Wialon Combine

The Wialon Combine (v. 1.1.5) binary communication protocol was developed by Gurtam for use in personal and automotive GPS and GLONASS trackers which transmit data to a satellite monitoring server using the TCP or the UDP protocol.

Specification

- Big-Endian is the order of bytes.
- Field_name * is an extensible one-byte field. A high-order bit indicates that there is an additional byte.
- Field_name ** is an extensible two-byte field. A high-order bit indicates that there are two additional bytes.
- All data is received in binary format.
- Data transmission is implemented using the TCP and UDP protocols.

Table of Contents

General Data Structure

Size (Bytes)	2	1-2	2	2-4			
Section	Head	Type*	Seq	Len**	Login (for UDP)	Data	CRC16

Head — 0x2424.

Type*:

- 0 is Login,
- 1 is Data,
- 2 is Keep-Alive,
- 3 is ACK.

Seq. Sequence number (cyclic order 0 — 65535).

Len**. The length of the *Data* field.

Data. Useful data. Depends on a packet type.

Login (for UDP). The field is provided only when using UDP.

CRC16. Checksum. It is calculated from the first byte of the head to the last byte of useful data.

Required Server Response

The server responds to every received packet. The server response looks as follows:

Size (Bytes)	2	1	2
Section	Head	Code	Seq

Head — 0x4040.

Code. Response code.

Seq. The sequence number of the received packet.

Response code	**Meaning**

0	Packet successfully registered
1	Authorization error
3	Packet not registered
4	CRC error
255	Command to the device

Command to the Device

Size (Bytes)	2	1	2-4	4	1-2		2
Section	Head	Code	Len**	Time	Type*	Data	CRC16

Head — 0x4040.

Code — 0xFF.

Len. Packet length (*Time*, *Type* and *Data* fields).

Time. The time the message was sent.

Type. Command type.

Data. The additional parameters of the command.

CRC16. Checksum. It is calculated from the beginning of the head to the last byte of useful data.

Command type	**Meaning**
0	Custom command
1	Firmware block
2	Config block

If you want to receive a response from the device, you can create the *ACK* message with the fields of the command type and the response code, or the *Custom parameters* message with a list of the required parameters. Subsequently, you can create the necessary notifications based on these parameters. You can also use the *Driver message* record. In this case, the received message will be displayed in the chat with the driver.

In the case of the *Firmware/Config block* command, a response from the device (the *ACK* packet) is required.

Firmware/Config Block Command

A part of the firmware file. The *Firmware/Config block* record looks as follows:

Size (Bytes)	1	2-4	1-2	Len
Section	Ind*	Len**	Count*	Bin

Ind*. The index number of the transmitted block (numbering from 0).

Len**. The size of the file block (no more than 1024 bytes).

Count*. The number of the last block (numbering from 0).

Bin. The binary block of the file.

If the connection is interrupted, the transmission continues with the last data packet that has not been received. If the device does not respond within 60 seconds, the transmission is interrupted.

Login Packet

The *Login* packet looks as follows:

Size (Bytes)	1-2	1		
Section	Protocol version*	Flags	ID	Pwd

Protocol version*: 1 is used currently.

Flags (bit field):

- 4 high-order bits are responsible for the type and size of the *ID* field.

- 4 low-order bits are responsible for the type and size of the *Pwd* field.

Value	ID type
1	unsigned short (2 bytes)
2	unsigned int (4 bytes)
3	unsigned long (8 bytes)
4	string (the last byte 0x00)

Value	Pwd type
0	password is missing
1	unsigned short (2 bytes)
2	unsigned int (4 bytes)
3	unsigned long (8 bytes)
4	string (the last byte 0x00)

Keep-Alive Packet

The *Keep-Alive* packet contains only the first three packet fields (*Head*, *Type*, *Seq*) and looks as follows:

Size (Bytes)	2	1-2	2
Section	Head	Type*	Seq

ACK Packet

The packet of this type is necessary to confirm that the firmware block has been received.

Size Bytes	1	1
Section	Type	Code

Type. The type of the command to which the response is sent.

Code. The response code.

Response code	**Meaning**
0	Packet successfully received
1	Reception error (interrupts the transmission)
2	Last block transmission retry
3	Incorrect file (interrupts the transmission)

Command type	**Meaning**
0	Custom command
1	Firmware block
2	Config block

Data Packet

The packet can contain several messages.

