAFE01	DD02

- **Example**

The hexadecimal stream of AVL Data Packet receiving and response in this example is given in the hexadecimal form. The different fields of the packet are separated into different table columns for better readability and some of them are converted to ASCII values for better understanding.

Received data in the hexadecimal stream:

```
005FCAFE0107000F333532303933303836343033363535 8E010000016B4F831C6801000000000
000000000000000000000000010005000100010100010011009D000100
10015E2C880002000B000000003544C87A000E000000001DD7E06A000001
```

Parsed:

AVL Data Packet

AVL Data Packet Part		HEX Code Part
UDP Channel Header	Length	00 5F
	Packet ID	CA FE
	Not usable byte	01
AVL Packet Header	AVL packet ID	07
	IMEI Length	00 0F
	IMEI	33 35 32 30 39 33 30 38 36 34 30 33 36 35 35

AVL Data Array	Codec ID	8E
	Number of Data 1 (Records)	01
	Timestamp	00 00 01 6B 4F 83 1C 68 (GMT: Thursday, June 13, 2019 6:25:21 AM)
	Priority	01
	Longitude	00 00 00 00
	Latitude	00 00 00 00
	Altitude	00 00
	Angle	00 00
	Satellites	00
	Speed	00 00
	Event IO ID	00 01
	N of Total ID	00 05
	N1 of One Byte IO	00 01
	1’st IO ID	00 01 (AVL ID: 1, Name: DIN1)
	1’st IO Value	00 01
	N2 of Two Bytes IO	00 01
	1’st IO ID	00 11 (AVL ID: 17, Name: Axis X)
	1’st IO Value	00 1D
	N4 of Four Bytes IO	00 01
	1’st IO ID	00 10 (AVL ID: 16, Name: Total Odometer)
	1’st IO Value	01 5E 2C 88
	N8 of Eight Bytes IO	00 02
	1’st IO ID	00 0B (AVL ID: 11, Name: ICCID1)
	1’st IO Value	00 00 00 00 35 44 C8 7A
	2’nd IO ID	00 0E (AVL ID: 14, Name: ICCID2)
	2’nd IO Value	00 00 00 00 1D D7 E0 6A
	NX of X Byte IO	00 00
	Number of Data 2 (Records)	01

The server response in the hexadecimal stream: `0005CAFE010701`

Parsed:

Server Response to AVL Data Packet

Server Response Part		HEX Code Part
UDP Channel Header	Length	00 05
	Packet ID	CA FE
	Not usable byte	01
AVL Packet Acknowledgment	AVL packet ID	07
	Number of Accepted Data	01

Codec 16

- **Protocol overview**

Codec16 is using for FMB630/FM63XY series devices. This protocol looks familiar like Codec8 but they have some differences. The main differences between them are shown in the table below:

	Codec8	Codec16
Codec ID	0x08	0x10
AVL Data IO element ID event length	1 byte	2 bytes
AVL Data IO element AVL ID length	1 byte	2 bytes
Generation Type	Not Using	Is Using

Note: Codec16 is supported from firmware – 00.03.xx and newer. (FMB630/FM63XY) || AVL IDs that are higher than 255 will can be used only in the Codec16 protocol.

- **Codec 16 protocol sending over TCP**
- **AVL data packet**

The below table represents the AVL data packet structure:

0x00000000 (Preamble)	Data Field Length	Codec ID	Number of Data 1	AVL Data	Number of Data 2	CRC-16
4 bytes	4 bytes	1 byte	1 byte	X bytes	1 byte	4 bytes

Preamble – the packet starts with four zero bytes.
Data Field Length – size is calculated starting from Codec ID to Number of Data 2.
Codec ID – in Codec16 it is always 0x10.
Number of Data 1 – a number that defines how many records are in the packet.
AVL Data – actual data in the packet (more information below).
Number of Data 2 – a number that defines how many records are in the packet. This number must be the same as “Number of Data 1”.
CRC-16 – calculated from Codec ID to the Second Number of Data. CRC (Cyclic Redundancy Check) is an error-detecting code used to detect accidental changes to RAW data. For calculation we are using CRC-16/IBM.

Note: for FMB630 and FM63XY, the minimum AVL record size is 45 bytes (all IO elements disabled). The maximum AVL record size is 255 bytes.

- AVL Data

The below table represents the AVL Data structure:

Timestamp	Priority	GPS Element	IO Element
8 bytes	1 byte	15 bytes	X bytes

Timestamp – a difference, in milliseconds, between the current time and midnight, January 1970 UTC (UNIX time).
Priority – a field that defines AVL data priority (more information below).

GPS Element – location information of the AVL data (more information below).
IO Element – additional configurable information from the device (more information below).

- Priority

The below table represents Priority values. Packet priority depends on device configuration and records sent.

Priority	
0	Low
1	High
2	Panic

- GPS element

The below table represents the GPS Element structure:

Longitude	Latitude	Altitude	Angle	Satellites	Speed
4 bytes	4 bytes	2 bytes	2 bytes	1 byte	2 bytes

Longitude – east-west position.
Latitude – north-south position.
Altitude – meters above sea level.
Angle – degrees from north pole.
Satellites – number of satellites in use.
Speed – speed calculated from satellites.

Note: Speed will be `0x0000` if GPS data is invalid.

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

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

Where:
d – Degrees; m – Minutes; s – Seconds; ms – Milliseconds; p – Precision (10000000)
If the longitude is in the west or latitude in the south, multiply the result by –1.

