Wialon Combine

Communication binary protocol 'Wialon Combine' is developed by Gurtam to be used in personal or automobile GPS and GLONASS trackers transmitting data to the monitoring system server via TCP or UDP protocols.

Specification:

- *Byte ordering is implemented in the 'Big-Endian' format.*
- (Field_name)* — *1 byte expandable field. High-order bit indicates an availability of the additional byte.*
- *(Field_name)** — 2 bytes expandable field. High-order bit indicates an availability of 2 additional bytes.*
- *All data is received in the binary format.*
- *Data transmitting is implemented via TCP or UDP protocols.*

Table of contents:

Ver. 1.0.2

Data General Structure

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

Head — 0x2424
Type* :
0 — Login
1 — Data
2 — Keep-Alive

Seq — Sequence number (cyclic order 0 — 65535).
Len** — Length of the 'Data' field.
Data — Useful data. Depends on a package type.
Login (for UDP) — The field is provided upon UDP usage.
CRC16 — A Cyclic Redundancy Check. Calculated from the beginning of the head to the last byte of useful data.

Obligatory Server Response

Server responds on every received package:

Bytes:	2	1	2
Section:	**Head**	**Code**	**Seq**

Head — 0x4040
Code — response code.
Seq — sequence number of a package received.

Response codes:

0	Package successfully registered
1	Authorization error
2	Incorrect password
3	Package unregistered

4	CRC error
255	Device command

Device Command Format

Bytes:	2	1	2-4	4	1-2		2
Section:	**Head**	**Code**	**Len****	**Time**	**Type***	**Data**	**CRC16**

Head — 0x4040
Code — 0xFF
Len — Package length ('**Time', 'Type', and 'Data' fields**)
Time — Message sending time
Type — Command type
Data — Additional parameters of a command
CRC16 — A Cyclic Redundancy Check. Calculated from the beginning of the head to the last byte of useful data.

Command types

0	Custom command

'Login' Package

Login package consists of the following:

Bytes:	1	1		
Section:	**Protocol version***	**Flags**	**ID**	**Pwd**

Protocol version* currently used is 1.

Flags (bit field)**:**
4 high order bits stand for the type and size of ID field.
4 low order bits stand for the type and size of the 'Pwd' field.

ID types:
1 — unsigned short (2 bytes)
2 — unsigned int (4 bytes)

3 — unsigned long (8 bytes)
4 — String (the last byte 0x00)

Pwd types:

0 — password missed
1 — unsigned short (2 байта)
2 — unsigned int (4 байта)
3 — unsigned long (8 байт)
4 — String (последний байт 0x00)

'Keep-Alive' Package

Contains first 3 fields of a package (Head, Type, Seq)

'Data' Package

A package of this type may contain several messages.

Each message contains time and length, as well as a set of subrecords. Generally, a message has the following view:

Bytes:	4	1	1-2		...	1-2	
Section:	**Time**	**Count**	**Subrecord type***	**Sub-record**	...	**Type sub-record N**	**Sub-record N**

Time — Message formation time.
Count — Number of subrecords.
Subrecord type* — A field containing subrecord type code.

Subrecord types implemented:

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

Subrecord — Data structure. A set of subrecord fields depends on its type.

'Custom Parameters' Subrecord Type

Custom fields data set. Subrecord has the following view:

Bytes:	1-2	
Section:	**Count***	**Params**

Count* — number of custom fields in a subrecord.
Params — a set of indexed parameters. Each parameter is registered as param№.
Has the following view:

Bytes:	1-2	1	
Section:	**№***	**Type sensor**	**Value**

№* — Sensor number.
Type sensor — a field indicating data type in the 'Value' parameter.
Has the following view (for integer value types only):

Bits:	3	5
Section:	**10**X**	**Type sensor**

For the types 8 and over, each of the first 3 bits always equals 0.

10X** — A power of number 10. Corresponds to a number by which a 'Value' parameter will be divided.

