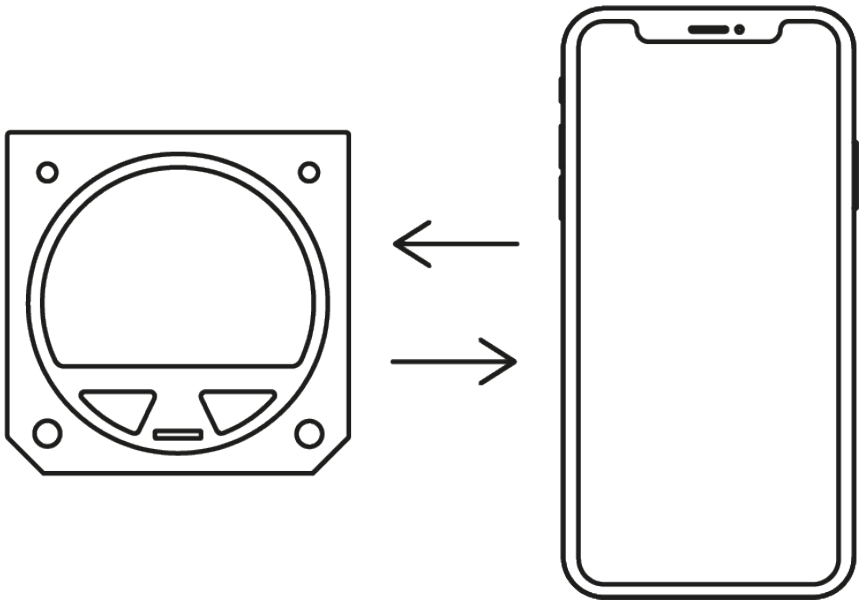
LX navigation Data Port



Communication Protocols

- LX navigation -

May, 2020

navigation

Tkalska ulica 10
SI-3000 Celje
Tel.: 00 386 3 490 46 70
Fax.: 00 386 3 490 46 71
info@lxnavigation.com
www.lxnavigation.com

Document information

0.1 Abstract

LX NMEA protocol is used for communication between LX navigation devices and third party peripherals (User devices hereafter).

Physical layer can be either LX user serial port or bluetooth interface on devices supporting wireless connectivity.

For NMEA communication check LX NMEA 1.0 Protocol and LX NMEA 2.0 Protocol. These subsections define additional requests and responses which supplement LX Binary Protocol datagrams or support new features.

This document is intended to aid developers of third party devices to enable full communication to LX navigation devices.

0.2 Document status

Document status: PUBLIC

Document status	Explanation
Internal	Intended only for LX navigation staff
Public	Available publicly to all
Personal	Intended for a specific person and/or company, noted on this page
Dealer	Intended for a specific dealer, noted on this page
Manufacturer	Intended for a specific manufacturer, noted on this page

0.3 List of applicable products

Device	Version
LX Eos 57	V1.9 or later
LX Eos 80	V1.5 or later
LX Era [57 & 80]	V1.5 or later
LX Colibri X	V1.5 or later
LX Zeus	V5.0 or later

0.4 Revision history

Document name	Document revision	Date	Written by	Approved by	Notes
LX_CP	R1	14.4.2020	A.S. A.S.	N.S. N.S.	LX communication protocol created Conversion to LaTeX

0.5 Disclaimer

Contents

LX NMEA 1.0 Protocol

LX NMEA protocol is used for communication between LX navigation devices and third party peripherals (User devices hereafter). It is based on NMEA 0183 standard.

Physical layer can be either LX user serial port or bluetooth interface on devices supporting wireless connectivity.

For additional sentences check LX NMEA 2.0 Protocol.

1.1 User port

LX user port is a RS232 UART serial interface intended for communication between LX devices and third party devices.

Following is a list of serial interface parameters:

- Baud Rate: 4800 - 115200
- Parity: none
- Data bits: 8
- Stop bits: 0
- Flow control: none/off

Physical connector pinout is described in LX device installation manual.

1.2 General

NMEA sentences are ASCII formatted strings of variable length. String is an array of parameters of various types separated by comma. Following is a general packet structure:

$<data_type>,<parameter_1>,<parameter_2>,...<parameter_n>*<CRC><CR><LF>

All packets starts with "$<data_type>," which is followed by a number of parameters.

The last parameter is followed by "*" and CRC. CRC contains two bytes which are ASCII representation of 8-bit CRC in hex.

The sentence is terminated with carriage return and line feed characters (0D 0A).

The following table lists NMEA sentences supported by LX devices. Note that some sentences are output from LX device. They are addressed as "responses". While other, that are expected to be received, are addressed "requests".

Data type	Response	Request
LXWP0	X	X *1
LXWP1	X	
LXWP2	X	
LXWP3	X	

Data type	Response	Request
GPRMB	X	
PFLX0		X
PFLX2		X

*1 - used for Condor simulator mode.

1.3 CRC calculation

A simple CRC is added to the end of each packet to detect data integrity faults. CRC is calculated from all bytes between "$" (excluding) and "*" (excluding).

Following is the algorithm for CRC calculation.

```
uint8_t byCRC = 0;

for(int32_t i=0; i<iN; i++)
{
    byCRC ^= pString[i];
}
```

1.4 Sentences

1.4.1 PFLX0 (User -> LX)

User device sets LXWP<N> sentences output intervals. Parameters are a list of <data_type_N,interval_N> pairs.

Sentence length is variable. Minimum pair count per sentence is 1, maximum is 4.

Available interval values are: -1 = send once, 0 = disabled, 1, 2, 3...= interval in seconds. After given sentence is requested once, it's interval is reset to default.

