
LwPKT

Tilen MAJERLE

Nov 15, 2020

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
	Index	25

Welcome to the documentation for version 1.0.1.

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

[Download library](#) [Getting started](#) [Open Github](#)

FEATURES

- Written in ANSI C99, compatible with `size_t` for size data types
- Platform independent, no architecture specific code
- Uses `LwRB` library for data read/write operations
- Optimized for embedded systems, allows high optimization for data transfer
- Configurable settings for packet structure and variable data length
- Allows multiple nodes 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

REQUIREMENTS

- C compiler
- Few kB of non-volatile memory

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

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.

TABLE OF CONTENTS

5.1 Getting started

5.1.1 Download library

Library is primarily 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/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 master 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 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 `lwpkt` folder to your project
- Add `lwpkt/src/include` folder to *include path* of your toolchain
- Add `libs/lwrb/src/include` folder to *include path* of your toolchain
- Add source files from `lwpkt/src/` folder to toolchain build
- Add source files from `libs/lwrb/src/` folder to toolchain build
- Build the project

5.1.3 Configuration file

Library comes with template config file, which can be modified according to needs. This file shall be named `lwpkt_opts.h` and its default template looks like the one below.

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 (or simply renamed in-place) where compiler include paths have access to it by using `#include "lwpkt_opts.h"`.

Tip: Check *Configuration* section for possible configuration settings

Listing 1: Template options file

```
1 /**
2  * \file          lwpkt_opts_template.h
3  * \brief        LwPKT configuration file
4  */
5
6 /**
7  * Copyright (c) 2020 Tilen MAJERLE
8  *
9  * Permission is hereby granted, free of charge, to any person
```

(continues on next page)

(continued from previous page)

```

10  * obtaining a copy of this software and associated documentation
11  * files (the "Software"), to deal in the Software without restriction,
12  * including without limitation the rights to use, copy, modify, merge,
13  * publish, distribute, sublicense, and/or sell copies of the Software,
14  * and to permit persons to whom the Software is furnished to do so,
15  * subject to the following conditions:
16  *
17  * The above copyright notice and this permission notice shall be
18  * included in all copies or substantial portions of the Software.
19  *
20  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
21  * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
22  * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
23  * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
24  * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
25  * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
26  * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
27  * OTHER DEALINGS IN THE SOFTWARE.
28  *
29  * This file is part of LwPKT - Lightweight packet protocol library.
30  *
31  * Author:          Tilen MAJERLE <tilen@majerle.eu>
32  * Version:         v1.0.1
33  */
34 #ifndef LWPKT_HDR_OPTS_H
35 #define LWPKT_HDR_OPTS_H
36
37 /* Rename this file to "lwpkt_opts.h" for your application */
38
39 /*
40  * Open "include/lwpkt/lwpkt_opt.h" and
41  * copy & replace here settings you want to change values
42  */
43
44 #endif /* LWPKT_HDR_OPTS_H */

```

5.1.4 Minimal example code

Run below example to test and verify library

Listing 2: Simple demo example to test functionality

```

1  #include <stdio.h>
2  #include "lwpkt/lwpkt.h"
3
4  /* LwPKT data */
5  static lwpkt_t pkt;
6  static lwrp_t pkt_tx_rb, pkt_rx_rb;
7  static uint8_t pkt_tx_rb_data[64], pkt_rx_rb_data[64];
8
9  /* Data to read and write */
10 static const char* data = "Hello World\r\n";
11
12 /**
13  * \brief          LwPKT example code

