Протокол бинарного формата данных

Трекер обменивается с сервером пакетами следующей структуры (бинарные контейнеры):

Заголовок пакета тип t_binary_container
Содержимое пакета - произвольные бинарные данные

В ответ сервер должен прислать трекеру пакет такой же структуры и пустым содержимым. В случае если трекер получает корректный пакет с данной структурой в ответ на свою посылку пакет считается переданным успешно и трекер переходит к отправке нового пакета. В случае если ответного пакета не получено или получен некорректный пакет (не совпадает контрольная сумма или преамбула), исходный пакет считается отправленным с ошибкой и его отправка повторяется.

Описание типов данных:
uint8_t - 8ми битный целый беззнаковый.
int8_t - 8ми битный целый знаковый.

uint16_t - 16ти битный целый беззнаковый. Порядок записи байтов - обратный (от младшего к старшему, интеловский, как в x86)
int16_t - 16ти битный целый знаковый.

uint32_t - 32х битный целый беззнаковый.
int32_t - 32х битный целый знаковый.

Все пакеты упакованы. Т.е. выравнивание элементов структуры - 1 байт.

Алгоритм расчета контрольной суммы:

```
 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 }
```

Описание структуры **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 - контрольная сумма пакета целиком начиная с поля preamble и заканчивая последним байтом содердимого пакета (кроме поля crc)
preamble - преамбула. Содердимое всегда 0x8A2C;
tracker_id - идентификатор трекера;
data_len - длина поля с данными;

Максимальная длина бинарного контейнера - 1400байт, включая заголовок.

В качестве полезного содержимого бинарного контейнера выступают данные со структурой **t_common_data_header** за которой следуют полезные данные пакета:

Заголовок - t_common_data_header
Полезные данные пакета

Таких пакетов в контейнере может быть несколько (а может не быть вовсе).

Пример пакта с данными:

t_binary_container
-- t_common_data_header
---- t_0x004C
-- t_common_data_header
---- t_0x0011
-- t_common_data_header
---- t_0x000A

Описание структуры **t_common_data_header**:

```
 1 typedef struct {
 2   uint16_t crc;
 3   uint16_t serial_id;
 4   uint32_t timestamp;
 5   uint16_t packet_type;
 6   uint16_t packet_len;
 7   uint32_t x_coord;
 8   uint32_t y_coord;
 9   uint8_t sf;
10 } t_common_data_header;
```

crc - контрольная сумма пакета, включая блок с данными (кроме поля crc);
serial_id - серийный номер пакета
timestamp - unixtime пакета (UTC+0)
packet_type - тип структуры, которая является полезными данными текущего пакета;
packet_len - длина полезных данных;
x_coord - широта - в соответсвии с правилами ЕГТС. Если равно 0xffffffff - значит нет фиксации валидных координат;
y_coord - долгота - в соответствии с правилапи ЕГТС. Если равно 0xffffffff - значит нет фиксации валидных координат;

sf - Дополнительные флаги пакета - битовая маска:

бит 0 - признак отправки данных из черного ящика (0 - свежесобранные данные, 1 - данные отправлены из черного ящика);
бит 6 - поле LAHS записи EGTS_SR_POS_DATA; 0 - северная широта; 1 - южная широта;
бит 7 - поле LOHS записи EGTS_SR_POS_DATA; 0 - восточная долгота; 1 - западная долгота;

Описание типов пакетов:

PACKET_TYPE_ALARM_DATA 0x000A - данные тревожной кнопки

```
1 typedef struct {

2   uint16_t gps_speed;             // Скорость в момент возникновения
тревожного события

3   uint16_t gps_angle;             // Направление в момент возникновения
тревожного события

4   uint32_t x_coord;               // Широта - милионные доли градуса

5   uint32_t y_coord;               // Долгота - милионные доли градуса

6   uint8_t  alarm_state;           // 1 - есть тревога; 0 - нет тревоги

7 } t_alarm_data_v1;
```

PACKET_TYPE_TEXT_CMD_DATA - 0x0008 - текстовая команда устройству из стека команд

```
1 typedef struct {

2   char     cmd_id[16]; - ID команды

3   char     cmd[160];   - Команда устройству

4 } t_text_cmd_data;
```

При получении этой команды терминал выполняет команду и этим же пакетом пересылает результат ее выполнения

Тип пакета 0x004C - универсальный пакет GPS:

Заголовок:

```
1 typedef struct {
2   uint8_t  ant_state;    // поля от сюда
3   uint8_t  fix_type;
4   uint8_t  sat_count;
5   uint16_t speed;
6   uint16_t course;
7   uint8_t  reason;   // до сюда присутсвуют в пакете всегда
8 } t_l4_data_base;
```

Далее за этим пакетом идет произвольный набор полей со следующей структурой:

тип поля 1 - 1 байт
данные поля 1 - длина зависит от типа поля
тип поля 2 - 1 байт
данные поля 2 - длина зависит от типа поля

Если тип поля = 255, то дальше данных нет - трекер не накопил данных что бы заполнить все поля полностью.

