LwPKT

Tilen MAJERLE

Apr 13, 2024

CONTENTS

1 Features **3**

2 Applications **5**

3 Requirements **7**

4 Contribute **9**

5 License **11**

6 Table of contents **13**

6.1 Getting started 13
6.2 User manual 18
6.3 API reference 23
6.4 Changelog 34
6.5 Authors 35

Index **37**

Welcome to the documentation for version latest-develop.

LwPKT is a generic packet protocol library optimized for embedded systems.

Download library Getting started Open Github Donate

CHAPTER

ONE

FEATURES

- Written in C (C11), compatible with `size_t` for size data types
- Platform independent, no architecture specific code
- Uses LwRB library for data read/write operations
- Support for events on packet ready, read or write operation
- Optimized for embedded systems, allows high optimization for data transfer
- Configurable settings for packet structure and variable data length
- Allows multiple notes in network with *from* and *to* addresses
- Separate optional field for *command* data type
- Variable data length to support theoretically unlimited packet length
- CRC check to handle data transmission errors
- User friendly MIT license

CHAPTER

TWO

APPLICATIONS

To name a few:

- Communication in RS-485 network between various devices
- Low-level point to point packet communication (UART, USB, ethernet, . . .)

CHAPTER

THREE

REQUIREMENTS

- C compiler
- Few kB of non-volatile memory

CHAPTER

FOUR

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

FIVE

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

SIX

TABLE OF CONTENTS

6.1 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

6.1.1 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/lwpkt` command to clone entire repository, including submodules
  - Run `git clone --recurse-submodules --branch develop https://github.com/MaJerle/lwpkt` to clone *development* branch, including submodules
  - Run `git clone --recurse-submodules --branch main https://github.com/MaJerle/lwpkt` 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.

6.1.2 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 `lwpkt` 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 `lwpkt` folder to your project, it contains library files
- Add `lwpkt/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 `lwpkt/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 `lwpkt/src/include/lwpkt/lwpkt_opts_template.h` to project folder and rename it to `lwpkt_opts.h`
- Build the project

6.1.3 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 `lwpkt_opts.h`

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

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

Tip: If you are using *CMake* build system, define the variable `LWPKT_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            lwpkt_opts_template.h
 * \brief           LwPKT 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 LwPKT - Lightweight packet protocol library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v1.3.0
 */
#ifndef LWPKT_OPTS_HDR_H
#define LWPKT_OPTS_HDR_H

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

/*
 * Open "include/lwpkt/lwpkt_opt.h" and
```

(continues on next page)

(continued from previous page)

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

#endif /* LWPKT_OPTS_HDR_H */
```

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

6.1.4 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

```
#include <stdio.h>
#include "lwpkt/lwpkt.h"

/* LwPKT data */
static lwpkt_t pkt;
static lwrb_t pkt_tx_rb, pkt_rx_rb;
static uint8_t pkt_tx_rb_data[64], pkt_rx_rb_data[64];

/* Data to read and write */
static const char* data = "Hello World\r\n";

/**
 * \brief           LwPKT example code
 */
void
example_lwpkt(void) {
    lwpktr_t res;
    uint8_t b;

    printf("---\r\nLwPKT default example..\r\n\r\n");

    /*
     * Initialize both ring buffers, for TX and RX operations
     *
     * Initialize LwPKT and link buffers together
     */
    lwrb_init(&pkt_tx_rb, pkt_tx_rb_data, sizeof(pkt_tx_rb_data));
    lwrb_init(&pkt_rx_rb, pkt_rx_rb_data, sizeof(pkt_rx_rb_data));
    lwpkt_init(&pkt, &pkt_tx_rb, &pkt_rx_rb);

#if LWPKT_CFG_USE_ADDR
    /* Set device address (if feature enabled) */
    lwpkt_set_addr(&pkt, 0x12);
#endif /* LWPKT_CFG_USE_ADDR */
```

(continues on next page)

