

# Codec

□

## Contents

- [1 Introduction](#)
- [2 Codec for device data sending](#)
  - [2.1 Codec 8](#)
  - [2.2 Codec 8 Extended](#)
  - [2.3 Codec 16](#)
  - [2.4 Differences between Codec 8, Codec 8 Extended and Codec 16](#)
- [3 Codec for communication over GPRS messages](#)
  - [3.1 Codec 12](#)
  - [3.2 Codec 13](#)
  - [3.3 Codec 14](#)
  - [3.4 Differences between Codec 12, Codec 13 and Codec 14](#)
- [4 24 Position SMS Data Protocol](#)
- [5 Sending data using SMS](#)
- [6 CRC-16](#)

## 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 ID's:

<b>Codec 8</b>	<b>Codec 8 Extended</b>	<b>Codec 16</b>	<b>Codec 12</b>	<b>Codec 13</b>	<b>Codec 14</b>
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 using for device data sending and differences between them.

# Codec 8

- **Protocol Overview**

Codec8 - a main FM device protocol that is used for sending data to 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**

Below table represents AVL Data Packet structure:

<b>0x00000000 (Preamble)</b>	<b>Data Field Length</b>	<b>Codec ID</b>	<b>Number of Data 1</b>	<b>AVL Data</b>	<b>Number of Data 2</b>	<b>CRC-16</b>
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 which defines how many records is in the packet.

**AVL Data** - actual data in the packet (more information below).

**Number of Data 2** - a number which defines how many records is 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 using for detect accidental changes to RAW data. For calculation we are using [CRC-16/IBM](#).

**Note:** for [FMB630](#), [FMB640](#) and [FM63XY](#), minimum AVL record size is 45 bytes (all IO elements disabled). Maximum AVL record size is 255 bytes. Maximum AVL packet size is 512 bytes. For other devices, minimum AVL record size is 45 bytes (all IO elements disabled). Maximum AVL packet size is 1280 bytes.

- AVL Data

Below table represents AVL Data structure.

<b>Timestamp</b>	<b>Priority</b>	<b>GPS Element</b>	<b>IO Element</b>
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** - field which define AVL data priority (more information below).

**GPS Element** - location information of the AVL data (more information below).

**IO Element** - additional configurable information from device (more information below).

- Priority

Below table represents Priority values. Packet priority depends on device configuration and records

sent.

### Priority

<b>0</b>	Low
<b>1</b>	High
<b>2</b>	Panic

- GPS element

Below table represents GPS Element structure:

<b>Longitude</b>	<b>Latitude</b>	<b>Altitude</b>	<b>Angle</b>	<b>Satellites</b>	<b>Speed</b>
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 visible satellites.

**Speed** - speed calculated from satellites.

**Note:** If record are 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:

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

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.

Note:

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

<b>Event IO ID</b>	1 byte
<b>N of Total IO</b>	1 byte
<b>N1 of One Byte IO</b>	1 byte
<b>1'st IO ID</b>	1 byte
<b>1'st IO Value</b>	1 byte
...	
<b>N1'th IO ID</b>	1 byte
<b>N1'th IO Value</b>	1 byte
<b>N2 of Two Bytes</b>	1 byte
<b>1'st IO ID</b>	1 byte
<b>1'st IO Value</b>	2 bytes
...	
<b>N2'th IO ID</b>	1 byte
<b>N2'th IO Value</b>	2 bytes
<b>N4 of Four Bytes</b>	1 byte
<b>1'st IO ID</b>	1 byte
<b>1'st IO Value</b>	4 bytes
...	
<b>N4'th IO ID</b>	1 byte
<b>N4'th IO Value</b>	4 bytes
<b>N8 of Eight Bytes</b>	1 byte
<b>1'st IO ID</b>	1 byte
<b>1'st IO Value</b>	8 bytes
...	
<b>N8'IO ID</b>	1 byte
<b>N8'IO Value</b>	8 bytes

**Event IO ID** - if data is acquired on event - this field defines which IO property has changed and generated an event. For example, when if Ignition state changed and it generate event, Event IO ID will be 0xEF (AVL ID: 239). If it's not 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 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 356307042441013 would be sent as 000F333536333037303432343431303133.