```

(continues on next page)

```

14  */
15  void
16  example_lwpkt(void) {
17      lwpktr_t res;
18      uint8_t b;
19
20      printf("---\r\nLwPKT default example..\r\n\r\n");
21
22      /*
23       * Initialize both ring buffers, for TX and RX operations
24       *
25       * Initialize LwPKT and link buffers together
26       */
27      lwrb_init(&pkt_tx_rb, pkt_tx_rb_data, sizeof(pkt_tx_rb_data));
28      lwrb_init(&pkt_rx_rb, pkt_rx_rb_data, sizeof(pkt_rx_rb_data));
29      lwpkt_init(&pkt, &pkt_tx_rb, &pkt_rx_rb);
30
31      #if LWPKT_CFG_USE_ADDR
32          /* Set device address (if feature enabled) */
33          lwpkt_set_addr(&pkt, 0x12);
34      #endif /* LWPKT_CFG_USE_ADDR */
35
36      /*
37       * Write packet to the TX ringbuffer,
38       * act as device wants to send some data
39       */
40      res = lwpkt_write(&pkt,
41      #if LWPKT_CFG_USE_ADDR
42          0x11, /* End address to whom to send */
43      #endif /* LWPKT_CFG_USE_ADDR */
44      #if LWPKT_CFG_USE_CMD
45          0x85, /* Command type */
46      #endif /* LWPKT_CFG_USE_CMD */
47          data, strlen(data)); /* Length of data and actual data */
48
49      /*
50       * LwPKT wrote data to pkt_tx_rb ringbuffer
51       * Now actually transmit data over your interface
52       * (USART for example, ...)
53       */
54
55      /*
56       * For the purpose of this example, application will
57       * fake data transmission by doing reading from TX buffer
58       * and writing it to RX buffer
59       */
60      while (lwrb_read(&pkt_tx_rb, &b, 1) == 1) {
61          lwrb_write(&pkt_rx_rb, &b, 1);
62      }
63
64      /*
65       * Here we have our data in RX buffer
66       * means we received data over network interface
67       */
68
69      /* Now read and process packet */
70      res = lwpkt_read(&pkt);

```

(continues on next page)

(continued from previous page)

```

71
72     if (res == lwpktVALID) {
73         size_t len;
74
75         /* Packet is valid */
76         printf("Packet is valid!\r\n");
77
78         /* Print debug messages for packet */
79 #if LwPKT_CFG_USE_ADDR
80         printf("Packet from: 0x%02X\r\n", (unsigned) lwpkt_get_from_addr(&pkt));
81         printf("Packet to: 0x%02X\r\n", (unsigned) lwpkt_get_to_addr(&pkt));
82 #endif /* LwPKT_CFG_USE_ADDR */
83 #if LwPKT_CFG_USE_CMD
84         printf("Packet cmd: 0x%02X\r\n", (unsigned) lwpkt_get_cmd(&pkt));
85 #endif /* LwPKT_CFG_USE_CMD */
86         printf("Packet data length: 0x%02X\r\n", (unsigned) lwpkt_get_data_len(&pkt));
87         if ((len = lwpkt_get_data_len(&pkt)) > 0) {
88             uint8_t* d = lwpkt_get_data(&pkt);
89             printf("Packet data: ");
90             for (size_t i = 0; i < len; ++i) {
91                 printf("0x%02X ", (unsigned) d[i]);
92             }
93             printf("\r\n");
94         }
95
96         /* Check who should be dedicated receiver */
97 #if LwPKT_CFG_USE_ADDR
98         if (lwpkt_is_for_me(&pkt)) {
99             printf("Packet is for me\r\n");
100         } else if (lwpkt_is_broadcast(&pkt)) {
101             printf("Packet is broadcast to all devices\r\n");
102         } else {
103             printf("Packet is for device ID: 0x%02X\r\n", (unsigned) lwpkt_get_to_
104 →addr(&pkt));
105         }
106 #endif /* LwPKT_CFG_USE_ADDR */
107         } else if (res == lwpktINPROG) {
108             printf("Packet is still in progress, did not receive yet all bytes..\r\n");
109         } else {
110             printf("Packet is not valid!\r\n");
111         }

```

5.2 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.

5.2.1 Packet structure

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

Fig. 1: Default packet structure

- **START:** Byte with fixed value to represent start of packet
- **FROM:** Byte from where this packet is coming from. Optional field, can be disabled with `LWPKT_CFG_USE_ADDR`
- **TO:** Byte to where this packet is targeting. Optional field, can be disabled with `LWPKT_CFG_USE_ADDR`
- **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

5.2.2 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.