(continued from previous page)

```

    /*
     * Write packet to the TX ringbuffer,
     * act as device wants to send some data
     */
    res = lwpkt_write(&pkt,
#if LWPKT_CFG_USE_ADDR
                      0x11, /* End address to whom to send */
#endif                      /* LWPKT_CFG_USE_ADDR */
#if LWPKT_CFG_USE_FLAGS
                      0x12345678,
#endif /* LWPKT_CFG_USE_FLAGS */
#if LWPKT_CFG_USE_CMD
                      0x85,                 /* Command type */
#endif                                      /* LWPKT_CFG_USE_CMD */
                      data, strlen(data)); /* Length of data and actual data */

    /*
     * LwPKT wrote data to pkt_tx_rb ringbuffer
     * Now actually transmit data over your interface
     * (USART for example, ...)
     */

    /*
     * For the purpose of this example, application will
     * fake data transmission by doing reading from TX buffer
     * and writing it to RX buffer
     */
    printf("Tx RB content len: %u, content: ", (unsigned)lwrb_get_full(&pkt_tx_rb));
    while (lwrb_read(&pkt_tx_rb, &b, 1) == 1) {
        printf("0x%02X, ", (unsigned)b);
        lwrb_write(&pkt_rx_rb, &b, 1);
    }
    printf("\r\n");

    /*
     * Here we have our data in RX buffer
     * means we received data over network interface
     */

    /* Now read and process packet */
    res = lwpkt_read(&pkt);

    if (res == lwpktVALID) {
        size_t len;

        /* Packet is valid */
        printf("Packet is valid!\r\n");

        /* Print debug messages for packet */
#if LWPKT_CFG_USE_ADDR
        printf("Packet from: 0x%08X\r\n", (unsigned)lwpkt_get_from_addr(&pkt));
```

(continues on next page)

(continued from previous page)

```
        printf("Packet to: 0x%08X\r\n", (unsigned)lwpkt_get_to_addr(&pkt));
#endif /* LWPKT_CFG_USE_ADDR */
#if LWPKT_CFG_USE_FLAGS
        printf("Packet flags: 0x%08X\r\n", (unsigned)lwpkt_get_flags(&pkt));
#endif /* LWPKT_CFG_USE_FLAGS */
#if LWPKT_CFG_USE_CMD
        printf("Packet cmd: 0x%02X\r\n", (unsigned)lwpkt_get_cmd(&pkt));
#endif /* LWPKT_CFG_USE_CMD */
        printf("Packet data length: 0x%08X\r\n", (unsigned)lwpkt_get_data_len(&pkt));
        if ((len = lwpkt_get_data_len(&pkt)) > 0) {
            uint8_t* d = lwpkt_get_data(&pkt);
            printf("Packet data: ");
            for (size_t i = 0; i < len; ++i) {
                printf("0x%02X ", (unsigned)d[i]);
            }
            printf("\r\n");
        }

        /* Check who should be dedicated receiver */
#if LWPKT_CFG_USE_ADDR
        if (lwpkt_is_for_me(&pkt)) {
            printf("Packet is for me\r\n");
        } else if (lwpkt_is_broadcast(&pkt)) {
            printf("Packet is broadcast to all devices\r\n");
        } else {
            printf("Packet is for device ID: 0x%08X\r\n", (unsigned)lwpkt_get_to_addr(&
↪pkt));
        }
#endif /* LWPKT_CFG_USE_ADDR */
    } else if (res == lwpktINPROG) {
        printf("Packet is still in progress, did not receive yet all bytes..\r\n");
    } else {
        printf("Packet is not valid!\r\n");
    }
}
```

6.2 User manual

LwPKT protocol library is a simple state-machine parser and raw data generator to allow *2* or more devices in a network to communicate in a structure way.

It is perfectly suitable for communication in embedded systems, suchs as *RS-485*, where multiple devices could be easily connected to one big network.

LwPKT library uses well known and easy implementation of LwRB library for data read and data write. It expects *2* different buffer instances.

Parser is simple state machine that reads and processes every received character from read buffer. When application wants to transmit data, LwPKT library generates raw data and writes them to TX buffer.

Combination of both gives embedded applications freedom to implement communication protocols for TX and RX.

6.2.1 Packet structure

Packet structure consists of several fields, where some are optional and some are mandatory.

Fig. 1: Full features structure format

- `START`: Byte with fixed value to represent start of packet
- `FROM`: Byte(s) from where this packet is coming from. Optional field, can be disabled with *LWPKT_CFG_USE_ADDR*
- `TO`: Byte(s) to where this packet is targeting. Optional field, can be disabled with *LWPKT_CFG_USE_ADDR*
- `FLAGS`: Variable length (unsigned 32-bit max) field for optional user flags. Optional field, can be disabled with *LWPKT_CFG_USE_FLAGS*
- `CMD`: Byte with optional command field to better align with multiple packets. Optional field, can be disabled with *LWPKT_CFG_USE_CMD*
- `LEN`: Length of *data* part field. This is variable multi-byte length to support data length >= `256` bytes. Always present
- `DATA`: Optional data field. Number of bytes is as in `LEN` field
- `CRC`: 8-bit CRC of all enabled fields except *START* and *STOP* bytes. Optional field, can be disabled with *LWPKT_CFG_USE_CRC*
- `STOP`: Byte with fixed value to represent stop of packet

Tip: If only 2 devices are communicating and are in the network, considering disabling *LWPKT_CFG_USE_ADDR* to improve data bandwidth and remove unnecessary packet overhead

6.2.2 Data input output

LwPKT library only reads and writes to `2` ringbuffers used for read and write operations. It is up to application to implement how buffers are actually later written for read operation and sent out on the network for write operation.

