Binary Data Protocol

The tracker communicates data packets of the following structure (binary containers) with the server:

Packet header type t_binary_container
Content of packet is arbitrary binary data

In response, the server shall send a data packet of the same structure and arbitrary content to the tracker. In case the tracker receives a valid data packet with the required structure in response to its sending, the data packet is considered to be transmitted successfully and the tracker sends a new data packet. If the response data packet is not received or an invalid data packet is received (the checksum or preamble does not match), the source data packet is considered to be sent with an error and it is sent again.

Data type description:
uint8_t – 8-bit unsigned integer.
int8_t – 8-bit signed integer.

uint16_t – 16-bit unsigned integer. The order of writing bytes is the reverse one (from the low bite to the high one, in the Intel way, like in x86).
int16_t – 16-bit signed integer.

uint32_t - 32-bit unsigned integer.
int32_t - 32-bit signed integer.

All data packets are packed. Structure elements alignment is 1 byte.

Algorithm for calculating the checksum:

```
1 /*
 2   Name  : CRC-16 CCITT
 3   Poly  : 0x1021    x^16 + x^12 + x^5 + 1
 4   Init  : 0xFFFF
 5   Revert: false
 6   XorOut: 0x0000
 7   Check : 0x29B1 ("123456789")
 8   MaxLen: 4095 (32767 bit)
 9 */
10
11 uint16_t crc16(unsigned char *pcBlock, uint16_t len)
```

```
12 {
13     unsigned short crc = 0xFFFF;
14     unsigned char i;
15
16     while(len--)
17     {
18         crc ^= *pcBlock++ << 8;
19
20         for(i = 0; i < 8; i++)
21             crc = crc & 0x8000 ? (crc << 1) ^ 0x1021 : crc << 1;
22     }
23
24     return crc;
25 }
```

Description of the structure **t_binary_container**:

```
1 typedef struct {
2   uint16_t crc;          // crc: sizeof (t_binary_container +
pay_load)
3   uint16_t preamble;
4   uint32_t tracker_id;
5   uint16_t data_len;
6 } t_binary_container;
```

crc – the checksum of the entire data packet starting from the preamble field and ending with the last byte of the packet contents (except for the crc field)
preamble – the preamble. The content is always 0x8A2C;
tracker_id – the tracker ID;
data_len – the data field length;

The maximum length of the binary container is1400 bytes, including the header.

The payload of the binary container is data with the structure **t_common_data_header** followed by the packet payload:

Header - t_common_data_header

Packet Payload

There may be a few of such packets (but there may be not at all).

Description of the structure **t_common_data_header**:

```
typedef struct {
  uint16_t crc;
  uint16_t serial_id;
  uint32_t timestamp;
  uint16_t packet_type;
  uint16_t packet_len;
  uint32_t x_coord;
  uint32_t y_coord;
  uint8_t sf;
} t_common_data_header;
```

crc - the checksum of the packet, including the data block (except crc field);
serial_id – the serial number of the data packet
timestamp - unixtime of the packet (UTC+0)
packet_type - the type of the structure that is the payload of the current packet;
packet_len – the length of the payload;
x_coord - latitude - in accordance with the rules of EGTS protocol. If it is equal to 0xffffffff, it means there is no fixation of valid coordinates;
y_coord - longitude - in accordance with the rules of EGTS protocol. If it is equal to 0xffffffff, it means there is no fixation of valid coordinates;
sf - Additional packet flags - bitmask:

бит 0 – a feature of sending data from the black box (0 - recently collected data, 1 - data sent from the black box);
бит 6 - LAHS field of recording EGTS_SR_POS_DATA; 0 - north latitude; 1 - south latitude;
бит 7 - LOHS field of recording EGTS_SR_POS_DATA; 0 - eastern longitude; 1 - зwestern longitude;

Description of packet types:

B_PACKET_TYPE_RCPTOK_SIMPLE = 0xFFFF - empty packet (to confirm the receipt of the container). No data, field packet_len=0

B_PACKET_TYPE_SENSORS_V1 = 0x0001 contains the data from the tracker sensors:

```
typedef struct {
  uint16_t update_reason;
```

```
 3   uint16_t d_state;
 4   uint16_t v_bat;
 5   uint16_t v_in;
 6   uint16_t v_5v;
 7   uint16_t v_1224v;
 8   uint16_t pwr_flags;
 9   uint16_t acc_x;
10   uint16_t acc_y;
11   uint16_t acc_z;
12   uint8_t  adc_chan_to_send;
13   uint8_t  discreete_count_to_send;
14   uint8_t  ow_count_to_send;
15   uint8_t  ibutton_data_send;
16   t_adc_data adc_data[adc_chan_to_send];
17   t_discreete_data discreete_data[discreete_count_to_send];
18   t_ow_data    ow_data[ow_count_to_send];
19   t_ibutton_data ibutton_data[ibutton_data_send];
20 } t_sensor_data;
```

update_reason - the reason for sending this message is a binary field:

SE_ADC_CHANGED 0x0001 - changes in the analog sensors values are above the critical value
SE_FREQ_CHANGED 0x0002 - changes in the frequency sensors values are above the critical value
SE_ACC_CRITICAL_EVENT 0x0004 - changes in the accelerometer values are above the critical value
SE_IBUTTON_EVENT 0x0008 – ibutton is pressed
SE_DS1820_EVENT 0x0010 - changes in the temperature sensors values are above the critical value
SE_TIMER_EVENT 0x8000 – update timer data.

