**Wialon IPS**

The Wialon IPS (v. 2.0) communication protocol was developed by Gurtam for use in personal and automotive GPS and GLONASS trackers which transfer data to a satellite monitoring server using the TCP or the UDP protocol.

Table of Contents

[TCP Data Transfer 3](#_Toc107947837)

[General Structure of TCP Messages 3](#_Toc107947838)

[Packet Types 5](#_Toc107947839)

[Login Packet 7](#_Toc107947840)

[Short Data Packet 8](#_Toc107947841)

[Extended Data Packet 9](#_Toc107947842)

[Additional Parameters (Params) 11](#_Toc107947843)

[Black Box Packet 14](#_Toc107947844)

[Video 15](#_Toc107947845)

[Request Live Stream Command 15](#_Toc107947846)

[Live Stream Packet 16](#_Toc107947847)

[Request Playback Command 17](#_Toc107947848)

[Playback Packet 17](#_Toc107947849)

[Request Video File Command 18](#_Toc107947850)

[Video File Packet 19](#_Toc107947851)

[Request Timeline Command 20](#_Toc107947852)

[Timeline Packet 20](#_Toc107947853)

[Ping Packet 21](#_Toc107947854)

[Commands 21](#_Toc107947855)

[Upload Firmware Command 21](#_Toc107947856)

[Upload Configuration Command 22](#_Toc107947857)

[Send Message to Driver Command 22](#_Toc107947858)

[Query Snapshot Command 23](#_Toc107947859)

[Snapshot Packet 24](#_Toc107947860)

[Query DDD File Command 25](#_Toc107947861)

[DDD File Information Packet 26](#_Toc107947862)

[DDD File Block Packet 27](#_Toc107947863)

[Send Custom Message Command 28](#_Toc107947864)

[UDP Data Transfer 29](#_Toc107947865)

[General Structure of UDP Messages 29](#_Toc107947866)

[Data Compression 30](#_Toc107947867)

[Checksum 32](#_Toc107947868)

[Annex 34](#_Toc107947869)

# TCP Data Transfer

The TCP connection must be maintained throughout the entire data transfer process. If the device disconnects immediately after sending the message, the server does not have time to send a response to the device, and traffic consumption increases.

While using one TCP connection, you should transfer data from one device. Otherwise, the system registers only the data of the device whose ID is the first in the incoming data list.

To save traffic, you can use the UDP protocol. However, it does not guarantee that the messages will be delivered.

## General Structure of TCP Messages

All data is received in text format as a packet which looks as follows:

#PT#msgCRC\r\n

|  |  |
| --- | --- |
| **Field** | **Description** |
| # | Start byte |
| PT | Packet type (see the *Packet types* table) |
| # | Delimiter |
| Msg | Text of the message |
| CRC | CRC16 checksum |
| \r\n | End of the packet (0x0D0A in HEX) |

## Packet Types

|  |  |  |
| --- | --- | --- |
| **Type** | **Description** | **Sender** |
| L | Login packet | Device |
| AL | Answer to the login packet | Server |
| SD | Short data packet | Device |
| ASD | Answer to the short data packet | Server |
| D | Extended data packet | Device |
| AD | Answer to the extended data packet | Server |
| B | Black box packet | Device |
| AB | Answer to the black box packet | Server |
| QLV | *Request live stream* command | Server |
| LV | Live stream packet | Device |
| QPB | *Request playback* command | Server |
| PB | Playback packet | Device |
| QVF | *Request video file* command | Server |
| VF | Video file packet | Device |
| QTM | *Request timeline* command | Server |
| TM | Timeline packet | Device |
| P | Ping packet | Device |
| AP | Answer to the ping packet | Server |
| US | *Upload firmware* command  | Server |
| UC | *Upload configuration* command | Server |
| M | Message to/from the driver | Server/Device |
| AM | Answer to the message from the driver | Server |
| QI | *Query snapshot* command | Server |
| I | Snapshot packet | Device |
| AI | Answer to the snapshot packet | Server |
| QT | *Query DDD file* command | Server |
| IT | DDD file information packet | Device |
| AIT | Answer to the DDD file information packet | Server |
| T | DDD file block packet | Device |
| AT | Answer to the DDD file block packet | Server |

# Login Packet

The packet is used for the device authorization on the server. Every TCP connection starts with sending this packet from the device to the server. Other data should be transferred only after the server confirms the successful authorization of the device.