Note:
To determine if the coordinate is negative, convert it to binary format and check the very first bit. If it is `0`, the coordinate is positive, if it is `1`, the 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 complement arithmetic.

- IO Element

Event IO ID	2 bytes
Generation Type	1 byte
N of Total IO	1 byte
N1 of One Byte IO	1 byte
1’st IO ID	2 bytes
1’st IO Value	1 byte
…	
N1’th IO ID	2 bytes
N1’th IO Value	1 byte
N2 of Two Bytes	1 byte
1’st IO ID	2 bytes
1’st IO Value	2 bytes
…	
N2’th IO ID	2 bytes
N2’th IO Value	2 bytes
N4 of Four Bytes	1 byte
1’st IO ID	2 bytes
1’st IO Value	4 bytes
…	
N4’th IO ID	2 bytes
N4’th IO Value	4 byte
N8 of Eight Bytes	1 byte
1’st IO ID	2 bytes
1’st IO Value	8 byte
…	
N8’IO ID	2 bytes
N8’IO Value	8 bytes

Event IO ID – if data is acquired on the event – this field defines which IO property has changed and generated an event. For example, when if the Ignition state changes and it generates an event, the Event IO ID will be 0xEF (AVL ID: 239). If it’s not an eventual record – the value is 0.
Generation type - data event generation type. More information about it you can find here.
N – a 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.
N’th IO ID - AVL ID.
N’th IO Value - AVL ID value.

- Generation type

Value	Record Created
0	On Exit
1	On Entrance
2	On Both
3	Reserved
4	Hysteresis
5	On Change

6	Eventual
7	Periodical

- **Communication with server**

Communication with the server is the same as with Codec8 protocol, except in Codec16 protocol Codec ID is `0x10` and has generation type.

- Example:

The module connects to the server and sends IMEI:
`000F333536333037303432343431303133`
The server accepts the module:
`01`
The module sends data packet:

AVL Data Packet Header	AVL Data Array	CRC-16
Four Zero Bytes – 0x00000000, "AVL Data Array" length – 0x000000FE	Codec ID – 0x10, Number of Data – **0x02** (Encoded using continuous bit stream. The last byte is padded to align to the byte boundary)	CRC of "AVL Data Array"
00000000000000FE	10**02**...(data elements)...**02**	00008612

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

- **Example**

The hexadecimal stream of AVL Data Packet receiving and response in this example is given in the hexadecimal form. The different fields of the packet are separated into different table columns for better readability and some of them are converted to ASCII values for better understanding.

Received data in the hexadecimal stream:
`000000000000005F10020000016BDBC7833000000000000000000000000000000000B05040200010000030002000B00270042563A00000000016BDBC78718`
`00000000000000000000000000000000B05040200010000030002000B00260042563A00000200005FB3`

Parsed data:

AVL Data Packet	
AVL Data Packet Part	**HEX Code Part**
Zero Bytes	00 00 00 00
Data Field Length	00 00 00 5F
Codec ID	10
Number of Data 1 (Records)	02

AVL Data (1'st record)	Timestamp	00 00 01 6B DB C7 83 30 (GMT: Wednesday, July 10, 2019, 12:06:54 PM)
	Priority	01
	Longitude	00 00 00 00
	Latitude	00 00 00 00
	Altitude	00 00
	Angle	00 00
	Satellites	00
	Speed	00 00
	Event IO ID	00 0B
	Generation Type	05
	N of Total ID	04
	N1 of One Byte IO	02
	1’st IO ID	00 01 (AVL ID: 1, Name: DIN1)
	1’st IO Value	00
	2’nd IO ID	00 03 (AVL ID: 3, Name: DIN3)
	2’nd IO Value	00
	N2 of Two Bytes IO	02
	1’st IO ID	00 0B (AVL ID: 11, Name: ICCID1)
	1’st IO Value	00 27
	2’nd IO ID	00 42 (AVL ID: 66, Name: External Voltage)
	2’nd IO Value	56 3A
	N4 of Four Bytes IO	00
	N8 of Eight Bytes IO	00

AVL Data (2'nd record)	Timestamp	00 00 01 6B DB C7 87 18 (GMT: Wednesday, July 10, 2019, 12:06:55 PM)
	Priority	01
	Longitude	00 00 00 00
	Latitude	00 00 00 00
	Altitude	00 00
	Angle	00 00
	Satellites	00
	Speed	00 00
	Event IO ID	00 0B
	Generation Type	05
	N of Total ID	04
	N1 of One Byte IO	02
	1’st IO ID	00 01 (AVL ID: 1, Name: DIN1)
	1’st IO Value	00
	2’nd IO ID	00 03 (AVL ID: 3, Name: DIN3)
	2’nd IO Value	00
	N2 of Two Bytes IO	02
	1’st IO ID	00 0B (AVL ID: 11, Name: ICCID1)
	1’st IO Value	00 26
	2’nd IO ID	00 42 (AVL ID: 66, Name: External Voltage)
	2’nd IO Value	56 3A
	N4 of Four Bytes IO	00
	N8 of Eight Bytes IO	00
	Number of Data 2 (Number of Total Records)	02
	CRC-16	00 00 5F B3

Server response: `00000002`

- **Codec16 protocol sending over UDP**
- **UDP channel protocol**

AVL data packet is the same as with Codec8, except Codec ID is changed to `0x10`. AVL Data encoding is performed according to the Codec16 protocol.

- **Communication with server**

The module sends the UDP channel packet with an encapsulated AVL data packet. The server sends the UDP channel packet with an encapsulated response module that validates the AVL Packet ID and the Number of accepted AVL elements. If the server responds with a valid AVL Packet ID that is not received within configured timeout, the module can retry sending.