There may be several reasons for the update. In this case they are combined according to OR;

d_state – the current state of the digital inputs

bit 0: state of the digital input 1 (1 – active; 0 – not active);
bit 1: state of the digital input 2 (1 - active; 0 – not active);
bit 2: state of the digital input 3 (1 - active; 0 – not active);
bit 3: state of the digital input 4 (1 - active; 0 – not active);

bit 4: state of the digital input 5 (1 - active; 0 – not active);
bit 5: state of the digital input 6 (1 - active; 0 – not active);
bit 6: reserved;
bit 7: reserved;

v_bat – battery voltage – 2 decimal places of a volt;
v_in – tracker input voltage - 2 decimal places of a volt;
v_5v - voltage in 5-v chain in the tracker - 2 decimal places of a volt;
v_1224v - voltage in 12-24-v chain - 2 decimal places of a volt;
pwr_flags - the current state of the tracker:

#define POWER_MODE_SLEEP 0x01 – the tracker is in sleep mode
#define POWER_MODE_ACTIVE 0x02 - the tracker is in active mode
#define MODE_ACC_STOP 0x04 – a stop is recorded according to the accelerometer
#define IGNITION_ON 0x08 – ignition is on
#define DEBUG_OUTPUT 0x10 - the output of debugging information to the console is on
#define SENSOR_OUTPUT 0x20 - the output of sensor values to the console is on
#define CAN_DEBUG_OUTPUT 0x40 - output CAN is on
#define NMEA_OUTPUT 0x80 - output NMEA is on

acc_x - axial acceleration X - from 0 to 32767. In the operating range of the accelerometer, which is specified by the variable ACC_RANGE
acc_y - axial acceleration Y - from 0 to 32767.
acc_z - axial acceleration Z - from 0 to 32767.
adc_chan_to_send - the quantity of ADC channels to be sent - from 0 to 2
discrete_count_to_send - the quantity of digital channels to be sent - from 0 to 6
ow_count_to_send - the quantity of thermometer channels to be sent - from 0 to 16
ibutton_data_send – the quantity of ibutton channels to be sent - 0 r 1
adc_data[N1] – ADC data. The quantity of fields depends on the field adc_chan_to_send. May be missing.
discrete_data[N2] – digital sensor data. The quantity of fields depends on the field discrete_count_to_send. May be missing.
ow_data[N3] – temperature digital sensor data. The quantity of fields depends on the field ow_count_to_send. May be missing.
ibutton_data[N4] – ibutton data. May be missing.

Structure Description:

```
 1 typedef struct {

 2   uint16_t adc_data; // Voltage on the corresponding ADC channel - 2
decimal places of a volt

 3 } t_adc_data;

 4

 5 typedef struct {

 6   int32_t freq; // Frequency on the corresponding digital sensor
channel

 7   uint32_t counter; // State (counter of shifts) of the corresponding
digital sensor;
```

```
 8 } t_discrete_data;

 9

10 typedef struct {

11   int16_t ow_data; // The temperature of the corresponding
temperature sensor channel, signed integer, 2 decimal places of a degree

12 } t_ow_data;

13

14 typedef struct {

15   uint8_t  i_button_id[8];  // ID of the key

16   uint32_t i_button_fix_time; // the exact time when the key is
pressed. unixtime UTC+0.

17 } t_ibutton_data;
```

B_PACKET_TYPE_GSM_V1 0x0002 – contains the data on GSM channel

```
 1 typedef struct {

 2   uint8_t module_temperature;  // the GSM module temperature

 3   uint8_t gsm_strength;        // the GSM-signal level0..30 units

 4   uint8_t network_register_state; // State of registration in the
network. 0 - no registration, 1 - home network, 5 - roaming;

 5   uint8_t active_sim_number; // the active SIM-card number

 6   uint8_t active_server_number; // the active server number

 7   char operator_name[8]; // the active mobile operator name (a part
of the name)

 8   uint16_t operator_id; // the mobile operator ID

 9   uint16_t mcc; // Data LBS - MCC

10   char lac[5]; // Data LBS - LAC

11   uint16_t mnc; // Data LBS - MNC

12   uint16_t rxl;  // Data LBS - RXL

13   uint16_t rxq; // Data LBS - RXQ

14   uint16_t ta;  // Data LBS - TA

15   uint16_t bsic;  // Data LBS - BSIC

16   char cell_id[5]; // Data LBS - CellID в HEX (!!!)

17   uint16_t mdm_cmd_flag; // reserved
```

```
18   uint8_t max_idle_counter; // reserved

19   uint8_t gprs_state; // the GPRS-connection state. 0 - not active, 1
- active;

20   uint8_t tcp_state; // the TCP -connection state. 0 - not active. 1
- active;

21   char imei[20]; // IMEI

22   char iccid[21]; // ICCID

23   uint8_t egts_last_error; // The state of the last access to the
EGTS server (see decoding below)

24   uint8_t egts_error_param_1; // The parameter of ЕГТС-server error
#1 (if applicable)

25   uint8_t egts_error_param_2; // The parameter of ЕГТС-server error
#2 (if applicable)

26 } t_gsm_info;
```

B_PACKET_TYPE_AV_V1 0x0003 – GPS data

```
 1 typedef struct {

 2   uint16_t v_in;             // voltage input, 2 decimal places of a
volt

 3   uint16_t v_bat;            // battery voltage, 2 decimal places of
a volt

 4   int16_t fs_data;           // fuel sensor data

 5   uint8_t  mode_acc_stop:4;  // stop mode in the accelerometer. 0 - a
stop is not recorded; 1 – a stop is recorded;

 6   uint8_t  ignition_on:4;    // ignition sensor state. 0 – ignition
is off; 1 – ignition is on;

 7   uint8_t  d_state;          // digital sensor state – see above

 8   uint8_t  fix_type;         // type of fixation GPS/GLONASS;

 9   uint8_t  sat_count;        // number of satellites tracked

10   uint16_t  hdop;            // hdop - 2 decimal places

11   uint16_t altitude;         // altitude of the antenna receiver
above / below the sea level, m;

12   uint16_t geoid_height;     // Geoidal difference is the difference
between the earth ellipsoid WGS-84 and the sea level (geoid), "-" = the
sea level is below the ellipsoid;

13   uint16_t speed;            // speed - 2 decimal places of km/h

14   uint16_t azimuth;          // displacement vector - 2 decimal
places of a degree
```

```
15   int32_t  x_coord;          // Latitude, coding in accordance with
the rules of EGTS; (4294967295 – the coordinate is not valid)

16   int32_t  y_coord;          // longitude, coding in accordance with
the rules of EGTS; (4294967295 - the coordinate is not valid)

17   uint8_t ant_state;         // GPS antenna state. 0 - Ok, 1 - short-
circuit failure, 2 – absent or lost connection.

18   uint8_t  egts_flags;       // EGTS flags to communicate to the
server

19   uint8_t  egts_src;         // The EGTS data source to communicate
to the server

20 } t_gps_data_v4;
```