Request: *$PFLX0,<data_type_1>,<interval_1>[,<data_type_2>,<interval_2>][,<data_type_3>,<interval_3>][,<data_type_4>,<interval_4>]*<CRC><CR><LF>*

Examples:

```
TX: $PFLX0,LXWP0,1,LXWP1,1,LXWP2,1,LXWP3,1*32
<- all LXWPx sentences will be output once per second

TX: $PFLX0,LXWP0,0,LXWP1,0,LXWP2,0,LXWP3,0*32
<- all LXWPx sentences will be disabled

TX: $PFLX0,LXWP1,0,LXWP3,5*35
<- disable LXWP1 sentence and set LXWP3 output interval to once per 5s

TX: $PFLX0,LXWP3,-1*0E
<- request LXWP3 sentence only once
```

NOTE: If one user device is connected to user port and the second via Bluetooth interface each user device sets it's own output intervals.

1.4.2 PFLX2 (User -> LX)

User device sets MacCready, ballast, bugs factor, polar and Volume on LX device.

Request: *$PFLX2,<mc>,<load_factor>,<bugs>,<polar_a>,<polar_b>,<polar_c>,<volume>*<CRC><CR><LF>*

Parameter	Data type	Description
<mc>	float	MacCready factor
<load_factor>	float	Total glider mass divided by polar reference mass
<bugs>	uint16_t	Bugs factor in percent
<polar_a>	float	Polar -square coefficient, velocity in m/s
<polar_b>	float	Polar -linear coefficient, velocity in m/s
<polar_c>	float	Polar -constant coefficient, velocity in m/s
<volume>	uint8_t	Variometer volume in percent

Example

```
$PFLX2,1.1,1.94,15,2.77,-3.12,1.20,75*14
```

1.4.3 LXWP0 (User <-> LX)

LX device outputs basic flight data parameters.

This sentence can be also used for supplying flight data to LX device during Condor simulator mode.

Response: *$LXWP0,<is_logger_running>,<tas>,<altitude>,<vario1>,<vario2>,<vario3>,<vario4>,<vario5>,<vario6>,<heading>,<wind_direction>,<wind_speed>*<CRC><CR><LF>*

Parameter	Data type	Description
<is_logger_running>	char	'Y'=yes, 'N'=no
<tas>	float	True airspeed in km/h
<altitude>	float	True altitude in meters
<varioN>	float	6 measurements of vario in last second in m/s
<heading>	uint16_t	True heading in degrees. Blank if compass not connected.
<wind_direction>	string	Wind direction in degrees. Blank if wind speed is 0.0.
<wind_speed>	string	Wind speed in km/h. Blank if wind speed is 0.0.

Example:

```
RX: $LXWP0,Y,119.4,1717.6,0.02,0.02,0.02,0.02,0.02,0.02,,000,107.2*5b
```

1.4.4 LXWP1 (User <- LX)

LX device outputs basic device information.

Response: *$LXWP1,<device_name>,<serial>,<sw_version>,<hw_version>*CRC<CR><LF>*

Parameter	Data type	Description
<device_name>	string	LX device name
<serial>	uint32_t	serial number

Parameter	Data type	Description
<sw_version>	float	firmware version
<hw_version>	float	hardware version

Example:

```
RX: $LXWP1,LX Eos,34949,1.5,1.4*7d
```

1.4.5 LXWP2 (User <- LX)

LX device outputs MacCready, load factor, bugs, volume and polar data.

Response: *$LXWP2,<mc>,<load_factor>,<bugs>,<polar_a>,<polar_b>,<polar_c>,<volume>*<CRC><CR><LF>*

Parameter	Data type	Description
<mc>	float	MacCready factor
<load_factor>	float	Total glider mass divided by polar reference mass
<bugs>	uint16_t	Bugs factor in percent
<polar_a>	float	Polar -square coefficient, velocity in m/s
<polar_b>	float	Polar -linear coefficient, velocity in m/s
<polar_c>	float	Polar -constant coefficient, velocity in m/s
<volume>	uint8_t	Variometer volume in percent

Example:

```
RX: $LXWP2,1.5,1.11,13,2.96,-3.03,1.35,45*02
```

1.4.6 LXWP3 (User <- LX)

LX device outputs detailed vario and speed command parameters.

Response: *$LXWP3,<alt_offset>,<sc_mode>,<filter>,<reserved>,<te_level>,<int_time>,<range>,<silence>,<switch_mode>,<speed>,<polar_name>*<CRC><CR><LF>*

Parameter	Data type	Description
<alt_offset>	int16_t	Difference between true and standard altitude in feet
<sc_mode>	uint8_t	SC mode. 0 = manual, 1 = circling, 2 = speed
<filter>	float	SC filter factor in seconds
<reserved>		Reserved
<te_level>	uint16_t	TE level in percent
<int_time>	uint16_t	SC integration time in seconds
<range>	uint8_t	SC range in m/s
<silence>	float	SC silence in m/s
<switch_mode>	uint8_t	SC switch mode. 0 = off, 1 = on, 2 = toggle.
<speed>	uint16_t	SC speed in km/h
<polar_name>	string	Self explanatory
<reserved>		Reserved

Example:

```
RX: $LXWP3,0,2,5.0,0,29,20,10.0,1.3,1,120,0,KA6e,0*74
```

1.4.7 GPRMB (User <- LX)

LX device outputs location and tracking data.