- Example:

The module sends the data:

UDP Channel Header	AVL Packet Header	AVL Data Array
Length – 0x00FE, Packet ID – 0xCAFE Not Usable Byte – 0x01	AVL Packet ID – 0xDD, IMEI Length – 0x000F IMEI – 0x313233343536373839303132333435 (Encoded using continuous bit stream. The last byte is padded to align to the byte boundary)	Codec ID – 0x10, Number of Data – 0x02 (Encoded using continuous bit stream)
00FECAFE01	DD000F313334353637383903132333435	1002...(data elements)...02

The server must respond with an acknowledgment:

UDP Channel Header	AVL Packet Acknowledgment
Length – 0x0005, Packet ID – 0xCAFE, Not Usable Byte – 0x01	AVL Packet ID – 0xDD, Number of Accepted Data – 0x02
0005CAFE01	DD02

- **Example**

The hexadecimal stream of AVL Data Packet receiving and response in this example is given in the hexadecimal form. The different fields of the packet are separated into different table columns for better readability and some of them are converted to ASCII values for better understanding.

Received data in the hexadecimal stream:
`015BCAFE0101000F333532303934303835323331353932100700000151 17E40FE800000000000000000000000000000000000EF050504000100000300 00B4000 0EF01010042111A000001`

Parsed:

AVL Data Packet

AVL Data Packet Part		HEX Code Part
UDP Channel Header	Length	01 5B
	Packet ID	CA FE
	Not usable byte	01
AVL Packet Header	AVL packet ID	07
	IMEI Length	00 0F
	IMEI	33 35 32 30 39 34 30 38 35 32 33 31 35 39 32

AVL Data Array	Codec ID	10
	Number of Data 1 (Records)	01
	Timestamp	00 00 01 51 17 E4 0F E8 (GMT: Wednesday, November 18, 2015, 12:00:01 AM)
	Priority	00
	Longitude	00 00 00 00
	Latitude	00 00 00 00
	Altitude	00 00
	Angle	00 00
	Satellites	00
	Speed	00 00
	Event IO ID	00 EF
	Generation type	05
	N of Total ID	05
	N1 of One Byte IO	04
	1’st IO ID	00 01 (AVL ID: 1, Name: DIN1)
	1’st IO Value	00
	2’nd IO ID	00 03 (AVL ID: 3, Name: DIN3)
	2’nd IO Value	00
	3’rd IO ID	00 B4 (AVL ID: 180, Name: DOUT2)
	3’rd IO Value	00
	4’th IO ID	00 EF (AVL ID: 239, Name: Ignition)
	4’th IO Value	00
	N2 of Two Bytes IO	01
	1’st IO ID	42 (AVL ID: 66, Name: External Voltage)
	1’st IO Value	11 1A
	N4 of Four Bytes IO	00
	N8 of Eight Bytes IO	00
	Number of Data 2 (Number of Total Records)	01

The server response in the hexadecimal stream: `0005CAFE010701`

Parsed:

Server Response to AVL Data Packet

Server Response Part		HEX Code Part
UDP Channel Header	Length	00 05
	Packet ID	CA FE
AVL Packet Acknowledgment	Not usable byte	01
	AVL packet ID	07
	Number of Accepted Data	01

Differences between Codec 8, Codec 8 Extended and Codec 16

In the table below you will see differences between Codec8, Codec8 Extended, and Codec16.

	Codec8	Codec8 Extended	Codec16
Codec ID	0x08	0x8E	0x10
AVL Data IO element length	1 byte	2 bytes	2 bytes
AVL Data IO element total IO count length	1 byte	2 bytes	2 bytes
Generation Type	Is Using	Not Using	Is Using
AVL Data IO element IO count length	1 byte	2 bytes	1 byte
AVL Data IO element AVL ID length	1 byte	2 bytes	2 bytes
Variable size IO elements	Does not include	Includes variable size elements	Does not include

Codec for communication over GPRS messages

In this chapter, you will find information about every Codec protocol which are used for communication over GPRS messages and the differences between them.

Codec 12

- **About Codec12**

Codec12 is the original and main Teltonika protocol for device-server communication over GPRS messages. Codec12 GPRS commands can be used for sending configuration, debug, digital outputs control commands, or other (special purpose commands on special firmware versions). This protocol is also necessary for using FMB630/FM6300/FM5300/FM5500/FM4200 features like Garmin, LCD communication, and COM TCP Link Mode.

• GPRS command session

The following figure shows how the GRPS command session is started over TCP.
First, the Teltonika device opens the GPRS session and sends AVL data to the server (refer to device protocols). Once all records are sent and the correct sent data array acknowledgment is received by the device then GPRS commands in Hex can be sent to the device.
The ACK (acknowledgment of IMEI from server) is a one-byte constant 0x01. The acknowledgment of each data array send from the device is four bytes integer – a number of received records.
Note, that the GPRS session should remain active between the device and server, while GPRS commands are sent. For this reason, active datalink timeout (global parameters in device configuration) is recommended to be set to 259200 (maximum value).

• General Codec12 message structure

The following diagram shows the basic structure of Codec12 messages.