B_PACKET_TYPE_DUT_E 0x0004 – fuel sensor data

```
 1 typedef struct {

 2   uint8_t  bind_place;  // sensor connection place (0 - RS232_MAIN; 1
- RS232_AUX; 2 - RS485);

 3   char     t;           // fuel temperature;

 4   uint16_t n;           // current fuel level;

 5   uint16_t f_curr;      // current frequency of the internal
electrical generation unit;

 6   uint32_t id;          // sensor ID;

 7   uint16_t f_max_t;     // frequency reading at maximum fuel level;

 8   uint16_t f_min_t;     // Frequency reading at minimum fuel level;

 9   uint16_t k_t;         // reserved;

10   char     k_t0;        // reserved;

11   uint16_t pwm_max;     // reserved;

12   uint16_t trl;         // reserved;

13   char net_addr;        // network address of the sensor;

14   char net_mode;        // sensor operating mode - 0 - single; 1 -
network;

15   char pwm_mode;        // reserved

16 } t_dut_e_struct;
```

B_PACKET_TYPE_CAN_V1 0x0005 - CAN-bus data

NB!!! The data packet contains 4 packets **t_can_data**

```
1 typedef struct {
```

```
2   uint32_t timestamp; // packet capture time - unixtime

3   uint32_t can_id;    // can_id of the packet

4   uint8_t  can_data[9];  // can-packet data (can_data[0] - packet
length; can_data[1]...can_data[8] – packet data)

5 } t_can_data;
```

PACKET_TYPE_ALARM_DATA 0x000A - alarm button data

```
1 typedef struct {

2   uint16_t gps_speed;             // current speed when an alarm event
occurred

3   uint16_t gps_angle;             // direction when an alarm event
occurred

4   uint32_t x_coord;               // latitude - 2 decimal places of a
degree

5   uint32_t y_coord;               // longitude - 2 decimal places of a
degree

6   uint8_t  alarm_state;           // 1 – alarm state; 0 – no alarm
state

7 } t_alarm_data_v1;
```

PACKET_TYPE_FW_ASK 0x0006 – Request the firmware block

```
1 typedef struct {

2   unsigned char cur_fw_version[12]; - current firmware version

3   unsigned char req_fw_version[12]; - requested firmware version. If
0x00, the latest available version is requested

4   unsigned char serial[12];         - serial number of the device

5   unsigned char device_type;        - type of the device

6   uint32_t      fw_flash_ptr;       - offset when the firmware is
requested

7 } t_fw_ask_data;
```

PACKET_TYPE_FW_PAGE_DATA 0x0007 – firmware block

```
1 #define FW_PAGE_DATA_SIZE  1028

2 typedef struct {

3   uint32_t offset;                 - offset of the block

4   uint32_t fw_size;                - total firmware size
```

```
5   unsigned char fw_version[12];    - firmware version

6   unsigned char device_type;       - device the firmware is designed
for

7   uint8_t  fw_page_data[FW_PAGE_DATA_SIZE]; - firmware block

8 } t_fw_page_data;
```

PACKET_TYPE_TEXT_CMD_DATA - 0x0008 - text command to the device from the command stack

```
1 typedef struct {

2   char     cmd_id[16]; - command ID

3   char     cmd[160];   - command to the device

4 } t_text_cmd_data;
```

When this command is received, the terminal executes it and forwards the result of its execution alongside the same packet.

PACKET_TYPE_CONFIG_POLL - 0x0009 - data request from the configuration server

```
1 typedef struct {

2   uint8_t serial[12]; - device serial number

3 } t_poll_data;
```

The tracker requests data from the configuration server. In response, the terminal expects PACKET_TYPE_TEXT_CMD_DATA structure in the data packet or an empty packet if there are no commands to this terminal.

PACKET_TYPE_DEBUG - debugging information

```
 1 typedef struct {

 2   uint32_t last_reset_state; - reason for the last reboot

 3   uint32_t flash_rd;         - memory pointer to reading

 4   uint32_t flash_wr;         - memory pointer to recording

 5   uint32_t uptime_cnt;       - uptime counter

 6   t_gps_stat gps_stat;       - statistics on the operation of the GPS
module

 7 } t_debug_packet;

 8

 9 typedef struct {

10   uint32_t crc_err_cnt; //+    L
```

```
11   uint32_t no_data_cnt;    //+ L
12   uint32_t not_processed_cnt; //+  L
13   uint8_t gps_power_state;   //+   C
14   uint32_t gps_power_cycles_counter;//+  L
15   uint32_t angle_int; //+               L
16   uint32_t speed_int; //+               L
17   uint8_t gps_ok;     //+               C
18   uint8_t stop_state; //+               C
19   uint8_t sleep_state; //+              C
20   uint8_t freeze_acc; //+               C
21   uint8_t freeze_ign; //+               C
22   uint8_t sat_count;  //+               C
23   uint8_t fix_type;   //+               C
24   uint32_t send_timer;    //+           L
25   uint32_t send_interval; //+           L
26   uint16_t speed_change;  //+           S
27   uint16_t angle_change;  //+           S
28   uint8_t last_gps_event; //+           C
29   uint8_t has_gps_fix; //+              C
30 } t_gps_stat;
31
```

PACKET_TYPE_CAN_LOG_DATA - 0x000D - CANLog data (daughter extension board or module connected via RS232/485 interface);

```
1 typedef struct {
2   t_can_log_frame can_log_frames[N];
3 } t_can_log_data;
4
5 typedef struct {
6   char     prefix[2]; // S,
A,B,C,D,E,F,G,R,H,I,J,K,L,M,N,O,P,U,V,WA,WB,WC,WD,WE,WF,WG,WH,TA,TB,TC,T
D,TE,XA,XB,Z
7   uint32_t data;
```

```
8 } t_can_log_frame;
```

Data packet t_can_log_data consists of several t_can_log_frame packets.

The exact number of packets in the message is calculated using the formula t_common_data_header.packet_len/sizeof(t_can_log_frame). The maximum number of packets t_can_log_frame is limited to 40;

Packet interpretation:

The prefix field specifies the contents of the packet. The field can consist of one or two characters. In case the character is one, the second character is 0x00.