First two bytes denote IMEI length. In this case 0x000F 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:  
 000F333536333037303432343431303133  
 Server accepts the module:  
 01  
 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 - <b>0x02</b> (Encoded using continuous bit stream. Last byte padded to align to byte boundary)	CRC of "AVL Data Array"
00000000000000FE	08 <b>02</b> ...(data elements)... <b>02</b>	00008612

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

- **Examples**

Hexadecimal stream of AVL Data Packet receiving and response in these examples are given in hexadecimal form. The different fields of packets are separate 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 byte and 8 byte).

Received data in hexadecimal stream:  
 00000000000000003608010000016B40D8EA300100000000000000000000000000000010502150  
 3010101425E0F01F10000601A014E000000000000000000010000C7CF

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

	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)
AVL Data	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 byte).

Received data in hexadecimal stream:

000000000000002808010000016B40D9AD8001000000000000000000000000000010302150  
3010101425E100000010000F22A

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

	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
AVL Data	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 10
	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 hexadecimal stream:

```
00000000000000004308020000016B40D57B48010000000000000000000000000000000000000000000010101010
000000000000000016B40D5C1980100000000000000000000000000000000000000000000000000000000
101010101000000020000252C
```

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

	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
AVL Data (1'st record)	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
	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
AVL Data (2'nd record)	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

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

### Acknowledgment Packet

<b>Packet Length</b>	<b>Packet ID</b>	<b>Not Usable Byte</b>
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

Below table represents Sending Packet Payload structure.

### AVL data encapsulated in UDP channel packet

<b>AVL Packet ID</b>	<b>IMEI Length</b>	<b>Module IMEI</b>	<b>AVL Data Array</b>
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** - array of encoded AVL data (same as TCP AVL Data Array).

- Server response Packet Payload using UDP channel

Below table represents Server Response Packet Payload structure.

### Server Response to AVL Data Packet

<b>AVL Packet ID</b>	<b>Number of Accepted AVL Elements</b>
1 byte	1 byte

- Communication with server

Module sends UDP channel packet with encapsulated AVL data packet. Server sends UDP channel packet with encapsulated response 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.

- Example:

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. Last byte padded to align to byte boundary)	Codec ID - 0x08, Number of Data - 0x02 (Encoded using continuous bit stream)
00FECAFE01	DD000F3133343536373839303132333435	0802...(data elements)...02

Server must respond with acknowledgment:

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

- Example

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

Received data in hexadecimal stream:

003DCAFE0105000F33353230393330383634303336353508010000016B4F815B3001000000000  
00000000000000000000000000103021503010101425DBC000001

Parsed:

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

	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
AVL Data Array	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 Eight Bytes IO	00
	Number of Data 2 (Number of Total Records)	01

Server response in hexadecimal stream: 0005CAFE010501

Parsed:

### Server Response to AVL Data Packet

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

## Codec 8 Extended

### • Protocols overview

Codec8 Extended is using for FMBXXX family devices. This protocol looks familiar like Codec8 but they have some differences. Main differences between are shown in below table:

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

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

Below table represents 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 which defines how many records is in the packet.

**AVL Data** - actual data in the packet (more information below).

**Number of Data 2** - a number which defines how many records is 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 using for detect accidental changes to RAW data. For calculation we are using [CRC-16/IBM](#).

**Note:** for [FMB630](#), [FMB640](#) and [FM63XY](#), minimum AVL packet size is 45 bytes (all IO elements disabled). Maximum AVL packet size is 255 bytes. For other devices, minimum AVL packet size is 45 bytes (all IO elements disabled). Maximum AVL packet size is 1280 bytes.

- AVL Data

Below table represents 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** - field which define AVL data priority (more information below).

**GPS Element** - locational information of the AVL data (more information below).

**IO Element** - additional configurable information from device (more information below).

- Priority

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

Priority	
0	Low
1	High
2	Panic

- GPS element

Below table represents 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 visible satellites.

