LwGPS

Tilen MAJERLE

Jul 03, 2020

CONTENTS

Welcome to the documentation for version latest-develop.

LwGPS is lightweight, platform independent library to parse NMEA statements from GPS receivers. It is highly optimized for embedded systems.

Download library · *Getting started* · Open Github

CHAPTER ONE

FEATURES

- Written in ANSI C99
- Platform independent, easy to use
- Built-in support for 4 GPS statements
  - `GPGGA` or `GNGGA`: GPS fix data
  - `GPGSA` or `GNGSA`: GPS active satellites and dillusion of position
  - `GPGSV` or `GNGSV`: List of satellites in view zone
  - `GPRMC` or `GNRMC`: Recommended minimum specific GPS/Transit data
- Optional `float` or `double` floating point units
- Low-level layer is separated from application layer, thus allows you to add custom communication with GPS device
- Works with operating systems
- Works with different communication interfaces
- User friendly MIT license

CHAPTER

TWO

REQUIREMENTS

- C compiler
- Driver for receiving data from GPS receiver
- Few *kB* of non-volatile memory

CHAPTER

THREE

CONTRIBUTE

Fresh contributions are always welcome. Simple instructions to proceed:

1. Fork Github repository
2. Respect C style & coding rules used by the library
3. Create a pull request to `develop` branch with new features or bug fixes

Alternatively you may:

1. Report a bug
2. Ask for a feature request

CHAPTER FOUR

LICENSE

```
MIT License

Copyright (c) 2020 Tilen MAJERLE

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

CHAPTER

FIVE

TABLE OF CONTENTS

5.1 Getting started

5.1.1 Download library

Library is primarly hosted on Github.

- Download latest release from releases area on Github
- Clone *develop* branch for latest development

Download from releases

All releases are available on Github releases area.

Clone from Github

First-time clone

- Download and install `git` if not already
- Open console and navigate to path in the system to clone repository to. Use command `cd your_path`
- Clone repository with one of available `3` options
  - Run `git clone --recurse-submodules https://github.com/MaJerle/lwgps` command to clone entire repository, including submodules
  - Run `git clone --recurse-submodules --branch develop https://github.com/MaJerle/lwgps` to clone *development* branch, including submodules
  - Run `git clone --recurse-submodules --branch master https://github.com/MaJerle/lwgps` to clone *latest stable* branch, including submodules
- Navigate to `examples` directory and run favourite example

Update cloned to latest version

- Open console and navigate to path in the system where your resources repository is. Use command `cd your_path`
- Run `git pull origin master --recurse-submodules` command to pull latest changes and to fetch latest changes from submodules
- Run `git submodule foreach git pull origin master` to update & merge all submodules

Note: This is preferred option to use when you want to evaluate library and run prepared examples. Repository consists of multiple submodules which can be automatically downloaded when cloning and pulling changes from root repository.

5.1.2 Add library to project

At this point it is assumed that you have successfully download library, either cloned it or from releases page.

- Copy `lwgps` folder to your project
- Add `lwgps/src/include` folder to *include path* of your toolchain
- Add source files from `lwgps/src/` folder to toolchain build
- Build the project

5.1.3 Minimal example code

Run below example to test and verify library

Listing 1: Test verification code

```c
/**
 * This example uses direct processing function
 * to process dummy NMEA data from GPS receiver
 */
#include "lwgps/lwgps.h"
#include <string.h>
#include <stdio.h>

/* GPS handle */
lwgps_t hgps;

/**
 * \brief           Dummy data from GPS receiver
 */
const char
gps_rx_data[] = ""
                "$GPRMC,183729,A,3907.356,N,12102.482,W,000.0,360.0,080301,015.5,E*6F\r\n"
                "$GPRMB,A,,,,,,,,,,,,V*71\r\n"
                "$GPGGA,183730,3907.356,N,12102.482,W,1,05,1.6,646.4,M,-24.1,M,,*75\r\n"
                "$GPGSA,A,3,02,,,07,,09,24,26,,,,,1.6,1.6,1.0*3D\r\n"
                "$GPGSV,2,1,08,02,43,088,38,04,42,145,00,05,11,291,00,07,60,043,35*71\r\n"
```

(continues on next page)

(continued from previous page)