Response: *$GPRMB,<gps_validity>,<parameter1>,<parameter2>,<parameter3>,<name>,<latitude>,<hemisphere_lat>,<longitude>,<hemisphere_lon>,<distance_to_tp>,<bearing_to_tp>,<approaching_speed>,<inside_600>*<CRC><CR><LF>*

Parameter	Data type	Description
<gps_validity>	char	'A'=valid, 'V'=invalid
<parameter1>	float	Reserved
<parameter2>	char	Reserved
<parameter3>	string	Reserved
<name>	string	Turnpoint name
<latitude>	string	Formated turnpoint latitude. 4614.367 = 46°14.367'. Only positive values are valid. Hemisphere is defined by
<hemisphere_lat>	char	'N'=north, 'S'=south
<longitude>	string	Formated turnpoint latitude. 01513.482 = 15°23.482'. Only positive values are valid. Hemisphere is defined by
<hemisphere_lon>	char	'E'=north, 'W'=south
<distance_to_tp>	float	In nautical miles
<bearing_to_tp>	float	In degrees
<approaching_speed>	float	In knots
<inside_600>	char	Are we inside the 600m circle around turnpoint. 'A'=inside, 'V'=outside

Example:

```
RX: $GPRMB,A,0.00,R,,CELJE,4614.367,N,01513.482,E,1.7,273.8,0.0,A*7f<CR>
<LF><CR><LF>
```

LX NMEA 2.0 Protocol

LX NMEA 2.0 is an extension of LX NMEA 1.0 protocol. It defines additional requests and responses which supplement LX binary datagrams or support new features.

For additional sentences check LX NMEA 1.0 Protocol.

2.1 General

NMEA sentences are ASCII formatted strings of variable length. String is an array of parameters of various types separated by comma. All packets starts with "$<data_type>". Currently two different data types are implemented:

- LXBC -Stands for BroadCast.
- LXDT -Stands for Data Transfer.

2.1.1 LXBC

Following is a general LXBC sentence structure:

$LXBC,<sentence_code>,<parameter_1>,<parameter_2>,...<parameter_n>*<CRC><CR><LF>

Sentence starts with "$LXBC,<sentence_code>" which is followed by a number of parameters depending on given sentence code.

Number of parameters can be more or equal 0. The last parameter is followed by "*" and CRC. CRC contains two bytes which are ASCII representation of 8-bit CRC in hex.

All sentences are terminated with carriage return and line feed characters (0x0D 0x0A).

2.1.2 LXDT

Following is a general LXDT sentence structure:

$LXDT,<sentence_action>,<sentence_code>,<parameter_1>,<parameter_2>,...<parameter_n>*<CRC><CR><LF>

Sentence starts with "$LXDT," which is followed by action (<sentence_action>) and code (<sentence_code>). There are three actions available:

- GET -User device requests data from LX device,
- SET -User device sends data to LX device,
- ANS -LX device responds to GET or SET action.

Following table shows all supported sentence codes.

Code	GET	SET	ANS	Description
INFO	X		X	Get LX device info.

Code	GET	SET	ANS	Description
TP	X	X	X	Get or set task turnpoint.
ZONE	X	X	X	Get or set task obs. zone.
GLIDER	X	X	X	Get or set glider data.
PILOT	X	X	X	Get or set pilot data.
TSK_PAR	X	X	X	Get or set AAT, finish altitude.
MC_BAL	X	X	X	Get or set MacCready, ballast, bugs, volume... Automatic output on change.
RADIO	X	X	X	Get or set Radio parameters. On change send it out.
R_SWITCH		X		Switch radio frequencies.
R_DUAL		X		Set radio dual mode.
R_SPACING		X		Set radio frequency spacing.
FLIGHTS_NO	X		X	Get number of flights.
FLIGHT_INFO	X		X	Get info about the flight.
ERROR			X	Error occurred.
OK			X	LX device acknowledges reception of SET action.

Code is followed by a number of parameters depending on the given code. Number of parameters can be more or equal 0.

The last parameter is followed by "*" and CRC. CRC contains two bytes which are ASCII representation of 8-bit CRC in hex.

All sentences are terminated with carriage return and line feed characters (0x0D 0x0A).

LX device responds with ANS action to all requests. If unknown request is received it responds with ANS,ERROR sentence. Otherwise it responds with ANS,OK on SET actions and with corresponding ANS, on GET actions. If no response was received after a sentence was sent to LX device, the reason in most cases is incorrect CRC calculation.

2.2 CRC calculation

A simple CRC is added to the end of each packet to detect data integrity faults. CRC is calculated from all bytes between "$" (excluding) and "*" (excluding).

Following is the algorithm for CRC calculation.

```
uint8_t byCRC = 0;

for(int32_t i=0; i<iN; i++)
{
    byCRC ^= pString[i];
}
```

2.3 Sentences

2.3.1 Get info

User devices requests LX device name, serial, version numbers etc.

Request: *$LXDT,GET,INFO*5C<CR><LF>*

Response: *$LXDT,ANS,INFO,<device_name>,<serial>,<sw_version>,<hw_vresion>,<id>,<checksum>,<as>,<apt>*<CRC><CR><LF>*

Parameter	Data type	Description
<device_name>	string	LX device name
<serial>	uint32_t	serial number
<sw_version>	float	firmware version
<hw_version>	float	hardware version
<id>		TBD
<checksum>		TBD
<as>		TBD
<apt>		TBD

Example:

```
TX: $LXDT,GET,INFO*5C<CR><LF>
RX: $LXDT,ANS,INFO,LX Era,34949,1.4,1.1,0-[0],00,Empty,Empty*29<CR><LF>
```

2.3.2 Get TP

User device requests task turnpoint data from LX device.

Request: *$LXDT,GET,TP,<id>*<CRC><CR><LF>*

Parameter	Data type	Description
<id>	uint16_t	Turnpoint id. id = 0 represents Takeoff TP.

Response: *$LXDT,ANS,TP,<id>,<type>,<lat>,<lon>,<name>*<CRC><CR><LF>*

Parameter	Data type	Description
<id>	uint16_t	Turnpoint id.
<type>	uint8_t	Turnpoint type. 1 = point, 2 = landing, 3 = takeoff
<lat>	int32_t	Latitude in thousands of minutes (60000 = 1° 0.0')
<lon>	int32_t	Longitude in thousands of minutes (60000 = 1° 0.0')
<name>	string	Turnpoint name.