Field data - data corresponding to the content

prefix = S - security state flags. data - security state flags (bit field) – find the description in the attached file;
A или B - total engine operating time. data – in 2 decimal places of an hour;
C или D – total vehicle mileage. data – in meters;
E или F – total fuel consumption. data - in 1 decimal place of a liter;
G – fuel level in the tank in %. data - in 2 decimal places of a %;
R – fuel level in the tank in liters. data - in 2 decimal places of a liter;
H – engine speed per minute (rpm). data – in rpm;
I – engine temperature (degrees C). data – in degrees C;
K - load on the axle 1. data - in 1 decimal place of a kilo;
L - load on the axle 1. data - in 1 decimal place of a kilo;
M - load on the axle 1. data - in 1 decimal place of a kilo;
N - load on the axle 1. data - in 1 decimal place of a kilo;
O - load on the axle 1. data - in 1 decimal place of a kilo;
P – accident controllers. data - (bit field) – find the description in the attached file;
U - adBlue liquid level in the tank in %. data - in 1decimal place of a %;;
V - adBlue liquid level in the tank in liters. data - in 1 decimal place of a liter;
WA – agricultural equipment state. data – bit mask
WB – reaper time. data - in 2 decimal places of an hour;
WC - harvested acreage. data - in 2 decimal places of a hectare
WD - productivity. data - in 2 decimal places of a hectare per hour
WE – amount of the harvest. data - in 2 decimal places of a ton
WF - crop moisture. data - in 1 decimal place of a %;
WG – beater drum speed. data - rpm
WH – concave gap in output. data - mm
TA,TB,TC,TD,TE - information from these CANLog frames is not processed
XA - throttle pedal position. data – in %
XB - engine load. data – in %

B_PACKET_TYPE_TEXT_DATA - 0x000F - contains ASCIIZ, received from 232/485 port.

```
1 typedef struct {

2   char data[N];

3 } t_str_data;
```

Field length is communicated in the header t_common_data_header.

PACKET_TYPE_FMS_DATA - 0x0011 - datd FMS (from CAN-bus);

```
1 typedef struct {
2   t_fms_field fms_frames[N];
3 } t_fms_data;
4
5 typedef struct {
6   char     field_name[12];
7   uint32_t field_value;
8 } t_fms_field;
```

Data packet t_fms_data consists of several t_fms_field packets

The exact number of packets in the message is calculated using the formula t_common_data_header.packet_len/sizeof(t_fms_field). The maximum number of packets t_fms_field is limited to 40;

Packet interpretation:

field_name (ASCIIZ) determines the packet content.
data field – data corresponding to the content

Filed description:

fuel_lfc - fuel consumption, ltr
fuel_prc - fuel level, %
torq_prc - Engine torque, %
rpm - Engine speed, rpm
hours - Engine total hours or operation, hours
mileage - High resolution total mileage, m
engine_tmp - Engine temperature, C
amb_tmp - Abmient temperature, C
fuel_rate - Fuel rate l/hour
fuel_eco - Fuel economy km/l
air_pres - Supply air pressure, kpa
fuel_hrlfc - Total fuel used, l
adblue_prc - Aftertreatment 1 Diesel Exhaust Fluid Tank 1 level, %
fms1_1 - FMS Tell Tale Status: FMS1 P1-P4
fms1_2 - FMS Tell Tale Status: FMS1 P5-P8
prk_brk - Parking brake state
speed - Wheel based speed, km/h
pedal_state - b8-7: clutch switch/b6-5: brake switch/b2-1: cruise control state
pto_state - PTO state
acc_pos - Accelerator pedal position, %

engine_load - Engine load, %
axle_tire_l - Axle/tire location
axle_wght - Axle weight, kg
srv_dst - Service distance, km
pto_de - PTO Drive Engagement: PTODE
gross_wght - Gross weight, kg
rtrd_mode - Retarder mode
rtrd_torq - Retarder torque, %
rtrd_sel - Retarder selection non-engine, %
door_st1 - Doors state 1
door_st2 - Doors state 2
door_st3 - Doors state 3
fms_time - FMS time
fms_date - FMS date
alt_state - Alternator state
sel_gear - Selected gear
cur_gear - Current gear
b_pres_fl - Below pressure front left, kpa
b_pres_fr - Below pressure front right, kpa
b_pres_rl - Below pressure rear left, kpa
b_pres_rr - Below pressure rear right, kpa

Packets numbers and their identifiers:

#define B_PACKET_TYPE_SENSORS_V1 0x0001
#define B_PACKET_TYPE_GSM_V1 0x0002
#define B_PACKET_TYPE_AV_V1 0x0003
#define B_PACKET_TYPE_DUT_E 0x0004
#define B_PACKET_TYPE_CAN_V1 0x0005
#define B_PACKET_TYPE_FW_ASK_V1 0x0006
#define B_PACKET_FW_PAGE_DATA_V1 0x0007
#define B_PACKET_TYPE_TEXT_CMD_DATA 0x0008
#define B_PACKET_TYPE_CONFIG_POLL 0x0009
#define B_PACKET_TYPE_ALARM_DATA 0x000A
#define B_PACKET_TYPE_DEBUG 0x000B
#define B_PACKET_TYPE_IMAGE_DATA 0x000C
#define B_PACKET_TYPE_CAN_LOG_DATA 0x000D
#define B_PACKET_TYPE_TEXT_DATA 0x000F

Example of data packet:

t_binary_container
-- t_common_data_header
---- t_gps_data_v4
-- t_common_data_header
---- t_sensor_data

-- t_common_data_header
---- t_gsm_info

Binary Data Protocol

This protocol is in addition to Binary Data Protocol PRO. Only packages that are missing in Binary Data Protocol PRO are described below.

