LwGPS

Tilen MAJERLE

Mar 23, 2024

CONTENTS

1 Features **3**

2 Requirements **5**

3 Contribute **7**

4 License **9**

5 Table of contents **11**
- 5.1 Getting started . . . 11
- 5.2 User manual . . . 15
- 5.3 API reference . . . 19
- 5.4 Examples and demos . . . 29
- 5.5 Changelog . . . 33
- 5.6 Authors . . . 33

Index **35**

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 Donate

CHAPTER

ONE

FEATURES

- Written in C (C11)
- 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) 2024 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

Getting started may be the most challenging part of every new library. This guide is describing how to start with the library quickly and effectively

5.1.1 Download library

Library is primarly hosted on Github.

You can get it by:

- Downloading latest release from releases area on Github
- Cloning `main` branch for latest stable version
- Cloning `develop` branch for latest development

Download from releases

All releases are available on Github releases area.

Clone from Github

First-time clone

This is used when you do not have yet local copy on your machine.

- Make sure `git` is installed.
- Open console and navigate to path in the system to clone repository to. Use command `cd your_path`
- Clone repository with one of available options below
  - 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 main 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 repository is located. Use command `cd your_path`
- Run `git pull origin main` command to get latest changes on `main` branch
- Run `git pull origin develop` command to get latest changes on `develop` branch
- Run `git submodule update --init --remote` to update submodules to latest version

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. Next step is to add the library to the project, by means of source files to compiler inputs and header files in search path.

CMake is the main supported build system. Package comes with the `CMakeLists.txt` and `library.cmake` files, both located in the `lwgps` directory:

- `CMakeLists.txt`: Is a wrapper and only includes `library.cmake` file. It is used if target application uses `add_subdirectory` and then uses `target_link_libraries` to include the library in the project
- `library.cmake`: It is a fully configured set of variables. User must use `include(path/to/library.cmake)` to include the library and must manually add files/includes to the final target

---

Tip: Open `library.cmake` file and manually analyze all the possible variables you can set for full functionality.

If you do not use the *CMake*, you can do the following:

- Copy `lwgps` folder to your project, it contains library files
- Add `lwgps/src/include` folder to *include path* of your toolchain. This is where *C/C++* compiler can find the files during compilation process. Usually using `-I` flag
- Add source files from `lwgps/src/` folder to toolchain build. These files are built by *C/C++* compiler. CMake configuration comes with the library, allows users to include library in the project as **subdirectory** and **library**.
- Copy `lwgps/src/include/lwgps/lwgps_opts_template.h` to project folder and rename it to `lwgps_opts.h`
- Build the project

5.1.3 Configuration file

Configuration file is used to overwrite default settings defined for the essential use case. Library comes with template config file, which can be modified according to the application needs. and it should be copied (or simply renamed in-place) and named `lwgps_opts.h`

Note: Default configuration template file location: `lwgps/src/include/lwgps/lwgps_opts_template.h`. File must be renamed to `lwgps_opts.h` first and then copied to the project directory where compiler include paths have

access to it by using `#include "lwgps_opts.h"`.

Tip: If you are using *CMake* build system, define the variable `LWGPS_OPTS_FILE` before adding library's directory to the *CMake* project. Variable must contain the path to the user options file. If not provided and to avoid build error, one will be generated in the build directory.

Configuration options list is available available in the *Configuration* section. If any option is about to be modified, it should be done in configuration file

Listing 1: Template configuration file

```
/**
 * \file            lwgps_opts_template.h
 * \brief           LwGPS configuration file
 */

/*
 * Copyright (c) 2024 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.
 *
 * This file is part of LwGPS - Lightweight GPS NMEA parser library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v2.2.0
 */
#ifndef LWGPS_OPTS_HDR_H
#define LWGPS_OPTS_HDR_H

/* Rename this file to "lwgps_opts.h" for your application */

/*
 * Open "include/lwgps/lwgps_opt.h" and
```

(continues on next page)

(continued from previous page)

```
 * copy & replace here settings you want to change values
 */

#endif /* LWGPS_OPTS_HDR_H */
```

Note: If you prefer to avoid using configuration file, application must define a global symbol `LWGPS_IGNORE_USER_OPTS`, visible across entire application. This can be achieved with `-D` compiler option.