Таким образом каждый пакет может содержать в себе до 255 полей.

Тип и название полей пока статичны и определяются следующей структурой:

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

В настоящее время определены следующие типы полей:

```
#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
104, FIELD_TYPE_16B,   "adc3",  // *      //3
105, FIELD_TYPE_16B,   "adc4",  // *      //3
106, FIELD_TYPE_16B,   "freq3",           //3
107, FIELD_TYPE_16B,   "freq4",           //3
108, FIELD_TYPE_16B,  "counter4",        //3
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
112, FIELD_TYPE_8B, "level",
113, FIELD_TYPE_32B, "sen1",
114, FIELD_TYPE_32B, "sen2",
115, FIELD_TYPE_32B, "sen3",
116, FIELD_TYPE_32B, "sen4",
117, FIELD_TYPE_32B, "sen5",
118, FIELD_TYPE_32B, "sen6",
119, FIELD_TYPE_32B, "sen7",
120, FIELD_TYPE_32B, "sen8",
121, FIELD_TYPE_32B, "sen9",
122, FIELD_TYPE_32B, "sen10",
123, FIELD_TYPE_32B, "sen11",
124, FIELD_TYPE_32B, "sen12",
```

```
125, FIELD_TYPE_32B, "sen13",
126, FIELD_TYPE_32B, "sen14",
127, FIELD_TYPE_32B, "sen15",
128, FIELD_TYPE_32B, "sen16",


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", // Израсходовано топлива, x0.5л

  176, FIELD_TYPE_32B, "fm_fuel_prc", // уровень топлива, x0.4%

  177, FIELD_TYPE_32B, "fms_rpm", // обороты двигателя, об/мин

  178, FIELD_TYPE_32B, "fms_hours", // моточасы двигателя, x0.05ч

  179, FIELD_TYPE_32B, "fms_milage", // пройденное расстояние, м

  180, FIELD_TYPE_32B, "fms_engine_temp".. // температура ОЖ, °C

  181, FIELD_TYPE_32B, "fms_amb_temp", // забортная температура, °C

  182, FIELD_TYPE_32B, "fms_fuel_rt", // мгновенный расход топлива, x0,05л/ч

  183, FIELD_TYPE_32B, "fms_fl_hrlf", // израсходовано топлива (высокая
точность), л

  184, FIELD_TYPE_32B, "fms_speed", // скорость, км/ч

  185, FIELD_TYPE_32B, "fms_eng_loa", // нагрузка на двигатель, %

  186, FIELD_TYPE_32B, "fms_axl_wgt", // масса оси, кг

  187, FIELD_TYPE_32B, "fms_srv_dst", // расстояние до ТО, км

  188, FIELD_TYPE_32B, "fms_gros_wg", // вес ТС, кг

  189, FIELD_TYPE_32B, "fms_taho_st", // данные собираемые тахографом

  190, FIELD_TYPE_32B, "fms_tah_spd", // скорость ТС, регистрируемая
тахографом, км/ч

  191, FIELD_TYPE_32B, "iqf_sp", // температура ХОУ, °C

  192, FIELD_TYPE_32B, "iqf_sp", // температура установленная, °C

  193, FIELD_TYPE_32B, "iqf_ambt", // температура окружающего воздуха, °C

  194, FIELD_TYPE_32B, "iqf_afzt", // температура ОЖ двигателя привода
компрессора, °C
```

```
  195, FIELD_TYPE_32B, "iqf_rpm", // обороты двигателя привода компрессора, об/мин

  196, FIELD_TYPE_32B, "iqf_conf", // режим работы компрессора (непрерывный/с остановкой двигателя)
```

```
  197, FIELD_TYPE_32B, "iqf_state", // состояние системы (нагрев, охлаждение, оттайка)

  198, FIELD_TYPE_32B, "iqf_dr", // состояние двери

  199, FIELD_TYPE_32B, "iqf_batv", // напряжение аккумулятора, В

  200, FIELD_TYPE_20Z, "imei",    // IMEI модема

  201, FIELD_TYPE_20Z, "iccid1",  // ICCID сим-карты 1

  202, FIELD_TYPE_20Z, "iccid2",  // ICCID сим-карты 2

  203, FIELD_TYPE_8B,  "sim_num",    // номер используемой сим-карты (1 или 2)

  204, FIELD_TYPE_8B,  "srv_num",    // номер сервера (основной, вспомогательный или конфигурационный (1,2 или 3))

  205, FIELD_TYPE_16Z, "op_name",    //  название оператора

  206, FIELD_TYPE_16B, "lac",        //  код локальной зоны

  207, FIELD_TYPE_32B, "cell_id"   // идентификатор соты мобильной сети

  208, FIELD_TYPE_8B  “gsm_power”  // уровень GSM-сигнала ( 31 – max)

  245, FIELD_TYPE_32B, "iqf_bata", // сила тока аккумулятора установки, А

  246, FIELD_TYPE_32B, "iqf_alc", // кол-во ошибок активных в текущий момент на установке

  247, FIELD_TYPE_32B, "iqf_al1", // код самой важной ошибки установки

  248, FIELD_TYPE_32B, "iqf_al2", // код 2й по важности ошибки

  249, FIELD_TYPE_32B, "iqf_al3", // код 3й по важности ошибки

  250, FIELD_TYPE_32B, "iqf_in"  // наличие напряжения на дискретном входе 1 (0 – менее 3.3В, 1 – более 3.3В)
```