The login package looks as follows:

 #L#Protocol\_version;IMEI;Password;CRC16\r\n

|  |  |
| --- | --- |
| **Field** | **Description** |
| L | Packet type: login packet. |
| Protocol\_version | Current protocol version. In this case, 2.0. |
| ; | Delimiter. |
| IMEI | IMEI, ID or serial number of the controller. |
| Password | Password to access the device. If there is none, NA is transmitted. |
| CRC16 | Checksum. See the *Checksum* section. |

**Server Response to the L Packet**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Code** | **Meaning** | **Example** |
| AL | 1 | Unit successfully authorized. | #AL#1\r\n |
| 0 | Connection rejected. Possible reasons:* Incorrect protocol version. The current one is 2.0;
* The unit is not created on the server;
* Incorrect packet structure.
 | #AL#0\r\n |
| 01 | Password verification error. | #AL#01\r\n |
| 10 | Checksum verification error. | #AL#10\r\n |

# Short Data Packet

The packet contains only navigation data and looks as follows:

 #SD#Date;Time;LatDeg;LatSign;LonDeg;LonSign;Speed;Course;Alt;Sats;CRC16\r\n

|  |  |
| --- | --- |
| **Field** | **Description** |
| SD | Packet type: short data packet. |
| Date | Date in the DDMMYY format, UTC±00:00. If there is no data, NA is transmitted. |
| Time | Time in the HHMMSS format, UTC±00:00. If there is no data, NA is transmitted. |
| LatDeg;LatSign | Latitude. LatDeg denotes degrees, LatSign denotes a cardinal point. If there is no data, NA;NA is transmitted. See Annex. |
| LonDeg;LonSign | Longitude. LonDeg denotes degrees, LonSign denotes a cardinal point. If there is no data, NA;NA is transmitted. See Annex. |
| Speed | Speed value, integer (km/h). If there is no data, NA is transmitted. |
| Course | Direction of movement, integer (from 0 to 359 degrees). If there is no data, NA is transmitted. |
| Alt | Altitude, integer (metres). If there is no data, NA is transmitted. |
| Sats | Number of satellites, integer. If there is no data, NA is transmitted. |
| CRC16 | Checksum. See the *Checksum* section.  |

|  |
| --- |
| If the *Date* and *Time* fields contain NA, the message is registered with the current server time. |

**Server Response to the SD Packet**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Code** | **Meaning** | **Example** |
| ASD | -1 | Incorrect packet structure. | #ASD#-1\r\n |
| 0 | Incorrect time. | #ASD#0\r\n |
| 1 | Packet successfully registered. | #ASD#1\r\n |
| 10 | Error receiving coordinates. | #ASD#10\r\n |
| 11 | Error receiving speed, course, or altitude. | #ASD#11\r\n |
| 12 | Error receiving the number of satellites. | #ASD#12\r\n |
| 13 | Checksum verification error. | #ASD#13\r\n |

# Extended Data Packet

The packet contains additional data structures and looks as follows:

#D#Date;Time;LatDeg;LatSign;LonDeg;LonSign;Speed;Course;Alt;Sats;HDOP;Inputs; Outputs;ADC;Ibutton;Params;CRC16\r\n

|  |  |
| --- | --- |
| **Field** | **Description** |
| D | Packet type: extended data packet. |
| Date | Date in the DDMMYY format, UTC±00:00. If there is no data, NA is transmitted. |
| Time | Time in the HHMMSS format, UTC±00:00. If there is no data, NA is transmitted. |
| LatDeg;LatSign | Latitude. LatDeg denotes degrees, LatSign denotes a cardinal point. If there is no data, NA;NA is transmitted. See Annex. |
| LonDeg;LonSign | Longitude. LonDeg denotes degrees, LonSign denotes a cardinal point. If there is no data, NA;NA is transmitted. See Annex. |
| Speed | Speed value, integer (km/h). If there is no data, NA is transmitted. |
| Course | Direction of movement, integer (from 0 to 359 degrees). If there is no data, NA is transmitted. |
| Alt | Altitude, integer (metres). If there is no data, NA is transmitted. |
| Sats | Number of satellites, integer. If there is no data, NA is transmitted. |
| HDOP | Horizontal Dilution of Precision. It shows the accuracy of the coordinates transmitted by the device. The smaller this value is, the more accurate the coordinates are. If there is no data, NA is transmitted. |
| Inputs | Digital inputs. Every bit of the number (beginning from the low-order one) corresponds to one input. Integer. If there are none, NA is transmitted. |
| Outputs | Digital outputs. Every bit of the number (beginning from the low-order one) corresponds to one output. Integer. If there are none, NA is transmitted. |
| ADC | Analog inputs. Fractional numbers separated by commas. The sensors are numbered from 1. If there are no analog inputs, an empty string is transmitted. Example: 14.77,0.02,3.6 |
| Ibutton | Driver key code. A string of arbitrary length. If there is none, NA is transmitted. |
| Params | Additional parameters. Separated by commas. See *Additional Parameters*.  |
| CRC16 | Checksum. See the *Checksum* section. |