Each message includes time and length as well as a set of records. In general, the message looks as follows:

Size (Bytes)	4	1	1-2		...	1-2	
Section	Time	Count	Sub-record type*	Sub-record	...	Type sub-record N	Sub-recordN

Time. The time the message was formed.

The time should be converted to UTC:0 without regard to the local time zone of the device. This is required in order to display the time correctly to the end user.

UTC is defined as the number of seconds that have elapsed since midnight (00:00:00 UTC), 1 January 1970.

Count. The number of records.

Sub-record type*. The field which contains the record type code.

Sub-record. Data structure. A set of record fields depends on its type.

Record types:

Code	Record type
0	Custom Parameters
1	Position Data
2	I/O Data
3	Picture
4	LBS Parameters
5	Fuel Parameters
6	Temperature Parameters
7	CAN Parameters
8	Counter Parameters
9	Analog Parameters (ADC)
10	Driver Code Parameters
11	Tacho File
12	Driver message
13	Wi-Fi Parameters
14	Extended Position Data

15	Named Parameters

Custom Parameters Record Type

The record is a set of custom fields data which looks as follows:

Size (Bytes)	1-2	
Section	Count*	Params

Count*. The number of custom fields in the record.

Params. A set of numbered parameters, each of which is registered as *param№*. It looks as follows:

Bytes	1-2	1	
Section	№*	Sensor type	Value

№. Sensor number.

Sensor type. The field which indicates the type of data in the *Value* field. It has the following structure (for integer types only):

Size (Bits)	3	5
Section	10**X	Sensor type

For type 8 and more, the first three bits always equal 0.

10**X. The degree of number 10. The value in the *Value* field will be divided by the number in this field.

Value. The sensor value according to the selected type.

Sensor types:

Code	**Sensor type**
0	unsigned byte (1 byte)

1	unsigned short (2 bytes)
2	unsigned int (4 bytes)
3	unsigned long (8 bytes)
4	signed byte (1 byte)
5	signed short (2 bytes)
6	signed int (4 bytes)
7	signed long (8 bytes)
8	float (4 bytes)
9	double (8 bytes)
10	String (the last byte 0x00)

Position Data Record Type

The record contains navigation data and looks as follows:

Size (Bytes)	4	4	2	2	2	1	2
Section	Lat	Lon	Speed	Course	Alt	Sats	HDOP

Lat. Latitude.

Lon. Longitude.

The coordinate value is of the *signed int* type. Example of value formation: a floating-point degree value multiplied by 1,000,000.

Speed. Speed value (km/h).

Course. Direction of movement (from 0 to 359 degrees).

Alt. Altitude. The *signed int* type.

Sats. The number of visible satellites.

HDOP. The Horizontal Dilution of Precision value multiplied by 100. It shows the accuracy of the coordinates transmitted by the device. The smaller this value is, the more accurate the coordinates are.

Extended Position Data Record Type

The record contains navigation data and looks as follows:

Size (Bytes)	8	8	2	2	8	1	2
Section	Lat	Lon	Speed	Course	Alt	Sats	HDOP

The coordinate values and altitude are of the *signed long* type. Coordinate values are multiplied by 10^16. Altitude value is multiplied by 10^14

Lat. Latitude.

Lon. Longitude.

Speed. Speed value (km/h).

Course. Direction of movement (from 0 to 359 degrees).

Alt. Altitude.

Sats. The number of visible satellites.

HDOP. The Horizontal Dilution of Precision value multiplied by 100. It shows the accuracy of the coordinates transmitted by the device. The smaller this value is, the more accurate the coordinates are.

I/O Record Type

A bit field. Contains digital input and output values. Each bit of the number corresponds to one input or output. The *I/O* record looks as follows:

Size (Bytes)	4	4
Section	Inputs	Outputs

Picture Record Type

The record contains a part of a picture made by the device camera.

The *Picture* record looks as follows:

Size (Bytes)	1	2-4	1-2		Len
Section	Ind*	Len**	Count*	Name	Bin

Ind*. The index number of the transmitted picture block (numbering from 0).

Len**. The size of the picture block.

Count*. The number of the last block (numbering from 0).

Name. The name of the transmitted picture. This is a text field which ends with 0x00.

Bin. Binary picture block.