Пример данных полей (без заголовка):

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 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

0B = 11 - тип поля 11 - stop_state длина - 1 байт значение 00
03 = 3 тип поля 3 - vbat - длина - 2 байта значение 0x05EB = 1515
01 = 1 тип поля 1 - v_in длина - 2 байта значение 0x095A= 2394
02 = 2 тип поля 2 - ign_state длина 1 байт значение 1
00 = 0 тип поля 0 - alt длина 2 байта значение 005B = 91
04 = 4 тип поля 4 = adc1 длина 2 байта значение 0060 = 96
05 = 5 тип поля 5 = adc2 длина 2 байта значение 0060 = 96

FF = данных дальше нет. Разбор пакета можно остановить. Так же нет смысла разбирать пакет дальше если встретился неизвестный тип поля.

Порядок полей в пакете может быть произвольным, при этом общая длина пакета будет соответствовать настроенному набору полей.

Так как список полей в пакете 0x004C будет расширяться, то чтобы разбор пакетов в системе мониторинга не прерывался, необходимо при разборе пакетов выполнение 3 постулатов:

1. Если структура t_binary_container корректна (совпадают преамбула и контрольная сумма всего пакета), то пакет подтверждается - трекер считает, что все в порядке и готовит к отправке следующий пакет.
2. Если структура t_common_data_header внутри t_binary_container корректна (совпадают преамбулы и контрольные суммы), но packet_type системе мониторинга неизвестен, то переходим к разбору следующего пакета в контейнере (поле packet_len позволяет это сделать). Если структура t_common_data_header некорректна, то разбор контейнера на этом заканчивается.
3. Если система мониторинга встретила неизвестное поле внутри пакета B_PACKET_TYPE_L4_UNIVERSAL (0x004C), то разбор пакета на этом заканчивается, но все поля, которые удалось разобрать, добавляются в базу. Нужно учитывать, что пакет данного типа может быть вообще пустой. В этом случае в базу добавляются только поля из t_l4_data_base.

Компрессия данных

Устройства поддерживают компрессию отправляемых данных.

Пакет t_binary_container сжимается полностью по алгоритму LZ. К получившемуся бинарному массиву сверху цепляется вот такая структура (все упаковано):

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

Преамбула другая:

```
#define B_COMPRESSED_PREAMBLE  0x9b3d
```

crc - контрольная сумма сжатого пакета
data_len - длина в сжатом виде

Т.е. на сервер передается сначала:
t_compressed_header
затем сжатый t_binary_container

Соответственно, алгоритм такой:

1. Проверяем что преамбула соответствует зашифрованному пакету
2. Проверяем контрольную сумму пакета
3. Расшифровываем (распаковываем) данные
4. Получившийся пакет обрабатываем как обычно

Пример сжатого пакета:

[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

После распаковки получаем:

[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

Далее обрабатываем как обычный пакет.

Библиотека сжатия описана в файлах LZ.c, LZ.h.