|  |
| --- |
| If the *Date* and *Time* fields contain NA, the message is registered with the current server time. |

## Additional Parameters (Params)

Each parameter has the following structure:

 Name:Type:Value

Examples of additional parameters: count1:1:564, fuel:2:45.8, hw:3:V4.5, SOS:1:1

|  |  |
| --- | --- |
| **Field** | **Description** |
| Name | Parameter name in Latin. In lowercase.  The maximum number of characters is 38. Invalid characters: space, comma, colon, number sign, line feed and carriage return (\r\n). |
| Type | Parameter type:1 — Integer / Long;2 — Double;3 — String (the maximum number of characters: 1344). |
| Value | Parameter value. Depends on the parameter type. |

|  |
| --- |
| If the value does not correspond to the parameter type, the parameter will not be registered. |

The maximum number of parameters that can be registered in Wialon is 200. The protocol does not limit the number of the transmitted parameters.

**Fixed parameters**

* **Alarm messages**. To transmit an alarm message highlighted in red, a parameter of the first (Integer) type is used. The parameter name is SOS, in uppercase. A value of 1 means the alarm is triggered.
* **Chat with drivers**. To display a message in the *Chat with drivers*pop-up window, a parameter of the third (String) type is used. The parameter name is *text*.
* **Positioning by LBS**. To determine the location by base stations (LBS), it is required to register the following parameters:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Value** |
| mcc# | Integer | Mobile [country code](https://en.wikipedia.org/wiki/Country_code) |
| mnc# | Mobile network code |
| lac# | Local area code |
| cell\_id# | Cell identification |

# is a parameter index. It is used if it is necessary to transmit several LBS structures. In this case, the parameter names should be numbered. Examples: mcc1=12, mnc1=12, lac1=12, cell\_id1=12, mcc2=13, mnc2=13, lac2=13, cell\_id2=13.

* **Positioning by Wi-Fi**. To determine the location by Wi-Fi points, it is required to register the following parameters:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Value** |
| wifi\_mac\_# | String | MAC address. Every byte is separated by the minus sign (-). Example: 74-D8-3E-40-B8-7A |
| wifi\_rssi\_# | Integer | Indicator of the signal level |

# is a parameter index. It is used if it is necessary to transmit several parameter pairs. In this case, the parameter names should be numbered. The parameter registered in Wialon will look like 74:d8:3e:40:b8:7a, that is, separated by colons and in lower case, even if it was sent in upper case. The initial value sent by the device can’t be like this, because the colons are not allowed. Example: wifi\_mac\_1=74:d8:3e:40:b8:7a, wifi\_rssi\_1 = 74, wifi\_mac\_2=34:a2:5e:30:b8:4a, wifi\_rssi\_2 = 72.