LBS Parameters Record Type

The *LBS Parameters* record looks as follows:

Size (Bytes)	1	
Section	Count	LBS params

Count. The number of the *LBS params* structures.

LBS params:

Size (Bytes)	2	2	2	2	2	2
Section	MCC	MNC	LAC	Cell ID	Rx level	TA

MCC. Mobile Country Code.

MNC. Mobile Network Code.

LAC. Local Area Code. A local area is a group of base stations serviced by a base station controller.

Cell ID. A cell identificator assigned by the operator to every base station sector.

Rx level. The level of the input radio signal received by the GSM modem through this channel.

TA. Timing Advance. This parameter is used to compensate for the propagation delay as the signal travels between the GSM modem and the base station. In effect, it is the distance to the base station.

Fuel Parameter Record Type

The *Fuel Parameter* record looks as follows:

Size (Bytes)	1	
Section	Count	Fuel (the *Params* structure analog)

Count. The number of the *Fuel* structures.

Each parameter of this field will be registered with the name *fuel№*.

Temperature Parameters Record Type

The *Temperature Parameters* record looks as follows:

Size (Bytes)	1	
Section	Count	Temp (the *Params* structure analog)

Count. The number of the *Temp* structures.

Each parameter of this field will be registered with the name *temp№*.

CAN Parameters Record Type

The *CAN Parameters* record looks as follows:

Size (Bytes)	1	
Section	Count	CAN (the *Params* structure analog)

Count. The number of the *CAN* structures.

Each parameter of this field will be registered with the name *can№*.

Counter Parameters Record Type

The *Counter Parameters* record looks as follows:

Size (Bytes)	1	
Section	Count	Counter (the *Params* structure analog)

Count. The number of the *Counter* structures.

Each parameter of this field will be registered with the name *counter№*.

Analog Parameters (ADC) Record Type

The *Analog Parameters (ADC)* record looks as follows:

Size (Bytes)	1	
Section	Count	ADC (the *Params* structure analog)

Count. The number of the *ADC* structures.

Each parameter of this field will be registered with the name *adc№*.

Driver Code Parameters Record Type

The *Driver Code Parameters* record looks as follows:

Size (Bytes)	1	

Section	Count	Driver code (the *Params* structure analog)

Count. The number of the *Driver code* structures.

Each parameter of this field will be registered with the name *driver_code№* *.

Tacho File Record Type

The record contains a part of a tachograph file. It looks as follows:

Size (Bytes)	1	2-4	1	Len
Section	Ind*	Len**	Count*	Bin

Ind*. The index number of the transmitted block (numbering from 0).

Len**. The size of the transmitted block.

Count*. The number of the last block (numbering from 0).

Bin. The binary block of the tachograph file.

Driver Message Record Type

The record contains a message to the driver. It looks as follows:

Size (Bytes)	Endian 0x00
Section	Text

Text. The message to the driver. The string ends with 0x00.

Wi-Fi Parameters Record Type

Wi-Fi parameters. The record looks as follows:

Size (Bytes)	1	

Section	Count	Wi-Fi params

Count. The number of *Wi-Fi params* structures.

Wi-Fi params:

Size (Bytes)	6	1
Section	MAC	Rssi

MAC (Media Access Control). The unique identifier assigned to a network interface controller.

Rssi (Received signal strength indicator). The values are in dBm. The *signed byte* type from −128 to 127.

Named Parameters Record Type

A set of custom fields data. The record looks as follows:

Size (Bytes)	1-2	
Section	Count*	Params

Count*. The number of custom fields in the record.

Params. A set of named parameters. It looks as follows:

Bytes		1	
Section	Param name	Param type	Param value

Param name. The parameter name, String (the last byte is 0x00). In lower case. The maximum number of characters is 38. The invalid characters are spaces, commas, colons, number signs, line feeds and carriage returns (\r\n).

Param type. The field indicating the *Param value* data type. (The *Sensor type* table is described in the *Custom Parameters Record Type* section).

If the value doesn’t correspond to the parameter type, the parameter won’t be registered. The maximum number of parameters that can be registered in Wialon is 200. The protocol doesn’t limit the number of transmitted parameters.