Sensor types:

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

Value — sensor value.

'Position Data' Subrecord Type

Navigation data:

Bytes:	4	4	2	2	2	1	2
Section:	**Lat**	**Lon**	**Speed**	**Course**	**Height**	**Sats**	**Hdop**

Lat — Latitude. 'signed int' type. Divided by 1000000.
Lon — Longitude. 'signed int' type. Divided by 1000000.
Speed — Speed value (km/h).
Course — Movement direction (degrees, 0 — 360).
Height — Elevation above sea level. 'Signed int' type.
Sats — Number of visible satellites.
Hdop — Horizontal dilution of precision. Multiplied by 100. Shows accuracy of coordinates provided by a device. The less a value of this parameter is, the more accurate coordinates are.

'I/O' Subrecord Type

Bit field. Values of digital inputs and outputs. Each bit of a number corresponds to one input or output:

Bytes:	4	4
Section:	**Inputs**	**Outputs**

'Picture' Subrecord Type

A part of a picture made by device camera.

Bytes:	1	2-4	1		Len
Section:	**Ind***	**Len****	**Count***	**Name**	**Bin**

Ind* — Index number of data message (numeration from 0).
Len** — Size of a picture block.
Count* — Number of the last block (numeration from 0).
Name — Name of a picture delivered. Text field which ends with 0x00.
Bin — Binary picture block.

'LBS Parameters' Subrecord Type

Bytes:	1	
Section:	**Count**	**LBS param**

Count — Number of the 'LBS param' structures
LBS param:

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. Local Area ia an aggregation of base stations serviced by one base stations controller.
Cell ID — Cell identificator assigned by an operator to each sector of a base station.
LAC — Local Area Code. Local Area ia an aggregation of base stations serviced by one base stations controller.

Rx level — Уровень принимаемого по данному каналу радиосигнала на входе в приёмник GSM-модема.
TA — Timing Advance - параметр компенсации времени прохождения сигнала от GSM-модема до БС. Фактически означает расстояние до БС.

'Fuel Parameters' Subrecord Type

Bytes:	1	
Section:	**Count**	**Fuel** ('Params' structure analog)

Count — Number of the 'Fuel' structures

(* 'Fuel№' name will be used to register each parameter of this field *)

'Temperature Parameters' Subrecord Type

Bytes:	1	
Section:	**Count**	**Temp** ('Params' structure analog)

Count — Number of the 'Temp' structures

(* 'temp№' name will be used to register each parameter of this field *)

'CAN Parameters' Subrecord Type

Bytes:	1	
Section:	**Count**	**Can** ('Params' structure analog)

Count — Number of the 'Can' structures

(* 'can№' name will be used to register each parameter of this field *)

'Counter Parameters' Subrecord Type

Bytes:	1	
Section:	**Count**	**Counter** ('Params' structure analog)

Count — Number of the 'Counter' structures

('counter№' name will be used to register each parameter of this field *)*

'Analog Parameters' (ADC) Subrecord Type

Bytes:	1	
Section:	**Count**	**ADC** ('Params' structure analog)

Count — Number of the 'Can' structures

(* 'can№' name will be used to register each parameter of this field *)

'Driver code Parameters' Subrecord Type

Bytes:	1	
Section:	**Count**	**Driver code** ('Params' structure analog)

Count — Number of the 'Driver code' structures

(* 'driver_code№' name will be used to register each parameter of this field *)

'Tacho File' Subrecord Type

File registered by a tachograph.

Bytes:	1	2-4	1	Len
Section:	**Ind***	**Len****	**Count***	**Bin**

Ind* — Index number of data message (numeration from 0).
Len** — Size of a picture block.
Count* — Number of the last block (numeration from 0).
Bin — Binary block of a tachograph file.

'Driver message' Subrecord Type

Message for a driver.

Bytes:	Endian 0x00
Section:	**Text**

Text — Message for a driver. A line ends with 0x00.

CRC 16 (C language 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;
}
```