Warning: LwPKT is platform independant and requires final application to actually take care of data being read/written from/to ringbuffers and transferred further over the network

6.2.3 Variable data length

Some fields implement variable data length feature, to optimize data transfer length. Currently supported fields are:

- `LEN` field is always enabled
- `FROM` and `TO` fields when *LWPKT_CFG_ADDR_EXTENDED* feature is enabled
- `FLAGS` field when *LWPKT_CFG_USE_FLAGS* feature is enabled

Variable data length is a feature that uses minimum number of bytes to transfer data. It uses `7 LSB bits` per byte for actual data, and `MSB` bit to indicate if there are more bytes coming after. For example, values between `0x00 - 0x7F` are codified within single byte, while values between `0x80 - 0x3F` require `2` bytes for transfer. To transfer `32-bit` variable, minimum `1-byte` and maximum `5-bytes` are used.

Tip: Data codification is always LSB Byte first.

6.2.4 Static & dynamic feature

LwPKT supports multiple instance in the same build, but there might be cases where each instance needs different protocol configuration, such as enabled/disabled **from/to** fields or enabled/disabled **command** feature.

Some configuration features (See configuration chapter for full list of options) support static or dynamic configuration:

- **static** configuration is one configuration for all instances. Globally enabled or disabled feature
- **dynamic** configuration allows that each instance keeps its own protocol configuration.

6.2.5 Event management

LwPKT may operate in event mode, meaning that application receives notifications on different events:

- New packet has been received
- Timeout during packet receive

Timeout function is used when network doesn't transmit all bytes or if data got lost in the middle of transmission. This is to make sure that packet protocol library easily recovers to be able to receive more packets in the future

Warning: To use this feature, application must provide accurate timing in units of milliseconds to be able to properly handle timeout function.

Listing 3: LwPKT example with events

```
#include <stdio.h>
#include "lwpkt/lwpkt.h"

/* LwPKT data */
static lwpkt_t pkt;
static lwrb_t pkt_tx_rb, pkt_rx_rb;
static uint8_t pkt_tx_rb_data[64], pkt_rx_rb_data[64];

/* Data to read and write */
static const char* data = "Hello World\r\n";

/**
 * \brief           LwPKT application callback
 */
static void
my_lwpkt_evt_fn(lwpkt_t* pkt, lwpkt_evt_type_t type) {
    switch (type) {
        case LWPKT_EVT_PKT: {
            printf("Valid packet received..\r\n");

            /* Packet is valid */
            printf("Packet is valid!\r\n");
```

(continues on next page)

(continued from previous page)

```
            /* Print debug messages for packet */
#if LWPKT_CFG_USE_ADDR
            printf("Packet from: 0x%08X\r\n", (unsigned)lwpkt_get_from_addr(pkt));
            printf("Packet to: 0x%08X\r\n", (unsigned)lwpkt_get_to_addr(pkt));
#endif /* LWPKT_CFG_USE_ADDR */
#if LWPKT_CFG_USE_CMD
            printf("Packet cmd: 0x%08X\r\n", (unsigned)lwpkt_get_cmd(pkt));
#endif /* LWPKT_CFG_USE_CMD */
            printf("Packet data length: 0x%08X\r\n", (unsigned)lwpkt_get_data_len(pkt));

            /* Do other thins... */
            break;
        }
        case LWPKT_EVT_TIMEOUT: {
            printf("Timeout detected during read operation..\r\n");
            break;
        }
        default: {
            break;
        }
    }
}

/**
 * \brief           LwPKT example code with event feature
 */
void
example_lwpkt_evt(void) {
    lwpktr_t res;
    uint32_t time;
    uint8_t b;

    printf("---\r\nLwPKT event type..\r\n\r\n");

    /*
     * Initialize both ring buffers, for TX and RX operations
     *
     * Initialize LwPKT and link buffers together
     */
    lwrb_init(&pkt_tx_rb, pkt_tx_rb_data, sizeof(pkt_tx_rb_data));
    lwrb_init(&pkt_rx_rb, pkt_rx_rb_data, sizeof(pkt_rx_rb_data));
    lwpkt_init(&pkt, &pkt_tx_rb, &pkt_rx_rb);

#if LWPKT_CFG_USE_ADDR
    /* Set device address (if feature enabled) */
    lwpkt_set_addr(&pkt, 0x12);
#endif /* LWPKT_CFG_USE_ADDR */

    /*
     * Write packet to the TX ringbuffer,
     * act as device wants to send some data
```

(continues on next page)