Packet Type 0x0040 - information about GPS-coordinates

```
1 typedef struct {

 2   uint16_t v1224; - battery voltage, 2 decimal places of a volt

 3   uint16_t v_bat; - tracker battery voltage - 2 decimal places of a
volt;

 4   uint16_t fs_data; - data from fuel sensor 1, channel RS232 or RS485

 5   uint8_t  stop_state; - 1 if vehicle is at standstill; 0 – if
vehicle is moving;

 6   uint8_t  ign_state; - 1 ignition is on; 0 – ignition is off;

 7   uint8_t  d_state; - digital input state – bit mask;

 8   uint16_t freq2; - frequency of digital input 2;

 9   uint16_t c_counter2; - impulse counter at digital input 2;

10   uint16_t freq1; - frequency of digital input 1;

11   uint16_t c_counter1; - impulse counter at digital input 1;

12   uint8_t  ant_state; - GPS antenna state;

13   uint8_t  fix_type; - type of satellite fixation;

14   uint8_t  sat_count; - number of satellites tracked;

15   uint16_t altitude; - heigth;

16   uint16_t geoid_height; - geoidal height;

17   uint32_t x_coord; - latitude, coding in accordance with the rules
of EGTS;

18   uint32_t y_coord; - longitude, coding in accordance with the rules
of EGTS;

19   uint16_t speed; - 2 decimal places of kilometers per hour;

20   uint16_t course; - angle of azimuth, degrees;

21   uint16_t adc1; - ADC 1 data - 2 decimal places of a volt;

22   uint16_t c_counter3; - digital input counter 3;

23   int16_t  ow_data; - 1-wire N1 sensor data, 2 decimal places of a
degree;

24   uint8_t  egts_flags; - EGTS flags;
```

```
25   uint8_t  egts_src; - EGTS data source;

26 } t_l2b_gps_info;
```

Packet type 0x0041 –GSM-network data

```
 1 typedef struct {

 2   char operator_name{8}; operator name. ASCIIZ line. Maximum 7
characters + 1 terminator

 3   uint8_t active_sim_number; - active SIM-card number(1 or 2)

 4   uint8_t active_server_number; - active server number (1 or 2)

 5   uint8_t network_register_state; - GSM-network registration
state(0,2 – not registered 1 – registered in home network; 5 - roaming)

 6   uint8_t gsm_strength; - GSM-signal strength - from 0 to 31 units

 7   uint8_t module_temperature; -  GSM-module temperature in degrees

 8   uint16_t operator_id; - mobile operator ID

 9   char lac{5}; LAC - ASCIIZ line

10   uint16_t mnc; - MNC

11   char cell_id{5}; - CELL_ID - ASCIIZ line;

12   uint16_t mcc; - MCC

13   uint16_t bsic;- BSIC

14   uint16_t rxl; -  RXL

15   uint16_t rxq; - RXQ

16   uint16_t ta; - TA

17 } t_l2b_gsm_info;
```

Packet type 0x0042 – sensor data

```
1 typedef struct {

2   uint16_t acc_max_x; - accelerometer data, axis X

3   uint16_t acc_max_y; - accelerometer data, axis Y

4   uint16_t acc_max_z; - accelerometer data, axis Z

5   uint16_t fuel_data{4}; - 4 fuel sensor RS485 data

6   uint32_t uptime_cnt; - time since tracker actuation, sec

7 } t_l2b_sd_line;
```

Packet type 0x0043 - debugging information

```
 1 typedef struct {
 2   uint32_t uptime_cnt; - time since tracker actuation, sec
 3   uint32_t wr_flash_address; - write address pointer in flash-memory
 4   uint32_t rd_flash_address; - read address pointer from flash-memory
 5   uint32_t rcc_csr; - tracker actuation flags
 6   char imei{20}; - modem IMEI
 7   char iccid{21}; - SIM-card ICCID
 8   uint16_t vcc; - 3.3V bus voltage, 2 decimal places of a volt
 9 } t_l2b_debug_info_line;
```

Packet type 0x0044 - 1-wire sensor data

```
1 typedef struct {
2   t_single_ow_data ow_data{4}; - 4 temperature sensor data
3   uint32_t i_button_fix_time; - time of last iButton reading, unixtime
4   uint8_t i_button_id{8}; - button ID - 8 bytes
5 } t_l2b_ow_info_line;
```

Structure t_single_ow_data is described below:

```
1 typedef struct {
2   uint8_t ow_id; - temperature sensor identifier – last byte OW-ID
sensor
3   int16_t ow_temp; - temperature, 2 decimal places of a degree
4 } t_single_ow_data;
```

Packet type 0x0045 – route minimal data

```
1 typedef struct {
2   uint8_t  ant_state;  - GPS-antenna state
3   uint8_t  fix_type;   - satellite fixation type
4   uint8_t  sat_count;  - number of satellites to connect
5   uint16_t speed;      - current speed - 2 decimal places of a km/h
6   uint16_t course;     - current displacement vector
7   uint8_t egts_flags;  - EGTS flags
8 } t_l2b_gps_info_min;
```

Packet type 0x004A - GPS-module expanded data

```
1 typedef struct {

2   uint8_t sv_gps;       - number of visible GPS satellites

3   uint8_t sv_glonass;   - number of visible GLONASS satellites

4   uint8_t sc_gps;       - number of tracked GPS satellites

5   uint8_t sc_glonass;   - number of tracked GLONASS satellites

6   uint8_t snr_min;      - minimal value SNR of tracked satellites,
dbHz

7   uint8_t snr_max;      - maximal value SNR of tracked satellites,
dbHz

8 } t_l2b_gps_ex_info;
```