```
                "$GPGSV,2,2,08,08,02,145,00,09,46,303,47,24,16,178,32,26,18,231,43*77\r\n"
                "$PGRME,22.0,M,52.9,M,51.0,M*14\r\n"
                "$GPGLL,3907.360,N,12102.481,W,183730,A*33\r\n"
                "$PGRMZ,2062,f,3*2D\r\n"
                "$PGRMM,WGS84*06\r\n"
                "$GPBOD,,T,,M,,*47\r\n"
                "$GPRTE,1,1,c,0*07\r\n"
                "$GPRMC,183731,A,3907.482,N,12102.436,W,000.0,360.0,080301,015.5,E*67\r\n"
                "$GPRMB,A,,,,,,,,,,,,V*71\r\n";

int
main() {
    /* Init GPS */
    lwgps_init(&hgps);

    /* Process all input data */
    lwgps_process(&hgps, gps_rx_data, strlen(gps_rx_data));

    /* Print messages */
    printf("Valid status: %d\r\n", hgps.is_valid);
    printf("Latitude: %f degrees\r\n", hgps.latitude);
    printf("Longitude: %f degrees\r\n", hgps.longitude);
    printf("Altitude: %f meters\r\n", hgps.altitude);

    return 0;
}
```

5.2 User manual

5.2.1 How it works

GPS NMEA Parser parses raw data formatted as NMEA 0183 statements from GPS receivers. It supports up to 4 different statements:

- GPGGA or GNGGA: GPS fix data
- GPGSA or GNGSA: GPS active satellites and dillusion of position
- GPGSV or GNGSV: List of satellites in view zone
- GPRMC or GNRMC: Recommended minimum specific GPS/Transit data

Tip: By changing different configuration options, it is possible to disable some statements. Check *GPS Configuration* for more information.

Application must assure to properly receive data from GPS receiver. Usually GPS receivers communicate with host embedded system with UART protocol and output directly formatted NMEA 0183 statements.

Note: Application must take care of properly receive data from GPS.

Application must use *lwgps_process()* function for data processing. Function will:

- Detect statement type, such as *GPGGA* or *GPGSV*
- Parse all the terms of specific statement
- Check valid CRC after each statement

Programmer's model is as following:

- Application receives data from GPS receiver
- Application sends data to `lwgps_process()` function
- Application uses processed data to display altitude, latitude, longitude, and other parameters

Check *Examples and demos* for typical example

5.2.2 Float/double precision

With configuration of `GSM_CFG_DOUBLE`, it is possible to enable `double` floating point precision. All floating point variables are then configured in *double precision*.

When configuration is set to `0`, floating point variables are configured in *single precision* format.

Note: Single precision uses less memory in application. As a drawback, application may be a subject of data loss at latter digits.

5.2.3 Thread safety

Library tends to be as simple as possible. No specific features have been implemented for thread safety.

When library is using multi-thread environment and if multi threads tend to access to shared resources, user must resolve it with care, using mutual exclusion.

Tip: When single thread is dedicated for GPS processing, no special mutual exclusion is necessary.

5.2.4 Tests during development

During the development, test check is performed to validate raw NMEA input data vs expected result.

Listing 2: Test code for development

```c
/*
 * This example uses direct processing function,
 * to process dummy NMEA data from GPS receiver
 */
#include <string.h>
#include <stdio.h>
#include "lwgps/lwgps.h"
#include "test_common.h"

/* GPS handle */
lwgps_t hgps;

```

(continues on next page)

(continued from previous page)