Example:

```
TX: $LXDT,GET,TP,2*48<CR><LF>
RX: $LXDT,ANS,TP,2,2,2748617,906762,NOVO MESTO *1d<CR><LF>
```

2.3.3 Set TP

User device sets task turnpoint on LX device. The lowest valid <id> is 0 and represents Start TP.

Request: *$LXDT,SET,TP,<id>,<total_tp_count>,<lat>,<lon>,<name>*<CRC><CR><LF>*

Parameter	Data type	Description
<id>	uint16_t	Turnpoint id.
<total_tp_count>	uint8_t	Number of all turnpoints in task including Takeoff and Landing.
<lat>	int32_t	Latitude in thousands of minutes (60000 = 1° 0.0')
<lon>	int32_t	Longitude in thousands of minutes (60000 = 1° 0.0')
<name>	string	Turnpoint name.

Example:

```
TX: $LXDT,SET,TP,0,5,2748617,906762,NOVO MESTO*26<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>
```

2.3.4 Get zone

User device requests task zone data from LX device. The lowest valid <id> is 0 and represents Start TP zone.

Request: *$LXDT,GET,ZONE,<id>*<CRC><CR><LF>*

Parameter	Data type	Description
<id>	uint16_t	Turnpoint id. id = 0 represents Start TP zone.

Response: *$LXDT,ANS,ZONE,<id>,<direction>,<is_auto_next>,<is_line>,<a1>,<a2>,<a21>,<r1>,<r2>, <elevation>*<CRC><CR><LF>*

Parameter	Data type	Description
<id>	uint16_t	Turnpoint id.
<direction>	uint8_t	Zone direction/orientation type. 0 = symmetric, 1 = Fixed (according to), 2 = to next, 3 = to previous, 4 = to start
<is_auto_next>	boolean	True or false.
<is_line>	boolean	True or false.
<a1>	uint16_t	Angle A1 in degrees
<a2>	uint16_t	Angle A2 in degrees
<a21>	uint16_t	Angle A21 in degrees
<r1>	uint16_t	Radius R1 in meters
<r2>	uint16_t	Radius R1 in meters
<elevation>	uint16_t	Turnpoint elevation in meters

Example:

```
TX: $LXDT,GET,ZONE,2*52<CR><LF>
RX: $LXDT,ANS,ZONE,2,3,0,1,90,60,309,5000,3500,174*42<CR><LF>
```

2.3.5 Set zone

User device sets task zone on LX device. The lowest valid is 0 and represents Start TP zone.

Request: *$LXDT,SET,ZONE,<id>,<direction>,<is_auto_next>,<is_line>,<a1>,<a2>,<a21>,<r1>,<r2>, <elevation>*<CRC><CR><LF>*

Parameter	Data type	Description
<id>	uint16_t	Turnpoint id.
<direction>	uint8_t	Zone direction/orientation type. 0 = symmetric, 1 = Fixed (according to<a21>), 2 = to next, 3 = to previous, 4 = to start
<is_auto_next>	boolean	Self explanatory. True or false.
<is_line>	boolean	Self explanatory. True or false.
<a1>	uint16_t	Angle A1 in degrees
<a2>	uint16_t	Angle A2 in degrees
<a21>	uint16_t	Angle A21 in degrees
<r1>	uint16_t	Radius R1 in meters
<r2>	uint16_t	Radius R1 in meters
<elevation>	uint16_t	Turnpoint elevation in meters

Example:

```
TX: $LXDT,SET,ZONE,2,1,1,1,90,60,309,5000,3500,174*5F<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>
```

2.3.6 Get glider

User device requests glider data from LX device.

Request: *$LXDT,GET,GLIDER*43<CR><LF>*

Response: *$LXDT,ANS,GLIDER,<polar_name>,<reg_no>,<comp_id>,<class>*<CRC><CR><LF>*

Parameter	Data type	Description
<polar_name>	string	Self explanatory.
<reg_no>	string	Registration number
<comp_id>	string	Competition id.
<class>	string	Class.

Example:

```
TX: $LXDT,GET,GLIDER*43<CR><LF>
RX: $LXDT,ANS,GLIDER,JS3 15m,D-KLXD,XD,OPEN*50<CR><LF>
```

2.3.7 Set glider

User device sets glider data on LX device.

Request: *$LXDT,SET,GLIDER,<reg_no>,<comp_id>,<class>*<CRC><CR><LF>*

Parameter	Data type	Description
<reg_no>	string	Registration number
<comp_id>	string	Competition id.
<class>	string	Class.

Example:

```
TX: $LXDT,SET,GLIDER,D-KLXD,XD,OPEN*01<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>
```

NOTE: Glider name can not be set because it depends on the selected polar.

2.3.8 Get pilot

User device requests pilot data from LX device.

Request: *$LXDT,GET,PILOT*1C<CR><LF>*

Response: *$LXDT,ANS,PILOT,<name>,<surname>*<CRC><CR><LF>*

Parameter	Data type	Description
<name>	string	Pilot name.
<surname>	string	Pilot surname.

Example:

```
TX: $LXDT,GET,PILOT*1C<CR><LF>
RX: $LXDT,ANS,PILOT,ACE,FLYER*15<CR><LF>
```

2.3.9 Set pilot

User device sets pilot data on LX device.

Request: *$LXDT,SET,PILOT,<name>,<surname>*<CRC><CR><LF>*

Parameter	Data type	Description
<name>	string	Pilot name.
<surname>	string	Pilot surname.

Example:

```
TX: $LXDT,SET,PILOT,ACE,FLYER*0B<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>
```

2.3.10 Get task parameters

User device requests additional task settings from LX device.

Request: *$LXDT,GET,TSK_PAR*02<CR><LF>*

Response: *$LXDT,ANS,TSK_PAR,<finish_1000>,<finish_alt_offset>,<aat_time>*<CRC><CR><LF>*

Parameter	Data type	Description
<finish_1000>	boolean	True if finish 1000m below starting point option is enabled and vice versa.
<finish_alt_offset>	uint16_t	Altitude offset in meters. Difference between finish point elevation and finish altitude.
<aat_time>	string	HH:MM formated time.