Packet type 0x004B - т GPS-Lite universal data packet

```
 1

 2 typedef struct {

 3   uint32_t flags1;       // bit field that specifies the presence of
group 0 fields in the packet

 4   uint8_t  ant_state;    // fields from here

 5   uint8_t  fix_type;

 6   uint8_t  sat_count;

 7   uint16_t speed;

 8   uint16_t course;

 9   uint8_t egts_flags;   // to here; are always present in the packet

10   // remaining fields are present in the packet only if the
corresponding bit is set in the flags1 or flags2 field

11   uint32_t flags2;            // 0.0 - bit field that defines the
presence of group 1 fields in the packet

12   t_altitude_info altitude_info; // 0.1 – altitude data

13   uint16_t v1224;        // 0.2 - tracker input power, 2 decimal
places of a volt

14   uint16_t v_bat;        // 0.3 - tracker battery power, 2 decimal
places of a volt

15   uint16_t adc1;         // 0.4 – tracker analog input 1 power, 2
decimal places of a volt

16
```

```
17   uint16_t freq1;        // 0.5 - frequency measured at frequency
input 1, Hz

18   uint16_t freq2;        // 0.6 - frequency measured at frequency
input 2, Hz

19   uint16_t c_counter1;   // 0.7 - number of impulses counted on
discrete input 1, pcs

20   uint16_t c_counter2;   // 0.8 - number of impulses counted on
discrete input 2, pcs

21   uint16_t c_counter3;   // 0.9 - number of impulses counted on
discrete input 3, pcs

22

23   uint16_t fuel_data1;   // 0.10 – 1st fuel sensor value RS485/232

24   uint16_t fuel_data2;   // 0.11 – 2nd fuel sensor value RS485

25   uint16_t fuel_data3;   // 0.12 – 3rd fuel sensor value RS485

26   uint16_t fuel_data4;   // 0.13 – 4th fuel sensor value RS485

27

28   uint8_t  stop_state;   // 0.14 - 1 - tracker is at rest, 0 -
tracker is in motion

29   uint8_t  ign_state;    // 0.15 - 1 – ignition is on, 0 - ignition
is off

30   uint8_t  d_state;      // 0.16 – digital input state (bit mask)

31   uint8_t  egts_src;     // 0.17 – EGTS data source

32

33   uint16_t acc_data[3];  // 0.18, wrray with accelerometer readings,
axes X,Y,Z

34

35   t_single_ow_data ow_data1; // 0.19 - number and results of
measurements of temperature sensor 1 (see the description of
t_single_ow_data type below)

36   t_single_ow_data ow_data2; // 0.20 - number and results of
measurements of temperature sensor 2

37   t_single_ow_data ow_data3; // 0.21 - number and results of
measurements of temperature sensor 3

38   t_single_ow_data ow_data4; // 0.22 - number and results of
measurements of temperature sensor 4

39

40   t_ibutton_data   ibut_data; // 0.23 - iButton data see the
description of t_ibutton_data below)
```

```
41
42   char operator_name[8];                    // 0.24 - ASCIIZ line –
current mobile operator name
43   uint8_t active_sim_number;                // 0.25 – current SIM-
card number (0 – SIM-card 1 number, 1 - SIM-card 2 number)
44   uint8_t active_server_number;             // 0.26 – current server
number (0/1)
45   uint8_t network_register_state;           // 0.27 – network
registration state (0 – not registered, 1 – registered in home network,
5 – registered in roaming)
46   uint8_t gsm_strength;                     // 0.28 – GSM-signal
power(0 - minimum, 31 - maximum)
47   uint8_t module_temperature;               // 0.29 - GSM-module
temperature
48   uint16_t operator_id;                     // 0.30 - operator_id
49   uint32_t uptime_cnt;                      // 0.31 – tracker
operating time since last start / restart, sec
50
51   char lac[5];                              //1.0 - lac (ASCIIZ)
52   uint16_t mnc;                             //1.1 - mnc
53   char cell_id[5];                          //1.2 - cell_id
(ASCIIZ)
54   uint16_t mcc;                             //1.3 - mcc
55   uint16_t bsic;                            //1.4 - bsic
56   uint16_t rxl;                             //1.5 - rxl
57   uint16_t rxq;                             //1.6 - rzq
58   uint16_t ta;                              //1.7 - ta
59   uint32_t distance_travalled;              //1.8 - пробег ТС
60 } t_universal_data_full;
61
62 typedef struct {
63   int16_t altitude;
64   int16_t geoid_height;
65 } t_altitude_info;
66
67 typedef struct {
```

```
68   uint8_t ow_id;
69   int16_t ow_temp;
70 } t_single_ow_data;
71
72 typedef struct {
73   uint8_t  i_button_id[8];
74   uint32_t i_button_fix_time;
75 } t_ibutton_data;
```

Bits that are included in flags1 and flags2 specify the presence of the corresponding fields in t_universal_data_full structure.

For example:

- bit 0 of flags1 field specifies the presence of field 0.0 (flags2). If flags2 is absent, all the fields of group 1 are absent. * (lac, mnc, cell_id, etc.)

- bit 1 of field flags1 specifies the presence of field 0.1 (altitude_info);

- bit 2 of flags1 field specifies the presence of field 0.2 (v1224);

- bit 3 of flags1 field specifies the presence of field 0.3 (v_bat);

and so on.

Packet type 0x004C - GPS-Lite3 universal packet:

Heading:

```
1 typedef struct {
2   uint8_t  ant_state;    // fields from here
3   uint8_t  fix_type;
4   uint8_t  sat_count;
5   uint16_t speed;
6   uint16_t course;
7   uint8_t  reason;   // to here, are always present in the packet
8 } t_l4_data_base;
```

This packet is followed by arbitrary set of fields with the following structure:

field type 1 - 1 byte
field data 1 – length depends of field type

field type 2 - 1 byte
field data 2 - length depends of field type

If the field type is 255, there is no further data - the tracker has not accumulated data in order to fill all the fields completely.

Thus, each packet can contain up to 255 fields.

The type and name of the fields are still static and are determined by the following structure:

```
1 typedef struct {
2   uint8_t field_idx;
3   uint8_t field_len;
4   char    field_name[12];
5 } t_field_def;
6
```

