LwCELL

Tilen MAJERLE

Mar 23, 2024

CONTENTS

Welcome to the documentation for version latest-develop.

LwCELL is lightweight, platform independent, cellular modem AT commands parser, targeting (as of today) communicion with *SIMCOM* based modules *SIM800/SIM900* or *SIM70xx*. Module is written in C11 and is independent from used platform. Its main targets are embedded system devices like ARM Cortex-M, AVR, PIC and others, but can easily work under *Windows*, *Linux* or *MAC* environments.

Download library Getting started Open Github Donate

CHAPTER

ONE

FEATURES

- Supports `SIM800/SIM900 (2G)` and `SIM7000/SIM7020 (NB-Iot LTE)` modules
- Platform independent and very easy to port
  - Development of library under Win32 platform
  - Provided examples for ARM Cortex-M or Win32 platforms
- Written in C language (C11)
- Allows different configurations to optimize user requirements
- Supports implementation with operating systems with advanced inter-thread communications
  - Currently only OS mode is supported
  - 2 different threads handling user data and received data
    - First (producer) thread (collects user commands from user threads and starts the command processing)
    - Second (process) thread reads the data from GSM device and does the job accordingly
- Allows sequential API for connections in client and server mode
- Includes several applications built on top of library
  - MQTT client for MQTT connection
- User friendly MIT license

CHAPTER

TWO

REQUIREMENTS

- C compiler
- Supported GSM Physical device

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/lwcell` command to clone entire repository, including submodules
  - Run `git clone --recurse-submodules --branch develop https://github.com/MaJerle/lwcell` to clone *development* branch, including submodules
  - Run `git clone --recurse-submodules --branch main https://github.com/MaJerle/lwcell` 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 `lwcell` 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 `lwcell` folder to your project, it contains library files
- Add `lwcell/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 `lwcell/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 `lwcell/src/include/lwcell/lwcell_opts_template.h` to project folder and rename it to `lwcell_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 `lwcell_opts.h`

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

have access to it by using `#include "lwcell_opts.h"`.

Tip: If you are using *CMake* build system, define the variable `LWCELL_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            lwcell_opts_template.h
 * \brief           Template config 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 LwCELL - Lightweight cellular modem AT library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v0.1.1
 */
#ifndef LWCELL_OPTS_HDR_H
#define LWCELL_OPTS_HDR_H

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

/*
 * Open "include/lwcell/lwcell_opt.h" and
```

(continues on next page)

(continued from previous page)

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

#endif /* LWCELL_OPTS_HDR_H */
```

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

5.2 User manual

5.2.1 Overview

IoT activity is embedded in today's application. Almost every device uses some type of communication, from WiFi, BLE, Sub-GHz or NB-IoT/LTE/2G/3G.

LwCELL has been developed to allow customers to:

- Develop on single (host MCU) architecture at the same time and do not care about device arch
- Shorten time to market

Customers using *LwCELL* do not need to take care about proper command for specific task, they can call API functions to execute the task. Library will take the necessary steps in order to send right command to device via low-level driver (usually UART) and process incoming response from device before it will notify application layer if it was successfuly or not.

To summarize:

- *GSM* device runs official *AT* firmware, provided by device vendor
- Host MCU runs custom application, together with *LwCELL* library
- Host MCU communicates with *GSM* device with UART or similar interface.

5.2.2 Architecture

Architecture of the library consists of 4 layers.

Fig. 1: LwCELL layer architecture overview

Application layer

User layer is the highest layer of the final application. This is the part where API functions are called to execute some command.

Middleware layer

Middleware part is actively developed and shall not be modified by customer by any means. If there is a necessity to do it, often it means that developer of the application uses it wrongly. This part is platform independent and does not use any specific compiler features for proper operation.

Note: There is no compiler specific features implemented in this layer.

System & low-level layer

Application needs to fully implement this part and resolve it with care. Functions are related to actual implementation with *GSM* device and are highly architecture oriented. Some examples for *WIN32* and *ARM Cortex-M* are included with library.

Tip: Check *Porting guide* for detailed instructions and examples.

System functions

System functions are bridge between operating system running on embedded system and LwCELL middleware. Functions need to provide:

- Thread management
- Binary semaphore management
- Recursive mutex management
- Message queue management
- Current time status information

Tip: System function prototypes are available in *System functions* section.

Low-level implementation

Low-Level, or *LWCELL_LL*, is part, dedicated for communication between *LwCELL* middleware and *GSM* physical device. Application needs to implement output function to send necessary *AT command* instruction aswell as implement *input module* to send received data from *GSM* device to *LwCELL* middleware.

Application must also assure memory assignment for api_lwcell_mem when default allocation is used.

Tip: Low level, input module & memory function prototypes are available in *Low-Level functions*, api_lwcell_input and api_lwcell_mem respectfully.

GSM physical device

5.2.3 Inter thread communication

LwCELL is only available with operating system. For successful resources management, it uses 2 threads within library and allows multiple application threads to post new command to be processed.

Fig. 2: Inter-thread architecture block diagram

Producing and *Processing* threads are part of library, its implementation is in `lwcell_threads.c` file.

Processing thread

Processing thread is in charge of processing each and every received character from *GSM* device. It can process *URC* messages which are received from *GSM* device without any command request. Some of them are:

- *+RECEIVE* indicating new data packet received from remote side on active connection
- *RING* indicating new call to be processed by *GSM*
- and more others

Note: Received messages without any command (URC messages) are sent to application layer using events, where they can be processed and used in further steps

---

This thread also checks and processes specific received messages based on active command. As an example, when application tries to make a new connection to remote server, it starts command with `AT+CIPSTART` message. Thread understands that active command is to connect to remote side and will wait for potential `0, CONNECT OK` message, indicating connection status. it will also wait for `OK` or `ERROR`, indicating *command finished* status before it unlocks **sync_sem** to unblock *producing thread*.

Tip: When thread tries to unlock **sync_sem**, it first checks if it has been locked by *producing thread*.

Producing thread

Producing thread waits for command messages posted from application thread. When new message has been received, it sends initial *AT message* over AT port.

- It checks if command is valid and if it has corresponding initial AT sequence, such as `AT+CIPSTART`
- It locks **sync_sem** semaphore and waits for processing thread to unlock it
  - *Processing thread* is in charge to read respone from *GSM* and react accordingly. See previous section for details.
- If application uses *blocking mode*, it unlocks command **sem** semaphore and returns response
- If application uses *non-blocking mode*, it frees memory for message and sends event with response message

Application thread

Application thread is considered any thread which calls API functions and therefore writes new messages to *producing message queue*, later processed by *producing thread*.

A new message memory is allocated in this thread and type of command is assigned to it, together with required input data for command. It also sets *blocking* or *non-blocking* mode, how command shall be executed.

When application tries to execute command in *blocking mode*, it creates new sync semaphore **sem**, locks it, writes message to *producing queue* and waits for **sem** to get unlocked. This effectively puts thread to blocked state by operating system and removes it from scheduler until semaphore is unlocked again. Semaphore **sem** gets unlocked in *producing thread* when response has been received for specific command.

Tip: **sem** semaphore is unlocked in *producing* thread after **sync_sem** is unlocked in *processing* thread

Note: Every command message uses its own **sem** semaphore to sync multiple *application* threads at the same time.

If message is to be executed in *non-blocking* mode, **sem** is not created as there is no need to block application thread. When this is the case, application thread will only write message command to *producing queue* and return status of writing to application.

5.2.4 Events and callback functions

Library uses events to notify application layer for (possible, but not limited to) unexpected events. This concept is used aswell for commands with longer executing time, such as *scanning access points* or when application starts new connection as client mode.

There are 3 types of events/callbacks available:

- *Global event* callback function, assigned when initializing library
- *Connection specific event* callback function, to process only events related to connection, such as *connection error*, *data send*, *data receive*, *connection closed*
- *API function* call based event callback function

Every callback is always called from protected area of middleware (when exclusing access is granted to single thread only), and it can be called from one of these 3 threads:

- *Producing thread*
- *Processing thread*
- *Input thread*, when `LWCELL_CFG_INPUT_USE_PROCESS` is enabled and `lwcell_input_process()` function is called

Tip: Check *Inter thread communication* for more details about *Producing* and *Processing* thread.

Global event callback

Global event callback function is assigned at library initialization. It is used by the application to receive any kind of event, except the one related to connection:

- GSM station successfully connected to access point
- GSM physical device reset has been detected
- Restore operation finished
- New station has connected to access point
- and many more..

Tip: Check api_lwcell_evt section for different kind of events

By default, global event function is single function. If the application tries to split different events with different callback functions, it is possible to do so by using `lwcell_evt_register()` function to register a new, custom, event function.

Tip: Implementation of *Netconn API* leverages `lwcell_evt_register()` to receive event when station disconnected from wifi access point. Check its source file for actual implementation.

Listing 2: Netconn API module actual implementation

```
/**
 * \file            lwcell_netconn.c
 * \brief           API functions for sequential calls
 */

/*
 * 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.
 *
```

(continues on next page)

(continued from previous page)

```
 * This file is part of LwCELL - Lightweight cellular modem AT library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v0.1.1
 */
#include "lwcell/lwcell_netconn.h"
#include "lwcell/lwcell_conn.h"
#include "lwcell/lwcell_mem.h"
#include "lwcell/lwcell_private.h"

#if LWCELL_CFG_NETCONN || __DOXYGEN__

/* Check conditions */
#if !LWCELL_CFG_CONN
#error "LWCELL_CFG_CONN must be enabled for NETCONN API!"
#endif /* !LWCELL_CFG_CONN */

#if LWCELL_CFG_NETCONN_RECEIVE_QUEUE_LEN < 2
#error "LWCELL_CFG_NETCONN_RECEIVE_QUEUE_LEN must be greater or equal to 2"
#endif /* LWCELL_CFG_NETCONN_RECEIVE_QUEUE_LEN < 2 */

/**
 * \brief           Sequential API structure
 */
typedef struct lwcell_netconn {
    struct lwcell_netconn* next;    /*!< Linked list entry */

    lwcell_netconn_type_t type;     /*!< Netconn type */

    size_t rcv_packets;             /*!< Number of received packets so far on this␣
↪connection */
    lwcell_conn_p conn;             /*!< Pointer to actual connection */

    lwcell_sys_mbox_t mbox_receive; /*!< Message queue for receive mbox */

    lwcell_linbuff_t buff;          /*!< Linear buffer structure */

    uint16_t conn_timeout;          /*!< Connection timeout in units of seconds when
                                         netconn is in server (listen) mode.
                                         Connection will be automatically␣
↪closed if there is no
                                         data exchange in time. Set to `0`␣
↪when timeout feature is disabled. */

#if LWCELL_CFG_NETCONN_RECEIVE_TIMEOUT || __DOXYGEN__
    uint32_t rcv_timeout; /*!< Receive timeout in unit of milliseconds */
#endif
} lwcell_netconn_t;

static uint8_t recv_closed = 0xFF;
static lwcell_netconn_t* netconn_list; /*!< Linked list of netconn entries */

