https://wiki.teltonika-networks.com/view/VCode

VCode

□

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.

Codec ID table

Codec 8	Codec 8 extended UDP channel header	Codec 12	Codec 13	Codec 16
0x08	0x8E	0x0C	0x0D	0x10

Codec 8

- *AVL data packet*

Because the smallest information amount that can be written is one bit, there can be some bits left unused when result is byte array. Any unused bits should be left blank.

Below table represents AVL data packet structure.

Example

Received data:

00000000000000c08010000013feb55ff74000f0ea850209a6900009400001200000 0

1e0901000200030004001601470 3f0001504c8000c0900730a00460b00501300464306

d7440000b5000bb60007422e9 f180000cd0386ce000107c700000000f10000601a4600

0001344800000bb84900000bb84a00000bb84c00000000024e0000000000000000cf00

00000000000000010000 3fca

In total 152 Bytes

4 zeros(4 bytes)	00000000 4 zeroes, 4 bytes
Data field length(4 bytes)	0000008c data length, 4 bytes
Codec ID(1 byte)	08 – Codec ID
Number of Data 1(1 byte)	01 Number of Data (1 record) 1’st record data

AVL Data(30-147 bytes)	Timestamp(8 bytes)		010000013feb55ff74 – Timestamp in milliseconds (1374042849140) GMT: Wed, 17 Jul 2013 06:34:09 GMT
	Priority(1 byte)		00 – Priority
	GPS Element(15 bytes)	Longitude (4 bytes)	0f0ea850 – Longitude 252618832 = 25,2618832º N
		Latitude (4 bytes)	209a6900 – Latitude 546990336 = 54,6990336 º E
		Altitude (2 bytes)	0094 – Altitude 148 meters
		Angle (2 bytes)	0000 – Angle 0
		Satellites (1 bytes)	12 – 18 Visible satellites
		Speed (2 bytes)	0000 – 0 km/h speed
	IO Element (6-123 bytes)		00 – IO element ID of Event generated (in this case when 00 – data generated not on event) 1e – 30 IO elements in record (total) 09 – 9 IO elements, which length is 1 Byte 0 – IO element ID = 01 0 – IO element's value = 0 02 – IO element ID = 02 0 – IO element's value = 0 03 – IO element ID = 03 0 – IO element's value = 0 4 – IO element ID = 04 0 – IO element's value = 0 16 – IO element ID = 22 (dec) 0 – IO element's value = 1 47 – IO element ID = 71 (dec) 03 – IO element's value = 3 F0 – IO element ID = 240 (dec) 0 – IO element's value = 0 15 – IO element ID = 21 (dec) 04 – IO element's value = 0 C8 – IO element ID = 200 (dec) 0 – IO element's value = 0 0c – 12 IO elements, which value length is 2 Bytes 09 – IO element ID = 9 (dec) 0073 – IO element's value 0a – IO element ID = 10 (dec) 0046 – IO element's value 0b – IO element ID = 11 (dec) 0050 – IO element's value 13 – IO element ID = 19 (dec) 0046 – IO element's value 43 – IO element ID = 67 (dec) 06d7 – IO element's value 1 – IO element ID = 68 (dec) 0 – IO element's value B5 – IO element ID = 181 (dec) 000b – IO element's value B6 – IO element ID = 182 (dec) 0007 – IO element's value 42 – IO element ID = 66 (dec) 2e9f – IO element's value 2 – IO element ID = 24 (dec) 0 – IO element's value cd – IO element ID = 205 (dec) 3 – IO element's value CE – IO element ID = 206 (dec) 0 – IO element's value 07 – 7 IO elements, which value length is 4 Bytes C7 – IO element ID = 199 (dec) 0 – IO element's value f1 – IO element ID = 241 (dec) 0000601a – IO element's value 46 – IO element ID = 70 (dec) 00000134 – IO element's value 48 – IO element ID = 72 (dec) 00000bb8 – IO element's value 4 – IO element ID = 73 (dec) 00000bb8 – IO element's value 4a – IO element ID = 74 (dec) 00000bb8 – IO element's value 4c – IO element ID = 76 (dec) 1 – IO element's value 02 – 2 IO elements, which value length is 8 Bytes 4e – IO element ID = 78 (dec) 0 – IO element's value cf – IO element ID = 207 (dec) 0 – IO element's value

Number of Data 2(1 byte)	01 – Number of Data (1 record)
CRC-16(4 bytes)	00003fca - CRC-16, 4 Bytes (first 2 are always zeros)

Number of data – number of encoded data (number of records). Codec ID is constant 08.

Data field length is the length of bytes [codec id, number of data 2]. Number of data 1 should always be equal to number of data 2 byte. CRC-16 is 4 bytes, but first two are zeroes and last two are CRC-16 calculated for [codec id, number of data 2] Minimum AVL packet size is 45 bytes (all IO elements disabled). Maximum AVL packet size for one record is 783 bytes.