5.1.4 Minimal example code

To verify proper library setup, minimal example has been prepared. Run it in your main application file to verify its proper execution

Listing 2: Absolute minimum example

```
/**
 * This example uses direct processing function
 * to process dummy NMEA data from GPS receiver
 */
#include <string.h>
#include <stdio.h>
#include "lwgps/lwgps.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"
                           "$GPGSV,2,1,08,02,43,088,38,04,42,145,00,05,11,291,00,07,60,
↪043,35*71\r\n"
                           "$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";
```

(continues on next page)

(continued from previous page)

```c

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

LwGPS 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 *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 NMEA data refresh

LwGPS is designed to parse standard NMEA output from GPS module.

Tip: You can read more about NMEA 0183 here.

GPS module outputs several NMEA statements periodically, for instance once a second. In rare cases, outputs can be even every *100ms*. The common *problem* we try to solve is what happens if application tries to access GPS parsed data, while library processed only part of new NMEA statement.

Depending on the application requirements, it is necessary to make sure data used by the application are all from the single NMEA output packet, and not split between different ones. Below are 2 examples of several statements GPS module will output every second.

First statement at any given time: `$GPRMC,183729,A,3907.356,N,12102.482,W,000.0,360.0,080301,015.5,E*6F` `$GPGGA,183730,3907.356,N,12102.482,W,1,05,1.6,646.4,M,-24.1,M,,*75` `$GPGSA,A,3,02,,,07,,09,24,26,,,,,1.6,1.6,1.0*3D` `$GPGSV,2,1,08,02,43,088,38,04,42,145,00,05,11,291,00,07,60,043,35*71` `$GPGSV,2,2,08,08,02,145,00,09,46,303,47,24,16,178,32,26,18,231,43*77`

New statement after one second: `$GPRMC,183729,A,3907.356,N,12102.482,W,000.0,360.0,080301,015.5,E*6F` `$GPGGA,183730,3907.356,N,12102.482,W,1,05,1.6,646.4,M,-24.1,M,,*75` `$GPGSA,A,3,02,,,07,,09,24,26,,,,,1.6,1.6,1.0*3D` `$GPGSV,2,1,08,02,43,088,38,04,42,145,00,05,11,291,00,07,60,043,35*71` `$GPGSV,2,2,08,08,02,145,00,09,46,303,47,24,16,178,32,26,18,231,43*77`

If application manages to check GPS parsed data after first packet has been processed and second didn't arrive yet, there is no issue. Application parsed data are all belonging to single packet, at specific time.

But what would happen if application starts using GPS data while GPGGA packet is being received for second time?

- Application has new GPRMC information, from new packet
- Application still keeps GPGGA, GPGSA and GPGSV data from old packets

This could be a major issue for some applications. Time, speed and position do not match anymore.

Common approach

A common approach to this is to have a source of time in the application. A set of timeouts could determine if packet has just started, or has just been completed and is now fully filled with new data.

An algorithm would be, assuming GPS sends packet data every 1 second:

- When character comes, if time of previous character is greater than maximum time between 2 characters (let's say 10ms, even if this is a lot), this is probably start of new packet.
- If new time is >10ms since last received character, it was probably the last character.
- Application can now use new data
- Application goes to *wait new packet mode*
- Go back to step nr.1

5.2.5 Tests during development

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

Listing 3: Test code for development

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

/* GPS handle */
lwgps_t hgps;

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

(continues on next page)

(continued from previous page)

```
↪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, 39.12260000000));
    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, 1.60000000000));
    RUN_TEST(INT_IS_EQUAL(hgps.dop_h, 1.60000000000));
    RUN_TEST(INT_IS_EQUAL(hgps.dop_v, 1.00000000000));
    RUN_TEST(FLT_IS_EQUAL(hgps.speed, 0.00000000000));
    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));
    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 LwGPS

group **LWGPS**

Lightweight GPS NMEA parser.

Defines

lwgps_speed_kps

Backward compatibility.

Deprecated:

Use *lwgps_speed_t* instead

lwgps_speed_kph

Backward compatibility.

Deprecated:

Use *lwgps_speed_t* instead

lwgps_speed_mps

Backward compatibility.

Deprecated:

Use *lwgps_speed_t* instead

lwgps_speed_mpm

Backward compatibility.

Deprecated:

Use *lwgps_speed_t* instead

lwgps_speed_mips

Backward compatibility.

Deprecated:

Use *lwgps_speed_t* instead

lwgps_speed_mph

Backward compatibility.