**Speed** - speed calculated from satellites.

**Note:** If record are 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:

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

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.

Note:

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

<b>Event IO ID</b>	2 bytes
<b>N of Total IO</b>	2 bytes
<b>N1 of One Byte IO</b>	2 bytes
<b>1'st IO ID</b>	2 bytes
<b>1'st IO Value</b>	1 byte
...	
<b>N1'th IO ID</b>	2 bytes
<b>N1'th IO Value</b>	1 byte
<b>N2 of Two Bytes</b>	2 bytes
<b>1'st IO ID</b>	2 bytes
<b>1'st IO Value</b>	2 bytes
...	
<b>N2'th IO ID</b>	2 bytes
<b>N2'th IO Value</b>	2 bytes
<b>N4 of Four Bytes</b>	2 bytes
<b>1'st IO ID</b>	2 bytes
<b>1'st IO Value</b>	4 bytes
...	
<b>N4'th IO ID</b>	2 bytes
<b>N4'th IO Value</b>	4 byte
<b>N8 of Eight Bytes</b>	2 bytes
<b>1'st IO ID</b>	2 bytes
<b>1'st IO Value</b>	8 bytes
...	
<b>N8'IO ID</b>	2 bytes
<b>N8'IO Value</b>	8 bytes
<b>NX of X Byte IO</b>	2 bytes
<b>1'st IO ID</b>	2 bytes
<b>1'st IO Length</b>	2 bytes
<b>1'st IO Value</b>	Defined by length
...	
<b>NX'th IO ID</b>	2 bytes
<b>NX'th Length</b>	2 bytes
<b>NX'th Value</b>	Defined by length

**Event IO ID** - if data is acquired on event - this field defines which IO property has changed and generated an event. For example, when if Ignition state changed and it generate event, Event IO ID will be 0x00EF (AVL ID: 239). If it's not 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 length element. **N'th IO ID** - AVL ID.

**N'th Length** - AVL ID value length.

**N'th IO Value** - AVL ID value.

- **Communication with server**

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

- Example:

Module connects to server and sends IMEI:  
 000F333536333037303432343431303133  
 Server accepts the module:  
 01  
 Module sends data packet:

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

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

- **Example**

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

Received data in hexadecimal stream:  
 000000000000004A8E010000016B412CEE0001000100050  
 00100010100010011001D00010010015E2C880002000B0000000003544C87  
 A000E000000001DD7E06A00000100002994

Parsed data:

<b>AVL Data Packet</b>	
<b>AVL Data Packet Part</b>	<b>HEX Code Part</b>
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 IO	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 performed according to Codec8 Extended protocol.

- **Communication with server**

Module sends UDP channel packet with encapsulated AVL data packet. Server sends UDP channel packet with encapsulated response 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.

- Example:

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. Last byte padded to align to byte boundary)	Codec ID - 0x8E, Number of Data - 0x02 (Encoded using continuous bit stream)
00FECAFE01	DD000F3133343536373839303132333435	8E02...(data elements)...02

Server must respond with acknowledgment:

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

• **Example**

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

Received data in hexadecimal stream:

```
005FCAFE0107000F3335323039333038363430333635358E010000016B4F831C6801000000000
000000000000000000000000010005000100010100010011009D000100
10015E2C880002000B000000003544C87A000E000000001DD7E06A000001
```

Parsed:

AVL Data Packet		
AVL Data Packet Part		HEX Code Part
	Length	00 5F
UDP Channel Header	Packet ID	CA FE
	Not usable byte	01
	AVL packet ID	05
AVL Packet Header	IMEI Length	00 0F
	IMEI	33 35 32 30 39 33 30 38 36 34 30 33 36 35 35

	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
AVL Data Array	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

Server response in hexadecimal stream: 0005CAFE010701

Parsed:

### Server Response to AVL Data Packet

Server Response Part		HEX Code Part
	Length	00 05
UDP Channel Header	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 devices. This protocol looks familiar like Codec8 but they have some differences. Main differences between are shown in table below:

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

**Note:** Codec16 is supported from firmware - 00.03.xx and newer. ([FMB630/FM63XY](#)) || AVL ID's which are higher than 255 will can be used only in Codec16 protocol.