(continued from previous page)

```
     */
    res = lwpkt_write(&pkt,
#if LWPKT_CFG_USE_ADDR
                      0x11, /* End address to whom to send */
#endif                      /* LWPKT_CFG_USE_ADDR */
#if LWPKT_CFG_USE_FLAGS
                      0x12345678, /* Custom flags added to the packet */
#endif                            /* LWPKT_CFG_USE_FLAGS */
#if LWPKT_CFG_USE_CMD
                      0x85,               /* Command type */
#endif                                    /* LWPKT_CFG_USE_CMD */
                      data, strlen(data)); /* Length of data and actual data */

    /*
     * LwPKT wrote data to pkt_tx_rb ringbuffer
     * Now actually transmit data over your interface
     * (USART for example, ...)
     */

    /*
     * For the purpose of this example, application will
     * fake data transmission by doing reading from TX buffer
     * and writing it to RX buffer
     */
    while (lwrb_read(&pkt_tx_rb, &b, 1) == 1) {
        lwrb_write(&pkt_rx_rb, &b, 1);
    }

    /*
     * Here we have our data in RX buffer
     * means we received data over network interface
     */

    /* Set callback function */
    lwpkt_set_evt_fn(&pkt, my_lwpkt_evt_fn);

    /* Now call process function instead */
    time = 100; /* Get_current_time_in_milliseconds */
    lwpkt_process(&pkt, time);

    (void)res;
}
```

6.3 API reference

List of all the modules:

6.3.1 LwPKT

group **LWPKT**

Lightweight packet protocol.

Defines

lwpkt_get_from_addr(pkt)

Get address from where packet was sent.

Parameters

- **pkt** – **[in]** LwPKT instance

Returns
Address

lwpkt_get_to_addr(pkt)

Get address to where packet was sent.

Parameters

- **pkt** – **[in]** LwPKT instance

Returns
Address

lwpkt_get_data_len(pkt)

Get length of packet.

Parameters

- **pkt** – **[in]** LwPKT instance

Returns
Number of data bytes in packet

lwpkt_get_data(pkt)

Get pointer to packet data.

Parameters

- **pkt** – **[in]** LwPKT instance

Returns
Pointer to data

lwpkt_get_cmd(pkt)

Get packet command data field.

Parameters

- **pkt** – **[in]** LwPKT instance

Returns
Command data field

`lwpkt_get_flags`(pkt)

Get packet flags.

Parameters

- **pkt** – **[in]** LwPKT instance

Returns
Last received packet flags

`lwpkt_is_for_me`(pkt)

Check if packet `to` field address matches device address.

Parameters

- **pkt** – **[in]** LwPKT instance

Returns
`1` on success, `0` otherwise

`lwpkt_is_broadcast`(pkt)

Check if packet was sent to all devices on network.

Parameters

- **pkt** – **[in]** LwPKT instance

Returns
`1` if broadcast, `0` otherwise

Typedefs

typedef void (*`lwpkt_evt_fn`)(struct lwpkt *pkt, *lwpkt_evt_type_t* evt_type)

Event function prototype.

Param pkt
[in] Packet structure

Param evt_type
[in] Event type

typedef uint32_t `lwpkt_addr_t`

Device address data type.

Enums

enum `lwpkt_state_t`

Packet state enumeration.

Values:

enumerator `LWPKT_STATE_START` = 0x00

Packet waits for start byte

enumerator **LWPKT_STATE_FROM**

Packet waits for “packet from” byte

enumerator **LWPKT_STATE_TO**

Packet waits for “packet to” byte

enumerator **LWPKT_STATE_CMD**

Packet waits for “packet cmd” byte

enumerator **LWPKT_STATE_FLAGS**

Packet waits for “packet flags” byte (custom user flags)

enumerator **LWPKT_STATE_LEN**

Packet waits for (multiple) data length bytes

enumerator **LWPKT_STATE_DATA**

Packet waits for actual data bytes

enumerator **LWPKT_STATE_CRC**

Packet waits for CRC data

enumerator **LWPKT_STATE_STOP**

Packet waits for stop byte

enumerator **LWPKT_STATE_END**

Last entry

enum **lwpktr_t**

Packet result enumeration.

Values:

enumerator **lwpktOK** = 0x00

Function returns successfully

enumerator **lwpktERR**

General error for function status

enumerator **lwpktINPROG**

Receive is in progress

enumerator **lwpktVALID**

packet valid and ready to be read as CRC is valid and STOP received

enumerator **lwpktERRCRC**

CRC integrity error for the packet. Will not wait STOP byte if received

enumerator **lwpktERRSTOP**

Packet error with STOP byte, wrong character received for STOP

enumerator **lwpktWAITDATA**

Packet state is in start mode, waiting start byte to start receiving

enumerator **lwpktERRMEM**

No enough memory available for write

enum **lwpkt_evt_type_t**

List of event types.