```
/**
 * \brief           Dummy data from GPS receiver
 */
const char
gps_rx_data[] = ""
                "$GPRMC,183729,A,3907.356,N,12102.482,W,000.0,360.0,080301,015.5,E*6F\r\n"
                "$GPGGA,183730,3907.356,N,12102.482,W,1,05,1.6,646.4,M,-24.1,M,,*75\r\n"
                "$GPGSA,A,3,02,,,07,,09,24,26,,,,,1.6,1.6,1.0*3D\r\n"
                "$GPGSV,2,1,08,02,43,088,38,04,42,145,00,05,11,291,00,07,60,043,35*71\r\n"
                "$GPGSV,2,2,08,08,02,145,00,09,46,303,47,24,16,178,32,26,18,231,43*77\r\n"
                "";

/**
 * \brief           Run the test of raw input data
 */
void
run_tests() {
    lwgps_init(&hgps);                          /* Init GPS */

    /* Process all input data */
    lwgps_process(&hgps, gps_rx_data, strlen(gps_rx_data));

    /* Run the test */
    RUN_TEST(!INT_IS_EQUAL(hgps.is_valid, 0));
    RUN_TEST(INT_IS_EQUAL(hgps.fix, 1));
    RUN_TEST(INT_IS_EQUAL(hgps.fix_mode, 3));
    RUN_TEST(FLT_IS_EQUAL(hgps.latitude, 39.1226000000));
    RUN_TEST(FLT_IS_EQUAL(hgps.longitude, -121.0413666666));
    RUN_TEST(FLT_IS_EQUAL(hgps.altitude, 646.4000000000));
    RUN_TEST(FLT_IS_EQUAL(hgps.course, 360.0000000000));
    RUN_TEST(INT_IS_EQUAL(hgps.dop_p, 1.6000000000));
    RUN_TEST(INT_IS_EQUAL(hgps.dop_h, 1.6000000000));
    RUN_TEST(INT_IS_EQUAL(hgps.dop_v, 1.0000000000));
    RUN_TEST(FLT_IS_EQUAL(hgps.speed, 0.0000000000));
    RUN_TEST(FLT_IS_EQUAL(hgps.geo_sep, -24.100000000));
    RUN_TEST(FLT_IS_EQUAL(hgps.variation, 15.500000000));
    RUN_TEST(INT_IS_EQUAL(hgps.sats_in_view, 8));

    RUN_TEST(INT_IS_EQUAL(hgps.sats_in_use, 5));
    RUN_TEST(INT_IS_EQUAL(hgps.satellites_ids[0], 2));
    RUN_TEST(INT_IS_EQUAL(hgps.satellites_ids[1], 0));
    RUN_TEST(INT_IS_EQUAL(hgps.satellites_ids[2], 0));
    RUN_TEST(INT_IS_EQUAL(hgps.satellites_ids[3], 7));
    RUN_TEST(INT_IS_EQUAL(hgps.satellites_ids[4], 0));
    RUN_TEST(INT_IS_EQUAL(hgps.satellites_ids[5], 9));
    RUN_TEST(INT_IS_EQUAL(hgps.satellites_ids[6], 24));
    RUN_TEST(INT_IS_EQUAL(hgps.satellites_ids[7], 26));
    RUN_TEST(INT_IS_EQUAL(hgps.satellites_ids[8], 0));
    RUN_TEST(INT_IS_EQUAL(hgps.satellites_ids[9], 0));
    RUN_TEST(INT_IS_EQUAL(hgps.satellites_ids[10], 0));
    RUN_TEST(INT_IS_EQUAL(hgps.satellites_ids[11], 0));

    RUN_TEST(INT_IS_EQUAL(hgps.date, 8));
```

(continues on next page)

(continued from previous page)