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

5.2.3 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

```

1  #include <stdio.h>
2  #include "lwpkt/lwpkt.h"
3
4  /* LwPKT data */
5  static lwpkt_t pkt;
6  static lwrb_t pkt_tx_rb, pkt_rx_rb;
7  static uint8_t pkt_tx_rb_data[64], pkt_rx_rb_data[64];
8
9  /* Data to read and write */
10 static const char* data = "Hello World\r\n";
11
12 /**
13  * \brief          LwPKT application callback
14  */
15 static void
16 my_lwpkt_evt_fn(lwpkt_t* pkt, lwpkt_evt_type_t type) {
17     switch (type) {
18         case LWPKT_EVT_PKT: {
19             size_t len;
20
21             printf("Valid packet received..\r\n");
22
23             /* Packet is valid */
24             printf("Packet is valid!\r\n");
25
26             /* Print debug messages for packet */
27 #if LWPKT_CFG_USE_ADDR
28             printf("Packet from: 0x%02X\r\n", (unsigned)lwpkt_get_from_addr(pkt));
29             printf("Packet to: 0x%02X\r\n", (unsigned)lwpkt_get_to_addr(pkt));
30 #endif /* LWPKT_CFG_USE_ADDR */
31 #if LWPKT_CFG_USE_CMD
32             printf("Packet cmd: 0x%02X\r\n", (unsigned)lwpkt_get_cmd(pkt));
33 #endif /* LWPKT_CFG_USE_CMD */
34             printf("Packet data length: 0x%02X\r\n", (unsigned)lwpkt_get_data_
↵ len(pkt));
35
36             /* Do other thins... */
37             break;
38         }
39         case LWPKT_EVT_TIMEOUT: {
40             printf("Timeout detected during read operation..\r\n");
41             break;
42         }
43     }
44 }
45
46 /**
47  * \brief          LwPKT example code with event feature
48  */
49 void
50 example_lwpkt_evt(void) {

```

(continues on next page)

```

51  lwpktr_t res;
52  uint32_t time;
53  uint8_t b;
54
55  printf("---\r\nLwPKT event type..\r\n\r\n");
56
57  /*
58   * Initialize both ring buffers, for TX and RX operations
59   *
60   * Initialize LwPKT and link buffers together
61   */
62  lwrb_init(&pkt_tx_rb, pkt_tx_rb_data, sizeof(pkt_tx_rb_data));
63  lwrb_init(&pkt_rx_rb, pkt_rx_rb_data, sizeof(pkt_rx_rb_data));
64  lwpkt_init(&pkt, &pkt_tx_rb, &pkt_rx_rb);
65
66  #if LWPKT_CFG_USE_ADDR
67  /* Set device address (if feature enabled) */
68  lwpkt_set_addr(&pkt, 0x12);
69  #endif /* LWPKT_CFG_USE_ADDR */
70
71  /*
72   * Write packet to the TX ringbuffer,
73   * act as device wants to send some data
74   */
75  res = lwpkt_write(&pkt,
76  #if LWPKT_CFG_USE_ADDR
77      0x11, /* End address to whom to send */
78  #endif /* LWPKT_CFG_USE_ADDR */
79  #if LWPKT_CFG_USE_CMD
80      0x85, /* Command type */
81  #endif /* LWPKT_CFG_USE_CMD */
82      data, strlen(data)); /* Length of data and actual data */
83
84  /*
85   * LwPKT wrote data to pkt_tx_rb ringbuffer
86   * Now actually transmit data over your interface
87   * (USART for example, ...)
88   */
89
90  /*
91   * For the purpose of this example, application will
92   * fake data transmission by doing reading from TX buffer
93   * and writing it to RX buffer
94   */
95  while (lwrb_read(&pkt_tx_rb, &b, 1) == 1) {
96      lwrb_write(&pkt_rx_rb, &b, 1);
97  }
98
99  /*
100   * Here we have our data in RX buffer
101   * means we received data over network interface
102   */
103
104  /* Now call process function instead */
105  time = 100; /* Get_current_time_in_milliseconds */
106  lwpkt_process(&pkt, time, my_lwpkt_evt_fn);
107 }

```

5.3 API reference

List of all the modules:

5.3.1 LwPKT

group **LWPKT**

Lightweight packet protocol.