Values:

enumerator **LWPKT_EVT_PKT**

Valid packet ready to read

enumerator **LWPKT_EVT_TIMEOUT**

Timeout on packat, reset event

enumerator **LWPKT_EVT_READ**

Packet read operation. Called when read operation happens from RX buffer

enumerator **LWPKT_EVT_WRITE**

Packet write operation. Called when write operation happens to TX buffer

enumerator **LWPKT_EVT_PRE_WRITE**

Packet pre-write operation. Called before write operation could even start. It can be used to get exclusive mutex access to the resource

enumerator **LWPKT_EVT_POST_WRITE**

Packet post-write operation. Called after write operation finished. It can be used to release exclusive mutex access from the resource

enumerator **LWPKT_EVT_PRE_READ**

Packet pre-read operation. Called before read operation could even start. It can be used to get exclusive mutex access to the resource

enumerator **LWPKT_EVT_POST_READ**

Packet post-read operation. Called after read operation finished. It can be used to release exclusive mutex access from the resource

Functions

lwpktr_t **lwpkt_init**(*lwpkt_t* *pkt, lwrb_t *tx_rb, lwrb_t *rx_rb)

Initialize packet instance and set device address.

Parameters

- **pkt** – **[in]** Packet instance
- **tx_rb** – **[in]** TX LwRB instance for data write
- **rx_rb** – **[in]** RX LwRB instance for data read

Returns

lwpktOK on success, member of *lwpktr_t* otherwise

lwpktr_t **lwpkt_set_addr**(*lwpkt_t* *pkt, *lwpkt_addr_t* addr)

Set device address for packet instance.

Parameters

- **pkt** – **[in]** Packet instance
- **addr** – **[in]** New device address

Returns

lwpktOK on success, member of *lwpktr_t* otherwise

lwpktr_t **lwpkt_read**(*lwpkt_t* *pkt)

Read raw data from RX ring buffer, parse the characters and try to construct the receive packet.

Parameters

pkt – **[in]** Packet instance

Returns

lwpktVALID when packet valid, member of *lwpktr_t* otherwise

lwpktr_t **lwpkt_write**(*lwpkt_t* *pkt, *lwpkt_addr_t* to, uint32_t flags, uint32_t cmd, const void *data, size_t len)

Write packet data to TX ringbuffer.

Parameters

- **pkt** – **[in]** Packet instance
- **to** – **[in]** End device address
- **cmd** – **[in]** Packet command
- **data** – **[in]** Pointer to input data. Set to `NULL` if not used
- **len** – **[in]** Length of input data. Must be set to `0` if `data == NULL`

Returns

lwpktOK on success, member of *lwpktr_t* otherwise

lwpktr_t **lwpkt_reset**(*lwpkt_t* *pkt)

Reset packet state.

Parameters

pkt – **[in]** Packet instance

Returns

lwpktOK on success, member of *lwpktr_t* otherwise

lwpktr_t **lwpkt_process**(*lwpkt_t* *pkt, uint32_t time)

Process packet instance and read new data.

Parameters

- **pkt** – **[in]** Packet instance
- **time** – **[in]** Current time in units of milliseconds

Returns
lwpktOK if processing OK, member of *lwpktr_t* otherwise

lwpktr_t **lwpkt_set_evt_fn**(*lwpkt_t* *pkt, *lwpkt_evt_fn* evt_fn)

Set event function for packet events.

Parameters

- **pkt** – **[in]** Packet structure
- **evt_fn** – **[in]** Function pointer for events

Returns
lwpktOK on success, member of *lwpktr_t* otherwise

void **lwpkt_set_addr_enabled**(*lwpkt_t* *pkt, uint8_t enable)

Enable addressing in the packet.

Note: This function is only available, if *LWPKT_CFG_USE_ADDR* is 2

Parameters

- **pkt** – LwPKT instance
- **enable** – 1 to enable, 0 otherwise

void **lwpkt_set_addr_extended_enabled**(*lwpkt_t* *pkt, uint8_t enable)

Enable extended addressing in the packet.

Note: This function is only available, if *LWPKT_CFG_ADDR_EXTENDED* is 2

Parameters

- **pkt** – LwPKT instance
- **enable** – 1 to enable, 0 otherwise

void **lwpkt_set_flags_enabled**(*lwpkt_t* *pkt, uint8_t enable)

Enable FLAGS mode in the packet.

Note: This function is only available, if *LWPKT_CFG_USE_FLAGS* is 2

Parameters

- **pkt** – LwPKT instance
- **enable** – 1 to enable, 0 otherwise

void **lwpkt_set_cmd_enabled**(*lwpkt_t* *pkt, uint8_t enable)

Enable CMD mode in the packet.