```
    RUN_TEST(INT_IS_EQUAL(hgps.month, 3));
    RUN_TEST(INT_IS_EQUAL(hgps.year, 1));
    RUN_TEST(INT_IS_EQUAL(hgps.hours, 18));
    RUN_TEST(INT_IS_EQUAL(hgps.minutes, 37));
    RUN_TEST(INT_IS_EQUAL(hgps.seconds, 30));
}
```

5.3 API reference

List of all the modules:

5.3.1 GPS NMEA Parser

group **LWGPS**

Lightweight GPS NMEA parser.

Defines

lwgps_is_valid(*_gh*)

Check if current GPS data contain valid signal.

Note *LWGPS_CFG_STATEMENT_GPRMC* must be enabled and GPRMC statement must be sent from GPS receiver

Return 1 on success, 0 otherwise

Parameters

- [in] _gh: GPS handle

Typedefs

typedef double **lwgps_float_t**

GPS float definition, can be either float or double

Note Check for *LWGPS_CFG_DOUBLE* configuration

typedef void (***lwgps_process_fn**)(*lwgps_statement_t* res)

Signature for caller-suplied callback function from gps_process.

Parameters

- [in] res: statement type of recently parsed statement

Enums

enum lwgps_statement_t

ENUM of possible GPS statements parsed.

Values:

enumerator STAT_UNKNOWN = 0

Unknown NMEA statement

enumerator STAT_GGA = 1

GPGGA statement

enumerator STAT_GSA = 2

GPGSA statement

enumerator STAT_GSV = 3

GPGSV statement

enumerator STAT_RMC = 4

GPRMC statement

enumerator STAT_UBX = 5

UBX statement (uBlox specific)

enumerator STAT_UBX_TIME = 6

UBX TIME statement (uBlox specific)

enumerator STAT_CHECKSUM_FAIL = UINT8_MAX

Special case, used when checksum fails

enum lwgps_speed_t

List of optional speed transformation from GPS values (in knots)

Values:

enumerator lwgps_speed_kps

Kilometers per second

enumerator lwgps_speed_kph

Kilometers per hour

enumerator lwgps_speed_mps

Meters per second

enumerator lwgps_speed_mpm

Meters per minute

enumerator lwgps_speed_mips

Miles per second

enumerator lwgps_speed_mph

Miles per hour

enumerator lwgps_speed_fps

Foots per second

enumerator lwgps_speed_fpm

Foots per minute

enumerator lwgps_speed_mpk

Minutes per kilometer

enumerator lwgps_speed_spk
Seconds per kilometer

enumerator lwgps_speed_sp100m
Seconds per 100 meters

enumerator lwgps_speed_mipm
Minutes per mile

enumerator lwgps_speed_spm
Seconds per mile

enumerator lwgps_speed_sp100y
Seconds per 100 yards

enumerator lwgps_speed_smph
Sea miles per hour

Functions

uint8_t **lwgps_init**(*lwgps_t *gh*)
Init GPS handle.

Return `1` on success, `0` otherwise

Parameters

- `[in] gh`: GPS handle structure

uint8_t **lwgps_process**(*lwgps_t *gh*, **const** void **data*, size_t *len*, *lwgps_process_fn evt_fn*)
Process NMEA data from GPS receiver.

Return `1` on success, `0` otherwise

Parameters

- `[in] gh`: GPS handle structure
- `[in] data`: Received data
- `[in] len`: Number of bytes to process
- `[in] evt_fn`: Event function to notify application layer

uint8_t **lwgps_distance_bearing**(*lwgps_float_t las*, *lwgps_float_t los*, *lwgps_float_t lae*, *lwgps_float_t loe*, *lwgps_float_t *d*, *lwgps_float_t *b*)
Calculate distance and bearing between `2` latitude and longitude coordinates.

Return `1` on success, `0` otherwise

Parameters

- `[in] las`: Latitude start coordinate, in units of degrees
- `[in] los`: Longitude start coordinate, in units of degrees
- `[in] lae`: Latitude end coordinate, in units of degrees
- `[in] loe`: Longitude end coordinate, in units of degrees
- `[out] d`: Pointer to output distance in units of meters

- `[out] b`: Pointer to output bearing between start and end coordinate in relation to north in units of degrees

lwgps_float_t **`lwgps_to_speed`**(*lwgps_float_t sik*, *lwgps_speed_t ts*)

Convert NMEA GPS speed (in knots = nautical mile per hour) to different speed format.

Return Speed calculated from knots

Parameters

- `[in] sik`: Speed in knots, received from GPS NMEA statement
- `[in] ts`: Target speed to convert to from knots

`struct lwgps_sat_t`

#include <lwgps.h> Satellite descriptor.

Public Members

uint8_t **`num`**

Satellite number

uint8_t **`elevation`**

Elevation value

uint16_t **`azimuth`**

Azimuth in degrees

uint8_t **`snr`**

Signal-to-noise ratio

`struct lwgps_t`

#include <lwgps.h> GPS main structure.

Public Members

lwgps_float_t **`latitude`**

Latitude in units of degrees