Command message structure:

0x00000000 (Preamble)	Data Size	Codec ID	Command Quantity 1	Type (0x05)	Command Size	Command	Command Quantity 2	CRC-16
4 bytes	4 bytes	1 byte	1 byte	1 byte	4 bytes	X bytes	1 byte	4 bytes

Response message structure:

0x00000000 (Preamble)	Data Size	Codec ID	Response Quantity 1	Type (0x06)	Response Size	Response	Response Quantity 2	CRC-16
4 bytes	4 bytes	1 byte	1 byte	1 byte	4 bytes	X bytes	1 byte	4 bytes

Preamble - the packet starts with four zero bytes.
Data Size - size is calculated from the Codec ID field to the second command or response quantity field.
Codec ID - in Codec12 it is always `0x0C`.
Command/Response Quantity 1 - it is ignored when parsing the message.
Type - it can be 0x05 to denote command or 0x06 to denote response.
Command/Response Size – command or response length.
Command/Response – command or response in HEX.
Command/Response Quantity 2 - a byte that defines how many records (commands or responses) are in the packet. This byte will not be parsed but it's recommended that it should contain the same value as Command/Response Quantity 1.
CRC-16 – calculated from Codec ID to the Command Quantity 2. CRC (Cyclic Redundancy Check) is an error-detecting code used to detect accidental changes to RAW data. For calculation we are using CRC-16/IBM.

Note that the difference between commands and responses is the message type field: `0x05` means command and `0x06` means response.

• Command coding table

Command has to be converted from ASCII characters (char) to hexadecimal (HEX):

- **Command parsing example**

The hexadecimal stream of command and answer in this example is given in the hexadecimal form. The different fields of the message are separated into different table columns for better readability and understanding.

- **GPRS commands examples**

The hexadecimal stream of GPRS command and answer in these examples are given in the hexadecimal form. The different fields of messages are separated into different table columns for better readability and some of them are converted to ASCII values for better understanding.

1'st example: Sending *getinfo* SMS command via GPRS Codec12

Server request in the hexadecimal stream:
`000000000000000F0C010500000007676574696E666F0100004312`

Parsed:

Server Command

Server Command Part	HEX Code Part
Zero Bytes	00 00 00 00
Data Size	00 00 00 0F
Codec ID	0C
Command Quantity 1	01
Command Type	05
Command Size	00 00 00 07
Command	67 65 74 69 6E 66 6F
Command Quantity 2	01
CRC-16	00 00 43 12

Note that Server Command converted from HEX to ASCII means *getinfo*

Device response in the hexadecimal stream:
`00000000000000900C010600000088494E493A323031392F372F323220373A3232205254433A323031392F372F323220373A3533205253543A32204552523A312053523A302042523A302043463A302046473A3020464C3A302054553A302F302055543A3020534D533A30204E4F4750533A303A3330204750533A31205341543A302052533A332052463A36352053463A31204D443A30010000C78F`

Parsed:

Device Answer

Device Answer Part	HEX Code Part
Zero Bytes	00 00 00 00
Data Size	00 00 00 90
Codec ID	0C
Response Quantity 1	01

Response Type	06
Response Size	00 00 00 88
Response	49 4E 49 3A 32 30 31 39 2F 37 2F 32 32 20 37 3A 32 32 20 52 54 43 3A 32 30 31 39 2F 37 2F 32 32 20 37 3A 35 33 20 52 53 54 3A 32 20 45 52 52 3A 31 20 53 52 3A 30 20 42 52 3A 30 20 43 46 3A 30 20 46 47 3A 30 20 46 4C 3A 30 20 54 55 3A 30 2F 30 20 55 54 3A 30 20 53 4D 53 3A 30 20 4E 4F 47 50 53 3A 30 3A 33 30 20 47 50 53 3A 31 20 53 41 54 3A 30 20 52 53 3A 33 20 52 46 3A 36 35 20 53 46 3A 31 20 4D 44 3A 30
Response Quantity 2	01
CRC-16	00 00 C7 8F

Note that Device Response converted from HEX to ASCII means:
INI:2019/7/22 7:22 RTC:2019/7/22 7:53 RST:2 ERR:1 SR:0 BR:0 CF:0 FG:0 FL:0 TU:0/0 UT:0 SMS:0 NOGPS:0:30 GPS:1 SAT:0 RS:3 RF:65 SF:1 MD:0

2'nd example: Sending *getio* SMS command via GPRS Codec12

Server request in the hexadecimal stream:
`000000000000000D0C010500000005676574696F01000000CB`

Parsed:

Server Command

Server Command Part	HEX Code Part
Zero Bytes	00 00 00 00
Data Size	00 00 00 0D
Codec ID	0C
Command Quantity 1	01
Command Type	05
Command Size	00 00 00 05
Command	67 65 74 69 6F
Command Quantity 2	01
CRC-16	00 00 00 CB

Note that Server Command converted from HEX to ASCII means *getio*

Device response in the hexadecimal stream:
`00000000000000370C01060000002F4449313A31204449323A30204449333A302041494E313A302041494E323A313639323420444F313A3020444F323A3101000066E3`

Parsed:

Device Answer

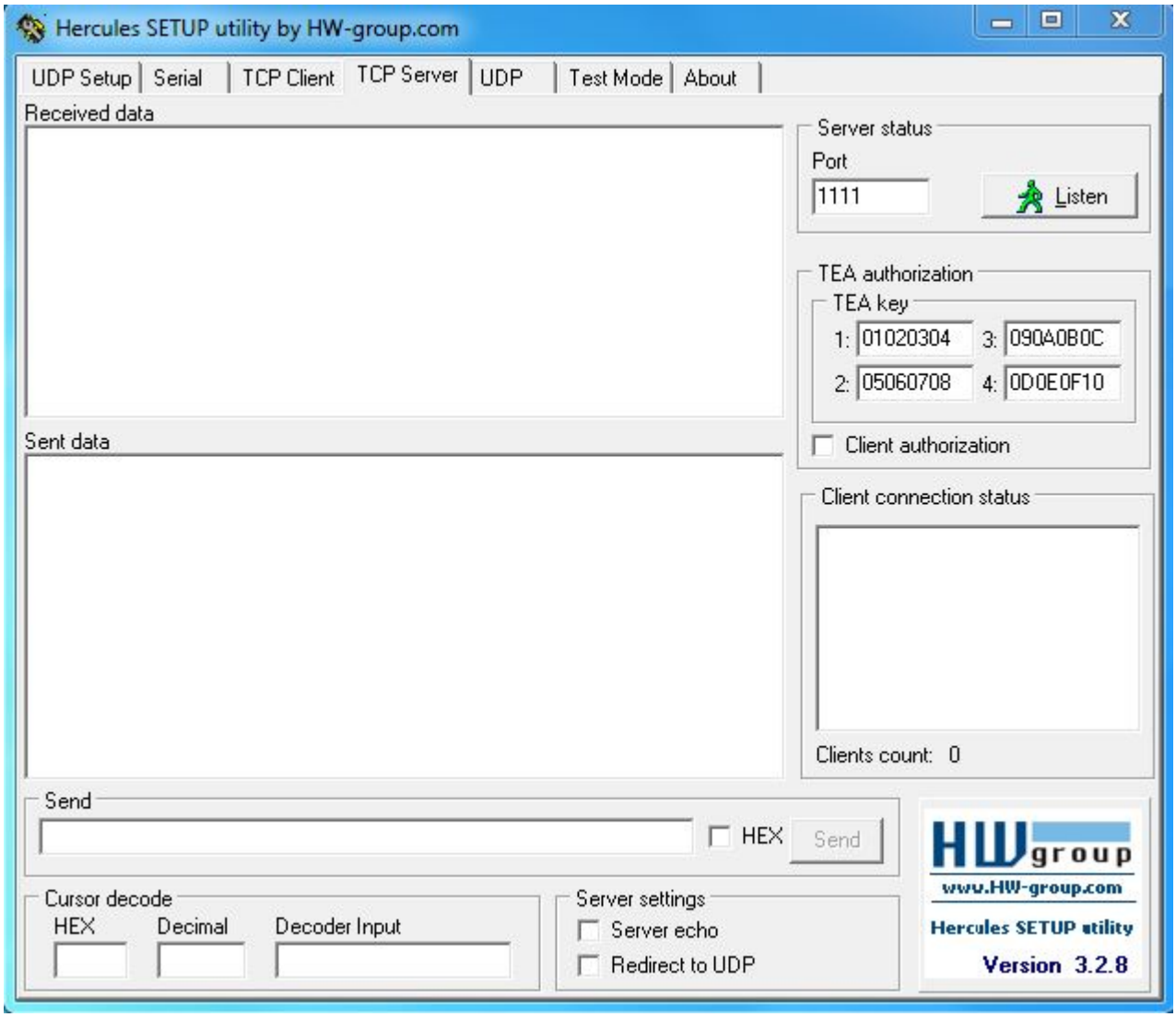
Device Answer Part	HEX Code Part
Zero Bytes	00 00 00 00

Data Size	00 00 00 37
Codec ID	0C
Response Quantity 1	01
Response Type	06
Response Size	00 00 00 2F
Response	44 49 31 3A 31 20 44 49 32 3A 30 20 44 49 33 3A 30 20 41 49 4E 31 3A 30 20 41 49 4E 32 3A 31 36 39 32 34 20 44 4F 31 3A 30 20 44 4F 32 3A 31
Response Quantity 2	01
CRC-16	00 00 66 E3

Note that Device Response converted from HEX to ASCII means:
DI1:1 DI2:0 DI3:0 AIN1:0 AIN2:16924 DO1:0 DO2:1

- **Communication with server**

The GSM/GPRS commands can be sent from a terminal program. We recommend using Hercules (in TCP server mode). Simply write the command into the Hercules Send field, check the HEX box and click Send button. Note that the TCP server must be listening on a specified port (see Port field and Listen button below).



- **FMXX and Codec12 functionality**
- **Garmin**

All information is provided in the “FMXX and Garmin development.pdf” document.

- **COM TCP Link Mode**

All information is provided in the “FMxx TCP Link mode test instructions.pdf” document.

Codec 13

- **About Codec13**

Codec13 is the original Teltonika protocol for device-server communication over GPRS messages and it is based on the Codec12 protocol. The main differences of Codec13 are that timestamp is used in messages and communication is one way only (Codec13 is used for Device -> Server sending).

- **General Codec13 message structure**

The following diagram shows the basic structure of Codec 13 messages:

0x00000000 (Preamble)	Data Size	Codec ID	Command Quantity 1	Type	Command Size	Timestamp	Command	Command Quantity 2	CRC-16
4 bytes	4 bytes	1 byte	1 byte	1 byte	4 bytes	4 bytes	X bytes	1 byte	4 bytes

Preamble – the packet starts with a preamble field (four zero bytes).
Data Size – size is calculated from the Codec ID field to the second Command Quantity field.
Codec ID – in Codec13 it is always `0x0D`.
Command Quantity 1 – `0x01`, it is ignored when parsing the message.
Command Type – it is always `0x06` since the packet is direction is FM->Server.
Command Size – command size field includes the size of the timestamp too, so it is equal to the size of the payload + the size of the timestamp.
Timestamp – a difference, in milliseconds, between the current time and midnight, January 1970 UTC (UNIX time).
Command – actual received data.
Command Quantity 2 – a byte that defines how many records (commands) are in the packet. This byte will not be parsed but it’s recommended that it should contain the same value as Command/Response Quantity 1.
CRC-16 – calculated from Codec ID to the Second Number of Data. CRC (Cyclic Redundancy Check) is an error-detecting code used to detect accidental changes to RAW data. For calculation we are using CRC-16/IBM.