- **Codec 16 protocol sending over TCP**

- **AVL data packet**

Below table represents AVL data packet structure:

<b>0x00000000 (Preamble)</b>	<b>Data Field Length</b>	<b>Codec ID</b>	<b>Number of Data 1</b>	<b>AVL Data</b>	<b>Number of Data 2</b>	<b>CRC-16</b>
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 which defines how many records is in the packet.

**AVL Data** - actual data in the packet (more information below).

**Number of Data 2** - a number which defines how many records is 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 using for detect accidental changes to RAW data. For calculation we are using [CRC-16/IBM](#).

**Note:** for [FMB630](#) and FM63XY, minimum AVL packet size is 45 bytes (all IO elements disabled). Maximum AVL packet size is 255 bytes.

- **AVL Data**

Below table represents AVL Data structure:

<b>Timestamp</b>	<b>Priority</b>	<b>GPS Element</b>	<b>IO Element</b>
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** - field which define AVL data priority (more information below).

**GPS Element** - location information of the AVL data (more information below).

**IO Element** - additional configurable information from device (more information below).

- Priority

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

<b>Priority</b>	
<b>0</b>	Low
<b>1</b>	High
<b>2</b>	Panic

- GPS element

Below table represents GPS Element structure:

<b>Longitude</b>	<b>Latitude</b>	<b>Altitude</b>	<b>Angle</b>	<b>Satellites</b>	<b>Speed</b>
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 visible satellites.

**Speed** - speed calculated from satellites.

**Note:** If record are 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:

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

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.

Note:

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

<b>Event IO ID</b>	2 bytes
<b>Generation Type</b>	1 byte
<b>N of Total IO</b>	1 byte
<b>N1 of One Byte IO</b>	1 byte
<b>1'st IO ID</b>	2 bytes
<b>1'st IO Value</b>	1 byte
...	
<b>N1'th IO ID</b>	2 bytes
<b>N1'th IO Value</b>	1 byte
<b>N2 of Two Bytes</b>	1 byte
<b>1'st IO ID</b>	2 bytes
<b>1'st IO Value</b>	2 bytes
...	
<b>N2'th IO ID</b>	2 bytes
<b>N2'th IO Value</b>	2 bytes
<b>N4 of Four Bytes</b>	1 byte
<b>1'st IO ID</b>	2 bytes
<b>1'st IO Value</b>	4 bytes
...	
<b>N4'th IO ID</b>	2 bytes
<b>N4'th IO Value</b>	4 byte
<b>N8 of Eight Bytes</b>	1 byte
<b>1'st IO ID</b>	2 bytes
<b>1'st IO Value</b>	8 byte
...	
<b>N8'IO ID</b>	2 bytes
<b>N8'IO Value</b>	8 bytes

**Event IO ID** - if data is acquired on event - this field defines which IO property has changed and generated an event. For example, when if Ignition state changed and it generate event, Event IO ID will be 0xEF (AVL ID: 239). If it's not 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

<b>Value</b>	<b>Record Created</b>
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 server is the same as with Codec8 protocol, except in Codec16 protocol Codec ID is 0x10 and has generation type.

- Example:

Module connects to server and sends IMEI:  
000F333536333037303432343431303133  
Server accepts the module:  
01  
Module sends data packet:

AVL Data Packet Header	AVL Data Array	CRC-16
Four Zero Bytes - 0x00000000, "AVL Data Array" length - 0x000000FE 00000000000000FE	Codec ID - 0x10, Number of Data - <b>0x02</b> (Encoded using continuous bit stream. Last byte padded to align to byte boundary) 1002...(data elements)...02	CRC of "AVL Data Array" 00008612

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

- **Example**

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

Received data in hexadecimal stream:  
000000000000005F10020000016BDBC7833000B05040  
200010000030002000B00270042563A00000000016BDBC78718  
00B05040200010000030002000B00260042563A00000  
200005FB3

Parsed data:

<b>AVL Data Packet</b>	
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

	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
AVL Data	N of Total ID	04
(1'st record)	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



	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
AVL Data (2'nd record)	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 performed according to Codec16 protocol.