lwgps_float_t **`longitude`**

Longitude in units of degrees

lwgps_float_t **`altitude`**

Altitude in units of meters

lwgps_float_t **`geo_sep`**

Geoid separation in units of meters

uint8_t **`sats_in_use`**

Number of satellites in use

uint8_t **`fix`**

Fix status. `0` = invalid, `1` = GPS fix, `2` = DGPS fix, `3` = PPS fix

uint8_t **`hours`**

Hours in UTC

uint8_t **`minutes`**

Minutes in UTC

uint8_t **seconds**
Seconds in UTC

lwgps_float_t **dop_h**
Dolution of precision, horizontal

lwgps_float_t **dop_v**
Dolution of precision, vertical

lwgps_float_t **dop_p**
Dolution of precision, position

uint8_t **fix_mode**
Fix mode. `1` = NO fix, `2` = 2D fix, `3` = 3D fix

uint8_t **satellites_ids**[12]
List of satellite IDs in use. Valid range is `0` to `sats_in_use`

uint8_t **sats_in_view**
Number of satellites in view

lwgps_sat_t **sats_in_view_desc**[12]

uint8_t **is_valid**
GPS valid status

lwgps_float_t **speed**
Ground speed in knots

lwgps_float_t **course**
Ground coarse

lwgps_float_t **variation**
Magnetic variation

uint8_t **date**
Fix date

uint8_t **month**
Fix month

uint8_t **year**
Fix year

lwgps_float_t **utc_tow**
UTC TimeOfWeek, eg 113851.00

uint16_t **utc_wk**
UTC week number, continues beyond 1023

uint8_t **leap_sec**
UTC leap seconds; UTC + leap_sec = TAI

uint32_t **clk_bias**
Receiver clock bias, eg 1930035

lwgps_float_t **clk_drift**
Receiver clock drift, eg -2660.664

uint32_t **tp_gran**
Time pulse granularity, eg 43

5.3.2 GPS Configuration

This is the default configuration of the middleware. When any of the settings shall be modified, it shall be done in library header file, `lwgps/src/include/lwgps/lwgps.h`

group **LWGPS_CONFIG**

Default configuration setup.

Defines

LWGPS_CFG_DOUBLE

Enables `1` or disables `0` `double precision` for floating point values such as latitude, longitude, altitude.

`double` is used as variable type when enabled, `float` when disabled.

LWGPS_CFG_STATUS

Enables `1` or disables `0` status reporting callback by *lwgps_process*.

Note This is an extension, so not enabled by default.

LWGPS_CFG_STATEMENT_GPGGA

Enables `1` or disables `0` `GGA` statement parsing.

Note This statement must be enabled to parse:

- Latitude, Longitude, Altitude
- Number of satellites in use, fix (no fix, GPS, DGPS), UTC time

LWGPS_CFG_STATEMENT_GPGSA

Enables `1` or disables `0` `GSA` statement parsing.

Note This statement must be enabled to parse:

- Position/Vertical/Horizontal dilution of precision
- Fix mode (no fix, 2D, 3D fix)
- IDs of satellites in use

LWGPS_CFG_STATEMENT_GPRMC

Enables `1` or disables `0` `RMC` statement parsing.

Note This statement must be enabled to parse:

- Validity of GPS signal
- Ground speed in knots and coarse in degrees
- Magnetic variation
- UTC date

LWGPS_CFG_STATEMENT_GPGSV

Enables `1` or disables `0` `GSV` statement parsing.

Note This statement must be enabled to parse:

- Number of satellites in view

- Optional details of each satellite in view. See *LWGPS_CFG_STATEMENT_GPGSV_SAT_DET*

LWGPS_CFG_STATEMENT_GPGSV_SAT_DET

Enables `1` or disables `0` detailed parsing of each satellite in view for `GSV` statement.

Note When this feature is disabled, only number of "satellites in view" is parsed

LWGPS_CFG_STATEMENT_PUBX

Enables `1` or disables `0` parsing and generation of PUBX (uBlox) messages.

PUBX are a nonstandard ublox-specific extensions, so disabled by default.

LWGPS_CFG_STATEMENT_PUBX_TIME

Enables `1` or disables `0` parsing and generation of PUBX (uBlox) TIME messages.

This is a nonstandard ublox-specific extension, so disabled by default.

Note TIME messages can be used to obtain:

- UTC time of week
- UTC week number
- Leap seconds (allows conversion to eg. TAI)

This configure option requires LWGPS_CFG_STATEMENT_PUBX

5.4 Examples and demos

There are `2` very basic examples provided with the library.

5.4.1 Parse block of data

In this example, block of data is prepared as big string array and sent to processing function in single shot. Application can then check if GPS signal has been detected as valid and use other data accordingly.

Listing 3: Minimum example code