**Server Response to the D Packet**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Code** | **Meaning** | **Example** |
| AD | -1 | Incorrect packet structure. | #AD#-1\r\n |
| 0 | Incorrect time. | #AD#0\r\n |
| 1 | Packet successfully registered. | #AD#1\r\n |
| 10 | Error receiving coordinates. | #AD#10\r\n |
| 11 | Error receiving speed, course, or altitude. | #AD#11\r\n |
| 12 | Error receiving the number of satellites or HDOP. | #AD#12\r\n |
| 13 | Error receiving Inputs or Outputs. | #AD#13\r\n |
| 14 | Error receiving ADC. | #AD#14\r\n |
| 15 | Error receiving additional parameters. | #AD#15\r\n |
| 15.1 | Error receiving additional parameters. The parameter name is more than 40 characters long. The parameter will not be registered. | #AD#15.1\r\n |
| 15.2 | Error receiving additional parameters. The parameter name contains a space, which is not allowed. The parameter will not be registered. | #AD#15.2\r\n |
| 16 | Checksum verification error. | #AD#16\r\n |

# Black Box Packet

The black box packet is used to transmit messages for the past period. The maximum number of messages that can be transmitted in one packet is 5000. The packet looks as follows:

#B#Date;Time;LatDeg;LatSign;LonDeg;LonSign;Speed;Course;Alt;Sats|Date;Time;LatDeg;LatSign;LonDeg;LonSign;Speed;Course;Alt;Sats|Date;Time;LatDeg;LatSign;LonDeg;LonSign;Speed;Course;Alt;Sats|CRC16\r\n

|  |  |
| --- | --- |
| **Field** | **Description** |
| B | Packet type: black box packet. |
| Data | A set of short (SD) or extended (D) data packets without the packet type field. The packets are separated by the vertical bar ( | ). |
| CRC16 | Checksum. See the *Checksum* section. |

**Server Response to the B Packet**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Value** | **Meaning** | **Example** |
| AB | Number | Number of packets successfully registered. | #AB#3\r\n |
| Empty string | Checksum verification error. | #AB#\r\n |

# Video

This part of the protocol describes the operation of the device and the server as far as getting and watching video is concerned. After sending the *Request live stream*, *Request playback*, *Request video file* commands, the device is connected to the remote server specified in the command. To authorize on the server, the device should first send the login packet and then data packets to the server.

Wialon

Device

Video server

Livestream, playback, video file data

Video Command

## Request Live Stream Command

The command is used for requesting live streams.

The packet looks as follows:

#QLV#hwsIP;chNum;streamType\r\n

|  |  |
| --- | --- |
| **Field** | **Description** |
| QLV | Packet type: *Request live stream* command. |
| hwsIP | IP address of the media server. |
| chNum | Channel number. Only one channel can be specified in a packet. |
| streamType | Video stream type. 0 is the main video stream. Ensures the highest video quality. 1 is a secondary video stream. Ensures a low video quality.  |
| \r\n | End of the packet. |

### Live Stream Packet

Before sending a live stream packet, it is required to send a login packet to authorize on the server. The login packet looks as follows:

#L#Protocol\_version;IMEI;Password;CRC16\r\n

After a positive server answer, the device can send a live stream packet:

#LV#date;time;chNum;encode;dataLen\r\ndata

|  |  |
| --- | --- |
| **Field** | **Description** |
| LV | Packet type: *Request live stream* command. |
| date | Date in the DDMMYY format, UTC±00:00. |
| time | Time in the HHMMSS format, UTC±00:00 |
| chNum | Channel number. Only one channel can be specified in a packet. |
| encode | Encoding:* 0-10 is video encoding;

0 – h264;* 11-20 is sound encoding;

11 – apdcm. |
| dataLen | Video data size. |
| \r\n | Indicator of the packet end. |
| data | Binary video data. Formed according to the specified encoding. |

## Request Playback Command

The command is used for requesting a video playback.

The packet looks as follows:

#QPB#hwsIP;date;time;chNum;streamType\r\n

|  |  |
| --- | --- |
| **Field** | **Description** |
| QPB | Packet type: *Request playback* command. |
| hwsIP | IP address of the media server. |
| date | Start date of the playback request in the DDMMYY format, UTC±00:0000. |
| time | Start time of the playback request in the HHMMSS format, UTC±00:0000. |
| chNum | Channel number. Only one channel can be specified in a packet. |
| streamType | Video stream type. 0 is the main video stream. Ensures the highest video quality. 1 is a secondary video stream. Ensures a low video quality. |
| \r\n | End of the packet. |

### Playback Packet

Before sending a playback packet, it is required to send a login packet to authorize on the server. The login packet looks as follows:

#L#Protocol\_version;IMEI;Password;CRC16\r\n

After a positive server answer, the device can send a playback packet:

#PB#date;time;chNum;encode;dataLen\r\nData

|  |  |
| --- | --- |
| **Field** | **Description** |
| PB | Packet type: *Request playback* command. |
| date | Frame date in the DDMMYY format, UTC±00:00. |
| time | Frame time in the HHMMSS format, UTC±00:00 |
| chNum | Channel number. Only one channel can be specified in a packet. |
| encode | Encoding:* 0-10 is video encoding;

0 – h264;* 11-20 is sound encoding;

11 – apdcm. |
| dataLen | Video data size. |
| \r\n | Indicator of the packet end. |
| Data | Binary video data. Formed according to the specified encoding. |

## Request Video File Command

The command is used for downloading a video file from the device.