Note: Codec13 packets are used only when the “Message Timestamp” parameter in RS232 settings is enabled.

Codec 14

- **About Codec14**

Codec14 is the original Teltonika protocol for device-server communication over GPRS messages

and it is based on the Codec12 protocol.
The main difference of Codec14 is that the device will answer the GPRS command if the device's physical IMEI number matches the specified IMEI number in the GPRS command.

Codec14 GPRS commands can be used for sending configuration, debug, digital outputs control commands, or other (special purpose commands on special firmware versions).

- ## FMB firmware requirements

Implemented in base firmware from FMB.Ver.03.25.04.Rev.00 and newer.

- ## General Codec14 message structure

The following diagram shows the basic structure of Codec14 messages.

Command message structure

0x00000000 (preamble)	Data size	0x0E (Codec ID)	Command quantity	0x05 (Message type)	Command size + IMEI size (8 bytes)	IMEI (HEX)	Command	Command quantity	CRC-16
4 bytes	4 bytes	1 bytes	1 bytes	1 bytes	4 bytes	8 bytes	X bytes	1 bytes	4 bytes

Response message structure

0x00000000 (preamble)	Data size	0x0E (Codec ID)	Response quantity	0x06 / 0x11 (Message type)	Response size + IMEI size (8 bytes)	IMEI (HEX)	Response	Response quantity	CRC-16
4 bytes	4 bytes	1 bytes	1 bytes	1 bytes	4 bytes	8 bytes	X bytes	1 bytes	4 bytes

Preamble – the packet starts with four zero bytes.
Data Size – size is calculated from the Codec ID field to the second command or response quantity field.
Codec ID – in Codec14 it is always `0x0E`.
Command/Response Quantity 1 – it is ignored when parsing the message.
Type – if it is a request command from the server it has to contain 0x05. The response type field will contain `0x06` if it's ACK or `0x11` if it's nACK.
Explanation: If command message IMEI is equal to actual device IMEI, received command will be executed and response will be sent with ACK (`0x06`) message type field value. If the command message IMEI doesn't match the actual device IMEI, the received command won't be executed and a response to the server will be sent with nACK (`0x11`) message type field value.
Command/Response Size – command or response length.
Note: make sure that size is IMEI size 8 + actual command size. The minimal value is 8 because Codec14 always contains IMEI and it's 8 bytes.
IMEI (HEX) – it is send as HEX value. For example, if the device IMEI is 123456789123456 then the IMEI data field will contain `0x0123456789123456` value.
Command/Response – command or response in HEX.
Command/Response Quantity 2 - a byte that defines how many records (commands or responses) are in the packet. This byte will not be parsed but it's recommended that it should contain the same value as Command/Response Quantity 1.
CRC-16 – calculated from Codec ID to the Second Number of Data. CRC (Cyclic Redundancy Check)

is an error-detecting code used to detect accidental changes to RAW data. For calculation we are using CRC-16/IBM.

- **GPRS in Codec14 examples**

The hexadecimal stream of the GPRS command and answer in this example is given in the hexadecimal form. The different fields of the message are separated into different table columns for better readability and some of them are converted to ASCII values for better understanding.

Sending *getver* SMS command via GPRS Codec14:

Server requests in Hexadecimal stream:
`00000000000000160E01050000000E035209308145225167657476657201000D2C1`

Parsed:

Server Command

Server Command Part	HEX Code Part
Zero Bytes	00 00 00 00
Data Size	00 00 00 16
Codec ID	0E
Command Quantity 1	01
Command Type	05
Command Size	00 00 00 0E
IMEI	03 52 09 30 81 45 22 51
Command	67 65 74 76 65 72
Command Quantity 2	01
CRC-16	00 00 D2 C1

Note that Server Command converted from HEX to ASCII means *getver*

Device ACK response in the hexadecimal stream:
`00000000000000AB0E0106000000A30352093081452251566572 3A30332E31382E31345F3034204750533A41584E5F352E31305F333333332048773A464D42313230204D6F643A313520494D45493A333532303933303831343532323531204 96E69743A323031382D31312D323220373A313320557074696D653A3137323334204D4143 3A3630424444303031363236312053043 3A31283029204 1584C3A30204F42443A3020424C3A312E362042543A340100007AAE`

Parsed:

Device Answer

Device Answer Part	HEX Code Part
Zero Bytes	00 00 00 00
Data Size	00 00 00 37
Codec ID	0E
Response Quantity 1	01
Response Type	06

Response Size	00 00 00 A3
IMEI	03 52 09 30 81 45 22 51
Response	56 65 72 3A 30 33 2E 31 38 2E 31 34 5F 30 34 20 47 50 53 3A 41 58 4E 5F 35 2E 31 30 5F 33 33 33 33 20 48 77 3A 46 4D 42 31 32 30 20 4D 6F 64 3A 31 35 20 49 4D 45 49 3A 33 35 32 30 39 33 30 38 31 34 35 32 32 35 31 20 49 6E 69 74 3A 32 30 31 38 2D 31 31 2D 32 32 20 37 3A 31 33 20 55 70 74 69 6D 65 3A 31 37 32 33 34 20 4D 41 43 3A 36 30 42 44 44 30 30 31 36 32 36 31 20 53 50 43 3A 31 28 30 29 20 41 58 4C 3A 30 20 4F 42 44 3A 30 20 42 4C 3A 31 2E 36 20 42 54 3A 34
Response Quantity 2	01
CRC-16	00 00 7A AE