Example:

```
TX: $LXDT,GET,TSK_PAR*02<CR><LF>
RX: $LXDT,ANS,TSK_PAR,1,700,02:30*19<CR><LF>
```

2.3.11 Set task parameters

User device sets additional task settings on LX device.

Request: *$LXDT,SET,TSK_PAR,<finish_1000>,<finish_alt_offset>,<aat_time>*<CRC><CR><LF>*

Parameter	Data type	Description
<finish_1000>	boolean	True if finish 1000m below starting point option is enabled and vice versa.
<finish_alt_offset>	uint16_t	Altitude offset in meters. Difference between finish point elevation and finish altitude. If <finish_1000> == 1, this parameter can be blank.
<aat_time>	string	HH:MM formated time.

Example:

```
TX: $LXDT,SET,TSK_PAR,0,700,02:30*06<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>

TX: $LXDT,SET,TSK_PAR,1,,02:30*30<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>
```

2.3.12 Get MC/Bal parameters

User device requests MacCready, ballast, bugs, brightness and volume level from LX device.

Request: *$LXDT,GET,MC_BAL*<CRC><CR><LF>*

Response: *$LXDT,ANS,MC_BAL,<mc>,<ballast>,<bugs>,<brightness>,<vario_vol>,<sc_vol>*<CRC><CR><LF>*

Parameter	Data type	Description
<mc>	float	MacCready factor
<ballast>	uint16_t	Ballast in kg
<bugs>	uint8_t	Bugs factor in percent
<brightness>	uint8_t	Screen brightness in percent
<vario_vol>	uint8_t	Variometer volume in percent
<sc_vol>	uint8_t	SC volume in percent

Example:

```
TX: $LXDT,GET,MC_BAL*4C<CR><LF>
RX: $LXDT,ANS,MC_BAL,1.1,200,30,55,70,20*5c<CR><LF>
```

NOTE: $LXDT,ANS,MC_BAL... sentence is sent automatically from LX device on any parameter change.

2.3.13 Set MC/Bal parameters

User device sets MacCready, ballast, bugs, brightness and volume level on LX device.

Request: *$LXDT,GET,MC_BAL,<mc>,<ballast>,<bugs>,<brightness>,<vario_vol>,<sc_vol>*<CRC><CR><LF>*

Parameter	Data type	Description
<mc>	float	MacCready factor
<ballast>	uint16_t	Ballast in kg
<bugs>	uint8_t	Bugs factor in percent
<brightness>	uint8_t	Screen brightness in percent
<vario_vol>	uint8_t	Variometer volume in percent
<sc_vol>	uint8_t	SC volume in percent

Example:

```
TX: $LXDT,SET,MC_BAL,1.1,200,30,55,70,20*42<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>
```

2.3.14 Get radio parameters

User device requests Radio parameters from LX device.

If no radio is connected to LX device or radio is disabled in settings, LX device responds with ANS, ERROR sentence.

Request: *$LXDT,GET,RADIO*03<CR><LF>*

Response: *$LXDT,ANS,RADIO,<active_freq>,<standby_freq>,<volume>,<squelch>,<vox>*<CRC><CR><LF>*

Parameter	Data type	Description
<active_freq>	float	Currently set active frequency
<standby_freq>	float	Currently set standby frequency
<volume>	uint16_t	Volume level
<squelch>	uint16_t	Squelch level
<vox>	uint16_t	VOX level

Example:

```
TX: $LXDT,GET,RADIO*03<CR><LF>
RX: $LXDT,ANS,RADIO,128.800,118.475,10,5,33*1c<CR><LF>
```

NOTE: $LXDT,ANS,RADIO... sentence is sent automatically from LX device on any radio parameter change.

2.3.15 Set radio parameters

User device sends commands to Radio via LX device. Not all parameters can be set on given radio. Table at the bottom shows supported functionalities.

Request: *$LXDT,SET,RADIO,<active_freq>,<standby_freq>,<volume>,<squelch>,<vox>*<CRC><CR><LF>*

Parameter	Data type	Description
<active_freq>	float	Active frequency
<standby_freq>	float	Standby frequency
<volume>	uint16_t	Volume level
<squelch>	uint16_t	Squelch level
<vox>	uint16_t	VOX level

Example:

```
TX: $LXDT,SET,RADIO,118.475,121.500,9,8,7*04<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>
```

2.3.16 Switch radio frequencies

User device switches between active and standby frequencies on the radio via LX device.

Request: *$LXDT,SET,R_SWITCH*59<CR><LF>*

NOTE: Check table at the bottom to see which radios are supported for this functionality.

2.3.17 Set radio dual mode

User device sets radio dual mode via LX device.

Request: *$LXDT,SET,R_DUAL,<enable>*<CRC><CR><LF>*

Parameter	Data type	Description
<enable>	boolean	0 = disable, 1 = enable

Example:

```
TX: $LXDT,SET,R_DUAL,1*4A – enable
RX: $LXDT,ANS,OK*5c<CR><LF>
```

NOTE: Check table at the bottom to see which radios are supported for this functionality.

2.3.18 Set radio frequency spacing

User device sets radio frequency spacing via LX device.

Request: *$LXDT,SET,R_SPACING,<spacing>*<CRC><CR><LF>*

Parameter	Data type	Description
<spacing>	boolean	0 = 25 kHz, 1 = 8.33 kHz

Example:

```
TX: $LXDT,SET,R_SPACING,1*17
RX: $LXDT,ANS,OK*5c<CR><LF>
```

NOTE: Check table at the bottom to see which radios are supported for this functionality.

2.3.19 Get number of flights

User device requests number of flights from LX device's logbook.