So far the following types of fields have been defined:

```
 #define FIELD_TYPE_8B     1
 #define FIELD_TYPE_16B    2
 #define FIELD_TYPE_32B    4
 #define FIELD_TYPE_FLOAT  4
 #define FIELD_TYPE_16Z    16
 #define FIELD_TYPE_20Z    20


 const t_field_def field_defs[]={
   //                    012345678901
0, FIELD_TYPE_16B,   "alt",   // *      //3 #alias:altitude
  1, FIELD_TYPE_16B,   "v_in",  // *      //3 #alias:innervoltage
  2, FIELD_TYPE_8B,    "ign_state", // *  //2 #alias:ign
  3, FIELD_TYPE_16B,   "vbat",  // *      //3 #alias:battery
  4, FIELD_TYPE_16B,   "adc1",  // *      //3
  5, FIELD_TYPE_16B,   "adc2",  // *      //3
  6, FIELD_TYPE_16B,   "freq1",           //3 #alias:freq_data_1
```

```
7, FIELD_TYPE_16B,   "freq2",           //3 #alias:freq_data_2
8, FIELD_TYPE_16B,   "counter1",        //3 #alias:c_data_1
9, FIELD_TYPE_16B,   "counter2",        //3 #alias:c_data_2
10, FIELD_TYPE_16B,  "counter3",        //3 #alias:c_data_3
11, FIELD_TYPE_8B,   "stop_state", // * //2
12, FIELD_TYPE_8B,   "d_state",         //2 #alias:sensordata
13, FIELD_TYPE_8B,   "snr_min",         //2
14, FIELD_TYPE_8B,   "snr_max",         //2
15, FIELD_TYPE_16B,  "ts_datai",        //2
16, FIELD_TYPE_16B,  "ts_data_0",       //2 #alias:ts_data
17, FIELD_TYPE_16B,  "ts_data_1",       //2 #alias:ow2_temp
18, FIELD_TYPE_16B,  "ts_data_2",       //2 #alias:ow3_temp
19, FIELD_TYPE_16B,  "ts_data_3",       //2 #alias:ow4_temp
20, FIELD_TYPE_32B,  "ibut_time",       //4 #alias:i_button_time
21, FIELD_TYPE_8A,   "ibut_id",         //8 #alias:i_button_id
22, FIELD_TYPE_16B,  "vbat_prc",        //8
23, FIELD_TYPE_16B,  "v_1224",          //8 #alias:innervoltage
24, FIELD_TYPE_32B,  "milage",          //8 #alias:milage


94, FIELD_TYPE_8B,  "fueltemp0",       //3
95, FIELD_TYPE_8B,  "fueltemp1",       //3
96, FIELD_TYPE_8B,  "fueltemp2",       //3
97, FIELD_TYPE_8B,  "fueltemp3",       //3
98, FIELD_TYPE_8B,  "fueltemp4",       //3
99, FIELD_TYPE_16B,  "fueldata0",       //3
100, FIELD_TYPE_16B, "fueldata1",       //3  #alias:fs_data_1
101, FIELD_TYPE_16B, "fueldata2",       //3  #alias:fs_data_2
102, FIELD_TYPE_16B, "fueldata3",       //3  #alias:fs_data_3
103, FIELD_TYPE_16B, "fueldata4",        //3 #alias:fs_data_4


109, FIELD_TYPE_16B, "acc_data_x",       //3 #alias:acc_x
```

```
110, FIELD_TYPE_16B, "acc_data_y",       //3 #alias:acc_y
111, FIELD_TYPE_16B, "acc_data_z",       //3 #alias:acc_z


140, FIELD_TYPE_32B, "can_log_s",
141, FIELD_TYPE_32B, "can_log_a",
142, FIELD_TYPE_32B, "can_log_b",
143, FIELD_TYPE_32B, "can_log_c",
144, FIELD_TYPE_32B, "can_log_d",
145, FIELD_TYPE_32B, "can_log_e",
146, FIELD_TYPE_32B, "can_log_f",
147, FIELD_TYPE_32B, "can_log_g",
148, FIELD_TYPE_32B, "can_log_r",
149, FIELD_TYPE_32B, "can_log_h",
150, FIELD_TYPE_32B, "can_log_i",
151, FIELD_TYPE_32B, "can_log_j",
152, FIELD_TYPE_32B, "can_log_k",
153, FIELD_TYPE_32B, "can_log_n",
154, FIELD_TYPE_32B, "can_log_o",
155, FIELD_TYPE_32B, "can_log_p",
156, FIELD_TYPE_32B, "can_log_u",
157, FIELD_TYPE_32B, "can_log_v",
158, FIELD_TYPE_32B, "can_log_wa",
159, FIELD_TYPE_32B, "can_log_wb",
160, FIELD_TYPE_32B, "can_log_wc",
161, FIELD_TYPE_32B, "can_log_wd",
162, FIELD_TYPE_32B, "can_log_we",
163, FIELD_TYPE_32B, "can_log_wf",
164, FIELD_TYPE_32B, "can_log_wg",
165, FIELD_TYPE_32B, "can_log_wh",
166, FIELD_TYPE_32B, "can_log_ta",
167, FIELD_TYPE_32B, "can_log_tb",
```

```
168, FIELD_TYPE_32B, "can_log_tc",
169, FIELD_TYPE_32B, "can_log_td",
170, FIELD_TYPE_32B, "can_log_te",
171, FIELD_TYPE_32B, "can_log_xa",
172, FIELD_TYPE_32B, "can_log_xb",
173, FIELD_TYPE_32B, "can_log_l",
174, FIELD_TYPE_32B, "can_log_m",
175, FIELD_TYPE_32B, "fm_fuel_lfc", // // #alias:fuel_lfc
176, FIELD_TYPE_32B, "fm_fuel_prc", // // #alias:fuel_prc
177, FIELD_TYPE_32B, "fms_rpm", // // #alias:rpm
178, FIELD_TYPE_32B, "fms_hours", // // #alias:hours
179, FIELD_TYPE_32B, "fms_milage", // // #alias:milage
180, FIELD_TYPE_32B, "fms_eng_tmp", // // #alias:engine_tmp
181, FIELD_TYPE_32B, "fms_amb_tmp", // // #alias:amb_tmp
182, FIELD_TYPE_32B, "fms_fuel_rt", // // #alias:fuel_rate
183, FIELD_TYPE_32B, "fms_fl_hrlf", // // #alias:fuel_hrlfc
184, FIELD_TYPE_32B, "fms_speed", // // #alias:speed
185, FIELD_TYPE_32B, "fms_eng_loa", // // #alias:engine_load
186, FIELD_TYPE_32B, "fms_axl_wgt", // // #alias:axle_wght
187, FIELD_TYPE_32B, "fms_srv_dst", // // #alias:srv_dst
188, FIELD_TYPE_32B, "fms_gros_wg", // // #alias:gross_wght
189, FIELD_TYPE_32B, "fms_taho_st", // // #alias:taho_st
190, FIELD_TYPE_32B, "fms_tah_spd", // // #alias:taho_speed
191, FIELD_TYPE_32B, "iqf_mt", // // #alias:iqf_mt
192, FIELD_TYPE_32B, "iqf_sp", // // #alias:iqf_sp
193, FIELD_TYPE_32B, "iqf_ambt", // // #alias:iqf_ambt
194, FIELD_TYPE_32B, "iqf_afzt", // // #alias:iqf_afzt
195, FIELD_TYPE_32B, "iqf_rpm", // // #alias:iqf_rpm
196, FIELD_TYPE_32B, "iqf_conf", // // #alias:iqf_conf
197, FIELD_TYPE_32B, "iqf_state", // // #alias:iqf_state
198, FIELD_TYPE_32B, "iqf_dr", // // #alias:iqf_dr
```

```
199, FIELD_TYPE_32B, "iqf_batv", // // #alias:iqf_batv

200, FIELD_TYPE_20Z, "imei",

201, FIELD_TYPE_20Z, "iccid1",

202, FIELD_TYPE_20Z, "iccid2",

203, FIELD_TYPE_8B,  "sim_num",    // #alias:sim_number

204, FIELD_TYPE_8B,  "srv_num",    // #alias:active_server

205, FIELD_TYPE_16Z, "op_name",

206, FIELD_TYPE_16B, "lac",

207, FIELD_TYPE_32B, "cell_id"

208, FIELD_TYPE_8B,  "gsm_power",  // #alias:gsmstrength


245, FIELD_TYPE_32B, "iqf_bata", // // #alias:iqf_bata

246, FIELD_TYPE_32B, "iqf_alc", // // #alias:iqf_alc

247, FIELD_TYPE_32B, "iqf_al1", // // #alias:iqf_al1

248, FIELD_TYPE_32B, "iqf_al2", // // #alias:iqf_al2

249, FIELD_TYPE_32B, "iqf_al3", // // #alias:iqf_al3

250, FIELD_TYPE_32B, "iqf_in"  // // #alias:iqf_in
```