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

- *Priority*

Priority level (AVL packet priority) can be:

- Low Priority
  - Module makes an additional record with an indication that the event was caused by an I/O element change (depending on Operands configuration).
- High Priority
  - Module makes an additional record with High priority flag and sends event packet immediately to the server using GPRS.
- Panic Priority
  - This priority triggers same actions as High priority, but if GPRS fails, it sends an AVL packet using SMS data if SMS data sending is enabled and the number is provided in SMS/Call Settings.

- *GPS Element*

X Longitude

Y Latitude1

Altitude In meters above sea level1

Angle In degrees, 0 is north, increasing clock-wise 1

Satellites Number of visible satellites1

Speed in km/h. 0x0000 if GPS data is invalid1

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

d - Degrees

m - Minutes

s - Seconds

ms - Milliseconds

p - Precision (10000000)

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

- **SENDING DATA OVER TCP/IP**

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 000F means, that IMEI is 15 bytes long.

After receiving IMEI, server should determine if it would accept data from this module. If yes server will reply to module 01 if not 00. Note that confirmation should be sent as binary packet. I.e. 1 byte 0x01 or 0x00.

Then module starts to send first AVL data packet. After server receives packet and parses it, server must report to module number of data received as integer (four bytes).

If sent data number and reported by server doesn't match module resends sent data.

Example:

Module connects to server and sends IMEI:

000f333536333037303432343431303133

Server accepts the module:

01

Module sends data packet:

AVL data packet header	AVL data array	CRC
Four zero bytes, 'AVL data array' length – 254	CodecId – 08, NumberOfData – 2. (Encoded using continuous bit stream. Last byte padded to align to byte boundary)	CRC of 'AVL data array'
00000000000000FE	0802...(data elements)...02	00008612

Server acknowledges data reception (2 data elements): 00000002

- **SENDING DATA OVER UDP/IP**

- *UDP channel protocol*

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

UDP datagram

UDP channel packet x N	Example	2 bytes	Packet length (excluding this field) in big ending byte order
	Packet Id	2 bytes	Packet id unique for this channel
	Not usable byte	1 byte	Not usable byte
	Packet payload	m bytes	Data payload

Not usable byte

1 Data packet requiring acknowledgment

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

Packet length	2 bytes	0x0003
Packet id	2 bytes	same as in acknowledged packet
Not usable byte	1 byte	0x00

- **Sending AVL data using UDP channel**

AVL data are sent encapsulated in UDP channel packets (Data payload field).

AVL data encapsulated in UDP channel packet

AVL packet id (1 byte) Module IMEI AVL data array

AVL packet id (1 byte) – id identifying this AVL packet Module IMEI – IMEI of a sending module encoded the same as with TCP AVL data array – array of encoded AVL data

Server response to AVL data packet

AVL packet id (1 byte) Number of accepted AVL elements (1 byte)

AVL packet id (1 byte) – id of received AVL data packet

AVL packet id (1 byte) – id of received AVL data packet

Number of AVL data elements accepted (1 byte) – number of AVL data array entries from the beginning of array, which were accepted by the server.

Scenario: Module sends UDP channel packet with encapsulated AVL data packet. 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
Len – 253, Id – 0xCAFE, Not usable byte – 00	AVL packet id – 0xDD, IMEI – 1234567890123456	CodecId – 08, NumberOfData – 2. (Encoded using continuous bit stream)
00FDCAFE01	DD000F313334353637383930313233343 5	0802…(data elements)…02

Server must respond with acknowledgment:

UDP channel header	AVL packet acknowledgment
Len – 5, Id – 0xABCD, Not usable byte – 00	AVL packet id – 0xDD, NumberOfAcceptedData – 2
0005ABCD01	DD02

Codec 8 Extended

- **Protocols overview**

Difference between codec8 and codec8 extended

	Codec 8	Codec 8 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

Table 2. Codec 8 and 8 Extended differences

Main differences between are shown in above table. AVL data element sizes in codec 8 extended protocol were increased to 2 bytes length and new variable type added. For more detailed description look in codec 8 and codec 8 extended chapters.

- **Codec 8 extended protocol sending over TCP**

AVL data packet

Below table represents AVL data packet structure.

4 zeros	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	38-768 Bytes	1 Byte	4 Bytes

Table 3. AVL data packet structure

Number of data – number of encoded data (number of records). Codec ID is constant 0x8E.

Data field length is the length of bytes [codec id, number of data 2]. Number of data 1 should always be equal to number of data 2 byte.

CRC-16 is 4 bytes, but first two are zeroes and last two are CRC-16 calculated for [codec id, number of data 2]

Minimum AVL packet size is 53 bytes (all IO elements disabled).