Request: *$LXDT,GET,FLIGHTS_NO*47<CR><LF>*

Response: *$LXDT,ANS,FLIGHTS_NO,<no_of_flights>*<CRC><CR><LF>*

Parameter	Data type	Description
<no_of_flights>	uint16_t	Total flights count.

Example:

```
TX: $LXDT,GET,FLIGHTS_NO*47<CR><LF>
RX: $LXDT,ANS,FLIGHTS_NO,9*58<CR><LF>
```

2.3.20 Get flight info

User device requests info for flight with given id from LX device's logbook.

Request: *$LXDT,GET,FLIGHT_INFO,<flight_id>*<CRC><CR><LF>*

Parameter	Data type	Description
<flight_id>	uint16_t	Lowest valid id is 1 and represents the latest flight.

Response: *$LXDT,ANS,FLIGHT_INFO,<flight_id>,<filename>,<date>,<take_off>,<landing>,<pilot_name>, <pilot_surname>,<reg_no>,<comp_id>,<min_gforce>,<max_gforce>,<max_alt>, <max_ias>*<CRC><CR><LF>*

Parameter	Data type	Description
<flight_id>	uint16_t	Lowest id is 1 and represents the latest flight.
<filename>	string	IGC filename.
<date>	string	Flight date. "DD.MM.YYYY" formated string.
<take_off>	string	Take off time. "HH:MM:SS" formated string
<landing>	string	Landing time. "HH:MM:SS" formated string
<pilot_name>	string	Self explanatory.
<pilot_surname>	string	Self explanatory.
<reg_no>	string	Registration number
<comp_id>	string	Competition id.
<min_gforce>	int8_t	Minimum g-force during flight. Value in tens of actual g-force value (15 = 1.5g).
<max_gforce>	int8_t	Minimum g-force during flight. Value in tens of actual g-force value.
<max_alt>	uint16_t	Maximum altitude during flight in meters.
<max_ias>	uint16_t	Maximum indicated airspeed in meters per second.

Example:

```
TX: $LXDT,GET,FLIGHT_INFO,3*04<CR><LF>
RX: $LXDT,ANS,FLIGHT_INFO,1,03JLQYT1,19.03.2020,07:08:24,07:11:27,ACE,
FLYER,D-KLXD,XD,0,10,1260,98*3c<CR><LF>
```

Note: File can be downloaded using LX Binary Protocol (check datagram Get flight block).

2.3.21 AHRS data

LX device sends out AHRS and G-force data. In case AHRS data is invalid, pitch, roll, yaw and slip parameters are blank.

Response: *$LXBC,AHRS,<pitch>,<roll>,<yaw>,<slip>,<gf_x>,<gf_y>,<gf_z>*<CRC><CR><LF>*

Parameter	Data type	Description	Range
<pitch>	float	Aircraft pitch angle in degrees.	-90 -> +90
<roll>	float	Roll angle in degrees.	-180 -> +180
<yaw>	float	Yaw angle in degrees.	0 -> +360
<slip>	float	Slip in degrees.	-90 -> +90
<gf_x>	float	G-force in X axis.	-10.0 -> 10.0
<gf_y>	float	G-force in Y axis.	-10.0 -> 10.0
<gf_z>	float	G-force in Z axis.	-10.0 -> 10.0

Examples:

```
RX: $LXBC,AHRS,15.9,10.0,310.6,9.9,0.8,-0.3,-0.6*36<CR><LF>

RX: $LXBC,AHRS,,,,,0.8,-0.3,-0.6*3e<CR><LF>
```

2.3.22 Error response

After a SET action from user device, LX device can respond with an error.

Response: *$LXDT,ANS,ERROR,<description>*<CRC><CR><LF>*

Parameter	Data type	Description
<description>	string	Error description text

Example:

```
RX: $LXDT,ANS,ERROR,Parameter count mismatch*02<CR><LF>
```

2.4 Radio supported functionalities

Following table shows which parameters can be set via LX device for the given radio.

Radio	Active	Standby	Volume	Squelch	VOX	Switch	Dual	Spacing
KRT2	X	X	X	X	X	X	X	
ATR833	X	X	X	X		X	X	
Becker	X	X				X		
Trig	X	X				X		
ACD	X	X				X		

2.5 Task declaration Example

Following is the communication log of task declaration with Takeoff, Start, one turnpoint, Finish and Landing points.

```
TX: $LXDT,SET,TP,0,5,2774736,913385,CELJE*1F<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>

TX: $LXDT,SET,TP,1,5,2774736,913385,CELJE*1E<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>

TX: $LXDT,SET,TP,2,5,2748616,906762,NOVO MESTO*25<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>

TX: $LXDT,SET,TP,3,5,2774736,913385,CELJE*1C<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>

TX: $LXDT,SET,TP,4,5,2774736,913385,CELJE*1B<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>

TX: $LXDT,SET,ZONE,1,2,1,1,90,0,0,5000,0,244*55<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>

TX: $LXDT,SET,ZONE,2,0,1,1,90,0,0,5000,0,169*58<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>

TX: $LXDT,SET,ZONE,3,3,1,1,90,0,0,5000,0,244*56<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>

TX: $LXDT,SET,TSK_PAR,0,700,02:30*06<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>

TX: $LXDT,SET,GLIDER,D-KLXD,XD,OPEN*01<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>

TX: $LXDT,SET,PILOT,ACE,FLYER*0B<CR><LF>
RX: $LXDT,ANS,OK*5c<CR><LF>
```

LX Binary Protocol

LX NMEA protocol is used for communication between LX navigation devices and third party peripherals (User devices hereafter).

Physical layer can be either LX user serial port or bluetooth interface on devices supporting wireless connectivity.

For NMEA communication check LX NMEA 1.0 Protocol and LX NMEA 2.0 Protocol.

3.1 General

LX User port protocol is a binary communication protocol. Following is a general message structure.