Deprecated:

Use *lwgps_speed_t* instead

`lwgps_speed_fps`

Backward compatibility.

Deprecated:

Use *lwgps_speed_t* instead

`lwgps_speed_fpm`

Backward compatibility.

Deprecated:

Use *lwgps_speed_t* instead

`lwgps_speed_mpk`

Backward compatibility.

Deprecated:

Use *lwgps_speed_t* instead

`lwgps_speed_spk`

Backward compatibility.

Deprecated:

Use *lwgps_speed_t* instead

`lwgps_speed_sp100m`

Backward compatibility.

Deprecated:

Use *lwgps_speed_t* instead

`lwgps_speed_mipm`

Backward compatibility.

Deprecated:

Use *lwgps_speed_t* instead

`lwgps_speed_spm`

Backward compatibility.

Deprecated:

Use *lwgps_speed_t* instead

`lwgps_speed_sp100y`

Backward compatibility.

Deprecated:

Use *lwgps_speed_t* instead

`lwgps_speed_smph`

Backward compatibility.

Deprecated:

Use *lwgps_speed_t* instead

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

---

Parameters

- **_gh** – **[in]** GPS handle

Returns

`1` on success, `0` otherwise

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.

Param res

[in] 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.

Parameters
ghandle – **[in]** GPS handle structure

Returns
1 on success, `0` otherwise

uint8_t **lwgps_process**(*lwgps_t* *gh, const void *data, size_t len, *lwgps_process_fn* evt_fn)

Process NMEA data from GPS receiver.

Parameters

- **ghandle** – **[in]** GPS handle structure
- **data** – **[in]** Received data
- **len** – **[in]** Number of bytes to process
- **evt_fn** – **[in]** Event function to notify application layer. This parameter is available only if *LWGPS_CFG_STATUS* is enabled

Returns
1 on success, `0` otherwise

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.

Parameters

- **las** – **[in]** Latitude start coordinate, in units of degrees
- **los** – **[in]** Longitude start coordinate, in units of degrees
- **lae** – **[in]** Latitude end coordinate, in units of degrees
- **loe** – **[in]** Longitude end coordinate, in units of degrees

- **d** – **[out]** Pointer to output distance in units of meters
- **b** – **[out]** Pointer to output bearing between start and end coordinate in relation to north in units of degrees

Returns
1 on success, 0 otherwise

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.

Parameters

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

Returns
Speed calculated 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 **dgps_age**

Age of DGPS correction data (in seconds)

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 Configuration

This is the default configuration of the middleware. When any of the settings shall be modified, it shall be done in dedicated application config `lwgps_opts.h` file.

Note: Check *Getting started* to create configuration file.

group **LWGPS_OPT**
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.

This configure option requires LWGPS_CFG_STATEMENT_PUBX

Note: TIME messages can be used to obtain:

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

LWGPS_CFG_CRC

Enables `1` or disables `0` CRC calculation and check.

Note: When not enabled, CRC check is ignored

LWESP_CFG_DISTANCE_BEARING

Enables `1` or disables `0` distance and bearing calculation.

Note: When not enabled, corresponding function is disabled

LWGPS_MEMSET(dst, val, len)

Memory set function.

Note: Function footprint is the same as memset

LWGPS_MEMCPY(dst, src, len)

Memory copy function.

Note: Function footprint is the same as memcpy

5.4 Examples and demos

There are several 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 4: Minimum example code

```
/**
 * This example uses direct processing function
 * to process dummy NMEA data from GPS receiver
 */
#include <string.h>
#include <stdio.h>
#include "lwgps/lwgps.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"
                           "$GPGSV,2,1,08,02,43,088,38,04,42,145,00,05,11,291,00,07,60,
↪043,35*71\r\n"
                           "$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"
```

(continues on next page)

(continued from previous page)

```
                            "$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 5: 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];

/**
 * \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"
```

(continues on next page)

(continued from previous page)

```
                "$GPGSA,A,3,02,,,07,,09,24,26,,,,,1.6,1.6,1.0*3D\r\n"
                "$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;
        }
    }

    return 0;
}

/**
 * \brief           Interrupt handler routing for UART received character
```

(continues on next page)

(continued from previous page)

```
 * \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;
    }
}
```

5.4.3 Distance and bearing

Library provides calculation of distance and bearing between 2 coordinates on earth. This is useful if used with autonomnous devices to understand in which direction device has to move to reach end point while knowing start coordinate.