CRC 16 (C Code Example):

```
static const unsigned short crc16_table[256] =
{
    0x0000,0xC0C1,0xC181,0x0140,0xC301,0x03C0,0x0280,0xC241,
    0xC601,0x06C0,0x0780,0xC741,0x0500,0xC5C1,0xC481,0x0440,
    0xCC01,0x0CC0,0x0D80,0xCD41,0x0F00,0xCFC1,0xCE81,0x0E40,
    0x0A00,0xCAC1,0xCB81,0x0B40,0xC901,0x09C0,0x0880,0xC841,
    0xD801,0x18C0,0x1980,0xD941,0x1B00,0xDBC1,0xDA81,0x1A40,
    0x1E00,0xDEC1,0xDF81,0x1F40,0xDD01,0x1DC0,0x1C80,0xDC41,
    0x1400,0xD4C1,0xD581,0x1540,0xD701,0x17C0,0x1680,0xD641,
    0xD201,0x12C0,0x1380,0xD341,0x1100,0xD1C1,0xD081,0x1040,
    0xF001,0x30C0,0x3180,0xF141,0x3300,0xF3C1,0xF281,0x3240,
    0x3600,0xF6C1,0xF781,0x3740,0xF501,0x35C0,0x3480,0xF441,
    0x3C00,0xFCC1,0xFD81,0x3D40,0xFF01,0x3FC0,0x3E80,0xFE41,
    0xFA01,0x3AC0,0x3B80,0xFB41,0x3900,0xF9C1,0xF881,0x3840,
    0x2800,0xE8C1,0xE981,0x2940,0xEB01,0x2BC0,0x2A80,0xEA41,
    0xEE01,0x2EC0,0x2F80,0xEF41,0x2D00,0xEDC1,0xEC81,0x2C40,
    0xE401,0x24C0,0x2580,0xE541,0x2700,0xE7C1,0xE681,0x2640,
    0x2200,0xE2C1,0xE381,0x2340,0xE101,0x21C0,0x2080,0xE041,
    0xA001,0x60C0,0x6180,0xA141,0x6300,0xA3C1,0xA281,0x6240,
    0x6600,0xA6C1,0xA781,0x6740,0xA501,0x65C0,0x6480,0xA441,
    0x6C00,0xACC1,0xAD81,0x6D40,0xAF01,0x6FC0,0x6E80,0xAE41,
    0xAA01,0x6AC0,0x6B80,0xAB41,0x6900,0xA9C1,0xA881,0x6840,
    0x7800,0xB8C1,0xB981,0x7940,0xBB01,0x7BC0,0x7A80,0xBA41,
    0xBE01,0x7EC0,0x7F80,0xBF41,0x7D00,0xBDC1,0xBC81,0x7C40,
    0xB401,0x74C0,0x7580,0xB541,0x7700,0xB7C1,0xB681,0x7640,
    0x7200,0xB2C1,0xB381,0x7340,0xB101,0x71C0,0x7080,0xB041,
    0x5000,0x90C1,0x9181,0x5140,0x9301,0x53C0,0x5280,0x9241,
    0x9601,0x56C0,0x5780,0x9741,0x5500,0x95C1,0x9481,0x5440,
    0x9C01,0x5CC0,0x5D80,0x9D41,0x5F00,0x9FC1,0x9E81,0x5E40,
    0x5A00,0x9AC1,0x9B81,0x5B40,0x9901,0x59C0,0x5880,0x9841,
    0x8801,0x48C0,0x4980,0x8941,0x4B00,0x8BC1,0x8A81,0x4A40,
    0x4E00,0x8EC1,0x8F81,0x4F40,0x8D01,0x4DC0,0x4C80,0x8C41,
    0x4400,0x84C1,0x8581,0x4540,0x8701,0x47C0,0x4680,0x8641,
    0x8201,0x42C0,0x4380,0x8341,0x4100,0x81C1,0x8081,0x4040
};

unsigned short crc16 (const void *data, unsigned data_size)
{
    if (!data || !data_size)
        return 0;

    unsigned short crc = 0;
    unsigned char* buf = (unsigned char*)data;

    while (data_size--)
        crc = (crc >> 8) ^ crc16_table[(unsigned char)crc ^ *buf++];

    return crc;
}
```

Message Examples

Login Message Example

The original message:

242400004000130144737472696E675F646576696365696400009B93

2424 is the head of the packet;

00 is the message type (Login);

0040 is the sequence number of the message;

0013 is the length of the message (the field is extensible, but because there is no high-order bit, the length is two bytes; otherwise, it would be 4 bytes);

01 is the protocol version;

44 is the flag. Binary representation (0100 0100), the *ID* type is 4 String, the *Pwd* type is 4 String;

737472696E675F646576696365696400 is the device ID. According to the protocol, the last byte after the string field is 0x00 to distinguish the border of the text data;

00 is the end byte of the password because in accordance with the flag, the password is transmitted. Regardless of whether there is a password value or not, there should be an end byte because in accordance with the flag, the packet has a password;

9B93 is the CRC.