Note: This function is only available, if *LWPKT_CFG_USE_CMD* is 2

Parameters

- **pkt** – LwPKT instance
- **enable** – 1 to enable, 0 otherwise

void **lwpkt_set_cmd_extended_enabled**(*lwpkt_t* *pkt, uint8_t enable)

Enable extended addressing in the packet.

Note: This function is only available, if *LWPKT_CFG_CMD_EXTENDED* is 2

Parameters

- **pkt** – LwPKT instance
- **enable** – 1 to enable, 0 otherwise

void **lwpkt_set_crc_enabled**(*lwpkt_t* *pkt, uint8_t enable)

Set CRC mode enabled.

Note: This function is only available, if *LWPKT_CFG_USE_CRC* is 2

Parameters

- **pkt** – LwPKT instance
- **enable** – 1 to enable, 0 otherwise

void **lwpkt_set_crc32_enabled**(*lwpkt_t* *pkt, uint8_t enable)

Enable extended addressing in the packet.

Note: This function is only available, if *LWPKT_CFG_CRC32* is 2

Parameters

- **pkt** – LwPKT instance
- **enable** – 1 to enable, 0 otherwise

struct **lwpkt_crc_t**

#include <lwpkt.h> CRC structure for packet.

Public Members

uint32_t **crc**
Current CRC value

struct **lwpkt_t**
#include <lwpkt.h> Packet structure.

Public Members

lwpkt_addr_t **addr**
Current device address

uint8_t **data**[LWPKT_CFG_MAX_DATA_LEN]
Memory to write received data

lwrb_t ***tx_rb**
TX ringbuffer

lwrb_t ***rx_rb**
RX ringbuffer

uint32_t **last_rx_time**
Last RX time in units of milliseconds

lwpkt_evt_fn **evt_fn**
Global event function for read and write operation

uint8_t **flags**
List of flags

lwpkt_state_t **state**
Actual packet state machine

lwpkt_crc_t **crc**
Packet CRC object

uint32_t **crc_data**
CRC data received from the other side

lwpkt_addr_t **from**
Device address packet is coming from

lwpkt_addr_t **to**
Device address packet is intended for

uint32_t **flags**

Custom flags

uint32_t **cmd**

Command packet

size_t **len**

Number of bytes to receive

size_t **index**

General index variable for multi-byte parts of packet

struct *lwpkt_t*::[anonymous] **m**

Module that is periodically reset for next packet

6.3.2 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 `lwpkt_opts.h` file.

Note: Check *Getting started* for guidelines on how to create and use configuration file.

group **LWPKT_OPT**

Default configuration setup.

Defines

LWPKT_MEMSET(dst, val, len)

Memory set function.

Note: Function footprint is the same as memset

LWPKT_MEMCPY(dst, src, len)

Memory copy function.

Note: Function footprint is the same as memcpy

LWPKT_CFG_MAX_DATA_LEN

Maximum length of `data` part of the packet in units of bytes.

LWPKT_CFG_ADDR_BROADCAST

Address identifying broadcast message to all devices.

LWPKT_CFG_USE_ADDR

Enables 1 or disables 0 from and to fields in the protocol.

This features is useful if communication is between 2 devices exclusively, without addressing requirements

Configuration options:

- 0: Feature is globally disabled in the library
- 1: Feature is globally enabled in the library
- 2: Feature is dynamically enabled/disabled in the library, according to the LwPKT object instance. If set to 2, feature is by default enabled, but it can be disabled with appropriate API function.

LWPKT_CFG_ADDR_EXTENDED

Enables 1 or disables 0 extended address length.

When enabled, multi-byte addresses are supported with MSB codification. Maximum address is limited to 32-bits.

When disabled, simple 8-bit address is fixed with single byte.

Configuration options:

- 0: Feature is globally disabled in the library
- 1: Feature is globally enabled in the library
- 2: Feature is dynamically enabled/disabled in the library, according to the LwPKT object instance. If set to 2, feature is by default enabled, but it can be disabled with appropriate API function.

Note: *LWPKT_CFG_USE_ADDR* must be enabled for this feature to work

Note: Feature is disabled by default to keep architecture compatibility

LWPKT_CFG_USE_CMD

Enables 1 or disables 0 cmd field in the protocol.

When disabled, command part is not used

Configuration options:

- 0: Feature is globally disabled in the library
- 1: Feature is globally enabled in the library
- 2: Feature is dynamically enabled/disabled in the library, according to the LwPKT object instance. If set to 2, feature is by default enabled, but it can be disabled with appropriate API function.