```

(continues on next page)

(continued from previous page)

```
/**
 * \brief           Flush all mboxes and clear possible used memories
 * \param[in]       nc: Pointer to netconn to flush
 * \param[in]       protect: Set to 1 to protect against multi-thread access
 */
static void
flush_mboxes(lwcell_netconn_t* nc, uint8_t protect) {
    lwcell_pbuf_p pbuf;
    if (protect) {
        lwcell_core_lock();
    }
    if (lwcell_sys_mbox_isvalid(&nc->mbox_receive)) {
        while (lwcell_sys_mbox_getnow(&nc->mbox_receive, (void**)&pbuf)) {
            if (pbuf != NULL && (uint8_t*)pbuf != (uint8_t*)&recv_closed) {
                lwcell_pbuf_free_s(&pbuf); /* Free received data buffers */
            }
        }
        lwcell_sys_mbox_delete(&nc->mbox_receive);  /* Delete message queue */
        lwcell_sys_mbox_invalid(&nc->mbox_receive); /* Invalid handle */
    }
    if (protect) {
        lwcell_core_unlock();
    }
}

/**
 * \brief           Callback function for every server connection
 * \param[in]       evt: Pointer to callback structure
 * \return          Member of \ref lwcellr_t enumeration
 */
static lwcellr_t
netconn_evt(lwcell_evt_t* evt) {
    lwcell_conn_p conn;
    lwcell_netconn_t* nc = NULL;
    uint8_t close = 0;

    conn = lwcell_conn_get_from_evt(evt); /* Get connection from event */
    switch (lwcell_evt_get_type(evt)) {
        /*
         * A new connection has been active
         * and should be handled by netconn API
         */
        case LWCELL_EVT_CONN_ACTIVE: {          /* A new connection active is active */
            if (lwcell_conn_is_client(conn)) {  /* Was connection started by us? */
                nc = lwcell_conn_get_arg(conn); /* Argument should be already set */
                if (nc != NULL) {
                    nc->conn = conn;            /* Save actual connection */
                } else {
                    close = 1;                  /* Close this connection, invalid netconn
↪*/
                }
            } else {
```

(continues on next page)

(continued from previous page)

```
                LWCELL_DEBUGF(LWCELL_CFG_DBG_NETCONN | LWCELL_DBG_TYPE_TRACE | LWCELL_
↪DBG_LVL_WARNING,
                              "[LWCELL NETCONN] Closing connection, it is not in client␣
↪mode!\r\n");
                close = 1; /* Close the connection at this point */
            }

            /* Decide if some events want to close the connection */
            if (close) {
                if (nc != NULL) {
                    lwcell_conn_set_arg(conn, NULL); /* Reset argument */
                    lwcell_netconn_delete(nc);       /* Free memory for API */
                }
                lwcell_conn_close(conn, 0);          /* Close the connection */
                close = 0;
            }
            break;
        }

        /*
         * We have a new data received which
         * should have netconn structure as argument
         */
        case LWCELL_EVT_CONN_RECV: {
            lwcell_pbuf_p pbuf;

            nc = lwcell_conn_get_arg(conn);            /* Get API from connection */
            pbuf = lwcell_evt_conn_recv_get_buff(evt); /* Get received buff */

            lwcell_conn_recved(conn, pbuf);            /* Notify stack about received␣
↪data */

            lwcell_pbuf_ref(pbuf);                     /* Increase reference counter */
            if (nc == NULL || !lwcell_sys_mbox_isvalid(&nc->mbox_receive)
                || !lwcell_sys_mbox_putnow(&nc->mbox_receive, pbuf)) {
                LWCELL_DEBUGF(LWCELL_CFG_DBG_NETCONN, "[LWCELL NETCONN] Ignoring more␣
↪data for receive!\r\n");
                lwcell_pbuf_free_s(&pbuf); /* Free pbuf */
                return lwcellOKIGNOREMORE; /* Return OK to free the memory and ignore␣
↪further data */
            }
            ++nc->rcv_packets;            /* Increase number of received packets */
            LWCELL_DEBUGF(LWCELL_CFG_DBG_NETCONN | LWCELL_DBG_TYPE_TRACE,
                          "[LWCELL NETCONN] Received pbuf contains %d bytes. Handle␣
↪written to receive mbox\r\n",
                          (int)lwcell_pbuf_length(pbuf, 0));
            break;
        }

        /* Connection was just closed */
        case LWCELL_EVT_CONN_CLOSE: {
            nc = lwcell_conn_get_arg(conn); /* Get API from connection */
```

(continues on next page)

(continued from previous page)

```
            /*
             * In case we have a netconn available,
             * simply write pointer to received variable to indicate closed state
             */
            if (nc != NULL && lwcell_sys_mbox_isvalid(&nc->mbox_receive)) {
                lwcell_sys_mbox_putnow(&nc->mbox_receive, (void*)&recv_closed);
            }

            break;
        }
        default: return lwcellERR;
    }
    return lwcellOK;
}

/**
 * \brief           Global event callback function
 * \param[in]       evt: Callback information and data
 * \return          \ref lwcellOK on success, member of \ref lwcellr_t otherwise
 */
static lwcellr_t
lwcell_evt(lwcell_evt_t* evt) {
    switch (lwcell_evt_get_type(evt)) {
        default: break;
    }
    return lwcellOK;
}

/**
 * \brief           Create new netconn connection
 * \param[in]       type: Netconn connection type
 * \return          New netconn connection on success, `NULL` otherwise
 */
lwcell_netconn_p
lwcell_netconn_new(lwcell_netconn_type_t type) {
    lwcell_netconn_t* a;
    static uint8_t first = 1;

    /* Register only once! */
    lwcell_core_lock();
    if (first) {
        first = 0;
        lwcell_evt_register(lwcell_evt); /* Register global event function */
    }
    lwcell_core_unlock();
    a = lwcell_mem_calloc(1, sizeof(*a)); /* Allocate memory for core object */
    if (a != NULL) {
        a->type = type;                 /* Save netconn type */
        a->conn_timeout = 0;            /* Default connection timeout */
        if (!lwcell_sys_mbox_create(
                &a->mbox_receive,
```

(continues on next page)

(continued from previous page)

```
                LWCELL_CFG_NETCONN_RECEIVE_QUEUE_LEN)) { /* Allocate memory for receiving message box */
            LWCELL_DEBUGF(LWCELL_CFG_DBG_NETCONN | LWCELL_DBG_TYPE_TRACE | LWCELL_DBG_LVL_DANGER,
                          "[LWCELL NETCONN] Cannot create receive MBOX\r\n");
            goto free_ret;
        }
        lwcell_core_lock();
        if (netconn_list == NULL) { /* Add new netconn to the existing list */
            netconn_list = a;
        } else {
            a->next = netconn_list; /* Add it to beginning of the list */
            netconn_list = a;
        }
        lwcell_core_unlock();
    }
    return a;
free_ret:
    if (lwcell_sys_mbox_isvalid(&a->mbox_receive)) {
        lwcell_sys_mbox_delete(&a->mbox_receive);
        lwcell_sys_mbox_invalid(&a->mbox_receive);
    }
    if (a != NULL) {
        lwcell_mem_free_s((void**)&a);
    }
    return NULL;
}

/**
 * \brief           Delete netconn connection
 * \param[in]       nc: Netconn handle
 * \return          \ref lwcellOK on success, member of \ref lwcellr_t enumeration otherwise
 */
lwcellr_t
lwcell_netconn_delete(lwcell_netconn_p nc) {
    LWCELL_ASSERT(nc != NULL);

    lwcell_core_lock();
    flush_mboxes(nc, 0); /* Clear mboxes */

    /* Remove netconn from linkedlist */
    if (netconn_list == nc) {
        netconn_list = netconn_list->next; /* Remove first from linked list */
    } else if (netconn_list != NULL) {
        lwcell_netconn_p tmp, prev;
        /* Find element on the list */
        for (prev = netconn_list, tmp = netconn_list->next; tmp != NULL; prev = tmp, tmp = tmp->next) {
            if (nc == tmp) {
                prev->next = tmp->next; /* Remove tmp from linked list */
                break;
```

(continues on next page)

(continued from previous page)

```
            }
        }
    }
    lwcell_core_unlock();

    lwcell_mem_free_s((void**)&nc);
    return lwcellOK;
}

/**
 * \brief           Connect to server as client
 * \param[in]       nc: Netconn handle
 * \param[in]       host: Pointer to host, such as domain name or IP address in string format
 * \param[in]       port: Target port to use
 * \return          \ref lwcellOK if successfully connected, member of \ref lwcellr_t otherwise
 */
lwcellr_t
lwcell_netconn_connect(lwcell_netconn_p nc, const char* host, lwcell_port_t port) {
    lwcellr_t res;

    LWCELL_ASSERT(nc != NULL);
    LWCELL_ASSERT(host != NULL);
    LWCELL_ASSERT(port > 0);

    /*
     * Start a new connection as client and:
     *
     *  - Set current netconn structure as argument
     *  - Set netconn callback function for connection management
     *  - Start connection in blocking mode
     */
    res = lwcell_conn_start(NULL, (lwcell_conn_type_t)nc->type, host, port, nc, netconn_evt, 1);
    return res;
}

/**
 * \brief           Write data to connection output buffers
 * \note            This function may only be used on TCP or SSL connections
 * \param[in]       nc: Netconn handle used to write data to
 * \param[in]       data: Pointer to data to write
 * \param[in]       btw: Number of bytes to write
 * \return          \ref lwcellOK on success, member of \ref lwcellr_t enumeration otherwise
 */
lwcellr_t
lwcell_netconn_write(lwcell_netconn_p nc, const void* data, size_t btw) {
    size_t len, sent;
    const uint8_t* d = data;
    lwcellr_t res;
```

(continues on next page)