The packet looks as follows:

#QVF#hwsIP;date;time;dur;chNum;streamType\r\n

|  |  |
| --- | --- |
| **File** | **Description** |
| QVF | Packet type: *Request video file* command. |
| hwsIP | IP address of the HWS server. |
| Date | Video start date in the DDMMYY format, UTC±00:00. |
| Time | Video start time in the HHMMSS format, UTC±00:00. |
| Dur | Video duration in seconds. |
| chNum | Channel number. Only one channel can be specified in a packet. |
| streamType | Video stream type. 0 is the main video stream. Ensures the highest video quality. 1 is a secondary video stream. Ensures a low video quality. |
| \r\n | End of the packet. |

### Video File Packet

Before sending a video file packet, it is required to send a login packet to authorize on the server. The login packet looks as follows:

#L#Protocol\_version;IMEI;Password;CRC16\r\n

After a positive server answer, the device can send a video file packet:

#VF#date;time;dur;chNum;container;encode;dataLen\r\ndata

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| VF | Packet type: *Request video file* command. |
| date | [Date](#Пример) in the DDMMYY format, UTC±00:00. |
| time | [Time](#Пример) in the HHMMSS format, UTC±00:00 |
| dur | Video [duration](#Пример) in seconds. |
| chNum | Channel number. Only one channel can be specified in a packet. |
| container | Media container:* 0 – mp4.
 |
| encode | Encoding:* 0-10 – video encoding;

0 – h264;* 11-20 – sound encoding;

11 – apdcm. |
| dataLen | Video data size. |
| \r\n | Indicator of the packet end. |
| Data | Binary video data. Formed according to the specified encoding. |

If the start time of a file from the device memory is later than the start time requested by the QVF command, the device sends the actual start time of the video file.

**Example:**

The start time 12:00:00 and a duration of 60 seconds have been requested by the QVF command. However, the device only has files starting from 12:00:30. In response to the VF command, the device will send the parameter **time=12:00:30**, and a shorter duration **dur=30**, respectively. In this case, the response to the command will be as follows:

#VF#date;120030;30;chNum;container;encode;dataLen\r\ndata

## Request Timeline Command

The command is used for requesting a list of the video files stored in the device.

The packet looks as follows:

#QTM#sDate;sTime;eDate;eTime;chNum\r\n

|  |  |
| --- | --- |
| **Field** | **Description** |
| QTM | Packet type: *Request timeline* command |
| sDate | Start date of the time interval in the DDMMYY format, UTC±00:00. |
| sTime | Start time of the interval in the HHMMSS format, UTC±00:00. |
| eDate | End date of the time interval in the DDMMYY format, UTC±00:00. |
| eTime | End time of the interval in the HHMMSS format, UTC±00:00. |
| chNum | Channel number. Several channels can be specified in a packet (separated by commas). Example: 1, 2, 3 |
| \r\n | End of the packet. |

### Timeline Packet

A packet with a list of video files is received in the same connection from which the command was sent. The packet looks as follows:

#TM#seqNum;count;list\r\n

|  |  |
| --- | --- |
| **Field** | **Description** |
| TM | Packet type: *Request timeline* command |
| seqNum | The number of files in the packet at the moment. |
| count | The total number of files to be transferred. |
| list | List of file names (separated by commas). A file name should be transferred in the format specified below. Otherwise, it will be ignored.**File name format:** {video start time DDMMYYHHMMSS utc 0}\_{duration in seconds}\_{channel number}.{file format}Example: 010521102256\_15\_1.mp4, 010521112341\_12\_1.mp4 |
| \r\n | End of the packet. |

# Ping Packet

The packet is used to maintain an active TCP connection with the server and to verify the availability of the channel. The packet looks as follows:

#P#\r\n

**Server Response to the P Packet**

|  |  |  |
| --- | --- | --- |
| **Type** | **Meaning** | **Example** |
| AP | Positive server response. | #AP#\r\n |

# Commands

## Upload Firmware Command

The command is used to transfer the firmware data from the server to the controller. The packet looks as follows:

#US#Sz;CRC16\r\nBIN

|  |  |
| --- | --- |
| **Field** | **Description** |
| US | Packet type: firmware packet. |
| Sz  | Size of the binary data of the firmware (bytes). |
| CRC16 | Checksum. See the *Checksum* section. |
| BIN | Firmware in binary format. |

## Upload Configuration Command

The command is used to transfer the configuration file from the server to the controller. The packet looks as follows:

#UC#Sz;CRC16\r\nBIN

|  |  |
| --- | --- |
| **Field** | **Description** |
| UC | Packet type: configuration packet. |
| Sz  | Size of the configuration file (bytes). |
| CRC16 | Checksum. See the *Checksum* section. |
| BIN | Contents of the configuration file. |

## Send Message to Driver Command

The command is used to exchange text messages between the server and the driver. The packet format is the same for the server and for the controller:

#M#Msg;CRC16\r\n

|  |  |
| --- | --- |
| **Field** | **Description** |
| М | Packet type: message to/from the driver. |
| Msg | Text of the message. If the message is sent from Wialon to the device, there is no size limit. If it is sent from the device to Wialon, the size limit is 4Kbytes. |
| CRC16 | Checksum. See the *Checksum* section. |

**Server Response to the M Packet**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Code** | **Meaning** | **Example** |
| AM | 1 | Message received. | #AM#1\r\n |
| 0 | Error receiving messages. | #AM#0\r\n |
| 01 | Checksum verification error. | #AM#01\r\n |

## Query Snapshot Command

The command is sent from the server to the controller to request a photograph.



The packet looks as follows:

#QI#\r\n

|  |  |
| --- | --- |
| **Field** | **Description** |
| QI | 1. Packet type: the *Query snapshot* command.
 |

### Snapshot Packet

The packet is used to transfer the image data to the Wialon server. The image is divided into blocks of bytes, each of which is sent to the server as a separate packet. The recommended block size is up to 50 KB. If the server cannot receive any image block, it disconnects. In this case, it is recommended to reduce the size of the blocks.

The packet looks as follows:

#I#Sz;Ind;Count;Date;Time;Name;CRC16\r\nBIN

|  |  |
| --- | --- |
| **Field** | **Description** |
| I | Packet type: snapshot packet. |
| Sz  | Size of the binary data of the packet (for example, 51200 bytes). |
| Ind | Index number of the transmitted block (numbering from zero). |
| Count | Number of the last block (numbering from 0). |
| Date | Date in the DDMMYY format, UTC±00:00. |
| Time | Time in the HHMMSS format, UTC±00:00. |
| Name | Name of the transmitted image. |
| CRC16 | Checksum. See the *Checksum* section. |
| BIN | Binary image block of the *Sz* size. |

**Server Response to the I Packet**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **Block number** | **Code** | **Meaning** | **Example** |
| AI | Ind  | 1 | Packet received. | #AI#Ind;1\r\n |
| AI | Ind | 0 | Error receiving packet. | #AI#Ind;0\r\n |
| AI | Ind | 01 | Checksum verification error. | #AI#Ind;01\r\n |
| AI | NA | 0 | Incorrect packet structure. | #AI#NA;0\r\n |
| AI | None | 1 | Image fully received and saved in Wialon. | #AI#1\r\n |

**Ind**. The index number of the transmitted image block. Integer.

When the image is fully received and saved in Wialon, the serverresponse contains only one parameter: code (#AI#1\r\n).

## Query DDD File Command

The command is sent from the server to the device to request a tachograph file.



The packet looks as follows:

#QT#DriverID\r\n

|  |  |
| --- | --- |
| **Field** | **Description** |
| QT | Packet type: the *Query DDD file* command. |
| DriverID  | Driver identification. |

### DDD File Information Packet

The packet contains information about the tachograph file transmitted to the server. All fields are required. This information is necessary to save the file correctly and bind it to the appropriate driver in Wialon. The saved file is named as follows: *driverid\_yyyymmdd\_hhmmss.ddd*. You should transfer this packet before transmitting the DDD file.