Note that Device Response converted from HEX to ASCII means:
Ver:03.18.14_04 GPS:AXN_5.10_3333 Hw:FMB120 Mod:15 IMEI:352093081452251 Init:2018-11-22 7:13 Uptime:17234 MAC:60BDD0016261 SPC:1(0) AXL:0 OBD:0 BL:1.6 BT:4

Device nACK response in the hexadecimal stream:
`000000000000000100E01110000000803520930814524680100003 2AC`

Parsed:

Device Answer

Device Answer Part	HEX Code Part
Zero Bytes	00 00 00 00
Data Size	00 00 00 10
Codec ID	0E
Response Quantity 1	01
Response Type	11
Response Size	00 00 00 08
IMEI	03 52 09 30 81 45 24 68
Response Quantity 2	01
CRC-16	00 00 32 AC

Codec 15

- **Protocol Overview**

Codec 15 relies on the Codec12 protocol and is employed when both message timestamp and device IMEI are enabled. It serves as the original Teltonika protocol for communication from the device to the server via GPRS messages. This protocol is exclusively applicable to FMX6 professional devices.

Codec15 is available in RS232 modes:

1. TCP/UDP Ascii
2. TCP/UDP Binary
3. TCP/UDP Ascii Buffered
4. TCP/UDP Binary Buffered.

- ## Structure of Codec 15 messages

0x00000000 (Preamble)	Data Size	0x0F (Codec ID)	Command quantity	Message type	Command size + timestamp + imei	Times tamp	IMEI	Command	Command quantity	CRC - 16
4 bytes	4 bytes	1 byte	1 byte	1 bytes	4 byte	4 bytes	8 bytes	X bytes	1 bytes	4 bytes

- ## Structure explanation

Preamble - four zero bytes.
Data size - size is calculated from codec id(0x0F) field to the second command quantity field.
Codec ID - in Codec 15 it is always 0x0F.
Command quantity - a number which defines how many commands are in the packet.
Message type - this value is configurable in RS232 settings box.
Command size + Timestamp + IMEI - it is equal to size of payload + size of timestamp + size of imei.
Timestamp – data record creation time in seconds(Unix timestamp).
IMEI - send as HEX value. Example if device IMEI is 123456789123456 then IMEI data field will contain 0x0123456789123456 value.
Command field - actual received data.
Command quantity - a number which defines how many commands are in the packet.
CRC field - calculated from Codec ID to the Second Number of Data.

- ## Codec 15 examples

Device sends message „Hello\n“ via GPRS Codec15:

```
000000000000001b0f010b00000013654b65a4012345678912345648656c6c6f210a01000093d6
```

Parsed command

Command Part	HEX Code Part
Zero Bytes	00 00 00 00
Data Size	00 00 00 1B
Codec ID	0F
Quantity of commands	01
Command type	0B
Command Size	00 00 00 13
Timestamp	65 4B 65 A4
IMEI	01 23 45 67 89 12 34 56
Command	48 65 6c 6c 6f 21 0a
Quantity of commands	01
CRC-16	00 00 93 D6

CRC: 0x 000093d6 The algorithm to calculate CRC is CRC-16 (also known as CRC-16-IBM). All the fields from codec ID to second command/response quantity field are used to calculate CRC.

Differences between Codec 12, Codec 13, Codec 14 and Codec 15

In the table below you will see differences between Codec12, Codec13, Codec14 and Codec 15.

	Codec12	Codec13	Codec14	Codec15
Communication	Server - Device Communication	One-way (Device -> Server communication)	Server - Device Communication	One-way (Device -> Server communication)
Codec ID	0x0C	0x0D	0x0E	0x0F
Response Message Type	0x06	-	0x06 (if it is ACK) or 0x11 (if it is nACK)	-
Command / Response size	Only Command/Response	Only Command	Command/Response + IMEI	Command + IMEI
Timestamp	Not Using	Is Using	Not Using	Is Using
IMEI	Not Using	Not Using	Is Using	Is Using

24 Position SMS Data Protocol

24-hour SMS is usually sent once every day and contains GPS data for the last 24 hours. The 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 uses standard AVL data protocol.

- **Encoding**

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

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

Bits in a byte are numbered starting from the 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 – is the most significant bit.

- **Structure**

Below in the tables, you will see the SMS Data Structure:

SMS Data Structure		
8	Codec ID	Codec ID = 4 (0x04)

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

SMS Data Structure

		GPSDataElement	GPS data elements
ElementCount *		Byte - align padding	Padding bits to align to 8 - bits boundary represented in seconds elapsed from 2000.01.01 00:00 EET.
	64	IMEI	IMEI of sending device as 8 byte long integer

The time of only the first GPS data element is specified in the Timestamp field. The time corresponding to each further element can be computed as elementTime = Timestamp + (1 hour * elementNumber).