Defines

lwpkt_get_from_addr (*pkt*)

Get address from where packet was sent.

Return Address

Parameters

- [*in*] *pkt*: LwPKT instance

lwpkt_get_to_addr (*pkt*)

Get address to where packet was sent.

Return Address

Parameters

- [*in*] *pkt*: LwPKT instance

lwpkt_get_data_len (*pkt*)

Get length of packet.

Return Number of data bytes in packet

Parameters

- [*in*] *pkt*: LwPKT instance

lwpkt_get_data (*pkt*)

Get pointer to packet data.

Return Pointer to data

Parameters

- [*in*] *pkt*: LwPKT instance

lwpkt_get_cmd (*pkt*)

Get packet command data field.

Return Command data field

Parameters

- [*in*] *pkt*: LwPKT instance

lw_pkt_is_for_me (*pkt*)

Check if packet to field address matches device address.

Return 1 on success, 0 otherwise

Parameters

- [in] *pkt*: LwPKT instance

lw_pkt_is_broadcast (*pkt*)

Check if packet was sent to all devices on network.

Return 1 if broadcast, 0 otherwise

Parameters

- [in] *pkt*: LwPKT instance

Typedefs

typedef void (***lw_pkt_evt_fn**) (*lw_pkt_t* **pkt*, *lw_pkt_evt_type_t* type)

LwPKT event function.

Parameters

- [in] *pkt*: LwPKT instance with valid packet
- [in] *type*: Event type

Enums

enum **lw_pkt_state_t**

Packet state enumeration.

Values:

enumerator **LWPKT_STATE_START**

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_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

enum lwpktr_t

Packet result enumeration.

Values:

enumerator lwpktOK

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

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 packet, reset event

Functions

lwpktr_t lwpkt_init (**lwpkt_t *pkt**, **LWRB_VOLATILE lwr_b_t *tx_rb**, **LWRB_VOLATILE lwr_b_t ***

Initialize packet instance and set device address.

Return *lwpktOK* on success, member of *lwpktr_t* otherwise

Parameters

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

lwpktr_t lwpkt_set_addr (**lwpkt_t *pkt**, **uint8_t addr**)

Set device address for packet instance.

Return *lwpktOK* on success, member of *lwpktr_t* otherwise

Parameters

- [in] pkt: Packet instance
- [in] addr: New device address

lwptr_t **lwpkt_read** (*lwpkt_t* *pkt)

Read raw data from RX buffer and prepare packet.

Return *lwpktVALID* when packet valid, member of *lwptr_t* otherwise

Parameters

- [in] pkt: Packet instance

lwptr_t **lwpkt_write** (*lwpkt_t* *pkt, uint8_t to, uint8_t cmd, **const** void *data, size_t len)

Write packet data to TX ringbuffer.

Return *lwpktOK* on success, member of *lwptr_t* otherwise

Parameters

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

lwptr_t **lwpkt_reset** (*lwpkt_t* *pkt)

Reset packet state.

Return *lwpktOK* on success, member of *lwptr_t* otherwise

Parameters

- [in] pkt: Packet instance

lwptr_t **lwpkt_process** (*lwpkt_t* *pkt, uint32_t time, *lwpkt_evt_fn* evt_fn)

Process packet instance and read new data.

Return *lwpktOK* if processing OK, member of *lwptr_t* otherwise

Parameters

- [in] pkt: Packet instance
- [in] time: Current time in units of milliseconds
- [in] evt_fn: Event function to be called on events

struct **lwpkt_crc_t**

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

Public Members

`uint8_t crc`
Current CRC value

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

Public Members

`uint8_t addr`
Current device address

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

`LWRB_VOLATILE lwrb_t * tx_rb`
TX ringbuffer

`LWRB_VOLATILE lwrb_t * rx_rb`
RX ringbuffer

`uint32_t last_rx_time`
Last RX time in units of milliseconds

`lwpkt_state_t state`
Actual packet state machine

`lwpkt_crc_t crc`
Packet CRC byte

`uint8_t from`
Device address packet is coming from

`uint8_t to`
Device address packet is intended for

`uint8_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

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 `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_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

LWPKT_CFG_USE_CMD

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

When disabled, command part is not used

LWPKT_CFG_USE_CRC

Enables 1 or disables 0 CRC check in the protocol.