- **Communication with server**

Module sends UDP channel packet with encapsulated AVL data packet. Server sends UDP channel packet with encapsulated response 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.

- Example:

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. Last byte padded to align to byte boundary)	Codec ID - 0x10, Number of Data - 0x02 (Encoded using continuous bit stream)
00FECAFE01	DD000F3133343536373839303132333435	1002...(data elements)...02

Server must respond with acknowledgment:

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

• **Example**

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

Received data in hexadecimal stream:

015BCAFE0101000F33353230393430383532333135393210070000015117E40FE8000000000000  
00000000000000000000000000000000EF05050400010000030000B4000 0EF01010042111A000001

Parsed:

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

	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
AVL Data Array	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

Server response in hexadecimal stream: 0005CAFE010701

Parsed:

### Server Response to AVL Data Packet

Server Response Part		HEX Code Part
	Length	00 05
UDP Channel Header	Packet ID	CA FE
	Not usable byte	01
AVL Packet Acknowledgment	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.

	<b>Codec8</b>	<b>Codec8 Extended</b>	<b>Codec16</b>
<b>Codec ID</b>	0x08	0x8E	0x10
<b>AVL Data IO element length</b>	1 byte	2 bytes	2 bytes
<b>AVL Data IO element total IO count length</b>	1 byte	2 bytes	2 bytes
<b>Generation Type</b>	Not Using	Not Using	Is Using
<b>AVL Data IO element IO count length</b>	1 byte	2 bytes	1 byte
<b>AVL Data IO element AVL ID length</b>	1 byte	2 bytes	2 bytes
<b>Variable size IO elements</b>	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 using for communication over GPRS messages and 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 command on special firmware versions). This protocol is also necessary for using [FMB630/FM6300](#)/FM5300/FM5500/FM4200 features like: Garmin, LCD communication, COM TCP Link Mode.

## • FM firmware requirements

Supported GPRS commands on each device depending on the firmware version. For available GPRS commands on each device, please refer to the table below.

Device	SMS over GPRS via TCP	SMS over GPRS via UDP
FM36YX	Available in base firmware	Since base firmware 01.06.01
FM63YX	Available in base firmware	Since base firmware 00.02.19
FMB, FMC, FMM, FMU family devices (exclude FMB6YX, <a href="#">FMC640</a> , <a href="#">FMM640</a> )	Available in base firmware	Available in base firmware
FMB6YX	Available in base firmware	Since base firmware 00.02.19
<a href="#">FMC640</a> , <a href="#">FMM640</a>	Available in base firmware	Available in base firmware

**Note:** "SMS over GPRS" means that all standard SMS commands text can be sent to the device via GPRS in Codec12 format.

**Note:** UDP commands are sent the same exact way as TCP commands.

## • 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 device protocols). Once all records are sent and correct sent data array acknowledgment is received by device then GPRS commands in Hex can be sent to the device.

The ACK (acknowledge of IMEI from server) is a one-byte constant 0x01. The acknowledgment of each data array send from the device is four bytes integer - the number of records received.

Note, that the GPRS session should remain active between 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 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:

<b>0x00000000 (Preamble)</b>	<b>Data Size</b>	<b>Codec ID</b>	<b>Response Quantity 1</b>	<b>Type (0x06)</b>	<b>Response Size</b>	<b>Response</b>	<b>Response Quantity 2</b>	<b>CRC-16</b>
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 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 which defines how many records (commands or responses) is in the packet. This byte will not be parsed but it's recommended that it should contain 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 using for detect accidental changes to RAW data. For calculation we are using [CRC-16/IBM](#).