 The packet looks as follows:

#IT#Date;Time;DriverID;Code;Count;CRC16\r\n

|  |  |
| --- | --- |
| **Field** | **Description** |
| IT | Packet type: DDD file information packet.  |
| Date | Date in the DDMMYY format, UTC±00:00. |
| Time | Time in the HHMMSS format, UTC±00:00. |
| DriverID | Driver identification. |
| Code | Error code. If there are no errors, an empty string is transmitted. |
| Count | Total amount of the DDD file blocks. |
| CRC16 | Checksum. See the *Checksum* section. |

**Server Response to the IT Packet**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Code** | **Meaning** | **Example** |
| AIT | 1 | Packet received. | #AIT#1\r\n |
| 0 | Error receiving packet. | #AIT#0\r\n |
| 01 | Checksum verification error. | #AIT#01\r\n |

### DDD File Block Packet

The packet is used to transfer DDD file data blocks and looks as follows:

#T#Code;Sz;Ind;CRC16\r\nBIN

|  |  |
| --- | --- |
| **Field** | **Description** |
| T | Packet type: DDD file block packet.  |
| Code | Error code. If there are no errors, an empty string is transmitted. |
| Sz | Size of the binary data of the packet (bytes). |
| Ind | Index number of the transmitted block (numbering from zero). |
| CRC16 | Checksum. See the *Checksum* section. |
| BIN | Binary file block of the *Sz* size. |

**Server Response to the T Packet**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **Block number** | **Code** | **Meaning** | **Example** |
| AT | Ind | 1 | Packet received. | #AT#Ind;1\r\n |
| Ind | 0 | Error receiving packet. | #AT#Ind;0\r\n |
| Ind | 01 | Checksum verification error. | #AT#Ind;01\r\n |
| None | 1 | DDD file fully received and saved in Wialon. | #AT#1\r\n |

**Ind**. The index number of the transmitted DDD file block. Integer.

When the image is fully received and saved in Wialon, the serverresponse contains only one parameter: code (#AT#1\r\n).

|  |
| --- |
| All DDD file block packets should be transmitted using the same TCP connection as the DDD file information packet. |

## Send Custom Message Command

The command is used to send custom messages to the device. It allows to implement additional features necessary for the controller.

In response to the command, you can send a *Message to/from the driver* packet. If you need to transfer the position data and other parameters, you can transmit an extended data packet.

The custom command sent to the device looks as follows:

Msg\r\n

|  |  |
| --- | --- |
| **Field** | **Description** |
| Msg | Text of the message. |

# UDP Data Transfer

The UDP protocol is used only to transfer data from the controller to the server. It is not possible to send commands from the server to the device using this protocol.

## General Structure of UDP Messages

A UDP packet has the same structure as a TCP packet with the only difference that the prefix *Protocol\_version;IMEI* is added at the beginning. The packet transferred using UDP looks as follows:

 Protocol\_version;IMEI#PT#MsgCRC\r\n

|  |  |
| --- | --- |
| **Field** | **Description** |
| Protocol\_version | Current protocol version. 2.0 is used now. |
| ; | Delimiter. |
| IMEI | IMEI of the device. |
| # | Start byte. |
| PT | Packet type. See the *Packet types* table. |
| # | Delimiter. |
| Msg | Text of the message. |
| CRC | CRC16 checksum. |
| \r\n | End of the packet (0x0D0A in HEX). |

The SD packet structure (UDP transfer):

2.0;IMEI#SD#Date;Time;LatDeg;LatSign;LonDeg;LonSign;Speed;Course;Alt;Sats;CRC16\r\n

# Data Compression

To save traffic, it is appropriate to use data compression while transferring packets which contain a large amount of data. The [DEFLATE](https://tools.ietf.org/html/rfc1951) algorithm of the cross-platform [«zlib»](http://www.zlib.net/)  library is used for compression. Both TCP and UDP transport protocols are supported. The container should consist of only one packet in text format.

**Compressed Packet Container Structure**

|  |  |  |  |
| --- | --- | --- | --- |
| **Size (bytes)** | 1  | 2  |  |
| **Field** | Head | Len | Data |

Head — 0xFF.