- **AVL Data**

Timestamp	Timestamp	Priority	GPS Element	IO Element
8 Bytes	1 Byte		15 Bytes	14 - 744

Table 4. AVL data structure

- **Priority**

0 Low
1 High
2 Panic

Table 5. Priority element values

- **GPS Element**

Longitude	Latitude	Altitude	Angle	Satellites	Speed
4 Bytes	4 Bytes	2 Bytes	2 Bytes	1 Byte	2 Bytes

Table 6. GPS element structure

- 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 bytes
N1'th IO ID	2 bytes
N1'st IO Value	1 bytes
N2 of Two Byte IO	2 bytes
1'st IO ID	2 bytes
1'st IO Value	2 bytes
N2'th IO ID	2 bytes
N2'st IO Value	2 bytes
N4 of Four Byte IO	2 bytes
1'st IO ID	2 bytes
1'st IO Value	4 bytes
N4'th IO ID	2 bytes
N4'st IO Value	4 bytes
N2 of Eight Byte IO	2 bytes
1'st IO ID	2 bytes
1'st IO Value	8 bytes
N8'th IO ID	2 bytes
N8'st 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'st IO ID	2 bytes
NX'st IO Length	2 bytes
NX'st IO Value	defined by length

Table 7. IO element structure

N - total number of properties coming with record (N=N1+N2+N4+N8+NX)

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 - number of properties, which length is defined by length element

- **Communication with server**

Communication with server is the same as with codec 8 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

AVL data packet header	AVL data array	CRC
Four zero bytes, 'AVL data array' length – 254	Codec Id – 8E, NumberOfData – 02. (Encoded using continuous bit stream. Last byte padded to align to byte boundary)	CRC of 'AVL data array'
00000000000000FE	8E02...(data elements)...02	00008612

Table 8. IO element structure

Server acknowledges data reception (2 data elements): 00000002

- **Example**

Received data:

00000000 0000005F 8E 01 0000015FBA40B620 00 0F0DCDE420959D30 008A 0000 06000000000600 01 00EF0
0 0001 0011001E 0001 00100000CBDF 0002 000B000000003544C875000E0000000029BFE4D1 0001 01
000011 00000000000000000000000000000000 01 0000D153

00000000 4 zeros, 4 bytes

0000005F data length, 4 bytes

8E – Codec ID

01- Number of Data (1 record)

1'st record data

0000015FBA40B620 – Timestamp in milliseconds (1510658324000)

GMT: Tuesday, November 14, 2017 11:18:44 AM

00 – Priority

GPS Element

0F0DCDE4 – Longitude 252562916 = 25, 2562916º N

20959D30 – Latitude 546676016 = 54,6676016 º E

008A – Altitude 138 meters

0000 – Angle 0º

06 – 6 Visible satellites

0000 – 0 km/h speed

IO Element

0000 – IO element ID of Event generated (in this case when 0000 – data generated not on event)

0006 – 6 IO elements in record (total)

0001 – 1 IO elements, which length is 1 Byte

00EF – IO element ID = 239 (dec)

00 – IO element's value

0001 – 1 IO elements, which length is 2 Byte

0011 – IO element ID = 17 (dec)

001E – IO element's value

0001 – 1 IO elements, which length is 4 Byte

0010 – IO element ID = 16 (dec)

0000CBDF – IO element's value = 52191 (dec)

0002 – 2 IO elements, which length is 2 Byte

000B – IO element ID = 11 (dec)

000000003544C875 – IO element's value

000E – IO element ID = 14 (dec)

0000000029BFE4D1 – IO element's value

01 – Number of Data (1 record)

0000D153 – CRC-16, 4 Bytes (first 2 are always zeros

Codec 8 extended protocol sending over UDP

- **AVL data packet**

AVL data packet is the same as with codec 8, except codec ID is changed to 0x8E.

Example:

Module sends the data:

UDP channel header	AVL packet header	AVL data array
Len – 253, Id – 0xCAFE, Not usable byte – 00	AVL packet id – 0xDD, IMEI – 1234567890123456	Codec Id – 8E, NumberOfData – 02. (Encoded using continuous bit stream
00FDCAFE00	DD000F3133343536373839303132333435	8E02…(data elements)…02

Table 9. Example packet send to server

Server must respond with acknowledgment:

UDP channel header	AVL packet acknowledgment
Len – 5, Id – 0xCAFE, Not usable byte – 00	AVL packet id -0xDD Number of Accepted Data -2
0005CAFE00	DD02

Table 10. Example packet server response

Example

Server received data:

00A1CAFE001B000F3335363330373034323434313031338E010000013FEBDD19C8000F0E9FF0209A718000690000120000

001E090100020003000400160147 03F0001504C8000C0900910A00440B004D13004443155544000 0B5000BB60005422E

9B180000CD0386CE000107C700000000F10000601A460000013C4800000BB84900000BB84A0000 0BB84C00000000024E

0000000000000000CF000000000000000001

Data length: 00A1 or 161 Bytes (not counting the first 2 data length bytes)

Packet identification: 0xCAFE 2 bytes

Not usable byte: 00

Packet id: 1B

Imei length: 000F

Actual imei: 33353633303730343234343130313133

Codec id: 8E

Number of data: 01

Timestamp: 0000013FEBDD19C8

Priority: 00

GPS data: 0F0E9FF0209A718000690000120000

Codec 12

- **About Codec12**

Codec12 is original 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 FMB63/FM63/FM5300/FM5500/FM4200 features like: Garmin, LCD communication, COM TCP Link Mode.