Server Response Message Example

The original message: 4040000040

4040 is the head of the packet;

00 is a response code (packet successfully registered);

0040 is the sequence number of the message

Keep-Alive Message Example

The original message: `2424020011`

2424 is the head of the packet;

02 is the message type (Keep-Alive);

0011 is the sequence number of the message.

Message Example of the Firmware Block Command

The original message:

```
4040FF035D5E4FAA5C01010354015FEA4C0C404141EB010111F4801FA4F10104E9D191F0000F
04BF01460020B1FA81F308BF2033A3F10B03B3F120020CDA0C3208DD02F1140CC2F10C0201F
A0CF021FA02F10CE002F11402D8BFC2F1200C01FA02F120FA0CFCDCBF41EA0C019040E41AA2
BF01EB0451294330BD6FEA04041F3C1CDA0C340EDC04F11404C4F1200220FA04F001FA02F340
EA030021FA04F345EA030130BDC4F10C04C4F1200220FA02F001FA04F340EA0300294630BD21
FA04F0294630BD94F0000F83F4801306BF81F480110134013D4EE77FEA645C18BF7FEA655C29
D094EA050F08BF90EA020F05D054EA000C04BF1946104630BD91EA030F1EBF0021002030BD5
FEA545C05D14000494128BF41F0004130BD14F580043CBF01F5801130BD01F0004545F0FE414
1F470014FF0000030BD7FEA645C1ABF194610467FEA655C1CBF0B46024650EA013406BF52EA0
33591EA030F41F4002130BD00BF90F0000F04BF0021704730B54FF4806404F132044FF000054F
F0000150E700BF90F0000F04BF0021704730B54FF4806404F1320410F0004548BF40424FF0000
13EE700BF42004FEAE2014FEA31014FEA02701FBF12F07F4393F07F4F81F06051704732F07F42
08BF704793F07F4F04BF41F40021704730B54FF4607401F0004521F000411CE700BF50EA01020
8BF704730B54FF000050AE050EA010208BF704730B511F0004502D5404261EB41014FF480640
4F132045FEA915C3FF4D8AE4FF003025FEADC0C18BF03325FEADC0C18BF033202EBDC02C2F12
00300FA03FC20FA02F001FA03FE40EA0E0021FA02F11444BDE600BF70B54FF0FF0C4CF4E06C1
CEA11541DBF1CEA135594EA0C0F95EA0C0F00F0DEF82C4481EA030621EA4C5123EA4C5350EA
013518BF52EA033541F4801143F4801338D0A0FB02CE4FF00005E1FB02E506F00042E0FB03E54
FF00006E1FB03569CF0000F18BF4EF0010EA4F1FF04B6F5007F64F5407404D25FEA4E0E6D4146
EB060642EAC62141EA55514FEAC52040EA5E504FEACE2EB4F1FD0C88BFBCF5E06F1ED8BEF100
4F08BF5FEA500E50F1000041EB045170BD06F0004646EA010140EA020081EA0301B4EB5C04C2
BFD4EB0C0541EA045170BD41F480114FF0000E013C00F3AB8014F1360FDEBF002001F0004170
BDC4F10004203C35DA0C341BDC04F11404C4F1200500FA05F3475FD6694BD8
```

4040 is the head of the packet;

FF is the response code (command to the device);

035D is the length of the packet (523 bytes);

5E4FAA5C is the time of sending;

01 is the *Firmware block* command type;

01 is the sequence number of the block (the second block);

0354 is the size of the file block (512 bytes);

01 is the number of the last block (43);

5FEA4C0C404141EB010… is the binary block of the file;

4BD8 is the CRC16.

ACK (Firmware/Config) Message Example

The original message: 24240302FC000201004C6A

2424 is the head of the packet;

03 is the message type (ACK (Firmware));

02FC is the sequence number of the message;

0002 is the length of useful data;

01 is the command type

00 is the response code (packet successfully received);

4C6A is the CRC.