LWPKT_CFG_CMD_EXTENDED

Enables 1 or disables 0 extended command length.

When enabled, multi-byte command length is supported in the protocol and is variable length integer encoded.

When disabled, simple `8-bit` address is fixed with single byte.

Configuration options:

- `0`: Feature is globally disabled in the library
- `1`: Feature is globally enabled in the library
- `2`: Feature is dynamically enabled/disabled in the library, according to the LwPKT object instance. If set to `2`, feature is by default enabled, but it can be disabled with appropriate API function.

Note: *LWPKT_CFG_USE_CMD* must be enabled for this feature to work

Note: Feature is disabled by default to keep architecture compatibility

LWPKT_CFG_USE_CRC

Enables `1` or disables `0` CRC check in the protocol.

Configuration options:

- `0`: Feature is globally disabled in the library
- `1`: Feature is globally enabled in the library
- `2`: Feature is dynamically enabled/disabled in the library, according to the LwPKT object instance. If set to `2`, feature is by default enabled, but it can be disabled with appropriate API function.

LWPKT_CFG_CRC32

Enables `1` or disables `0` CRC-32 type.

It controls if CRC type is set to `8-bits` or `32-bits`

Configuration options:

- `0`: CRC is ˙8-bits, fixed value. -1: Feature is globally enabled in the library and CRC is set fixed to32-bits -2: Feature is dynamically enabled/disabled in the library, according to the LwPKT object instance. If set to2`, feature is by default enabled, but it can be disabled with appropriate API function.

Note: *LWPKT_CFG_USE_CRC* must be enabled for this feature to work

LWPKT_CFG_USE_FLAGS

Enables `1` or disables `0` flags field in the protocol.

When enabled, multi-byte addresses are supported with MSB codification. Maximum address is limited to `32-bits`.

Configuration options:

- `0`: Feature is globally disabled in the library
- `1`: Feature is globally enabled in the library
- `2`: Feature is dynamically enabled/disabled in the library, according to the LwPKT object instance. If set to `2`, feature is by default enabled, but it can be disabled with appropriate API function.

Note: Feature is disabled by default to keep architecture compatibility

LWPKT_CFG_PROCESS_INPROG_TIMEOUT

Defines timeout time before packet is considered as not valid when too long time in data-read mode.

Used with *lwpkt_process* function

LWPKT_CFG_USE_EVT

Enables `1` or disables `0` event functions for read and write operations.

6.4 Changelog

```
# Changelog

## Develop

- Add python file prototype.
- Add support for `CRC-32` with `LWPKT_CFG_CRC32` option
- Add support for extended command length (variable length), configurable through `LWPKT_
↪CFG_CMD_EXTENDED`

## v1.3.0

- Split CMakeLists.txt files between library and executable
- Change license year to 2022
- Add `.clang-format` draft
- Change license year to 2023
- Add memory overflow check
- Add more events for pre and post write & read operations
- Add more recent version of LwRB
- Fix compilation error if CRC mode is disabled
- Add support for dynamic configuration, to support multiple LwPKT instances in one␣
↪project
- Add flags support to allow customer user flags in packet

## v1.2.0

- Added support for events on packet ready, read or write operation
- Add `library.json` for platform.io

## v1.1.0
```

(continues on next page)

(continued from previous page)

```
- Added support for variable length for address fields

## v1.0.1

- Added sphinx documentation to the repository
- Improved code documentation for doxygen compliancy

## v1.0.0

- First stable release
```

6.5 Authors

List of authors and contributors to the library

```
Tilen Majerle <tilen.majerle@gmail.com>
Tilen Majerle <tilen@majerle.eu>
```