- **FM firmware requirements**

Supported GPRS commands on each device depend on firmware version. For available GPRS commands on each device, please refer to Table1 FM firmware requirement “SMS over GPRS” means that all standard SMS commands text can be sent to device via GPRS in Codec12 format.

Device	SMS over GPRS	SMS over GPRS via UDP	Special Codec12 GPRS commands
FM11YX	Available since base firmware 01.11.XX	Since base firmware 1.26.00	Available in FM1100 and FM1110 special firmware version 12.XX.XX*
FM12YX	Available since base firmware 01.03.XX	-	Available in FM12YX special firmware 09.XX.XX*
FM10YX	Available since base firmware 00.03.XX	-	Available since base firmware 00.06.XX and later versions**
FM3400	Available since base firmware 01.01.XX	-	-
FM36YX	Available in base firmware	Since base firmware 01.06.01	-
FM5300	Available in all firmware versions	-	Available in all firmware versions**
FM5500	Available in all firmware versions	-	Available in all firmware versions**
FM2200	Not available	-	Special firmware version 07.XX.XX*
FM4200	Not available	-	Special firmware version 42.XX.XX*

FM63YX	Available in all firmware versions	Since base firmware 00.02.19	Available in all firmware versions**
FMB9YX	Available in base firmware	Available in base firmware	-
FMB0YX	Available in base firmware	Available in base firmware	-
FMB1YX	Available in base firmware	Available in base firmware	-
FMB6YX	Available in all firmware versions	Since base firmware 00.02.19	Available in all firmware versions**
FMA1YX	Available in base firmware	Since base firmware 1.26.00	Available in special firmware version 12.XX.XX*
FMA2YX	Available in base firmware	Available in base firmware	-

Table 1. FM firmware requirement

'*' - supported special commands are listed on special firmware description. Please contact Your Teltonika sales manager for more details;

'**' - supported special commands are listed on device user manual, chapter “GPRS Commands”

- **GPRS command session**

Following figure shows how GRPS command session is started over TCP.

First FM opens GPRS session and sends AVL data to server (refer FM protocols). Once all records are sent and correct sent data array acknowledgment is received by FM then GPRS commands in Hex can be sent to FM.

The ACK (acknowledge of IMEI from server) is a one byte constant 0x01. The acknowledgement of each data array send from FMXXXX is four bytes integer – number of records received.

Note, that GPRS session should remain active between FM and server, while GPRS commands are sent. For this reason active datalink timeout (global parameters in FMXXXX configuration) is recommended to be set to 259200 (maximum value).

- **General Codec12 message structure**

The following diagram shows basic structure of Codec 12 messages.

Command message structure

0x00000000	**Data size**	**0x0C**	**Command quantity**	**0x05**	**Command size**	**Command**	**<CR><LF>**	**Command quantity**	**CRC**
4 bytes	4 bytes	1 byte	1 byte	1 byte	4 bytes	X bytes	0D0A	1 byte	4 bytes

Response message structure

0x00000000	Data size	0x0C	Command quantity	0x06	Command size	Command	Command quantity	CRC
4 bytes	4 bytes	1 byte	1 byte	1 byte	4 bytes	X bytes	1 byte	4 bytes

Structure explanation:

The message starts with preamble field - four zero bytes. Then goes four bytes data size field (size is calculated from 0x0C field to the second command or response quantity field). Then follows one byte Codec ID field (in Codec 12 it is always 0x0C). Then goes command or response quantity field (it is ignored when parsing the message). After that goes one byte message type field. It can be 0x05 to denote command or 0x06 to denote response. Then follows four bytes command or response size field. After it follows the command or response field itself. After that goes the second command or response quantity field. At the end there's four bytes CRC field.

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

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. The algorithm of how to calculate

CRC is shown in Figure 3 CRC calculation algorithm.

Figure 3 CRC calculation algorithm

- **Command coding table**

Command has to be convert from ASCII characters (char) to hexadecimal (Hex)

Table 2 ASCII conversion table

- **Command parsing example**

Hexadecimal stream of command and answer in this example are given in hexadecimal form. The different fields of message are highlighted in gray and yellow for better readability and command is converted in ASCII for better understanding.