(continued from previous page)

```
323
324     LWCELL_ASSERT(nc != NULL);
325     LWCELL_ASSERT(nc->type == LWCELL_NETCONN_TYPE_TCP || nc->type == LWCELL_NETCONN_TYPE_
↪SSL);
326     LWCELL_ASSERT(lwcell_conn_is_active(nc->conn));
327
328     /*
329      * Several steps are done in write process
330      *
331      * 1. Check if buffer is set and check if there is something to write to it.
332      *    1. In case buffer will be full after copy, send it and free memory.
333      * 2. Check how many bytes we can write directly without need to copy
334      * 3. Try to allocate a new buffer and copy remaining input data to it
335      * 4. In case buffer allocation fails, send data directly (may affect on speed and␣
↪effectivenes)
336      */
337
338     /* Step 1 */
339     if (nc->buff.buff != NULL) {                            /* Is there a write buffer␣
↪ready to accept more data? */
340         len = LWCELL_MIN(nc->buff.len - nc->buff.ptr, btw); /* Get number of bytes we␣
↪can write to buffer */
341         if (len > 0) {
342             LWCELL_MEMCPY(&nc->buff.buff[nc->buff.ptr], data, len); /* Copy memory to␣
↪temporary write buffer */
343             d += len;
344             nc->buff.ptr += len;
345             btw -= len;
346         }
347
348         /* Step 1.1 */
349         if (nc->buff.ptr == nc->buff.len) {
350             res = lwcell_conn_send(nc->conn, nc->buff.buff, nc->buff.len, &sent, 1);
351
352             lwcell_mem_free_s((void**)&nc->buff.buff);
353             if (res != lwcellOK) {
354                 return res;
355             }
356         } else {
357             return lwcellOK; /* Buffer is not yet full yet */
358         }
359     }
360
361     /* Step 2 */
362     if (btw >= LWCELL_CFG_CONN_MAX_DATA_LEN) {
363         size_t rem;
364         rem = btw % LWCELL_CFG_CONN_MAX_DATA_LEN;                /* Get remaining bytes␣
↪for max data length */
365         res = lwcell_conn_send(nc->conn, d, btw - rem, &sent, 1); /* Write data directly␣
↪*/
366         if (res != lwcellOK) {
367             return res;
```

(continues on next page)

(continued from previous page)

```
        }
        d += sent;   /* Advance in data pointer */
        btw -= sent; /* Decrease remaining data to send */
    }

    if (btw == 0) { /* Sent everything? */
        return lwcellOK;
    }

    /* Step 3 */
    if (nc->buff.buff == NULL) {                    /* Check if we should allocate a new buffer */
        nc->buff.buff = lwcell_mem_malloc(sizeof(*nc->buff.buff) * LWCELL_CFG_CONN_MAX_DATA_LEN);
        nc->buff.len = LWCELL_CFG_CONN_MAX_DATA_LEN; /* Save buffer length */
        nc->buff.ptr = 0;                           /* Save buffer pointer */
    }

    /* Step 4 */
    if (nc->buff.buff != NULL) {                                /* Memory available? */
        LWCELL_MEMCPY(&nc->buff.buff[nc->buff.ptr], d, btw);   /* Copy data to buffer */
        nc->buff.ptr += btw;
    } else {                                                    /* Still no memory available? */
        return lwcell_conn_send(nc->conn, data, btw, NULL, 1); /* Simply send directly blocking */
    }
    return lwcellOK;
}

/**
 * \brief           Extended version of \ref lwcell_netconn_write with additional
 *                  option to set custom flags.
 *
 * \note            It is recommended to use this for full features support
 *
 * \param[in]       nc: Netconn handle used to write data to
 * \param[in]       data: Pointer to data to write
 * \param[in]       btw: Number of bytes to write
 * \param           flags: Bitwise-ORed set of flags for netconn.
 *                      Flags start with \ref LWCELL_NETCONN_FLAG_xxx
 * \return          \ref lwcellOK on success, member of \ref lwcellr_t enumeration otherwise
 */
lwcellr_t
lwcell_netconn_write_ex(lwcell_netconn_p nc, const void* data, size_t btw, uint16_t flags) {
    lwcellr_t res = lwcell_netconn_write(nc, data, btw);
    if (res == lwcellOK) {
        if (flags & LWCELL_NETCONN_FLAG_FLUSH) {
            res = lwcell_netconn_flush(nc);
        }
```

(continues on next page)

(continued from previous page)

```
    }
    return res;
}

/**
 * \brief           Flush buffered data on netconn \e TCP/SSL connection
 * \note            This function may only be used on \e TCP/SSL connection
 * \param[in]       nc: Netconn handle to flush data
 * \return          \ref lwcellOK on success, member of \ref lwcellr_t enumeration␣
↪otherwise
 */
lwcellr_t
lwcell_netconn_flush(lwcell_netconn_p nc) {
    LWCELL_ASSERT(nc != NULL);
    LWCELL_ASSERT(nc->type == LWCELL_NETCONN_TYPE_TCP || nc->type == LWCELL_NETCONN_TYPE_
↪SSL);
    LWCELL_ASSERT(lwcell_conn_is_active(nc->conn));

    /*
     * In case we have data in write buffer,
     * flush them out to network
     */
    if (nc->buff.buff != NULL) {                                        /* Check␣
↪remaining data */
        if (nc->buff.ptr > 0) {                                         /* Do we␣
↪have data in current buffer? */
            lwcell_conn_send(nc->conn, nc->buff.buff, nc->buff.ptr, NULL, 1); /* Send␣
↪data */
        }
        lwcell_mem_free_s((void**)&nc->buff.buff);
    }
    return lwcellOK;
}

/**
 * \brief           Send data on \e UDP connection to default IP and port
 * \param[in]       nc: Netconn handle used to send
 * \param[in]       data: Pointer to data to write
 * \param[in]       btw: Number of bytes to write
 * \return          \ref lwcellOK on success, member of \ref lwcellr_t enumeration␣
↪otherwise
 */
lwcellr_t
lwcell_netconn_send(lwcell_netconn_p nc, const void* data, size_t btw) {
    LWCELL_ASSERT(nc != NULL);
    LWCELL_ASSERT(nc->type == LWCELL_NETCONN_TYPE_UDP);
    LWCELL_ASSERT(lwcell_conn_is_active(nc->conn));

    return lwcell_conn_send(nc->conn, data, btw, NULL, 1);
}

/**
```

(continues on next page)

(continued from previous page)

```
 * \brief           Send data on \e UDP connection to specific IP and port
 * \note            Use this function in case of UDP type netconn
 * \param[in]       nc: Netconn handle used to send
 * \param[in]       ip: Pointer to IP address
 * \param[in]       port: Port number used to send data
 * \param[in]       data: Pointer to data to write
 * \param[in]       btw: Number of bytes to write
 * \return          \ref lwcellOK on success, member of \ref lwcellr_t enumeration␣
↪otherwise
 */
lwcellr_t
lwcell_netconn_sendto(lwcell_netconn_p nc, const lwcell_ip_t* ip, lwcell_port_t port,␣
↪const void* data, size_t btw) {
    LWCELL_ASSERT(nc != NULL);
    LWCELL_ASSERT(nc->type == LWCELL_NETCONN_TYPE_UDP);
    LWCELL_ASSERT(lwcell_conn_is_active(nc->conn));

    return lwcell_conn_sendto(nc->conn, ip, port, data, btw, NULL, 1);
}

/**
 * \brief           Receive data from connection
 * \param[in]       nc: Netconn handle used to receive from
 * \param[in]       pbuf: Pointer to pointer to save new receive buffer to.
 *                     When function returns, user must check for valid pbuf value `pbuf␣
↪!= NULL`
 * \return          \ref lwcellOK when new data ready,
 * \return          \ref lwcellCLOSED when connection closed by remote side,
 * \return          \ref lwcellTIMEOUT when receive timeout occurs
 * \return          Any other member of \ref lwcellr_t otherwise
 */
lwcellr_t
lwcell_netconn_receive(lwcell_netconn_p nc, lwcell_pbuf_p* pbuf) {
    LWCELL_ASSERT(nc != NULL);
    LWCELL_ASSERT(pbuf != NULL);

    *pbuf = NULL;
#if LWCELL_CFG_NETCONN_RECEIVE_TIMEOUT
    /*
     * Wait for new received data for up to specific timeout
     * or throw error for timeout notification
     */
    if (nc->rcv_timeout == LWCELL_NETCONN_RECEIVE_NO_WAIT) {
        if (!lwcell_sys_mbox_getnow(&nc->mbox_receive, (void**)pbuf)) {
            return lwcellTIMEOUT;
        }
    } else if (lwcell_sys_mbox_get(&nc->mbox_receive, (void**)pbuf, nc->rcv_timeout) ==␣
↪LWCELL_SYS_TIMEOUT) {
        return lwcellTIMEOUT;
    }
#else  /* LWCELL_CFG_NETCONN_RECEIVE_TIMEOUT */
    /* Forever wait for new receive packet */
```

(continues on next page)

(continued from previous page)

```
    lwcell_sys_mbox_get(&nc->mbox_receive, (void**)pbuf, 0);
#endif /* !LWCELL_CFG_NETCONN_RECEIVE_TIMEOUT */

    /* Check if connection closed */
    if ((uint8_t*)(*pbuf) == (uint8_t*)&recv_closed) {
        *pbuf = NULL; /* Reset pbuf */
        return lwcellCLOSED;
    }
    return lwcellOK; /* We have data available */
}

/**
 * \brief           Close a netconn connection
 * \param[in]       nc: Netconn handle to close
 * \return          \ref lwcellOK on success, member of \ref lwcellr_t enumeration␣
↪otherwise
 */
lwcellr_t
lwcell_netconn_close(lwcell_netconn_p nc) {
    lwcell_conn_p conn;

    LWCELL_ASSERT(nc != NULL);
    LWCELL_ASSERT(nc->conn != NULL);
    LWCELL_ASSERT(lwcell_conn_is_active(nc->conn));

    lwcell_netconn_flush(nc); /* Flush data and ignore result */
    conn = nc->conn;
    nc->conn = NULL;

    lwcell_conn_set_arg(conn, NULL); /* Reset argument */
    lwcell_conn_close(conn, 1);      /* Close the connection */
    flush_mboxes(nc, 1);            /* Flush message queues */
    return lwcellOK;
}

/**
 * \brief           Get connection number used for netconn
 * \param[in]       nc: Netconn handle
 * \return          `-1` on failure, connection number between `0` and \ref LWCELL_CFG_
↪MAX_CONNS otherwise
 */
int8_t
lwcell_netconn_getconnnum(lwcell_netconn_p nc) {
    if (nc != NULL && nc->conn != NULL) {
        return lwcell_conn_getnum(nc->conn);
    }
    return -1;
}

#if LWCELL_CFG_NETCONN_RECEIVE_TIMEOUT || __DOXYGEN__

/**
```