```c
/**
 * This example uses direct processing function
 * to process dummy NMEA data from GPS receiver
 */
#include "lwgps/lwgps.h"
#include <string.h>
#include <stdio.h>

/* GPS handle */
lwgps_t hgps;

/**
 * \brief           Dummy data from GPS receiver
 */
const char
gps_rx_data[] = ""
                "$GPRMC,183729,A,3907.356,N,12102.482,W,000.0,360.0,080301,015.5,E*6F\r\n"
                "$GPRMB,A,,,,,,,,,,,,V*71\r\n"
```

(continues on next page)

(continued from previous page)

```
                "$GPGGA,183730,3907.356,N,12102.482,W,1,05,1.6,646.4,M,-24.1,M,,*75\r\n"
                "$GPGSA,A,3,02,,,07,,09,24,26,,,,,1.6,1.6,1.0*3D\r\n"
                "$GPGSV,2,1,08,02,43,088,38,04,42,145,00,05,11,291,00,07,60,043,35*71\r\n"
                "$GPGSV,2,2,08,08,02,145,00,09,46,303,47,24,16,178,32,26,18,231,43*77\r\n"
                "$PGRME,22.0,M,52.9,M,51.0,M*14\r\n"
                "$GPGLL,3907.360,N,12102.481,W,183730,A*33\r\n"
                "$PGRMZ,2062,f,3*2D\r\n"
                "$PGRMM,WGS84*06\r\n"
                "$GPBOD,,T,,M,,*47\r\n"
                "$GPRTE,1,1,c,0*07\r\n"
                "$GPRMC,183731,A,3907.482,N,12102.436,W,000.0,360.0,080301,015.5,E*67\r\n"
                "$GPRMB,A,,,,,,,,,,,,V*71\r\n";

int
main() {
    /* Init GPS */
    lwgps_init(&hgps);

    /* Process all input data */
    lwgps_process(&hgps, gps_rx_data, strlen(gps_rx_data));