Server command

Hexadecimal stream:

Parsed:

Preamble: 0x00000000

Packet Length: 0x00000018

Codec: 0x0C

Quantity of commands: 0x01

Command type: 0x05

Command size: 0x00000010

Command in ASCII after conversion: #GET DATAORDER<CR><LF>

Quantity of commands: 0x01

CRC: 0x00004990

Device answer

Hexadecimal stream:

Parsed:

Preamble: 0x00000000

Packet Length: 0x00000016

Codec: 0x0C

Quantity of commands: 0x01

Command type: 0x06

Command size: 0x0000000E

Command response is in ASCII after conversion: #DATAORDER=1<CR><LF>

Quantity of commands: 0x01

CRC: 0x00000095

- **Codec12 GPRS commands examples**

The example commands given in hexadecimal form separated by dollar signs are suitable to be sent from TCP server during data exchange session between FMXXXX device and server (for more details see Figure 1 Command session) The command can be sent from a terminal program such as Hercules (in TCP server mode). Simply write command as explained below into Hercules Send field and click Send button. The TCP server must be listening on specified port (see field Port and button Listen in Figure 4).

Figure 4 Hercules terminal GUI

- **SMS over GPRS in Codec12 examples**

In case of sending SMS commands over GPRS, do not use SMS logins set during configuration or do not leave empty spaces before command. Devices and firmware versions that support SMS over GPRS are listed in Table1.

Example 1: getinfo

Sending “getinfo” SMS command via GPRS Codec12:

Server requests:

Hexadecimal stream:

00000000000000110C010500000009676574696E666F0D0A010000DA7E

Parsed:

Zero: 0x00000000

Packet Length: 0x00000011

Codec: 0x0C

Quantity of commands: 0x01

Command type: 0x05

Command size: 0x00000009

Command: 0x676574696e666f (HEX of getinfo)

Command end symbol: 0x0D0A

Quantity of commands: 01

CRC: 0x0000DA7E

Device response:

Hexadecimal stream:

00000000000000820C01060000007A494E493A323031312F312F3120303A30205254433A32303131
2F312F3120373A3333205253543A33204552523A302053523A3134372042523A302043463A312046
473A3020464C3A302055543A3020534D533A30204E4F4750533A303A3134204750533A322053415
43A302052533A36204D443A302052463A30010000B8AA

Parsed:

Zero: 0x00000000

Packet Length: 0x00000082

Codec: 0C

Quantity of commands: 01

Command type: 06

Command size: 0x0000007A

Command response in ASCII after conversion: INI:2011/1/1 0:0 RTC:2011/1/1 7:33 RST:3 ERR:0 SR:147 BR:0 CF:1 FG:0 FL:0 UT:0 SMS:0 NOGPS:0:14 GPS:2 SAT:0

RS:6 MD:0 RF:0 (without <CR><LF>)

Quantity of commands: 01

CRC: 0x0000B8AA

Example 2: getio

Sending “getio” SMS command via GPRS Codec12:

Server request:

Hexadecimal stream:

000000000000000F0C010500000007676574696F0D0A0100003349

Parsed:

Zero: 0x00000000

Packet Length: 0x0000000F

Codec: 0x0C

Quantity of commands: 0x01

Command type: 0x05

Command size: 0x00000007

Command: 0x676574696f (HEX of getio)

Command end symbol: 0x0D0A

Quantity of commands: 01

CRC: 00003349

Device response:

Hexadecimal stream:

000000000000002C0C010600000244449313A30204449323A30204449333A302041494E3A323420444F313A3020444F323A30010000F925

Parsed:

Zero: 0x00000000

Packet Length: 0x0000002C

Codec: 0C

Quantity of commands: 01

Command type: 06

Command size: 0x00000024

Command response in ASCII after conversion: DI1:0 DI2:0 DI3:0 AIN:24 DO1:0 DO2:0 (without <CR><LF>)

Quantity of commands: 01

CRC: 0x0000F925

- **Special Codec12 commands:**

Example 1: #GET VERSION

Server command - #GET VERSION<CR><LF>

Device response - #VERSION=XXXXXXXX<CR><LF>

XXXXXXXX – Device Firmware Version (up to 8 characters)

$00$00$00$00$00$00$00$16$0C$01$05$00$00$00$0E$23$47$45$54$20$56$45$52$53$49$4f$4e$0D$0A$01$00$00$D0$C8

Example 2: #GET NETWORK

Server command - #GET NETWORK<CR><LF>

Device response - #NETWORK=XXXXXX<CR><LF>

XXXXXX – GSM Operator Network [0 - 999999]

$00$00$00$00$00$00$00$16$0C$01$05$00$00$00$0E$23$47$45$54$20$4e$45$54$57$4f$52$4b
$0D$0A$01$00$00$ED$61

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 (Codec 13 is used for FM->Server sending).

- **FM firmware requirements**

Codec13 availability depends on device and firmware version.

Device	Availability
FM11YX	Available since base firmware 01.18.XX
FM12YX	Not available
FM10YX	Not available
FM3400	Not available
FM5300	Not available
FM5500	Not available
FM2200	Not available
FM4200	Not available
FM6320	Available in base
FMAXX	Not available
FMBXX	Available in base
FMB630	Available in base

Table 1. FM firmware requirements

- **GPRS command session**

Following figure shows how GRPS command session is started over TCP.