Example of field data (no heading):

0B 00 03 EB 05 01 5A 09 02 01 00 5B 00 04 60 00 05 60 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

0B = 11 - field type 11 - stop_state length - 1 byte value 00
03 = 3 field type 3 - vbat - length - 2 bytes value 0x05EB = 1515
01 = 1 field type 1 - v_in length - 2 bytes value 0x095A = 2394
02 = 2 field type 2 - ign_state length 1 byte value 1
00 = 0 field type 0 - alt length 2 bytes value 005B = 91
04 = 4 field type 4 = adc1 length 2 bytes value 0060 = 96
05 = 5 field type 5 = adc2 length 2 bytes value 0060 = 96

FF = no further data. You can stop parsing the packet. It also makes no sense to parse the packet further if an unknown type of field is encountered.

The order of the fields in the packet can be arbitrary, and the total length of the packet will correspond to the configured set of fields.

Since the list of fields in the 0x004C packet will expand, in order not to interrupt the parsing of the packets in the monitoring system, it is necessary to perform 3 postulates when parsing packetss:

1. If the t_binary_container structure is correct (the preamble and the checksum of the whole package match), the packet is confirmed - the tracker considers that everything is correct and prepares the next packet for sending.

2. If the structure t_common_data_header inside t_binary_container is correct (the preambles and checksums match), but the packet_type is unknown to the monitoring system, you should proceed to parsing the next packet in the container (packet_len field allows this). If the structure t_common_data_header is incorrect, the parsing of the container finishes here.

3. If the monitoring system has encountered an unknown field inside the packet B_PACKET_TYPE_L4_UNIVERSAL (0x004C), the parsing of the packet finishes here, but all the fields that have been parsed are added to the database. It should be taken into consideration that a package of this type can be completely empty. In this case, only fields from t_l4_data_base are added to the database.

Packet type 0x81 - DDD tachograph file.

```
typedef struct {
  uint16_t file_id;    // card file id.
  uint8_t  file_type;  // card file type
  uint16_t file_len;   // total file lenght
  uint8_t  count_mes;  // number of times the file has been sent
  uint8_t  number_mes; // sending number
} t_taho_data_header;

```

Data Compression

The new device SAT-LITE 3 supports the compression of the data being sent.
The t_binary_container packet is completely compressed by the LZ algorithm. Such a structure (all packed) clings to the resultant binary array on the top:

```
typedef struct {
  uint16_t crc;
  uint16_t preamble;
  uint16_t data_len;
} t_compressed_header;
```

Other preambule:

```
#define B_COMPRESSED_PREAMBLE  0x9b3d
```

Crc is the checksum of the compressed packet
Data_len - length in compressed form

I.e. the following things are transmitted to the server first:
T_compressed_header
Then compressed t_binary_container

Accordingly, the algorithm is as follows:

1. Verify that the preamble corresponds to the encrypted packet
2. Check the checksum of the packet
3. Decrypt (unpack) the data
4. Process the resulting package as usual

Example of a compressed package:
[26.01.17 18:28:24] Raw Data (binary): 9A 60 **3D 9B** **87 00** 03 EF FA 2C 8A 70 11 01 00 3F 01 67 D1 3D 53 61 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 0A 00 00 00 00 00 00 E8 3E 53 66 03 12 1D 09 03 05 1C C1 B3 3F 53 6B 03 18 1D DF 69 40 53 70 03 18 1D 1C F1 41 53 75 03 18 74 99 C5 42 53 7A 03 18 3A BA 93 43 53 7F 03 18 1D 2A 69 44 53 84 03 18 57 A0 80 46 53 89 03 12 1D 0C 03 05 1D 3A 3C 4F 53 B1 03 0F 1D 00 00 02 0B 03 05 1D 54 73 50 53 B6 03 18 1D

After unpacking, we get:

[26.01.17 18:28:24] Uncompressed Data (binary): EF FA 2C 8A 70 11 01 00 3F 01 67 D1 3D 53 61 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 0A 00 00 00 00 00 00 E8 3E 53 66 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 09 00 00 00 00 00 C1 B3 3F 53 6B 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 09 00 00 00 00 00 DF 69 40 53 70 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 09 00 00 00 00 00 1C F1 41 53 75 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 0A 00 00 00 00 00 99 C5 42 53 7A 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 09 00 00 00 00 00 BA 93 43 53 7F 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 09 00 00 00 00 00 2A 69 44 53 84 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 0A 00 00 00 00 00 A0 80 46 53 89 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 0C 00 00 00 00 00 3A 3C 4F 53 B1 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 00 00 02 0B 00 00 00 00 00 54 73 50 53 B6 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 00 00 02 0B 00 00 00 00 00

Next, we process it as an ordinary Light-3 package.

The compression library is described in the files LZ.c, LZ.h.