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

L

- LWPKT_CFG_ADDR_BROADCAST (*C macro*), 24
- LWPKT_CFG_MAX_DATA_LEN (*C macro*), 24
- LWPKT_CFG_PROCESS_INPROG_TIMEOUT (*C macro*), 24
- LWPKT_CFG_USE_ADDR (*C macro*), 24
- LWPKT_CFG_USE_CMD (*C macro*), 24
- LWPKT_CFG_USE_CRC (*C macro*), 24
- lwpkt_crc_t (*C++ struct*), 22
- lwpkt_crc_t::crc (*C++ member*), 23
- lwpkt_evt_fn (*C++ type*), 20
- lwpkt_evt_type_t (*C++ enum*), 21
- lwpkt_evt_type_t::LWPKT_EVT_PKT (*C++ enumerator*), 21
- lwpkt_evt_type_t::LWPKT_EVT_TIMEOUT (*C++ enumerator*), 21
- lwpkt_get_cmd (*C macro*), 19
- lwpkt_get_data (*C macro*), 19
- lwpkt_get_data_len (*C macro*), 19
- lwpkt_get_from_addr (*C macro*), 19
- lwpkt_get_to_addr (*C macro*), 19
- lwpkt_is_broadcast (*C macro*), 20
- lwpkt_is_for_me (*C macro*), 19
- lwpkt_process (*C++ function*), 22
- lwpkt_read (*C++ function*), 21
- lwpkt_reset (*C++ function*), 22
- lwpkt_set_addr (*C++ function*), 21
- lwpkt_state_t (*C++ enum*), 20
- lwpkt_state_t::LWPKT_STATE_CMD (*C++ enumerator*), 20
- lwpkt_state_t::LWPKT_STATE_CRC (*C++ enumerator*), 20
- lwpkt_state_t::LWPKT_STATE_DATA (*C++ enumerator*), 20
- lwpkt_state_t::LWPKT_STATE_FROM (*C++ enumerator*), 20
- lwpkt_state_t::LWPKT_STATE_LEN (*C++ enumerator*), 20
- lwpkt_state_t::LWPKT_STATE_START (*C++ enumerator*), 20
- lwpkt_state_t::LWPKT_STATE_STOP (*C++ enumerator*), 20
- lwpkt_state_t::LWPKT_STATE_TO (*C++ enumerator*), 20
- lwpkt_t (*C++ struct*), 23
- lwpkt_t::addr (*C++ member*), 23
- lwpkt_t::cmd (*C++ member*), 23
- lwpkt_t::crc (*C++ member*), 23
- lwpkt_t::data (*C++ member*), 23
- lwpkt_t::from (*C++ member*), 23
- lwpkt_t::index (*C++ member*), 23
- lwpkt_t::last_rx_time (*C++ member*), 23
- lwpkt_t::len (*C++ member*), 23
- lwpkt_t::m (*C++ member*), 23
- lwpkt_t::state (*C++ member*), 23
- lwpkt_t::to (*C++ member*), 23
- lwpkt_write (*C++ function*), 22
- lwpktr_t (*C++ enum*), 20
- lwpktr_t::lwpktERR (*C++ enumerator*), 21
- lwpktr_t::lwpktERRCRC (*C++ enumerator*), 21
- lwpktr_t::lwpktERRSTOP (*C++ enumerator*), 21
- lwpktr_t::lwpktINPROG (*C++ enumerator*), 21
- lwpktr_t::lwpktOK (*C++ enumerator*), 21
- lwpktr_t::lwpktVALID (*C++ enumerator*), 21
- lwpktr_t::lwpktWAITDATA (*C++ enumerator*), 21