Figure 1. Command session

First FM opens GPRS session and sends AVL data to server (refer FM protocols).

After all records are sent and correct sent data array acknowledgment is received by FM, it will begin TCP link mode message sending.

TCP Link mode messages do not require ACK.

- **General Codec13 message structure**

The following diagram shows basic structure of Codec 13 messages.

0x00000000	**Data size**	**0x0D**	**0x01**	**0x06**	**Command size**	**Timestamp**	**Payload**	**0x01**	**CRC**
4 bytes	4 bytes	1 byte	1 byte	1 byte	4 bytes	4 bytes	X bytes	1 byte	1 byte

0x06 **Message type filed - 0x06 = packet FM -> Server**

0x01 **Ignored bytes**

Figure 2. Structure of Codec 13 messages

Structure explanation:

- **Preamble field** - four zero bytes.
- **Data size field** (size is calculated from:

CID(0x0D = 1 byte)

NOD(0x01 = 1 byte)

CMD_TYPE(0x06 = 1 byte)

CMD_SIZE(variable = 4 bytes, includes size of timestamp field too)

PAYLOAD(variable size, stored in CMD_SIZE field)

NOD(0x01 = 1 byte))

- **Codec ID field** (in Codec 13 it is always 0x0D).
- **NOD field** (0x01, it is ignored when parsing the message).
- **Message type field**. It is always 0x06 since the packet is direction is FM->Server.
- **Command size field**. Command size field includes size of timestamp too, so it is equal to size of payload + size of timestamp.
- **Timestamp field** – UNIX timestamp (since 1970/01/01 00:00:00 UTC)
- **Payload field** – actual received data
- **NOD field** (0x01, it is ignored when parsing the message)
- **CRC field** – CRC-16-IBM

The algorithm to calculate CRC is CRC-16 (also known as CRC-16-IBM). All the fields from codec ID to last NOD field are used to calculate CRC. The algorithm of how to calculate CRC is shown in figure 3.

NOTE – Codec13 packets are used only when “Message Timestamp” parameter in RS232 settings is enabled.

Figure 3. CRC calculation algorithm

- **Command parsing example**

The different fields of message are highlighted in gray and yellow for better readability.

Hexadecimal stream:

Parsed:

Preamble: 0x00000000

Packet Length: 0x0000001C

Codec: 0x0D

NOD: 0x01

Command type: 0x06

Command size: 0x00000014

Timestamp: 1458119714 – 03/16/2016 @ 9:15am UTC

Command: #GET DATAORDER<CR><LF>

NOD: 0x01

CRC: 0x00004990

- **FMB630/FM6320/5300/FM5500 and Codec12 functionality**
- **Garmin**

All information is provided in “FMB6 FM6320 FM5300 FM5500 and Garmin development.pdf” document.

- **COM TCP Link Mode**

All information is provided in “FMB6 FM6320 FM5300 and FM5500 TCP Link mode test instructions.pdf” document.

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
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
4 bytes	4 bytes	1 bytes	1 bytes	1 bytes	4 bytes	8 bytes	X bytes	1 bytes	4 bytes

- **Structure explanation:**

1. The message starts with **preamble field** - four zero bytes.
2. Then goes four bytes **data size** field (size is calculated from 0x0E field to the second command or response quantity field).
3. Then follows one byte **Codec ID field (in Codec 14 it is always 0x0E)**.
4. Then goes **command or response quantity field.** Response message will contain same quantity field value as request command quantity 1st byte (quantity byte that's located before message field type byte). 2nd byte (quantity byte that's located at the end before CRC) will not be parsed but it's recommended that it should contain same value as 1st byte.
5. After that goes one byte **message type field.** If it's 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.

6. Then follows four bytes **command or response size field.** (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.
7. After it follows the **IMEI**. IMEI is send as HEX value. Example if device IMEI is 123456789123456 then IMEI data field will contain 0x0123456789123456 value.

8. After it follows the **command or response field** itself. If message type field is nACK (0x11) this data field will be empty.
9. After that goes the second **command or response quantity field.**
10. At the end there's four bytes **CRC field**.

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. The algorithm of how to calculate CRC is shown in Figure 3 CRC calculation algorithm.

CRC is shown in Figure 3 CRC calculation algorithm.