(continues on next page)

(continued from previous page)

```
 * \brief           Set timeout value for receiving data.
 *
 * When enabled, \ref lwcell_netconn_receive will only block for up to
 * \e timeout value and will return if no new data within this time
 *
 * \param[in]       nc: Netconn handle
 * \param[in]       timeout: Timeout in units of milliseconds.
 *                      Set to `0` to disable timeout feature. Function blocks until data receive or connection closed
 *                      Set to `> 0` to set maximum milliseconds to wait before timeout
 *                      Set to \ref LWCELL_NETCONN_RECEIVE_NO_WAIT to enable non-blocking receive
 */
void
lwcell_netconn_set_receive_timeout(lwcell_netconn_p nc, uint32_t timeout) {
    nc->rcv_timeout = timeout;
}

/**
 * \brief           Get netconn receive timeout value
 * \param[in]       nc: Netconn handle
 * \return          Timeout in units of milliseconds.
 *                  If value is `0`, timeout is disabled (wait forever)
 */
uint32_t
lwcell_netconn_get_receive_timeout(lwcell_netconn_p nc) {
    return nc->rcv_timeout; /* Return receive timeout */
}

#endif /* LWCELL_CFG_NETCONN_RECEIVE_TIMEOUT || __DOXYGEN__ */

#endif /* LWCELL_CFG_NETCONN || __DOXYGEN__ */
```

Connection specific event

This events are subset of global event callback. They work exactly the same way as global, but only receive events related to connections.

Tip: Connection related events start with `LWCELL_EVT_CONN_*`, such as `LWCELL_EVT_CONN_RECV`. Check api_lwcell_evt for list of all connection events.

Connection events callback function is set when client (application starts connection) starts a new connection with `lwcell_conn_start()` function

Listing 3: An example of client with its dedicated event callback function

```
#include "client.h"
#include "lwcell/lwcell.h"
#include "lwcell/lwcell_network_api.h"

```

(continues on next page)

(continued from previous page)

```
#if LWCELL_CFG_CONN

/* Host parameter */
#define CONN_HOST "example.com"
#define CONN_PORT 80

static lwcellr_t conn_callback_func(lwcell_evt_t* evt);

/**
 * \brief           Request data for connection
 */
static const uint8_t req_data[] = ""
                                  "GET / HTTP/1.1\r\n"
                                  "Host: " CONN_HOST "\r\n"
                                  "Connection: close\r\n"
                                  "\r\n";

/**
 * \brief           Start a new connection(s) as client
 */
void
client_connect(void) {
    lwcellr_t res;

    /* Attach to GSM network */
    lwcell_network_request_attach();

    /* Start a new connection as client in non-blocking mode */
    if ((res = lwcell_conn_start(NULL, LWCELL_CONN_TYPE_TCP, "example.com", 80, NULL,␣
↪conn_callback_func, 0))
        == lwcellOK) {
        printf("Connection to " CONN_HOST " started...\r\n");
    } else {
        printf("Cannot start connection to " CONN_HOST "!\r\n");
    }
}

/**
 * \brief           Event callback function for connection-only
 * \param[in]       evt: Event information with data
 * \return          \ref lwcellOK on success, member of \ref lwcellr_t otherwise
 */
static lwcellr_t
conn_callback_func(lwcell_evt_t* evt) {
    lwcell_conn_p conn;
    lwcellr_t res;
    uint8_t conn_num;

    conn = lwcell_conn_get_from_evt(evt); /* Get connection handle from event */
    if (conn == NULL) {
        return lwcellERR;
    }
```

(continues on next page)

(continued from previous page)

```
    conn_num = lwcell_conn_getnum(conn); /* Get connection number for identification */
    switch (lwcell_evt_get_type(evt)) {
        case LWCELL_EVT_CONN_ACTIVE: { /* Connection just active */
            printf("Connection %d active!\r\n", (int)conn_num);
            res = lwcell_conn_send(conn, req_data, sizeof(req_data) - 1, NULL,
                                   0); /* Start sending data in non-blocking mode */
            if (res == lwcellOK) {
                printf("Sending request data to server...\r\n");
            } else {
                printf("Cannot send request data to server. Closing connection manually..
↪.\r\n");
                lwcell_conn_close(conn, 0); /* Close the connection */
            }
            break;
        }
        case LWCELL_EVT_CONN_CLOSE: { /* Connection closed */
            if (lwcell_evt_conn_close_is_forced(evt)) {
                printf("Connection %d closed by client!\r\n", (int)conn_num);
            } else {
                printf("Connection %d closed by remote side!\r\n", (int)conn_num);
            }
            break;
        }
        case LWCELL_EVT_CONN_SEND: { /* Data send event */
            lwcellr_t res = lwcell_evt_conn_send_get_result(evt);
            if (res == lwcellOK) {
                printf("Data sent successfully on connection %d...waiting to receive␣
↪data from remote side...\r\n",
                       (int)conn_num);
            } else {
                printf("Error while sending data on connection %d!\r\n", (int)conn_num);
            }
            break;
        }
        case LWCELL_EVT_CONN_RECV: { /* Data received from remote side */
            lwcell_pbuf_p pbuf = lwcell_evt_conn_recv_get_buff(evt);
            lwcell_conn_recved(conn, pbuf); /* Notify stack about received pbuf */
            printf("Received %d bytes on connection %d..\r\n", (int)lwcell_pbuf_
↪length(pbuf, 1), (int)conn_num);
            break;
        }
        case LWCELL_EVT_CONN_ERROR: { /* Error connecting to server */
            const char* host = lwcell_evt_conn_error_get_host(evt);
            lwcell_port_t port = lwcell_evt_conn_error_get_port(evt);
            printf("Error connecting to %s:%d\r\n", host, (int)port);
            break;
        }
        default: break;
    }
    return lwcellOK;
}

```

(continues on next page)

(continued from previous page)

```
#endif /* LWCELL_CFG_CONN */
```

API call event

API function call event function is special type of event and is linked to command execution. It is especially useful when dealing with non-blocking commands to understand when specific command execution finished and when next operation could start.

Every API function, which directly operates with AT command on physical device layer, has optional 2 parameters for API call event:

- Callback function, called when command finished
- Custom user parameter for callback function

Below is an example code for SMS send. It uses custom API callback function to notify application when command has been executed successfully

Listing 4: Simple example for API call event, using DNS module

```
/* Somewhere in thread function */

/* Get device hostname in blocking mode */
/* Function returns actual result */
if (lwcell_sms_send("+0123456789", "text", NULL, NULL, 1 /* 1 means blocking call */) ==␣
↪lwcellOK) {
    /* At this point we have valid result from device */
    printf("SMS sent successfully\r\n");
} else {
    printf("Error trying to send SMS..\r\n");
}
```

5.2.5 Blocking or non-blocking API calls

API functions often allow application to set `blocking` parameter indicating if function shall be blocking or non-blocking.

Blocking mode

When the function is called in blocking mode `blocking = 1`, application thread gets suspended until response from *GSM* device is received. If there is a queue of multiple commands, thread may wait a while before receiving data.

When API function returns, application has valid response data and can react immediately.

- Linear programming model may be used
- Application may use multiple threads for real-time execution to prevent system stalling when running function call

Warning: Due to internal architecture, it is not allowed to call API functions in *blocking mode* from events or callbacks. Any attempt not to do so will result in function returning error.

Example code:

Listing 5: Blocking command example

```
/* Somewhere in thread function */

/* Get device hostname in blocking mode */
/* Function returns actual result */
if (lwcell_sms_send("+0123456789", "text", NULL, NULL, 1 /* 1 means blocking call */) ==␣
↪lwcellOK) {
    /* At this point we have valid result from device */
    printf("SMS sent successfully\r\n");
} else {
    printf("Error trying to send SMS..\r\n");
}
```

Non-blocking mode

If the API function is called in non-blocking mode, function will return immediately with status indicating if command request has been successfully sent to internal command queue. Response has to be processed in event callback function.

Warning: Due to internal architecture, it is only allowed to call API functions in *non-blocking mode* from events or callbacks. Any attempt to do so will result in function returning error.

Example code:

Listing 6: Non-blocking command example

```
/* Hostname event function, called when lwcell_sms_send() function finishes */
void
sms_send_fn(lwcellr_t res, void* arg) {
    /* Check actual result from device */
    if (res == lwcellOK) {
        printf("SMS sent successfully\r\n");
    } else {
        printf("Error trying to send SMS\r\n");
    }
}

/* Somewhere in thread and/or other GSM event function */

/* Send SMS in non-blocking mode */
/* Function now returns if command has been sent to internal message queue */
if (lwcell_sms_send("number", "text message", sms_send_fn, NULL, 0 /* 0 means non-
↪blocking call */) == lwcellOK) {
    /* At this point we only know that command has been sent to queue */
    printf("SMS send message command sent to queue.\r\n");
} else {
    /* Error writing message to queue */
    printf("Cannot send SMS send message command to queue. Maybe out of memory? Check␣
↪result from function\r\n");
}
```

Warning: When using non-blocking API calls, do not use local variables as parameter. This may introduce *undefined behavior* and *memory corruption* if application function returns before command is executed.

Example of a bad code:

Listing 7: Example of bad usage of non-blocking command

```
/* Hostname event function, called when lwcell_sms_send() function finishes */
void
sms_send_fn(lwcellr_t res, void* arg) {
    /* Check actual result from device */
    if (res == lwcellOK) {
        printf("SMS sent successfully\r\n");
    } else {
        printf("Error trying to send SMS\r\n");
    }
}

/* Check hostname */
void
check_hostname(void) {
    char message[] = "text message";

    /* Send SMS in non-blocking mode */
    /* Function now returns if command has been sent to internal message queue */
    /* It uses pointer to local data but w/o blocking command */
    if (lwcell_sms_send("number", message, sms_send_fn, NULL, 0 /* 0 means non-blocking␣
↪call */) == lwcellOK) {
        /* At this point we only know that command has been sent to queue */
        printf("SMS send message command sent to queue.\r\n");
    } else {
        /* Error writing message to queue */
        printf("Cannot send SMS send message command to queue. Maybe out of memory?␣
↪Check result from function\r\n");
    }
}
```

5.2.6 Porting guide

High level of *LwCELL* library is platform independent, written in C (C11), however there is an important part where middleware needs to communicate with target *GSM* device and it must work under different optional operating systems selected by final customer.