Data Message Example

The original message:

```
24240149F3006F5CF6150303010350A6EC023C5938000F012C01060B00640200000001000000
0000050100000200070300040861367E09610FEF5CF6150204010350A6C8023C59880000011F
01060C005E0200000000100000000000050100000200070300050861367409610FEC0D010A0B0C
0D0E0F81B913
```

2424 is the head of the packet;

01 is the message type (Data);

49F3 is the sequence number of the message;

006F is the length of useful data;

5CF61503 is time;

03 is the number of records;

01 is the *Position Data* record type;

350A6EC is a latitude of 55.61726 degrees. The value has been transferred to the decimal system (55617260) and divided by 1,000,000;

23C5938 is a longitude of 37.509432 degrees. The value has been transferred to the decimal system (37509432) and divided by 1,000,000;

000F is a speed of 15 km / h;

012C is a course of 300 degrees;

0106 is an altitude of 262 meters;

0B is the number of satellites (11);

0064 is 1 HDOP. The value has been transferred to the decimal system (100) and divided by 100;

02 is the *I/O Data* record type;

00000001 refers to inputs;

00000000 refers to outputs;

00 is the *Custom Parameters* record type;

05 is the number of records;

01 is the number of the sensor;

00 is the sensor type (0 is an unsingned byte (1 byte));

00 is the value of the sensor;

The final parameter form in Wialon: param1=0.

02 00 07 is param2=7;

03 00 04 is param3=4;

08 is the number of the sensor;

61 is the sensor type. Here the sensor type has an additional multiplier, that is three high-order 'X' bits. In a binary representation 0x61 => 0110 0001. According to the protocol, 10**X is a degree of number 10. The parameter value will be divided by 10**X;

367E is param8=13.95;

09 61 0FEF is param9=4.079;

5CF61501 is time;

04 is the numder of records;

01 is the *Position Data* record type;

0350A6C8 is a latitude of 55.617224 degrees. The value has been transferred to the decimal system (55617224) and divided by 1,000,000;

023C5988 is a longitude of 37.509512 degrees. The value has been transferred to the decimal system (37509512) and divided by 1,000,000;

0000 is speed;

011F is a course of 287 degrees;

0106 is an altitude of 262 meters;

0C is the number of satellites (12);

005E is 0.94 HDOP;

02 is the *I/O Data* record type;

00000001 refers to inputs;

00000000 refers to outputs;

00 is the *Custom Parameters* record type;

05 is the number of records;

01 is the number of the sensor;

00 is the sensor type (0 is an unsingned byte (1 byte));

00 is the value of the sensor;

02 00 07 is param2=7;

03 00 05 is param3=5;

08 61 3674 is param8=13.94;

09 61 0FEC is param9=4.076

0D is the Wi-Fi *Parameters* record type;

01 is the number of records;

0A0B0C0D0E0F — MAC-adress (0a:0b:0c:0d:0e:0f);

81 – RSSI (-127);

B913 – CRC.

Picture/Tachograph File Message Example

The original messages:

Message №1:

```
242401000102155CF78ACF01030002000274657374 5F696D61676500ffd8ffe000104
a46494600010101004800480000fffe00134372656174656420776974682047494d50f
fdb0043002016181c1814201c1a1c24222026305034302c2c3062464a3a5074667a78
7266706e8090b89c8088ae8a6e70a0daa2aebec4ced0ce7c9ae2f2e0c8f0b8cacec6ffd
b00430122242430 2a305e34345ec6847084c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c
6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6c6ffc2001
108001900850301110002110103110 1ffc400170001010101000000000000000000000
000000010203ffc400180101010101010000000000000000000000000001020304ffda00
0c03010002100310000001e1cbdf5811adce38bd8a080b5916640cde84eb3c664a23a
4686e79b37bd6235598dc6862fa05649662341740882 2eaccc5025d8000000000000ff
fc4001f1000010304030100000000000000000000011000110213141422 0223040ffda0
0080101000105025656815f0c0a8ac383b2cd99d34611ec9edb6dc0ba28c98307e6ffc
40028110002020201020309000000000000000000010200110312311021134161202 23
2334051627181ffda0008010301013f0188e1c5888e1c58e8737ba5ab8f6d8d0b8a6c5
c0f6c57ed0640537872009bf5c2ae50534b2300a8aacac0815fd8df2dff007080f908698
792a3899bb534b201fca3282e10f13efcea
```