INDEX

L

lwpkt_addr_t (*C++ type*), 24
LWPKT_CFG_ADDR_BROADCAST (*C macro*), 31
LWPKT_CFG_ADDR_EXTENDED (*C macro*), 32
LWPKT_CFG_CMD_EXTENDED (*C macro*), 32
LWPKT_CFG_CRC32 (*C macro*), 33
LWPKT_CFG_MAX_DATA_LEN (*C macro*), 31
LWPKT_CFG_PROCESS_INPROG_TIMEOUT (*C macro*), 34
LWPKT_CFG_USE_ADDR (*C macro*), 31
LWPKT_CFG_USE_CMD (*C macro*), 32
LWPKT_CFG_USE_CRC (*C macro*), 33
LWPKT_CFG_USE_EVT (*C macro*), 34
LWPKT_CFG_USE_FLAGS (*C macro*), 33
lwpkt_crc_t (*C++ struct*), 29
lwpkt_crc_t::crc (*C++ member*), 30
lwpkt_evt_fn (*C++ type*), 24
lwpkt_evt_type_t (*C++ enum*), 26
lwpkt_evt_type_t::LWPKT_EVT_PKT (*C++ enumerator*), 26
lwpkt_evt_type_t::LWPKT_EVT_POST_READ (*C++ enumerator*), 26
lwpkt_evt_type_t::LWPKT_EVT_POST_WRITE (*C++ enumerator*), 26
lwpkt_evt_type_t::LWPKT_EVT_PRE_READ (*C++ enumerator*), 26
lwpkt_evt_type_t::LWPKT_EVT_PRE_WRITE (*C++ enumerator*), 26
lwpkt_evt_type_t::LWPKT_EVT_READ (*C++ enumerator*), 26
lwpkt_evt_type_t::LWPKT_EVT_TIMEOUT (*C++ enumerator*), 26
lwpkt_evt_type_t::LWPKT_EVT_WRITE (*C++ enumerator*), 26
lwpkt_get_cmd (*C macro*), 23
lwpkt_get_data (*C macro*), 23
lwpkt_get_data_len (*C macro*), 23
lwpkt_get_flags (*C macro*), 24
lwpkt_get_from_addr (*C macro*), 23
lwpkt_get_to_addr (*C macro*), 23
lwpkt_init (*C++ function*), 27
lwpkt_is_broadcast (*C macro*), 24
lwpkt_is_for_me (*C macro*), 24
LWPKT_MEMCPY (*C macro*), 31
LWPKT_MEMSET (*C macro*), 31
lwpkt_process (*C++ function*), 27
lwpkt_read (*C++ function*), 27
lwpkt_reset (*C++ function*), 27
lwpkt_set_addr (*C++ function*), 27
lwpkt_set_addr_enabled (*C++ function*), 28
lwpkt_set_addr_extended_enabled (*C++ function*), 28
lwpkt_set_cmd_enabled (*C++ function*), 28
lwpkt_set_cmd_extended_enabled (*C++ function*), 29
lwpkt_set_crc32_enabled (*C++ function*), 29
lwpkt_set_crc_enabled (*C++ function*), 29
lwpkt_set_evt_fn (*C++ function*), 28
lwpkt_set_flags_enabled (*C++ function*), 28
lwpkt_state_t (*C++ enum*), 24
lwpkt_state_t::LWPKT_STATE_CMD (*C++ enumerator*), 25
lwpkt_state_t::LWPKT_STATE_CRC (*C++ enumerator*), 25
lwpkt_state_t::LWPKT_STATE_DATA (*C++ enumerator*), 25
lwpkt_state_t::LWPKT_STATE_END (*C++ enumerator*), 25
lwpkt_state_t::LWPKT_STATE_FLAGS (*C++ enumerator*), 25
lwpkt_state_t::LWPKT_STATE_FROM (*C++ enumerator*), 24
lwpkt_state_t::LWPKT_STATE_LEN (*C++ enumerator*), 25
lwpkt_state_t::LWPKT_STATE_START (*C++ enumerator*), 24
lwpkt_state_t::LWPKT_STATE_STOP (*C++ enumerator*), 25
lwpkt_state_t::LWPKT_STATE_TO (*C++ enumerator*), 25
lwpkt_t (*C++ struct*), 30
lwpkt_t::addr (*C++ member*), 30
lwpkt_t::cmd (*C++ member*), 31
lwpkt_t::crc (*C++ member*), 30
lwpkt_t::crc_data (*C++ member*), 30

lwpkt_t::data (*C++ member*), 30
lwpkt_t::evt_fn (*C++ member*), 30
lwpkt_t::flags (*C++ member*), 30
lwpkt_t::from (*C++ member*), 30
lwpkt_t::index (*C++ member*), 31
lwpkt_t::last_rx_time (*C++ member*), 30
lwpkt_t::len (*C++ member*), 31
lwpkt_t::m (*C++ member*), 31
lwpkt_t::rx_rb (*C++ member*), 30
lwpkt_t::state (*C++ member*), 30
lwpkt_t::to (*C++ member*), 30
lwpkt_t::tx_rb (*C++ member*), 30
lwpkt_write (*C++ function*), 27
lwpktr_t (*C++ enum*), 25
lwpktr_t::lwpktERR (*C++ enumerator*), 25
lwpktr_t::lwpktERRCRC (*C++ enumerator*), 25
lwpktr_t::lwpktERRMEM (*C++ enumerator*), 26
lwpktr_t::lwpktERRSTOP (*C++ enumerator*), 25
lwpktr_t::lwpktINPROG (*C++ enumerator*), 25
lwpktr_t::lwpktOK (*C++ enumerator*), 25
lwpktr_t::lwpktVALID (*C++ enumerator*), 25
lwpktr_t::lwpktWAITDATA (*C++ enumerator*), 26