GPS Data Element

		Size (bits)	Field	Description
ValidElement == 1		1	ValidElement	ValidElement = 1 – there is a valid Gps Data Element following, ValidElement = 0 – no element at this position
		1	DifferentialCoords	Format of following data
	DifferentialCoords == 1	14	LongitudeDiff	Difference from previous element‘s longitude. LongitudeDiff = prevLongitude – Longitude + 213 – 1
		14	LatitudeDiff	Difference from previous element‘s latitude LatitudeDiff = prevLatitude – Latitude + 213 – 1
	DifferentialCoords == 0	21	Longitude	Longitude = {(LongDegMult + 18 * 108) * (221 – 1)} over {36*108}
		20	Latitude	Latitude = (LatDegMult + 9*108) * (220 – 1) over {18*108}
		8	Speed	Speed in km/h

Longitude - longitude field value of GPSDataElement
Latitude - latitude field value of GPSDataElement
LongDegMult - longitude in degrees multiplied by 107 (integer part)
LatDegMult - latitude in degrees multiplied by 107 (integer part)
prevLongitude - longitude field value of previous GPSDataElemen
prevLatitude - latitude field value of previous GPSDataElement

- **Decoding GPS position**

When decoding GPS data with DifferentialCoords = 1, Latitude and Longitude values can be computed as follows: Longitude = prevLongitude – LongitudeDiff + 213 – 1, Latitude = prevLatitude – LatitudeDiff + 213 – 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:

$$LongDeg = \frac{Longitude * 360}{2^{21} - 1} - 180 \qquad LatDeg = \frac{Latitude * 180}{2^{20} - 1} - 90$$

- **SMS Events**

When configured to generate an SMS event user will get this SMS upon event:
<Year/Month/Day> <Hour:Minute:Second> P:<profile_nr> <SMS Text> Val:<Event Value> Lon:<longitude> Lat:<latitude> Q:<HDOP>

Example:
2016./04/11 12:00:00 P:3 Digital Input 1 Val:1 Lon:51.12258 Lat: 25.7461 Q:0.6

Sending data using SMS

This type of data sent is used for FMBXXX devices which can be configured in SMS Data Sending settings.

- **Data sending via SMS**

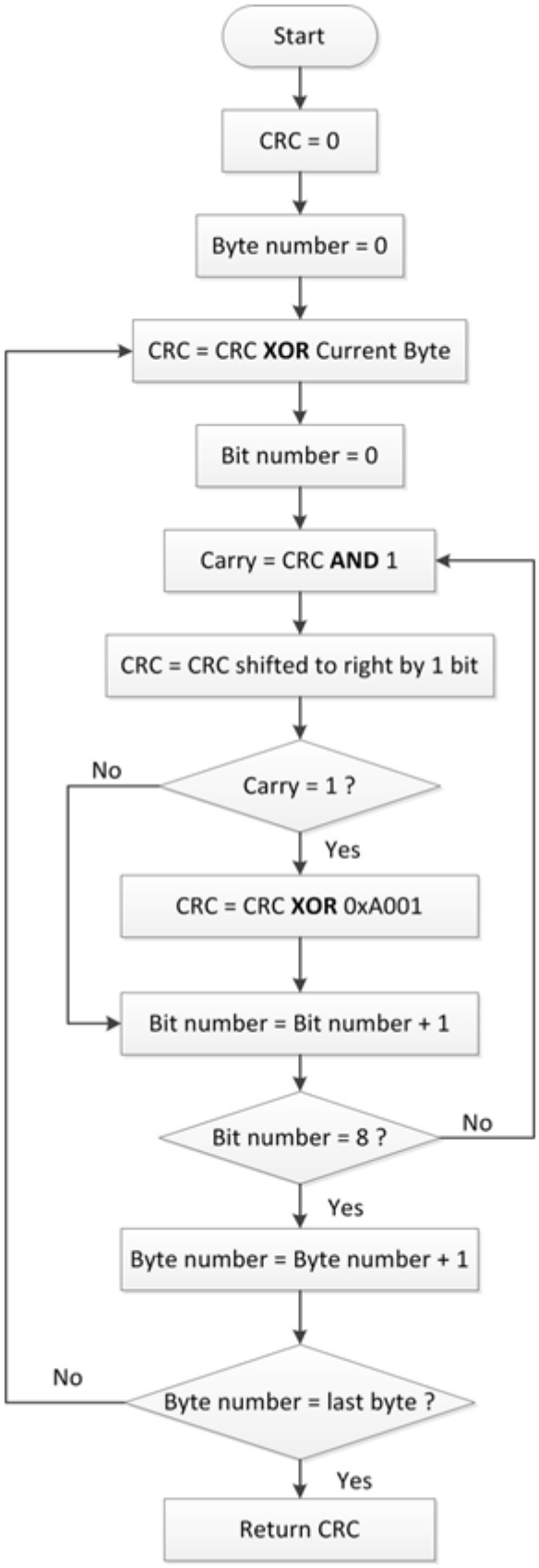
AVL data or events can be sent encapsulated in binary SMS. The TP-DCS field of these SMS should indicate that message contains 8-bit data (for example TP-DCS can be `0x04`).

SMS data (TP-UD)	
AVL data array	**IMEI**
X bytes	8 bytes

AVL data array – array of encoded AVL data.
IMEI – IMEI of sending module encoded as a big-endian 8-byte long number.

CRC-16

CRC (Cyclic Redundancy Check) is an error-detecting code used to detect accidental changes to RAW data. The algorithm on how to calculate CRC-16 (also known as CRC-16/IBM) you will find below.


Start
CRC = 0
Byte number = 0
CRC = CRC XOR Current Byte
Bit number = 0
Carry = CRC AND 1
CRC = CRC shifted to right by 1 bit
Carry = 1 ?
No
Yes
CRC = CRC XOR 0xA001
Bit number = Bit number + 1
Bit number = 8 ?
No
Yes
Byte number = Byte number + 1
Byte number = last byte ?
No
Yes
Return CRC