- **GPRS in Codec14 examples**

Sending "*getver*" SMS command via GPRS Codec14:
Server requests:
Hexadecimal stream:
00000000000000160E01050000000E03520930814522516765747665720100 00D2C1
Parsed:
Zero: 0x00000000
Packet Length: 0x00000016
Codec: **0E**
Quantity of commands: 0x01
Command type: 0x05
Command size: 0x0000000E
IMEI: 0x0352093081452251 (HEX string)
Command: 0x676574766572 (HEX of **getver**)
Quantity of commands: 01
CRC: 0x0000D2C1

Device ACK response:
Hexadecimal stream:
00000000000000AB0E01060000000A3035209308145225156657 23A30332E31382E3
1345F3034204750533A41584E5F352E31305F333333332048773A464D4231323020
4D6F643A313520494D45493A3335323039333038313435323235312049 6E69743A3
23031382D31312D323220373A313320557074696D653A3137323334204D41433A3
6304244443030313632363120535043 3A312830292041584C3A30204F42443A3020
424C3A312E362042543A3401000007AAE
Parsed:
Zero: 0x00000000
Packet Length: 0x000000AB
Codec: 0x**0E**
Quantity of commands: 01
Command type: 0x06
Command size: 0x000000A3
IMEI: 0x0352093081452251
Command response in ASCII after conversion:
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

Quantity of commands: 01
CRC: 0x00007AAE

Device nACK response:
Hexadecimal stream:
00000000000000010**0E**01110000000803520930814524680 1000032AC
Parsed: Zero: 0x00000000
Packet Length: 0x00000010
Codec: 0x**0E**
Quantity of commands: 01
Command type: 0x11
Command size: 0x00000008
IMEI: 0x0352093081452468
Quantity of commands: 01
CRC: 0x000032AC

Codec 16

- **Description and example**

Records to server will be sent as shown in table below. The main difference between CODEC8 and CODEC16 is CODEC ID which will be 0x10 instead of 0x08, AVL ID‘s in AVL data is sent in 2 bytes, instead of 1 byte.

Also new parameter – Generation type is added.

By receiving 0x10 codec ID server must know that AVL data record will be parsed different.

Codec16 is supported from firmware – 00.03.xx and newer. (FMB630/FM63XY) ll AVL ID‘s which are higher than 255 will can be used only in CODEC16 protocol.

Generation type elements:

Value	Record created
0	On exit
1	On Entrance
2	On Both
3	Reserved
4	Hysterisis
5	On Change
6	Eventual
7	Periodical

Codec16 TCP packet frame:

Header	Data length	Codec ID	NOD1	AVL DATA	NOD2	CRC16
4 bytes	4 bytes	0x10	1 byte	Variable	1 byte	4 Bytes

NOD1, NOD2 – number of data (number of packed records)

Codec ID – constant 0x10.

Data length – the length of packet from CodecID to NOD2.

NOD2 should be equal to NOD1.

CRC16 is 4 bytes, but first two are zeroes and last two are CRC-16 calculated for CodecID to NOD2.

Received data:

00000000 4 zeros, 4 bytes

0000009D data length, 4 bytes

10 – Codec ID

02- Number of Data (2 records)

1’st record data

0000013feb55ff74 – Timestamp in milliseconds (1374042849140)

GMT: Wed, 17 Jul 2013 06:34:09 GMT

00 – Priority

GPS Elements

0f0ea850 – Longitude 252618832 = 25,2618832º N

209a6900 – Latitude 546990336 = 54,6990336 º E

00AE – Altitude 174 meters

00B9 – Angle 185º

0B – 11 Visible satellites

0000 – 0 km/h speed

IO Elements

0000 – IO element ID of Event generated (in this case when 0000 – data generated not on event)

07 - Generation type

0A – 10 IO elements in record (total)

05 – 5 IO elements, which length is 1 Byte

0001 – IO element ID = 01

00 – IO element’s value = 0

0002 – IO element ID = 02

00 – IO element’s value = 0

0003 – IO element ID = 03

00 – IO element’s value = 0

0004 – IO element ID = 04

00 – IO element’s value = 0

0120 – IO element ID = 288 (dec)

00 – IO element’s value = 0

02 – 2 IO elements, which value length is 2 Bytes

0018 – IO element ID = 24 (dec)

0000 – IO element’s value

0046 – IO element ID = 70 (dec)

0129 – IO element’s value

02 – 2 IO elements, which value length is 4 Bytes

00C7 – IO element ID = 199 (dec)

00000000 – IO element’s value

0046 – IO element ID = 70 (dec)

00000000 – IO element’s value

01 – 1 IO elements, which value length is 8 Bytes

003E – IO element ID = 62 (dec)

0000000000000000 – IO element’s value

2’st record data

0000015B198C7498000F0DBC502095872F00AE00B90B00000000070A0500010000020

000030000040001200002001800000046012902 00C700000000004C0000000001003E00000

00000000000

02 – Number of Data (2 records)

000009A5 - CRC-16, 4 Bytes (first 2 are always zeros)

- **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 123456789012345 would be sent as 000f333536333037303432343431303133

First two bytes denote IMEI length. In this case 000F means, that IMEI is 15 bytes long.

After receiving IMEI, server should determine if it would accept data from this module. If yes server will reply to module 01 if not 00. Note that confirmation should be sent as binary packet. I.e. 1 byte 0x01 or 0x00.

Then module starts to send first AVL data packet. After server receives packet and parses it, server must report to module number of data received as integer (four bytes).