Note that difference between commands and responses is 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

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

### • GPRS commands examples

Hexadecimal stream of GPRS command and answer in these examples are given in hexadecimal form. The different fields of messages are separate 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 hexadecimal stream:

0000000000000000F0C010500000007676574696E666F0100004312

Parsed:

#### Server Command

<b>Server Command Part</b>	<b>HEX Code Part</b>
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 hexadecimal stream:

```
000000000000000900C010600000088494E493A323031392F372F323220373A3232205254433A3
23031392F372F323220373A3533205253543A32204552523A
312053523A302042523A302043463A302046473A3020464C3A302054553A302F302055543A302
0534D533A30204E4F4750533A303A3330204750533A312053
41543A302052533A332052463A36352053463A31204D443A30010000C78F
```

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 hexadecimal stream:

```
00000000000000D0C010500000005676574696F01000000CB
```

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 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

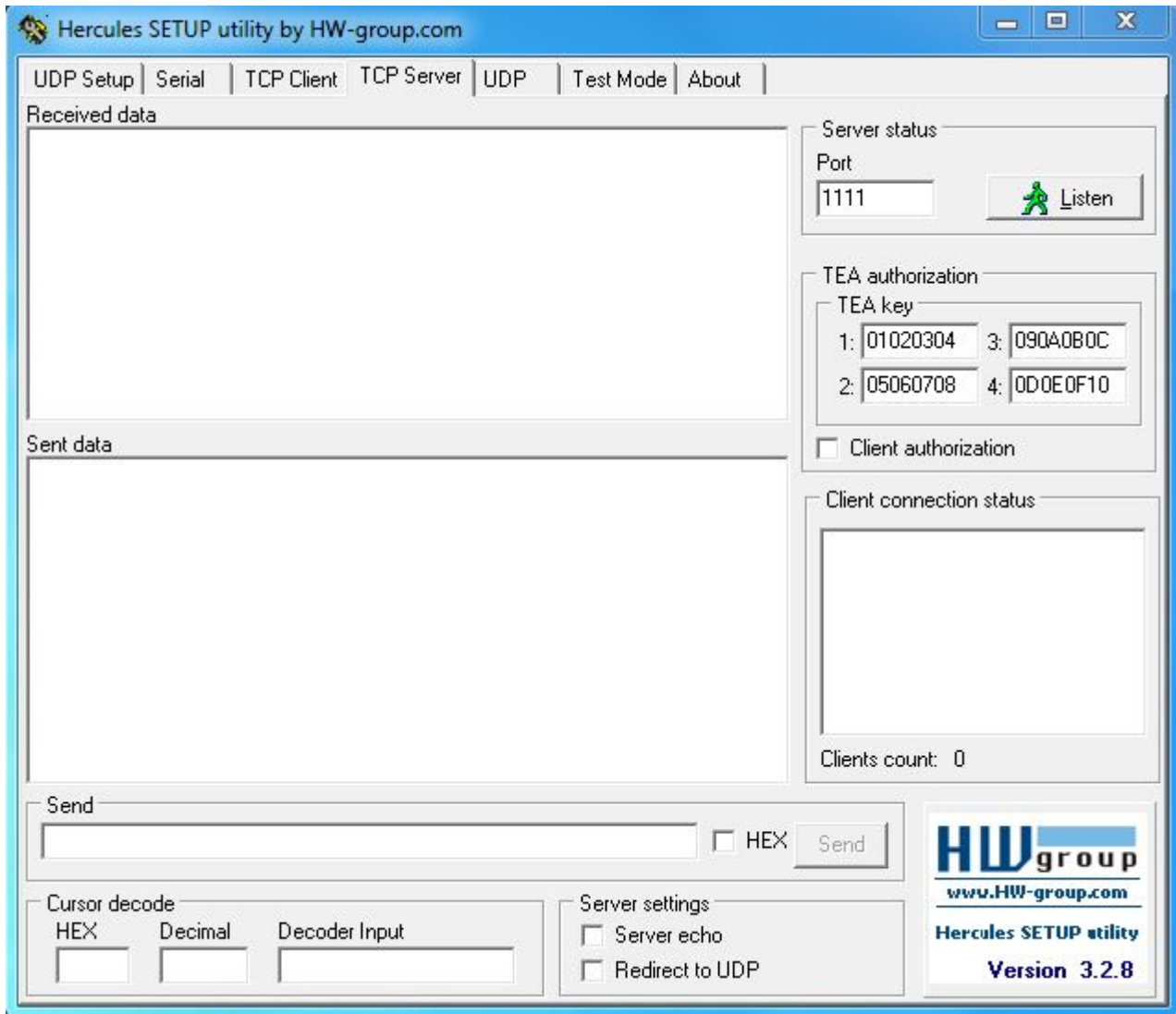
Note that Device Response converted from HEX to ASCII means:

```
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 to use Hercules (in TCP server mode). Simply write command as explained below into Hercules Send field, check HEX box and click Send button. Note that the TCP server must be listening on specified port (see Port field and Listen button below).





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

All information is provided in "FMXX and Garmin development.pdf" document.

- **COM TCP Link Mode**

All information is provided in "FMxx TCP Link mode test instructions.pdf" document.

## Codec 13

- **About Codec13**