Porting consists of:

- Implementation of *low-level* part, for actual communication between host device and *GSM* device
- Implementation of system functions, link between target operating system and middleware functions
- Assignment of memory for allocation manager

Implement low-level driver

To successfully prepare all parts of *low-level* driver, application must take care of:

- Implementing `lwcell_ll_init()` and `lwcell_ll_deinit()` callback functions
- Implement and assign *send data* and optional *hardware reset* function callbacks
- Assign memory for allocation manager when using default allocator or use custom allocator
- Process received data from *ESP* device and send it to input module for further processing

Tip: Port examples are available for STM32 and WIN32 architectures. Both actual working and up-to-date implementations are available within the library.

Note: Check api_lwcell_input for more information about direct & indirect input processing.

Implement system functions

System functions are bridge between operating system calls and *GSM* middleware. *GSM* library relies on stable operating system features and its implementation and does not require any special features which do not normally come with operating systems.

Operating system must support:

- Thread management functions
- Mutex management functions
- Binary semaphores only functions, no need for counting semaphores
- Message queue management functions

Warning: If any of the features are not available within targeted operating system, customer needs to resolve it with care. As an example, message queue is not available in WIN32 OS API therefore custom message queue has been implemented using binary semaphores

Application needs to implement all system call functions, starting with `lwcell_sys_`. It must also prepare header file for standard types in order to support OS types within *GSM* middleware.

An example code is provided latter section of this page for WIN32 and STM32.

Note: Check *System functions* for function prototypes.

Steps to follow

- Copy `lwcell/src/system/lwcell_sys_template.c` to the same folder and rename it to application port, eg. `lwcell_sys_win32.c`
- Open newly created file and implement all system functions
- Copy folder `lwcell/src/include/system/port/template/*` to the same folder and rename *folder name* to application port, eg. `cmsis_os`
- Open `lwcell_sys_port.h` file from newly created folder and implement all *typedefs* and *macros* for specific target
- Add source file to compiler sources and add path to header file to include paths in compiler options

Note: Check *System functions* for function prototypes.

Example: Low-level driver for WIN32

Example code for low-level porting on *WIN32* platform. It uses native *Windows* features to open *COM* port and read/write from/to it.

Notes:

- It uses separate thread for received data processing. It uses `lwcell_input_process()` or `lwcell_input()` functions, based on application configuration of *LWCELL_CFG_INPUT_USE_PROCESS* parameter.
  - When *LWCELL_CFG_INPUT_USE_PROCESS* is disabled, dedicated receive buffer is created by *LwCELL* library and `lwcell_input()` function just writes data to it and does not process received characters immediately. This is handled by *Processing* thread at later stage instead.
  - When *LWCELL_CFG_INPUT_USE_PROCESS* is enabled, `lwcell_input_process()` is used, which directly processes input data and sends potential callback/event functions to application layer.
- Memory manager has been assigned to 1 region of `LWCELL_MEM_SIZE` size
- It sets *send* and *reset* callback functions for *LwCELL* library

Listing 8: Actual implementation of low-level driver for WIN32

```
/**
 * \file            lwcell_ll_win32.c
 * \brief           Low-level communication with GSM device for WIN32
 */

/*
 * 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:
 *
```

(continues on next page)

(continued from previous page)

```
 * 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 LwCELL - Lightweight cellular modem AT library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v0.1.1
 */
#include "lwcell/lwcell_input.h"
#include "lwcell/lwcell_mem.h"
#include "lwcell/lwcell_types.h"
#include "lwcell/lwcell_utils.h"
#include "system/lwcell_ll.h"
#include "system/lwcell_sys.h"
#include "windows.h"

#if !__DOXYGEN__

static uint8_t initialized = 0;
static HANDLE thread_handle;
static volatile HANDLE com_port;    /*!< COM port handle */
static uint8_t data_buffer[0x1000]; /*!< Received data array */

static void uart_thread(void* param);

/**
 * \brief           Send data to GSM device, function called from GSM stack when we have data to send
 * \param[in]       data: Pointer to data to send
 * \param[in]       len: Number of bytes to send
 * \return          Number of bytes sent
 */
static size_t
send_data(const void* data, size_t len) {
    DWORD written;
    if (com_port != NULL) {
#if !LWCELL_CFG_AT_ECHO
        const uint8_t* d = data;
        HANDLE hConsole;

        hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
        SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
        for (DWORD i = 0; i < len; ++i) {
```

(continues on next page)

(continued from previous page)

```
            printf("%c", d[i]);
        }
        SetConsoleTextAttribute(hConsole, FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_
↪BLUE);
#endif /* !LWCELL_CFG_AT_ECHO */

        /* Write data to AT port */
        WriteFile(com_port, data, len, &written, NULL);
        FlushFileBuffers(com_port);
        return written;
    }
    return 0;
}

/**
 * \brief           Configure UART (USB to UART)
 */
static uint8_t
configure_uart(uint32_t baudrate) {
    DCB dcb = {0};
    dcb.DCBlength = sizeof(dcb);

    /*
     * On first call,
     * create virtual file on selected COM port and open it
     * as generic read and write
     */
    if (!initialized) {
        static const char* com_ports[] = {"\\\\.\\COM23", "\\\\.\\COM12", "\\\\.\\COM9",
↪"\\\\.\\COM8", "\\\\.\\COM4"};
        for (size_t i = 0; i < sizeof(com_ports) / sizeof(com_ports[0]); ++i) {
            com_port = CreateFileA(com_ports[i], GENERIC_READ | GENERIC_WRITE, 0, 0,␣
↪OPEN_EXISTING, 0, NULL);
            if (GetCommState(com_port, &dcb)) {
                printf("COM PORT %s opened!\r\n", (const char*)com_ports[i]);
                break;
            }
        }
    }

    /* Configure COM port parameters */
    if (GetCommState(com_port, &dcb)) {
        COMMTIMEOUTS timeouts;

        dcb.BaudRate = baudrate;
        dcb.ByteSize = 8;
        dcb.Parity = NOPARITY;
        dcb.StopBits = ONESTOPBIT;

        if (!SetCommState(com_port, &dcb)) {
            printf("Cannot set COM PORT info\r\n");
            return 0;
```

(continues on next page)

(continued from previous page)

```
        }
        if (GetCommTimeouts(com_port, &timeouts)) {
            /* Set timeout to return immediately from ReadFile function */
            timeouts.ReadIntervalTimeout = MAXDWORD;
            timeouts.ReadTotalTimeoutConstant = 0;
            timeouts.ReadTotalTimeoutMultiplier = 0;
            if (!SetCommTimeouts(com_port, &timeouts)) {
                printf("Cannot set COM PORT timeouts\r\n");
            }
            GetCommTimeouts(com_port, &timeouts);
        } else {
            printf("Cannot get COM PORT timeouts\r\n");
            return 0;
        }
    } else {
        printf("Cannot get COM PORT info\r\n");
        return 0;
    }

    /* On first function call, create a thread to read data from COM port */
    if (!initialized) {
        lwcell_sys_thread_create(&thread_handle, "lwcell_ll_thread", uart_thread, NULL,␣
↪0, 0);
    }
    return 1;
}

/**
 * \brief           UART thread
 */
static void
uart_thread(void* param) {
    DWORD bytes_read;
    lwcell_sys_sem_t sem;
    FILE* file = NULL;

    LWCELL_UNUSED(param);

    lwcell_sys_sem_create(&sem, 0); /* Create semaphore for delay functions */

    while (com_port == NULL) {
        lwcell_sys_sem_wait(&sem, 1); /* Add some delay with yield */
    }

    fopen_s(&file, "log_file.txt", "w+"); /* Open debug file in write mode */
    while (1) {
        /*
         * Try to read data from COM port
         * and send it to upper layer for processing
         */
        do {
            ReadFile(com_port, data_buffer, sizeof(data_buffer), &bytes_read, NULL);
```

(continues on next page)

(continued from previous page)

```
            if (bytes_read > 0) {
                HANDLE hConsole;
                hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
                SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
                for (DWORD i = 0; i < bytes_read; ++i) {
                    printf("%c", data_buffer[i]);
                }
                SetConsoleTextAttribute(hConsole, FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE);

                /* Send received data to input processing module */
#if LWCELL_CFG_INPUT_USE_PROCESS
                lwcell_input_process(data_buffer, (size_t)bytes_read);
#else  /* LWCELL_CFG_INPUT_USE_PROCESS */
                lwcell_input(data_buffer, (size_t)bytes_read);
#endif /* !LWCELL_CFG_INPUT_USE_PROCESS */

                /* Write received data to output debug file */
                if (file != NULL) {
                    fwrite(data_buffer, 1, bytes_read, file);
                    fflush(file);
                }
            }
        } while (bytes_read == (DWORD)sizeof(data_buffer));

        /* Implement delay to allow other tasks processing */
        lwcell_sys_sem_wait(&sem, 1);
    }
}

/**
 * \brief           Callback function called from initialization process
 *
 * \note            This function may be called multiple times if AT baudrate is changed from application.
 *                  It is important that every configuration except AT baudrate is configured only once!
 *
 * \note            This function may be called from different threads in GSM stack when using OS.
 *                  When \ref LWCELL_CFG_INPUT_USE_PROCESS is set to 1, this function may be called from user UART thread.
 *
 * \param[in,out]   ll: Pointer to \ref lwcell_ll_t structure to fill data for communication functions
 * \return          \ref lwcellOK on success, member of \ref lwcellr_t enumeration otherwise
 */
lwcellr_t
lwcell_ll_init(lwcell_ll_t* ll) {
#if !LWCELL_CFG_MEM_CUSTOM
    /* Step 1: Configure memory for dynamic allocations */
```

(continues on next page)

(continued from previous page)

```
    static uint8_t memory[0x10000]; /* Create memory for dynamic allocations with␣
↪specific size */

    /*
     * Create memory region(s) of memory.
     * If device has internal/external memory available,
     * multiple memories may be used
     */
    lwcell_mem_region_t mem_regions[] = {{memory, sizeof(memory)}};
    if (!initialized) {
        lwcell_mem_assignmemory(mem_regions,
                                LWCELL_ARRAYSIZE(mem_regions)); /* Assign memory for␣
↪allocations to GSM library */
    }
#endif /* !LWCELL_CFG_MEM_CUSTOM */

    /* Step 2: Set AT port send function to use when we have data to transmit */
    if (!initialized) {
        ll->send_fn = send_data; /* Set callback function to send data */
    }

    /* Step 3: Configure AT port to be able to send/receive data to/from GSM device */
    if (!configure_uart(ll->uart.baudrate)) { /* Initialize UART for communication */
        return lwcellERR;
    }
    initialized = 1;
    return lwcellOK;
}

#endif /* !__DOXYGEN__ */
```

Example: Low-level driver for STM32

Example code for low-level porting on *STM32* platform. It uses *CMSIS-OS* based application layer functions for implementing threads & other OS dependent features.