Message №2:

```
242401000202155CF78AD001030102000274657374 5F696D6167650012ca38a8100
c5b7a4ad8aab71530f676a8a8462d632138b5eaaa145081140d60c280d813c35a2236
357e62a85142328614614535e91915f9831a81426a2b5871a91462a2af10200baf942
80aebe5f47fffc4001411010000000000000000000000000000000060ffda000801020101
3f0109ffc4001c100002020203000000000000000000000000110112031305071ffda0
008010100063f02e6a2a751b161e162862e9bffc4002010010101000201040300000 0
00000000000111002131411020518140 61a1ffda0008010100013f21c953254f4ea2f6
7bc2b32466481f9cc7b00e96bef4af288dafac7f3d629ba8f9773a7ceed30790eeeee9e
ee6a3c5dd08eefa453961397aad6bab6f9cb79d547e308e9cabde18d3573fbc29d6a5b
75e6e10db957bd5b7ceadbe7f0fffda000c03010002000300000010925564b600d294d
3066c124b720bff00026e16fe4924924924924fffc400261101000201030303050100 00
```

```
0000000000010011213141a15171c110b1f12040618191d1ffda0008010301013f1080
74980749f4a37456bdbebbfe92e53f5973176ce6549a678972698e7d56aa338a1de1d
6a6ba6eef2806bce0d9394f723b78028f7660a5871e665e9bc32ccea39bff0018c3e07f
58837d47b31c6e7c2036a0f76002acc7997d99a79b87499a38af50474128231d26be1
174985b7bcd3b7298d1101d8cca0fc21913f6
```

Message №3:

```
242401000301345CF78AD2010302011F02746573745F696D6167650046e3f2c3362
c557ea1e2c1a4554d5f89441f29649f0fb3ffc4001d110002020203010000000000000
00000000110110213120304140ffda0008010201013f10232453ea63c3b8d1e08f24dc
904d7a6858a8d8b02c7150213e0854a2d7c7ffc400231001010002020202010500000
0000000000111002131514161107120408 1a1e1f0ffda0008010100013f10c755298ea
a53af82bb42cfce0f66651ae199b4bcb589525a1827716a7c90aaef4c455e0a6bd6015
3346b33fcbe9c109dab599ba881bcd1d6d7de401e1cfe3fac60cdf97595269a6f662069
1c7f7c54b7297a31aba4778e9baa386bbaabf28c95716bbf6c022a60346c433c5988d5
5c424898068fb7bc76a9888a938cae6ddb8b042f38282b38c5aefdb029bf6fd1ff00ffd96
84f
```

Message №1 is used as an example of data parsing for a file transfer. For the other messages (№2 and №3), the principle remains the same. Data parsing:

2424 is the head of the packet;

01 is the message type (Data);

0001 is the sequence number (cyclic order 0 — 65535);

0215 is the length of the *Data* field;

5CF78ACF is the time the message was formed;

01 is the number of records;

03 is the *Picture* record type;

00 is the index number of the transmitted block (numbering from 0);

0200 is the size of the picture block (only the binary part of the picture block);

02 is the number of the last block (numbering from 0);

746573745F696D61676500 is the name of the transmitted picture. A text field, ends with 0x00;

ffd8ffe000104a4649 ... f13e is the binary part of the picture;

fcea is the CRC.

UDP Message Example

The message example is based on the *Data* type.

```
2424	01	49F3	0066	0144737472696E675F64657669636569640000
5CF6150303010350A6EC023C5938000F012C01060B006402000000010000000000
0501000000200070300040861367E09610FEF5CF6150103010350A6C8023C598800
00011F01060C005E020000000100000000005010000002000703000508613674096
10FEC3EA9
```

2424 is the head of the packet;

01 is the message type (Data);

49F3 is the sequence number of the message;

0066 is the length of the *Data* field;

01 44 737472696E675F646576696365696400 00 is the *login* structure.

(Contains:

01 is the protocol version,

44 is the flag,

737472696E675F646576696365696400 is the ID,

00 is the password).

This is followed by the data structure without any changes. When calculating the CRC, the login is also included.