Listing 6: Distance and bearing calculation

```
#include "lwgps/lwgps.h"

/* Distance and bearing results */
lwgps_float_t dist, bear;

/* New York coordinates */
lwgps_float_t lat1 = 40.685721;
lwgps_float_t lon1 = -73.820465;

/* Munich coordinates */
lwgps_float_t lat2 = 48.150906;
lwgps_float_t lon2 = 11.554176;

/* Go from New York to Munich */
/* Calculate distance and bearing related to north */
lwgps_distance_bearing(lat1, lon1, lat2, lon2, &dist, &bear);
printf("Distance: %f meters\r\n", (float)dist);
printf("Initial bearing: %f degrees\r\n", (float)bear);

/* Go from Munich to New York */
/* Calculate distance and bearing related to north */
lwgps_distance_bearing(lat2, lon2, lat1, lon1, &dist, &bear);
printf("Distance: %f meters\r\n", (float)dist);
printf("Initial bearing: %f degrees\r\n", (float)bear);
```

5.5 Changelog

```
# Changelog

## Develop

- Add support for differential GPS last time

## v2.2.0

- Split `CMakeLists.txt` files between library and executable
- Change license year to `2023`
- Add `.clang-format` draft
- Deprecate lowercase `lwgps_speed_xxx` enumeration. Temporary implement macro to keep␣
↪backward compatibility. Will be removed in next major release
- Improve `C++` port

## v2.1.0

- Add configuration settings to be consistend with other LwXX libraries
- Apply code style settings with Artistic style options

## v2.0.0

- Break compatibility with v1.x
- Function prefix set to `lwgps_`
- Macros prefix set to `LWGPS_`
- Added support for PUBX Ublox statement

## v1.1.0

- Use pre-increment instead of post-increment
- Remove buffer library and propose ringbuff instead
- Other code style enhancements

## v1.0.0

- Initial release
```

5.6 Authors

List of authors and contributors to the library

```
Tilen Majerle <tilen.majerle@gmail.com>
Tilen Majerle <tilen@majerle.eu>
Jesse Hoogervorst <hoogervorstjesse@gmail.com>
Gonçalo Salazar <glbsalazar@gmail.com>
Brian <bayuan@purdue.edu>
Sirio Balmelli <sirio@b-ad.ch>
```

(continues on next page)

(continued from previous page)

```
Sebastian Krebs <Sebastian.Krebs@yardeye.com>
Robin Mueller <robin.mueller.m@gmail.com>
```