Notes:

- It uses separate thread for received data processing. It uses `lwcell_input_process()` function to directly process received data without using intermediate receive buffer
- Memory manager has been assigned to 1 region of `LWCELL_MEM_SIZE` size
- It sets *send* and *reset* callback functions for *LwCELL* library

Listing 9: Actual implementation of low-level driver for STM32

```
/**
 * \file            lwcell_ll_stm32.c
 * \brief           Generic STM32 driver, included in various STM32 driver variants
 */

/*
```

(continues on next page)

(continued from previous page)

```
 * 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 LwCELL - Lightweight cellular modem AT library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v0.1.1
 */

/*
 * How it works
 *
 * On first call to \ref lwcell_ll_init, new thread is created and processed in usart_ll_thread function.
 * USART is configured in RX DMA mode and any incoming bytes are processed inside thread function.
 * DMA and USART implement interrupt handlers to notify main thread about new data ready to send to upper layer.
 *
 * More about UART + RX DMA: https://github.com/MaJerle/stm32-usart-dma-rx-tx
 *
 * \ref LWCELL_CFG_INPUT_USE_PROCESS must be enabled in `lwcell_config.h` to use this driver.
 */
#include "lwcell/lwcell_input.h"
#include "lwcell/lwcell_mem.h"
#include "lwcell/lwcell_types.h"
#include "lwcell/lwcell_utils.h"
#include "system/lwcell_ll.h"
#include "system/lwcell_sys.h"

#if !__DOXYGEN__

```

(continues on next page)

(continued from previous page)

```
#if !LWCELL_CFG_INPUT_USE_PROCESS
#error "LWCELL_CFG_INPUT_USE_PROCESS must be enabled in `lwcell_config.h` to use this driver."
#endif /* LWCELL_CFG_INPUT_USE_PROCESS */

#if !defined(LWCELL_USART_DMA_RX_BUFF_SIZE)
#define LWCELL_USART_DMA_RX_BUFF_SIZE 0x1000
#endif /* !defined(LWCELL_USART_DMA_RX_BUFF_SIZE) */

#if !defined(LWCELL_MEM_SIZE)
#define LWCELL_MEM_SIZE 0x1000
#endif /* !defined(LWCELL_MEM_SIZE) */

#if !defined(LWCELL_USART_RDR_NAME)
#define LWCELL_USART_RDR_NAME RDR
#endif /* !defined(LWCELL_USART_RDR_NAME) */

/* USART memory */
static uint8_t usart_mem[LWCELL_USART_DMA_RX_BUFF_SIZE];
static uint8_t is_running, initialized;
static size_t old_pos;

/* USART thread */
static void usart_ll_thread(void* arg);
static osThreadId_t usart_ll_thread_id;

/* Message queue */
static osMessageQueueId_t usart_ll_mbox_id;

/**
 * \brief           USART data processing
 */
static void
usart_ll_thread(void* arg) {
    size_t pos;

    LWCELL_UNUSED(arg);

    while (1) {
        void* d;
        /* Wait for the event message from DMA or USART */
        osMessageQueueGet(usart_ll_mbox_id, &d, NULL, osWaitForever);

        /* Read data */
#if defined(LWCELL_USART_DMA_RX_STREAM)
        pos = sizeof(usart_mem) - LL_DMA_GetDataLength(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM);
#else
        pos = sizeof(usart_mem) - LL_DMA_GetDataLength(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_CH);
#endif /* defined(LWCELL_USART_DMA_RX_STREAM) */
        if (pos != old_pos && is_running) {
```

(continues on next page)

(continued from previous page)

```
            if (pos > old_pos) {
                lwcell_input_process(&usart_mem[old_pos], pos - old_pos);
            } else {
                lwcell_input_process(&usart_mem[old_pos], sizeof(usart_mem) - old_pos);
                if (pos > 0) {
                    lwcell_input_process(&usart_mem[0], pos);
                }
            }
            old_pos = pos;
            if (old_pos == sizeof(usart_mem)) {
                old_pos = 0;
            }
        }
    }
}

/**
 * \brief           Configure UART using DMA for receive in double buffer mode and IDLE␣
↪line detection
 */
static void
configure_uart(uint32_t baudrate) {
    static LL_USART_InitTypeDef usart_init;
    static LL_DMA_InitTypeDef dma_init;
    LL_GPIO_InitTypeDef gpio_init;

    if (!initialized) {
        /* Enable peripheral clocks */
        LWCELL_USART_CLK;
        LWCELL_USART_DMA_CLK;
        LWCELL_USART_TX_PORT_CLK;
        LWCELL_USART_RX_PORT_CLK;

#if defined(LWCELL_RESET_PIN)
        LWCELL_RESET_PORT_CLK;
#endif /* defined(LWCELL_RESET_PIN) */

        /* Global pin configuration */
        LL_GPIO_StructInit(&gpio_init);
        gpio_init.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
        gpio_init.Pull = LL_GPIO_PULL_UP;
        gpio_init.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
        gpio_init.Mode = LL_GPIO_MODE_OUTPUT;

#if defined(LWCELL_RESET_PIN)
        /* Configure RESET pin */
        gpio_init.Pin = LWCELL_RESET_PIN;
        LL_GPIO_Init(LWCELL_RESET_PORT, &gpio_init);
#endif /* defined(LWCELL_RESET_PIN) */

        /* Configure USART pins */
        gpio_init.Mode = LL_GPIO_MODE_ALTERNATE;
```

(continues on next page)

(continued from previous page)

```

        /* TX PIN */
        gpio_init.Alternate = LWCELL_USART_TX_PIN_AF;
        gpio_init.Pin = LWCELL_USART_TX_PIN;
        LL_GPIO_Init(LWCELL_USART_TX_PORT, &gpio_init);

        /* RX PIN */
        gpio_init.Alternate = LWCELL_USART_RX_PIN_AF;
        gpio_init.Pin = LWCELL_USART_RX_PIN;
        LL_GPIO_Init(LWCELL_USART_RX_PORT, &gpio_init);

        /* Configure UART */
        LL_USART_DeInit(LWCELL_USART);
        LL_USART_StructInit(&usart_init);
        usart_init.BaudRate = baudrate;
        usart_init.DataWidth = LL_USART_DATAWIDTH_8B;
        usart_init.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
        usart_init.OverSampling = LL_USART_OVERSAMPLING_16;
        usart_init.Parity = LL_USART_PARITY_NONE;
        usart_init.StopBits = LL_USART_STOPBITS_1;
        usart_init.TransferDirection = LL_USART_DIRECTION_TX_RX;
        LL_USART_Init(LWCELL_USART, &usart_init);

        /* Enable USART interrupts and DMA request */
        LL_USART_EnableIT_IDLE(LWCELL_USART);
        LL_USART_EnableIT_PE(LWCELL_USART);
        LL_USART_EnableIT_ERROR(LWCELL_USART);
        LL_USART_EnableDMAReq_RX(LWCELL_USART);

        /* Enable USART interrupts */
        NVIC_SetPriority(LWCELL_USART_IRQ, NVIC_EncodePriority(NVIC_
↪GetPriorityGrouping(), 0x07, 0x00));
        NVIC_EnableIRQ(LWCELL_USART_IRQ);

        /* Configure DMA */
        is_running = 0;
#if defined(LWCELL_USART_DMA_RX_STREAM)
        LL_DMA_DeInit(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM);
        dma_init.Channel = LWCELL_USART_DMA_RX_CH;
#else
        LL_DMA_DeInit(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_CH);
        dma_init.PeriphRequest = LWCELL_USART_DMA_RX_REQ_NUM;
#endif /* defined(LWCELL_USART_DMA_RX_STREAM) */
        dma_init.PeriphOrM2MSrcAddress = (uint32_t)&LWCELL_USART->LWCELL_USART_RDR_NAME;
        dma_init.MemoryOrM2MDstAddress = (uint32_t)usart_mem;
        dma_init.Direction = LL_DMA_DIRECTION_PERIPH_TO_MEMORY;
        dma_init.Mode = LL_DMA_MODE_CIRCULAR;
        dma_init.PeriphOrM2MSrcIncMode = LL_DMA_PERIPH_NOINCREMENT;
        dma_init.MemoryOrM2MDstIncMode = LL_DMA_MEMORY_INCREMENT;
        dma_init.PeriphOrM2MSrcDataSize = LL_DMA_PDATAALIGN_BYTE;
        dma_init.MemoryOrM2MDstDataSize = LL_DMA_MDATAALIGN_BYTE;
        dma_init.NbData = sizeof(usart_mem);
```

(continues on next page)

(continued from previous page)

```
        dma_init.Priority = LL_DMA_PRIORITY_MEDIUM;
#if defined(LWCELL_USART_DMA_RX_STREAM)
        LL_DMA_Init(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM, &dma_init);
#else
        LL_DMA_Init(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_CH, &dma_init);
#endif /* defined(LWCELL_USART_DMA_RX_STREAM) */

        /* Enable DMA interrupts */
#if defined(LWCELL_USART_DMA_RX_STREAM)
        LL_DMA_EnableIT_HT(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM);
        LL_DMA_EnableIT_TC(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM);
        LL_DMA_EnableIT_TE(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM);
        LL_DMA_EnableIT_FE(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM);
        LL_DMA_EnableIT_DME(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM);
#else
        LL_DMA_EnableIT_HT(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_CH);
        LL_DMA_EnableIT_TC(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_CH);
        LL_DMA_EnableIT_TE(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_CH);
#endif /* defined(LWCELL_USART_DMA_RX_STREAM) */

        /* Enable DMA interrupts */
        NVIC_SetPriority(LWCELL_USART_DMA_RX_IRQ, NVIC_EncodePriority(NVIC_
↪GetPriorityGrouping(), 0x07, 0x00));
        NVIC_EnableIRQ(LWCELL_USART_DMA_RX_IRQ);

        old_pos = 0;
        is_running = 1;

        /* Start DMA and USART */
#if defined(LWCELL_USART_DMA_RX_STREAM)
        LL_DMA_EnableStream(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM);
#else
        LL_DMA_EnableChannel(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_CH);
#endif /* defined(LWCELL_USART_DMA_RX_STREAM) */
        LL_USART_Enable(LWCELL_USART);
    } else {
        osDelay(10);
        LL_USART_Disable(LWCELL_USART);
        usart_init.BaudRate = baudrate;
        LL_USART_Init(LWCELL_USART, &usart_init);
        LL_USART_Enable(LWCELL_USART);
    }

    /* Create mbox and start thread */
    if (usart_ll_mbox_id == NULL) {
        usart_ll_mbox_id = osMessageQueueNew(10, sizeof(void*), NULL);
    }
    if (usart_ll_thread_id == NULL) {
        const osThreadAttr_t attr = {.stack_size = 1024};
        usart_ll_thread_id = osThreadNew(usart_ll_thread, usart_ll_mbox_id, &attr);
    }
}
```