If sent data number and reported by server doesn't match module resends sent data.

Example:

Module connects to server and sends IMEI:

000f333536333037303432343431303133

Server accepts the module:

01

Module sends data packet:

Codec type	AVL data packet header	AVL data array	CRC
	Four zero bytes, 'AVL data array' length – 254	CodecId – 08 or codec 16, NumberOfData – 2. (Encoded using continuous bit stream. Last byte padded to align to byte boundary)	CRC of 'AVL data array'
Codec8	00000000000000FE	0802...(data elements)...02	00008612
Codec16	00000000000000FE	1002...(data elements)...02	00008612

Server acknowledges data reception (2 data elements):

00000002

Sending data over UDP/IP

- **UDP channel protocol**

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

UDP datagram

UDP channel packet x N	Example	2 bytes	Packet length (excluding this field) in big ending byte order
	Packet Id	2 bytes	Packet id unique for this channel
	Not usable byte	1 byte	Not usable byte
	Packet payload	m bytes	Data payload

Not usable byte

1 Data packet requiring acknowledgment

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

Packet length	2 bytes	0x0003
Packet id	2 bytes	same as in acknowledged packet
Not usable byte	1 byte	0x00

- **Sending AVL data using UDP channel**

AVL data are sent encapsulated in UDP channel packets (Data payload field).

AVL data encapsulated in UDP channel packet

AVL packet id (1 byte) Module IMEI AVL data array

AVL packet id (1 byte) – id identifying this AVL packet

Module IMEI – IMEI of a sending module encoded the same as with TCP

AVL data array – array of encoded AVL data

Server response to AVL data packet

AVL packet id (1 byte) Number of accepted AVL elements (1 byte)

AVL packet id (1 byte) – id of received AVL data packet

Number of AVL data elements accepted (1 byte) – number of AVL data array entries from the beginning of array, which were accepted by the server

Scenario:

Module sends UDP channel packet with encapsulated AVL data packet. 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
Len – 253, Id – 0xCAFE, Not usable byte – 00	AVL packet id – 0xDD, IMEI – 1234567890123456	CodecId – 08, NumberOfData – 2. (Encoded using continuous bit stream)
00FDCAFE01	DD000F313334353637383930313233343 5	0802…(data elements)…02

Server must respond with acknowledgment:

UDP channel header	AVL packet acknowledgment
Len – 5, Id – 0xABCD, Not usable byte – 00	AVL packet id – 0xDD, NumberOfAcceptedData – 2
0005ABCD01	DD02

- **Another example, with all IO id’s enabled**

Server received data:

Data length: 00a1 or 161 Bytes (not counting the first 2 data length bytes)

Packet identification: 0xCAFE 2 bytes

Not usable byte : 00

Packet id: 1b

Imei length: 000f

Actual imei: 333536333037303432343431303133

Codec id: 08

Number of data: 01

Timestamp: 0000013febdd19c8

Priority: 00

GPS data: 0f0e9ff0209a718000690000120000

UDP protocol is the same as TCP except message header is 7 bytes, which consist of: data length, packet identification, not usable byte and packet id.

Then goes imei length and imei itself.

And after that goes AVL data.

And at the very end number of data byte. There is no CRC in UDP.

- **Sending data using SMS**

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

SM data (TP-UD)
AVL data array IMEI: 8 bytes

AVL data array – array of encoded AVL data

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

- **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	Bytes 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

SMS Data Structure

8	Codec ID	Codec ID = 4
35	Timestamp	Time corresponding to the first (oldest) GPS data element, represented in seconds elapsed from 2000.01.01 00:00 EET.
5	ElementCount	Number of GPS data elements

SMS Data Structure

ElementCount *		GPSDataElement	GPS data elements.
		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 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:

- **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

No information available	0	Iceland	1C
Austria	1	Kazakhstan	1D
Albania	2	Luxembourg	1E
Andorra	3	Lithuania	1F
Armenia	4	Latvia	20
Azerbaijan	5	Malta	21
Belgium	6	Monaco	22
Bulgaria	7	Republic of Moldova	23
Bosnia and Herzegovina	8	Macedonia	24
Belarus	9	Norway	25
Switzerland	0A	Netherlands	26
Cyprus	0B	Portugal	27
Czech Republic	0C	Poland	28
Germany	0D	Romania	29
Denmark	0E	San Marino	2A
Spain	0F	Russian Federation	2B
Estonia	10	Sweden	2C
France	11	Slovakia	2D

Finland	12	Slovenia	2E
Liechtenstein	13	Turkmenistan	2F
Faeroe Islands	14	Turkey	30
United Kingdom	15	Ukraine	31
Georgia	16	Vatican City	32
Greece	17	Yugoslavia	33
Hungary	18	RFU	34..FC
Croatia	19	European Community	FD
Italy	1A	Example	FE
Ireland	1B	Rest of the world	FF