Codec13 is original Teltonika protocol for device-server communication over GPRS messages. This protocol is necessary for using following FM features: COM TCP Link Mode (binary/ASCII/binary buffered/ASCII buffered) if message timestamp parameter is enabled in device configuration. Codec13 messages are one way only (Device → Server sending).

- **General Codec13 message structure**

The following diagram shows basic structure of Codec 13 messages:

<b>0x00000000 (Preamble)</b>	<b>Data Size</b>	<b>Codec ID</b>	<b>Response Quantity 1</b>	<b>Type</b>	<b>Response Size</b>	<b>Timestamp</b>	<b>Response</b>	<b>Response Quantity 2</b>	<b>CRC-16</b>
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 preamble field (four zero bytes).

**Data Size** - size is calculated from Codec ID field to the second Response Quantity field.

**Codec ID** - in Codec13 it is always 0x0D.

**Response Quantity 1** - 0x01, it is ignored when parsing the message.

**Response Type** - it is always 0x06 since the packet is direction is FM->Server.

**Response Size** - response size field includes size of timestamp too, so it is equal to size of payload + size of timestamp.

**Timestamp** - a difference, in seconds, between the current time and midnight, January, 1970 UTC (UNIX time).

**Response** - actual received data.

**Response Quantity 2** - a byte which defines how many records (responses) is in the packet. This byte will not be parsed but it's recommended that it should contain same value as Response Quantity 1.

**CRC-16** - calculated from Codec ID to the Second Number of Data. CRC (Cyclic Redundancy Check) is an error-detecting code using for detect accidental changes to RAW data. For calculation we are using [CRC-16/IBM](#).

**Note:** Codec13 packets are used only when "Message Timestamp" parameter in RS232 settings is enabled.

### • Command parsing example

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

Receiving "hello lets test" SMS response via GPRS Codec13

Hexadecimal stream:

```
0000000000000001D0D01060000001564E8328168656C6C6F206C65747320746573740D0A010003548
```

Parsed:

#### Device answer

<b>Device answer part</b>	<b>HEX Code Part</b>
Zero Bytes	00 00 00 00
Data Size	00 00 00 1D
Codec ID	0D
Response Quantity 1	01
Response Type	06
Response Size	00 00 00 15
Timestamp	64 E8 32 81
Response	68 65 6C 6C 6F 20 6C 65 74 73 20 74 65 73 74 0D 0A

Note that Server Response converted from HEX to ASCII means "hello lets test"

## Codec 14

### • About Codec14

Codec14 is original Teltonika protocol for device-server communication over GPRS messages and it is based on Codec12 protocol.

Main difference of Codec14 is that, device will answer to GPRS command if device physical IMEI number matches specified IMEI number in GPRS command.

Codec14 GPRS commands can be used for sending configuration, debug, digital outputs control commands or other (special purpose command 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 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 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 request command from 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 command message IMEI doesn't match actual device IMEI, received command won't be executed and response to 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. Minimal value is 8 because Codec14 always contain IMEI and it's 8 bytes.

**IMEI (HEX)** - it is send as HEX value. Example if device IMEI is 123456789123456 then IMEI data field will contain 0x0123456789123456 value.

**Command/Response** - command or response in HEX.

**Command/Response Quantity 2** - a byte which defines how many records (commands or responses) is in the packet. This byte will not be parsed but it's recommended that it should contain 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 using for detect accidental changes to RAW data. For calculation we are using [CRC-16/IBM](#).

## • GPRS in Codec14 examples

Hexadecimal stream of GPRS command and answer in this example are given in hexadecimal form. The different fields of message are separate 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:

0000000000000000160E01050000000E0352093081452251676574766572010000D2C1

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 hexadecimal stream:

0000000000000000AB0E0106000000A303520930814522515665723A30332E31382E31345F30342  
04750533A41584E5F352E31305F333333332048773A464D42313230  
204D6F643A313520494D45493A33353230393330383134353232353120496E69743A323031382  
D31312D323220373A313320557074696D653A3137323334204D4143  
3A363042444430303136323631205350433A312830292041584C3A30204F42443A3020424C3A3  
12E362042543A340100007AAE

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 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
Response	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 hexadecimal stream:

000000000000000100E011100000008035209308145246801000032AC

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

# Differences between Codec 12, Codec 13 and Codec 14

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

	Codec12	Codec13	Codec14
<b>Communication</b>	Server ⇌ Device Communication	One-way (Device → Server communication)	Server ⇌ Device Communication
<b>Codec ID</b>	0x0C	0x0D	0x0E
<b>command message type</b>	0x05	-	0x05
<b>Response Message Type</b>	0x06	0x06	0x06 (if it is ACK) or 0x11 (if it is nACK)
<b>Command / Response size</b>	Command/Response	Only Response	Command/Response + IMEI
<b>Timestamp</b>	Not Using	Is Using	Not Using
<b>IMEI</b>	Not Using	Not Using	Is Using

## 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.

### • 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	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 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.

### • Structure

Below in the tables you will see 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.

### SMS Data Structure

	GPSElement	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 Timestamp field. Time corresponding to each further element can be computed as  $\text{elementTime} = \text{Timestamp} + (1 \text{ hour} * \text{elementNumber})$ .

### GPS Data Element

	Size (bits)	Field	Description
	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 Difference from previous element's longitude.
	14	LongitudeDiff	LongitudeDiff = $\text{prevLongitude} - \text{Longitude} + 213 - 1$
DifferentialCoords == 1			Difference from previous element's latitude
ValidElement == 1	14	LatitudeDiff	LatitudeDiff = $\text{prevLatitude} - \text{Latitude} + 213 - 1$
	21	Longitude	Longitude = $\{(\text{LongDegMult} + 18 * 108) * (221 - 1)\} \text{ over } \{36 * 108\}$
DifferentialCoords == 0			Latitude = $(\text{LatDegMult} + 9 * 108) * (220 - 1) \text{ over } \{18 * 108\}$
	20	Latitude	
	8	Speed	Speed in km/h

**Longitude** - longitude field value of GPSElement

**Latitude** - latitude field value of GPSElement

**LongDegMult** - longitude in degrees multiplied by 107 (integer part)

**LatDegMult** - latitude in degrees multiplied by 107 (integer part)

**prevLongitude** - longitude field value of previous GPSElement

**prevLatitude** - latitude field value of previous GPSElement

#### • Decoding GPS position

When decoding GPS data with DifferentialCoords = 1, Latitude and Longitude values can be computed as follows:  $\text{Longitude} = \text{prevLongitude} - \text{LongitudeDiff} + 213 - 1$ ,  $\text{Latitude} = \text{prevLatitude}$

- LatitudeDiff + 213 - 1.

If there were no previous non-differential positions, differential coordinates should be computed assuming  $\text{prevLongitude} = \text{prevLatitude} = 0$ .

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

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

### • SMS Events

When Configured to generate 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 data sending is using 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. 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**

X bytes

**IMEI**

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 using for detect accidental changes to RAW data. The algorithm how to calculate CRC-16 (also known as CRC-16/IBM) you will find below.