1B	1B	0-N Bytes	1B
STX	CMD	. . . . . Data	CRC

Each message starts with STX (0x02). This is followed by command code (CMD) and data bytes. Data bytes length is variable and determined according to CMD. Minimum data length is 0, maximum is not defined. For data integrity, CRC is added to and end of the message*.

Following is a list of supported command codes.

CMD	Description	Message size
0x16	Syncronization byte (deprecated)	1 B
0xC4	Get logger info	3 B
0xCA	Set task	352 B
0xCB	Get task	3 B
0xD0	Set Class	12 B
0xF0	Get flight info	4 B
0xF1	Get flight block	7 B
0xF2	Get number of flights	3 B
0xF3	Send CMD to Radio	Variable
0xF4	Set Obs Zone	31 B
0xF5	Get Obs Zone	4 B

LX device responds on each request sent from user device by either*:

- ACK byte and requested data (if adequate); or
- NACK byte.

Code	Size	in Hex
ACK	1 B	0x06
NACK	1 B	0x15

* -Radio commands are a special messages with no CRC byte and no response from LX device to user device.

3.2 CRC calculation

For data integrity fault detection an 8-bit CRC is added to an end of the message. CRC is calclulated on all bytes including STX and CMD.

Following is the algorithm for calculating CRC:

```
#define CRCPOLY    0x69

uint8_t m_byCrc = 0xff;
uint8_t m_datagram[1024];

for (uint8_t byte = 0; byte <= byteCount; byte++) {
  int8_t d = m_datagram[byte];
  int8_t tmp = d;

  for (uint8_t bit = 0; ++bit <= 8; d <<= 1) {
    tmp = m_byCrc ^ d;
    m_byCrc <<= 1;
    if (tmp < 0)
      m_byCrc ^= CRCPOLY;
  }
}
```

3.3 Message structure

3.3.1 Get logger info

User devices requests LX device name, serial, version numbers etc.

CMD: 0xC4

Request:

Byte	Name	Size	Value
0	STX	1 B	0x02
1	CMD	1 B	0xC4
2	CRC	1 B	

Response: ASCII string

```
Version LX ERA
```

```
SN34949,HW1.0
ID:0-[0]
Checksum:00
AS:Empty
APT:Empty
```

3.3.2 Set declaration

User device sets declaration on LX device.

CMD: 0xCA

Request:

Byte	Name	Size	Value
0	STX	1 B	0x02
1	CMD	1 B	0xCA
2 -120	Structure sOldFlight	119 B	binary data
121 -350	Structure sTask	230 B	binary data
351	CRC	1 B	

Response: ACK or NACK

Data structures:

```
struct sOldFlight {
    uint8_t flag;           //< Not used
    uint16_t oo_id;         //< Not used
    char pilot[19];         //< "Name Surname"
    char glider[12];        //< Polar name
    char reg_num[8];        //< Acft registration number
    char cmp_num[4];        //< Competition id
    uint8_t byClass;        //< 0=STANDARD, 1=15-METER, 2=OPEN,
                          //< 3=18-METER, 4=WORLD, 5=DOUBLE,
                          //< 6=MOTOR_GL
    char observer[10];      //< Not used
    uint8_t gpsdatum;       //< Not used
    uint8_t fix_accuracy;   //< Not used
    char gps[60];           //< Not used
}__attribute__ ((packed));  //size 119byte

#define TPNUM       12

struct sTask
{
    /* auto defined */
    uint8_t flag;           //< Not used
    int32_t input_time;     //< Not used
    uint8_t di;             //< Not used
    uint8_t mi;             //< Not used
    uint8_t yi;             //< Not used
```

```
        /* user defined */
        uint8_t fd;              //< Not used
        uint8_t fm;              //< Not used
        uint8_t fy;              //< Not used

        int16_t taskid;          //< Not used
        char num_of_tp;          //< Number of TP without Takeoff, Start,
                                //<  Finish and Landing.
        uint8_t prg[TPNUM];      //< 1=Turnpoint (also Start and Finish),
                                //< 2=Landing, 3=Takeoff
        int32_t lon[TPNUM];      //< TP Longitude in degrees multiplied
                                //< by 60000.0f
        int32_t lat[TPNUM];      //< TP Latitude in degrees multiplied
                                //< by 60000.0f
        char name[TPNUM][9];     //< TP Name
    }__attribute__ ((packed));  //size 230byte
```

Example:

```
TX: 02 CA <DATA> <CRC>
RX: 06
```

3.3.3 Get declaration

User device requests declaration from LX device.

CMD: 0xCB

Request:

Byte	Name	Size	Value
0	STX	1 B	0x02
1	CMD	1 B	0xCB
3	CRC	1 B	

Response: ACK + Structure sOldFlight + Structure sTask; or NACK

Example:

```
TX: 02 CB <CRC>
RX: 06 <DATA> <CRC>
```

3.3.4 Set ObsZone

User device sets task zone with given Id on LX device. Lowest valid id is 1 and represents Start TP zone.

CMD: 0xF4

Request:

Byte	Name	Size	Value
0	STX	1 B	0x02
1	CMD	1 B	0xF4
2 -29	Structure sObsZoneData	28 B	binary data
30	CRC	1 B	

Response: ACK or NACK

Data structure:

```
struct sObsZoneData
{
    void Clear();
    uint8_t uiTpNr;          //!< TP number [example: 0=Takeof,
                            //< 1=Start, 2 = TP1, 3=TP2,
                            //< 4=Finish, 5=landing]
    uint8_t uiDirection;     //!< direction [0= Symmetric (default),
                            //< 1=Fixed, 2=Next, 3=Previous, Start]
    uint8_t bAutoNext;       //!< Is this auto next TP or AAT TP
    uint8_t bIsLine;         //!< Is this line flag
    float fA1;               //!< Angle A1 in radians
    float fA2;               //!< Angle A2 in radians
    float fA21;              //!< Angle A21 in radians
    uint32_t uiR1;           //!< Radius R1 in meters
    uint32_t uiR2;           //!< Radius R2 in meters
    float fElevation;        //!< Turnpoint elevation
}__attribute__ ((packed));  //size 28byte
```

Example:

```
TX: 02 F4 02 01 01 00 C2 B8 B2 3E C2 B8 32 3F C2 B8 32 3E E4 0C 00 00 7C
15 00 00 00 00 00 00 8A
RX: 06
```

3.3.5 Get ObsZone

User device requests task zone with given Id on LX device. Lowest valid id is 1 and represents Start TP zone.