Len. The *Data* field length (little-endian, 16-bit integer).

Data. The compressed binary data block of the specified size. Transmitted as it is.

You can transfer the compressed and regular packets of the Wialon IPS protocol simultaneously. The packets sent from the server are always regular (not compressed) because of their small size.

When implementing the library, the identifiers Z\_DEFAULT\_COMPRESSION, Z\_DEFLATED, Z\_DEFAULT\_STRATEGY affect the result, but the message is valid in any case.

**Compressed L Packet Example**

HEX:

|  |
| --- |
| FF1B00780153F65136D233B0CECC4DCDB4F673B476B4343602002FF404E6  |

Text:

|  |
| --- |
| #L#2.0;imei;NA;A932  |

**Compressed D Packet Example**

HEX:

|  |
| --- |
| FF76007801258CCB0AC24010043F26D77599DE9931ECF4C9D7351EF2050145024625F8FF9818EA540DD5CDB9290A41215CAB57BABA65AB652FEC28A55564B35A8517CA828AB02532FE86242BEC0E1C1FAF4020DD3EC33C4C5142330CBE1C79FA6E9BC6F33DDFA7346E8AD8B9A7FEDAAF1DED78D21FEF7522F7 |

Text:

|  |
| --- |
| 231012;153959;5354.49260;N;02731.44990;E;0;0;300;7;1.1;0;0;1,0,0,0;NA;ign:1:1,dparam:2:3.14159265,tparam:3:lorem,iparam:1:-55,SOS:1:1;4BC3 |

# Checksum

The CRC16 checksum should be added to the message as a hexadecimal number in ASCII characters. The byte order is big-endian.

 Example: 0xFC45 => 0x46433435

**Checksum Calculation**

|  |  |
| --- | --- |
| **Packet type** | **Explanation** |
| SD | Message example: #SD#Date;Time;LatDeg;LatSign;LonDeg;LonSign;Speed;Course;Alt;Sats; CRC16\r\nThe checksum is calculated for the following part of the packet: Date;Time;LatDeg;LatSign;LonDeg;LonSign;Speed;Course;Alt;Sats; |
| B | Message example: #B#Date;Time;LatDeg;LatSign;LonDeg;LonSign;Speed;Course;Alt;Sats|Date;Time;LatDeg;LatSign;LonDeg;LonSign;Speed;Course;Alt;Sats|CRC16\r\nThe checksum is calculated for the following part of the packet: Date;Time;LatDeg;LatSign;LonDeg;LonSign;Speed;Course;Alt;Sats|Date;Time;LatDeg;LatSign;LonDeg;LonSign;Speed;Course;Alt;Sats| |
| IUS UC T | Message example: #I#51200;0;1;070512;124010;sample.jpg;CRC16\r\nBINThe checksum is calculated for the *BIN* field only.  |
| LSDDBMIT | The checksum is calculated for the part of the packet between the packet type and the *CRC16* field. |

 **C Code Example for CRC16 Calculation**

|  |
| --- |
| 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;} |

# Annex

The coordinates are compliant with the NMEA 0183 standard.

**DDMM.MM** is the format of latitude. Two digits of degrees (DD). If the degree value consists of one digit, the degree field still contains two digits. That is, the field is filled with zeros, for example, 01. The degrees are followed by two digits of integer minutes, a point, and a fractional part of minutes of variable length. The leading zeros are not omitted. N denotes north (positive) latitude, S denotes south (negative) latitude.

Example: 5544.6025;N (LatDeg - 5544.6025, LatSign - N)

55 is a degree value.

44.6025 / 60 = 0,743375 is a minute value.

N is north latitude (positive sign).

55 + 0,743375 = +55,743375‬

**DDDMM.MM** is the format of longitude. Three digits of degrees (DDD). If the degree value consists of one digit, the degree field still contains three digits. That is, the field is filled with zeros, for example, 001. The degrees are followed by two digits of integer minutes, a point, and a fractional part of minutes of variable length. The leading zeros are not omitted. E denotes east (positive) longitude, W denotes west (negative) longitude.

Example: 03739.6834;E (LonDeg - 03739.6834, LonSign - E)

037 is a degree value.

39.6834 is a minute value.

E is east longitude (positive sign).

037 + 39.6834 = +37,66139