(continues on next page)

(continued from previous page)

```

#if defined(LWCELL_RESET_PIN)
/**
 * \brief           Hardware reset callback
 */
static uint8_t
reset_device(uint8_t state) {
    if (state) { /* Activate reset line */
        LL_GPIO_ResetOutputPin(LWCELL_RESET_PORT, LWCELL_RESET_PIN);
    } else {
        LL_GPIO_SetOutputPin(LWCELL_RESET_PORT, LWCELL_RESET_PIN);
    }
    return 1;
}
#endif /* defined(LWCELL_RESET_PIN) */

/**
 * \brief           Send data to GSM device
 * \param[in]       data: Pointer to data to send
 * \param[in]       len: Number of bytes to send
 * \return          Number of bytes sent
 */
static size_t
send_data(const void* data, size_t len) {
    const uint8_t* d = data;

    for (size_t i = 0; i < len; ++i, ++d) {
        LL_USART_TransmitData8(LWCELL_USART, *d);
        while (!LL_USART_IsActiveFlag_TXE(LWCELL_USART)) {}
    }
    return len;
}

/**
 * \brief           Callback function called from initialization process
 * \note            This function may be called multiple times if AT baudrate is changed from application
 * \param[in,out]   ll: Pointer to \ref lwcell_ll_t structure to fill data for communication functions
 * \param[in]       baudrate: Baudrate to use on AT port
 * \return          Member of \ref lwcellr_t enumeration
 */
lwcellr_t
lwcell_ll_init(lwcell_ll_t* ll) {
#if !LWCELL_CFG_MEM_CUSTOM
    static uint8_t memory[LWCELL_MEM_SIZE];
    lwcell_mem_region_t mem_regions[] = {{memory, sizeof(memory)}};

    if (!initialized) {
        lwcell_mem_assignmemory(mem_regions, LWCELL_ARRAYSIZE(mem_regions)); /* Assign memory for allocations */
    }
```

(continues on next page)

(continued from previous page)

```
#endif /* !LWCELL_CFG_MEM_CUSTOM */

    if (!initialized) {
        ll->send_fn = send_data; /* Set callback function to send data */
#if defined(LWCELL_RESET_PIN)
        ll->reset_fn = reset_device; /* Set callback for hardware reset */
#endif                               /* defined(LWCELL_RESET_PIN) */
    }

    configure_uart(ll->uart.baudrate); /* Initialize UART for communication */
    initialized = 1;
    return lwcellOK;
}

/**
 * \brief           Callback function to de-init low-level communication part
 * \param[in,out]   ll: Pointer to \ref lwcell_ll_t structure to fill data for communication functions
 * \return          \ref lwcellOK on success, member of \ref lwcellr_t enumeration otherwise
 */
lwcellr_t
lwcell_ll_deinit(lwcell_ll_t* ll) {
    if (usart_ll_mbox_id != NULL) {
        osMessageQueueId_t tmp = usart_ll_mbox_id;
        usart_ll_mbox_id = NULL;
        osMessageQueueDelete(tmp);
    }
    if (usart_ll_thread_id != NULL) {
        osThreadId_t tmp = usart_ll_thread_id;
        usart_ll_thread_id = NULL;
        osThreadTerminate(tmp);
    }
    initialized = 0;
    LWCELL_UNUSED(ll);
    return lwcellOK;
}

/**
 * \brief           UART global interrupt handler
 */
void
LWCELL_USART_IRQHANDLER(void) {
    LL_USART_ClearFlag_IDLE(LWCELL_USART);
    LL_USART_ClearFlag_PE(LWCELL_USART);
    LL_USART_ClearFlag_FE(LWCELL_USART);
    LL_USART_ClearFlag_ORE(LWCELL_USART);
    LL_USART_ClearFlag_NE(LWCELL_USART);

    if (usart_ll_mbox_id != NULL) {
        void* d = (void*)1;
        osMessageQueuePut(usart_ll_mbox_id, &d, 0, 0);
```

(continues on next page)

(continued from previous page)

```
    }
}

/**
 * \brief           UART DMA stream/channel handler
 */
void
LWCELL_USART_DMA_RX_IRQHANDLER(void) {
    LWCELL_USART_DMA_RX_CLEAR_TC;
    LWCELL_USART_DMA_RX_CLEAR_HT;

    if (usart_ll_mbox_id != NULL) {
        void* d = (void*)1;
        osMessageQueuePut(usart_ll_mbox_id, &d, 0, 0);
    }
}

#endif /* !__DOXYGEN__ */
```

Example: System functions for WIN32

Listing 10: Actual header implementation of system functions for WIN32

```
/**
 * \file            lwcell_sys_port.h
 * \brief           WIN32 based system file implementation
 */

/*
 * 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.
 *
```

(continues on next page)

(continued from previous page)

```
 * This file is part of LwCELL - Lightweight cellular modem AT library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v0.1.1
 */
#ifndef LWCELL_SYSTEM_PORT_HDR_H
#define LWCELL_SYSTEM_PORT_HDR_H

#include <stdint.h>
#include <stdlib.h>
#include "lwcell/lwcell_opt.h"
#include "windows.h"

#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

#if LWCELL_CFG_OS && !__DOXYGEN__

typedef HANDLE lwcell_sys_mutex_t;
typedef HANDLE lwcell_sys_sem_t;
typedef HANDLE lwcell_sys_mbox_t;
typedef HANDLE lwcell_sys_thread_t;
typedef int lwcell_sys_thread_prio_t;

#define LWCELL_SYS_MUTEX_NULL  ((HANDLE)0)
#define LWCELL_SYS_SEM_NULL    ((HANDLE)0)
#define LWCELL_SYS_MBOX_NULL   ((HANDLE)0)
#define LWCELL_SYS_TIMEOUT     (INFINITE)
#define LWCELL_SYS_THREAD_PRIO (0)
#define LWCELL_SYS_THREAD_SS   (4096)

#endif /* LWCELL_CFG_OS && !__DOXYGEN__ */

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif /* LWCELL_SYSTEM_PORT_HDR_H */
```

Listing 11: Actual implementation of system functions for WIN32

```
/**
 * \file            lwcell_sys_win32.c
 * \brief           System dependant functions for WIN32
 */

/*
 * Copyright (c) 2024 Tilen MAJERLE
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
```

(continues on next page)

(continued from previous page)

```
 * 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 LwCELL - Lightweight cellular modem AT library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v0.1.1
 */
#include <stdlib.h>
#include <string.h>
#include "lwcell/lwcell_private.h"
#include "system/lwcell_sys.h"

#if !__DOXYGEN__

/**
 * \brief           Custom message queue implementation for WIN32
 */
typedef struct {
    lwcell_sys_sem_t sem_not_empty; /*!< Semaphore indicates not empty */
    lwcell_sys_sem_t sem_not_full;  /*!< Semaphore indicates not full */
    lwcell_sys_sem_t sem;           /*!< Semaphore to lock access */
    size_t in, out, size;
    void* entries[1];
} win32_mbox_t;

static LARGE_INTEGER freq, sys_start_time;
static lwcell_sys_mutex_t sys_mutex; /* Mutex ID for main protection */

static uint8_t
mbox_is_full(win32_mbox_t* m) {
    size_t size = 0;
    if (m->in > m->out) {
        size = (m->in - m->out);
    } else if (m->out > m->in) {
        size = m->size - m->out + m->in;
    }
```

(continues on next page)

(continued from previous page)

```
    return size == m->size - 1;
}

static uint8_t
mbox_is_empty(win32_mbox_t* m) {
    return m->in == m->out;
}

static uint32_t
osKernelSysTick(void) {
    LONGLONG ret;
    LARGE_INTEGER now;

    QueryPerformanceFrequency(&freq); /* Get frequency */
    QueryPerformanceCounter(&now);    /* Get current time */
    ret = now.QuadPart - sys_start_time.QuadPart;
    return (uint32_t)(((ret)*1000) / freq.QuadPart);
}

uint8_t
lwcell_sys_init(void) {
    QueryPerformanceFrequency(&freq);
    QueryPerformanceCounter(&sys_start_time);

    lwcell_sys_mutex_create(&sys_mutex);
    return 1;
}

uint32_t
lwcell_sys_now(void) {
    return osKernelSysTick();
}

uint8_t
lwcell_sys_protect(void) {
    lwcell_sys_mutex_lock(&sys_mutex);
    return 1;
}

uint8_t
lwcell_sys_unprotect(void) {
    lwcell_sys_mutex_unlock(&sys_mutex);
    return 1;
}

uint8_t
lwcell_sys_mutex_create(lwcell_sys_mutex_t* p) {
    *p = CreateMutex(NULL, FALSE, NULL);
    return *p != NULL;
}

uint8_t
```

(continues on next page)