CMD: 0xF5

Request:

Byte	Name	Size	Value
0	STX	1 B	0x02
1	CMD	1 B	0xF5
2	Zone id	28 B	integer
3	CRC	1 B	

Response: ACK + Structure sObsZoneData; or NACK

Example:

```
TX: 02 F5 01 69
RX: 06 01 01 01 00 C2 B8 B2 3E C2 B8 32 3F C2 B8 32 3E E4 0C 00 00 7C
15 00 00 00 00 00 00 3D
```

3.3.6 Set Class

User devices sets competition class on LX device.

CMD: 0xD0

Request:

Byte	Name	Size	Value
0	STX	1 B	0x02
1	CMD	1 B	0xD0
2 -10	Class	9 B	ASCII chars
11	CRC	1 B	

Response: ACK or NACK

Example:

```
TX: 02 D0 63 6C 75 62 00 00 00 00 00 8A
RX: 06
```

3.3.7 Get number of flights

User devices requests number of flights from LX device.

CMD: 0xF2

Request:

Byte	Name	Size	Value
0	STX	1 B	0x02
1	CMD	1 B	0xF2
2	CRC	1 B	

Response:

Byte	Name	Size	Value
0	ACK	1 B	0x06
1 -2	Number of flights	2 B	9 => 0x09 0x00 (LSB first)
3	CRC	1 B	

Example:

```
TX: 02 F2 1F
RX: 06 09 00 50
```

3.3.8 Get flight info

User device requests info for flight with given id from LX device. Lowest valid id is 1 and represents the latest flight.

CMD: 0xF0

Request:

Byte	Name	Size	Value
0	STX	1 B	0x02
1	CMD	1 B	0xF0
2 -3	Flight Id	2 B	1 => 0x01 0x00 (LSB first)
4	CRC	1 B	

Response:

Byte	Name	Size	Value
0	ACK	1 B	0x06
1 -93	Structure EosFlightInfo	92 B	binary data
94	CRC	1 B	

Data structure:

```
struct FlightInfo {
    uint16_t    uiFlightID;                 //< Flight id
    char        acIGCFileName[10];          //< IGC file name for
                                            //< file copy
    uint32_t    uiDate;                     //< Date (Julian day)
    uint32_t    uiTakeOff;                  //< Takeoff time (seconds
                                            //< after midnight)
    uint32_t    uiLanding;                  //< Landing time (seconds
                                            //< after midnight)
    char        acName[12];                 //< Pilot name
    char        acSurname[12];              //< Pilot surname
    char        acRegNr[8];                 //< Registration number.
    char        acCompId[8];                //< Competition ID.
    int8_t      iMinGforce;                 //< Minimum G-force (need
                                            //< to be divided by 10)
    int8_t      iMaxGforce;                 //< Maximum G-force (need
                                            //< to be divided by 10)
    uint16_t    uiMaxALT;                   //< Maximum altitude
    uint16_t    uiMaxIAS;                   //< Maximum indicated
                                            //< air speed

    //! Free space for future.
```

```
    uint8_t m_abyFree[16];
};

struct EosFlightInfo {
    FlightInfo  m_fi;       //< basic flight info struct
    uint32_t    m_iSize;    //< flight file size
};
```

Example:

```
TX: 02 F0 01 00 3D
RX: 06 08 00 35 ... AB
```

3.3.9 Get flight block

User devices requests a block of *.igc file from LX device.

CMD: 0xF1

Request:

Byte	Name	Size	Value
0	STX	1 B	0x02
1	CMD	1 B	0xF1
2 -3	Flight Id	2 B	9 => 0x09 0x00 (LSB first)
4 -5	Block Id	2 B	1 => 0x01 0x00 (LSB first)
6	CRC	1 B	

Response:

Byte	Name	Size	Value
0	ACK	1 B	0x06
1 -2	Block size	2 B	120 => 0x78 0x00 (LSB first)
3 -4	Block Id	2 B	1 => 0x01 0x00 (LSB first)
5 -N	Block data	Block size	Binary
N	CRC	1 B	

Example:

```
TX: 02 F1 09 00 01 00 C8
RX: 06 78 00 01 00 47 4C ... 7A
```

3.3.10 Radio commands

User devices sends a command to Radio unit connected to LX device. Bacause Radio is connected to LX device via User port, those functions are available only when user device is connected to LX device via BT interface.

CMD: 0xF3

Request:

Byte	Name	Size	Value
0	STX	1 B	0x02
1	CMD	1 B	0xF3
2 -N	Radio command	Variable	Size is defined within Radio commmand

IMPORTANT: There is no CRC added to the end of message because CRC is already encoded in Radio command.

Supported Radio devices are:

- KRT 2
- ATR 833
- Becker (planned)
- Trig (planned)
- AIR Avionics ACD (planned)

Example:

```
TX: 02 F3 43
```

Contact

Headquarters
LX navigation d.o.o.
Tkalska ulica 10
SI-3000 Celje
Slovenia

VAT ID
Company is registered in Slovenia,
EU under the VAT ID: SI40539601

Webpage
www.lxnavigation.com

Phone
+386 (0)3 490 46 70

Fax
+386 (0)3 490 46 71

Sales
sales@lxnavigation.com

Support
info@lxnavigation.com