INDEX

L

LWESP_CFG_DISTANCE_BEARING (*C macro*), 28
LWGPS_CFG_CRC (*C macro*), 28
LWGPS_CFG_DOUBLE (*C macro*), 27
LWGPS_CFG_STATEMENT_GPGGA (*C macro*), 27
LWGPS_CFG_STATEMENT_GPGSA (*C macro*), 27
LWGPS_CFG_STATEMENT_GPGSV (*C macro*), 27
LWGPS_CFG_STATEMENT_GPGSV_SAT_DET (*C macro*), 28
LWGPS_CFG_STATEMENT_GPRMC (*C macro*), 27
LWGPS_CFG_STATEMENT_PUBX (*C macro*), 28
LWGPS_CFG_STATEMENT_PUBX_TIME (*C macro*), 28
LWGPS_CFG_STATUS (*C macro*), 27
lwgps_distance_bearing (*C++ function*), 23
lwgps_float_t (*C++ type*), 21
lwgps_init (*C++ function*), 23
lwgps_is_valid (*C macro*), 21
LWGPS_MEMCPY (*C macro*), 28
LWGPS_MEMSET (*C macro*), 28
lwgps_process (*C++ function*), 23
lwgps_process_fn (*C++ type*), 21
lwgps_sat_t (*C++ struct*), 24
lwgps_sat_t::azimuth (*C++ member*), 24
lwgps_sat_t::elevation (*C++ member*), 24
lwgps_sat_t::num (*C++ member*), 24
lwgps_sat_t::snr (*C++ member*), 24
lwgps_speed_fpm (*C macro*), 20
lwgps_speed_fps (*C macro*), 19
lwgps_speed_kph (*C macro*), 19
lwgps_speed_kps (*C macro*), 19
lwgps_speed_mipm (*C macro*), 20
lwgps_speed_mips (*C macro*), 19
lwgps_speed_mph (*C macro*), 19
lwgps_speed_mpk (*C macro*), 20
lwgps_speed_mpm (*C macro*), 19
lwgps_speed_mps (*C macro*), 19
lwgps_speed_smph (*C macro*), 20
lwgps_speed_sp100m (*C macro*), 20
lwgps_speed_sp100y (*C macro*), 20
lwgps_speed_spk (*C macro*), 20
lwgps_speed_spm (*C macro*), 20
lwgps_speed_t (*C++ enum*), 22
lwgps_speed_t::LWGPS_SPEED_FPM (*C++ enumerator*), 22
lwgps_speed_t::LWGPS_SPEED_FPS (*C++ enumerator*), 22
lwgps_speed_t::LWGPS_SPEED_KPH (*C++ enumerator*), 22
lwgps_speed_t::LWGPS_SPEED_KPS (*C++ enumerator*), 22
lwgps_speed_t::LWGPS_SPEED_MIPM (*C++ enumerator*), 23
lwgps_speed_t::LWGPS_SPEED_MIPS (*C++ enumerator*), 22
lwgps_speed_t::LWGPS_SPEED_MPH (*C++ enumerator*), 22
lwgps_speed_t::LWGPS_SPEED_MPK (*C++ enumerator*), 22
lwgps_speed_t::LWGPS_SPEED_MPM (*C++ enumerator*), 22
lwgps_speed_t::LWGPS_SPEED_MPS (*C++ enumerator*), 22
lwgps_speed_t::LWGPS_SPEED_SMPH (*C++ enumerator*), 23
lwgps_speed_t::LWGPS_SPEED_SP100M (*C++ enumerator*), 22
lwgps_speed_t::LWGPS_SPEED_SP100Y (*C++ enumerator*), 23
lwgps_speed_t::LWGPS_SPEED_SPK (*C++ enumerator*), 22
lwgps_speed_t::LWGPS_SPEED_SPM (*C++ enumerator*), 23
lwgps_statement_t (*C++ enum*), 21
lwgps_statement_t::STAT_CHECKSUM_FAIL (*C++ enumerator*), 22
lwgps_statement_t::STAT_GGA (*C++ enumerator*), 21
lwgps_statement_t::STAT_GSA (*C++ enumerator*), 21
lwgps_statement_t::STAT_GSV (*C++ enumerator*), 21
lwgps_statement_t::STAT_RMC (*C++ enumerator*), 21
lwgps_statement_t::STAT_UBX (*C++ enumerator*),

22
lwgps_statement_t::STAT_UBX_TIME (*C++ enumerator*), 22
lwgps_statement_t::STAT_UNKNOWN (*C++ enumerator*), 21
lwgps_t (*C++ struct*), 24
lwgps_t::altitude (*C++ member*), 24
lwgps_t::clk_bias (*C++ member*), 26
lwgps_t::clk_drift (*C++ member*), 26
lwgps_t::course (*C++ member*), 26
lwgps_t::date (*C++ member*), 26
lwgps_t::dgps_age (*C++ member*), 25
lwgps_t::dop_h (*C++ member*), 25
lwgps_t::dop_p (*C++ member*), 25
lwgps_t::dop_v (*C++ member*), 25
lwgps_t::fix (*C++ member*), 25
lwgps_t::fix_mode (*C++ member*), 25
lwgps_t::geo_sep (*C++ member*), 24
lwgps_t::hours (*C++ member*), 25
lwgps_t::is_valid (*C++ member*), 25
lwgps_t::latitude (*C++ member*), 24
lwgps_t::leap_sec (*C++ member*), 26
lwgps_t::longitude (*C++ member*), 24
lwgps_t::minutes (*C++ member*), 25
lwgps_t::month (*C++ member*), 26
lwgps_t::satellites_ids (*C++ member*), 25
lwgps_t::sats_in_use (*C++ member*), 25
lwgps_t::sats_in_view (*C++ member*), 25
lwgps_t::sats_in_view_desc (*C++ member*), 25
lwgps_t::seconds (*C++ member*), 25
lwgps_t::speed (*C++ member*), 25
lwgps_t::tp_gran (*C++ member*), 26
lwgps_t::utc_tow (*C++ member*), 26
lwgps_t::utc_wk (*C++ member*), 26
lwgps_t::variation (*C++ member*), 26
lwgps_t::year (*C++ member*), 26
lwgps_to_speed (*C++ function*), 24