    /* Print messages */
    printf("Valid status: %d\r\n", hgps.is_valid);
    printf("Latitude: %f degrees\r\n", hgps.latitude);
    printf("Longitude: %f degrees\r\n", hgps.longitude);
    printf("Altitude: %f meters\r\n", hgps.altitude);

    return 0;
}
```

5.4.2 Parse received data from interrupt/DMA

Second example is a typical use case with interrupts on embedded systems. For each received character, application uses `ringbuff` as intermediate buffer. Data are later processed outside interrupt context.

Note: For the sake of this example, application *implements* interrupts as function call in *while loop*.

Listing 4: Example of buffer

```
#include "lwgps/lwgps.h"
#include "lwrb/lwrb.h"
#include <string.h>

/* GPS handle */
lwgps_t hgps;

/* GPS buffer */
lwrb_t hgps_buff;
uint8_t hgps_buff_data[12];
```

(continues on next page)

(continued from previous page)

```c

/**
 * \brief           Dummy data from GPS receiver
 * \note            This data are used to fake UART receive event on microcontroller
 */
const char
gps_rx_data[] = ""
                "$GPRMC,183729,A,3907.356,N,12102.482,W,000.0,360.0,080301,015.5,E*6F\r\n"
                "$GPRMB,A,,,,,,,,,,,,V*71\r\n"
                "$GPGGA,183730,3907.356,N,12102.482,W,1,05,1.6,646.4,M,-24.1,M,,*75\r\n"
                "$GPGSA,A,3,02,,,07,,09,24,26,,,,,1.6,1.6,1.0*3D\r\n"
                "$GPGSV,2,1,08,02,43,088,38,04,42,145,00,05,11,291,00,07,60,043,35*71\r\n"
                "$GPGSV,2,2,08,08,02,145,00,09,46,303,47,24,16,178,32,26,18,231,43*77\r\n"
                "$PGRME,22.0,M,52.9,M,51.0,M*14\r\n"
                "$GPGLL,3907.360,N,12102.481,W,183730,A*33\r\n"
                "$PGRMZ,2062,f,3*2D\r\n"
                "$PGRMM,WGS84*06\r\n"
                "$GPBOD,,T,,M,,*47\r\n"
                "$GPRTE,1,1,c,0*07\r\n"
                "$GPRMC,183731,A,3907.482,N,12102.436,W,000.0,360.0,080301,015.5,E*67\r\n"
                "$GPRMB,A,,,,,,,,,,,,V*71\r\n";
static size_t write_ptr;
static void uart_irqhandler(void);

int
main() {
    uint8_t rx;

    /* Init GPS */
    lwgps_init(&hgps);

    /* Create buffer for received data */
    lwrb_init(&hgps_buff, hgps_buff_data, sizeof(hgps_buff_data));

    while (1) {
        /* Add new character to buffer */
        /* Fake UART interrupt handler on host microcontroller */
        uart_irqhandler();

        /* Process all input data */
        /* Read from buffer byte-by-byte and call processing function */
        if (lwrb_get_full(&hgps_buff)) {        /* Check if anything in buffer now */
            while (lwrb_read(&hgps_buff, &rx, 1) == 1) {
                lwgps_process(&hgps, &rx, 1);   /* Process byte-by-byte */
            }
        } else {
            /* Print all data after successful processing */
            printf("Latitude: %f degrees\r\n", hgps.latitude);
            printf("Longitude: %f degrees\r\n", hgps.longitude);
            printf("Altitude: %f meters\r\n", hgps.altitude);
            break;
        }
```

(continues on next page)

(continued from previous page)

```
    }

    return 0;
}

/**
 * \brief           Interrupt handler routing for UART received character
 * \note            This is not real MCU, it is software method, called from main
 */
static void
uart_irqhandler(void) {
    /* Make interrupt handler as fast as possible */
    /* Only write to received buffer and process later */
    if (write_ptr < strlen(gps_rx_data)) {
        /* Write to buffer only */
        lwrb_write(&hgps_buff, &gps_rx_data[write_ptr], 1);
        ++write_ptr;
    }
}
```

INDEX