(continued from previous page)

```
lwcell_sys_mutex_delete(lwcell_sys_mutex_t* p) {
    return CloseHandle(*p);
}

uint8_t
lwcell_sys_mutex_lock(lwcell_sys_mutex_t* p) {
    DWORD ret;
    ret = WaitForSingleObject(*p, INFINITE);
    if (ret != WAIT_OBJECT_0) {
        return 0;
    }
    return 1;
}

uint8_t
lwcell_sys_mutex_unlock(lwcell_sys_mutex_t* p) {
    return (uint8_t)ReleaseMutex(*p);
}

uint8_t
lwcell_sys_mutex_isvalid(lwcell_sys_mutex_t* p) {
    return p != NULL && *p != NULL;
}

uint8_t
lwcell_sys_mutex_invalid(lwcell_sys_mutex_t* p) {
    *p = LWCELL_SYS_MUTEX_NULL;
    return 1;
}

uint8_t
lwcell_sys_sem_create(lwcell_sys_sem_t* p, uint8_t cnt) {
    HANDLE h;
    h = CreateSemaphore(NULL, !!cnt, 1, NULL);
    *p = h;
    return *p != NULL;
}

uint8_t
lwcell_sys_sem_delete(lwcell_sys_sem_t* p) {
    return CloseHandle(*p);
}

uint32_t
lwcell_sys_sem_wait(lwcell_sys_sem_t* p, uint32_t timeout) {
    DWORD ret;

    if (timeout == 0) {
        ret = WaitForSingleObject(*p, INFINITE);
        return 1;
    } else {
        ret = WaitForSingleObject(*p, timeout);
```

(continues on next page)

(continued from previous page)

```
        if (ret == WAIT_OBJECT_0) {
            return 1;
        } else {
            return LWCELL_SYS_TIMEOUT;
        }
    }
}

uint8_t
lwcell_sys_sem_release(lwcell_sys_sem_t* p) {
    return ReleaseSemaphore(*p, 1, NULL);
}

uint8_t
lwcell_sys_sem_isvalid(lwcell_sys_sem_t* p) {
    return p != NULL && *p != NULL;
}

uint8_t
lwcell_sys_sem_invalid(lwcell_sys_sem_t* p) {
    *p = LWCELL_SYS_SEM_NULL;
    return 1;
}

uint8_t
lwcell_sys_mbox_create(lwcell_sys_mbox_t* b, size_t size) {
    win32_mbox_t* mbox;

    *b = NULL;

    mbox = malloc(sizeof(*mbox) + size * sizeof(void*));
    if (mbox != NULL) {
        memset(mbox, 0x00, sizeof(*mbox));
        mbox->size = size + 1; /* Set it to 1 more as cyclic buffer has only one less␣
↪than size */
        lwcell_sys_sem_create(&mbox->sem, 1);
        lwcell_sys_sem_create(&mbox->sem_not_empty, 0);
        lwcell_sys_sem_create(&mbox->sem_not_full, 0);
        *b = mbox;
    }
    return *b != NULL;
}

uint8_t
lwcell_sys_mbox_delete(lwcell_sys_mbox_t* b) {
    win32_mbox_t* mbox = *b;
    lwcell_sys_sem_delete(&mbox->sem);
    lwcell_sys_sem_delete(&mbox->sem_not_full);
    lwcell_sys_sem_delete(&mbox->sem_not_empty);
    free(mbox);
    return 1;
}
```

(continues on next page)

(continued from previous page)

```

uint32_t
lwcell_sys_mbox_put(lwcell_sys_mbox_t* b, void* m) {
    win32_mbox_t* mbox = *b;
    uint32_t time = osKernelSysTick(); /* Get start time */

    lwcell_sys_sem_wait(&mbox->sem, 0); /* Wait for access */

    /*
     * Since function is blocking until ready to write something to queue,
     * wait and release the semaphores to allow other threads
     * to process the queue before we can write new value.
     */
    while (mbox_is_full(mbox)) {
        lwcell_sys_sem_release(&mbox->sem);          /* Release semaphore */
        lwcell_sys_sem_wait(&mbox->sem_not_full, 0); /* Wait for semaphore indicating
↪not full */
        lwcell_sys_sem_wait(&mbox->sem, 0);          /* Wait availability again */
    }
    mbox->entries[mbox->in] = m;
    if (++mbox->in >= mbox->size) {
        mbox->in = 0;
    }
    lwcell_sys_sem_release(&mbox->sem_not_empty); /* Signal non-empty state */
    lwcell_sys_sem_release(&mbox->sem);           /* Release access for other threads */
    return osKernelSysTick() - time;
}

uint32_t
lwcell_sys_mbox_get(lwcell_sys_mbox_t* b, void** m, uint32_t timeout) {
    win32_mbox_t* mbox = *b;
    uint32_t time;

    time = osKernelSysTick();

    /* Get exclusive access to message queue */
    if (lwcell_sys_sem_wait(&mbox->sem, timeout) == LWCELL_SYS_TIMEOUT) {
        return LWCELL_SYS_TIMEOUT;
    }
    while (mbox_is_empty(mbox)) {
        lwcell_sys_sem_release(&mbox->sem);
        if (lwcell_sys_sem_wait(&mbox->sem_not_empty, timeout) == LWCELL_SYS_TIMEOUT) {
            return LWCELL_SYS_TIMEOUT;
        }
        lwcell_sys_sem_wait(&mbox->sem, timeout);
    }
    *m = mbox->entries[mbox->out];
    if (++mbox->out >= mbox->size) {
        mbox->out = 0;
    }
    lwcell_sys_sem_release(&mbox->sem_not_full);
    lwcell_sys_sem_release(&mbox->sem);
```

(continues on next page)

(continued from previous page)

```
    return osKernelSysTick() - time;
}

uint8_t
lwcell_sys_mbox_putnow(lwcell_sys_mbox_t* b, void* m) {
    win32_mbox_t* mbox = *b;

    lwcell_sys_sem_wait(&mbox->sem, 0);
    if (mbox_is_full(mbox)) {
        lwcell_sys_sem_release(&mbox->sem);
        return 0;
    }
    mbox->entries[mbox->in] = m;
    if (mbox->in == mbox->out) {
        lwcell_sys_sem_release(&mbox->sem_not_empty);
    }
    if (++mbox->in >= mbox->size) {
        mbox->in = 0;
    }
    lwcell_sys_sem_release(&mbox->sem);
    return 1;
}

uint8_t
lwcell_sys_mbox_getnow(lwcell_sys_mbox_t* b, void** m) {
    win32_mbox_t* mbox = *b;

    lwcell_sys_sem_wait(&mbox->sem, 0); /* Wait exclusive access */
    if (mbox->in == mbox->out) {
        lwcell_sys_sem_release(&mbox->sem); /* Release access */
        return 0;
    }

    *m = mbox->entries[mbox->out];
    if (++mbox->out >= mbox->size) {
        mbox->out = 0;
    }
    lwcell_sys_sem_release(&mbox->sem_not_full); /* Queue not full anymore */
    lwcell_sys_sem_release(&mbox->sem);          /* Release semaphore */
    return 1;
}

uint8_t
lwcell_sys_mbox_isvalid(lwcell_sys_mbox_t* b) {
    return b != NULL && *b != NULL; /* Return status if message box is valid */
}

uint8_t
lwcell_sys_mbox_invalid(lwcell_sys_mbox_t* b) {
    *b = LWCELL_SYS_MBOX_NULL; /* Invalidate message box */
    return 1;
```

(continues on next page)

(continued from previous page)

```
}

uint8_t
lwcell_sys_thread_create(lwcell_sys_thread_t* t, const char* name, lwcell_sys_thread_fn␣
↪thread_func, void* const arg,
                         size_t stack_size, lwcell_sys_thread_prio_t prio) {
    HANDLE h;
    DWORD id;

    LWCELL_UNUSED(name);
    LWCELL_UNUSED(stack_size);
    LWCELL_UNUSED(prio);

    h = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)thread_func, arg, 0, &id);
    if (t != NULL) {
        *t = h;
    }
    return h != NULL;
}

uint8_t
lwcell_sys_thread_terminate(lwcell_sys_thread_t* t) {
    if (t == NULL) { /* Shall we terminate ourself? */
        ExitThread(0);
    } else {
        /* We have known thread, find handle by looking at ID */
        TerminateThread(*t, 0);
    }
    return 1;
}

uint8_t
lwcell_sys_thread_yield(void) {
    /* Not implemented */
    return 1;
}

#endif /* !__DOXYGEN__ */
```

Example: System functions for CMSIS-OS

Listing 12: Actual header implementation of system functions for CMSIS-OS based operating systems

```
/**
 * \file            lwcell_sys_port.h
 * \brief           System dependent functions for CMSIS-OS based operating system
 */

/*
 * Copyright (c) 2024 Tilen MAJERLE
```

(continues on next page)

(continued from previous page)

```
 *
 * 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 LwCELL - Lightweight cellular modem AT library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v0.1.1
 */
#ifndef LWCELL_SYSTEM_PORT_HDR_H
#define LWCELL_SYSTEM_PORT_HDR_H

#include <stdint.h>
#include <stdlib.h>
#include "cmsis_os.h"
#include "lwcell/lwcell_opt.h"

#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

#if LWCELL_CFG_OS && !__DOXYGEN__

typedef osMutexId_t lwcell_sys_mutex_t;
typedef osSemaphoreId_t lwcell_sys_sem_t;
typedef osMessageQueueId_t lwcell_sys_mbox_t;
typedef osThreadId_t lwcell_sys_thread_t;
typedef osPriority_t lwcell_sys_thread_prio_t;

#define LWCELL_SYS_MUTEX_NULL  ((lwcell_sys_mutex_t)0)
#define LWCELL_SYS_SEM_NULL    ((lwcell_sys_sem_t)0)
#define LWCELL_SYS_MBOX_NULL   ((lwcell_sys_mbox_t)0)
#define LWCELL_SYS_TIMEOUT     ((uint32_t)osWaitForever)
#define LWCELL_SYS_THREAD_PRIO (osPriorityNormal)
#define LWCELL_SYS_THREAD_SS   (512)
```

(continues on next page)

(continued from previous page)

```

#endif /* LWCELL_CFG_OS && !__DOXYGEN__ */

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif /* LWCELL_SYSTEM_PORT_HDR_H */
```

Listing 13: Actual implementation of system functions for CMSIS-OS based operating systems

```
/**
 * \file            lwcell_sys_cmsis_os.c
 * \brief           System dependent functions for CMSIS-OS based operating system
 */

/*
 * 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 LwCELL - Lightweight cellular modem AT library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v0.1.1
 */
#include "cmsis_os.h"
#include "system/lwcell_sys.h"

#if !__DOXYGEN__

static osMutexId_t sys_mutex;

```

(continues on next page)

(continued from previous page)

```
uint8_t
lwcell_sys_init(void) {
    lwcell_sys_mutex_create(&sys_mutex);
    return 1;
}

uint32_t
lwcell_sys_now(void) {
    return osKernelGetTickCount();
}

uint8_t
lwcell_sys_protect(void) {
    lwcell_sys_mutex_lock(&sys_mutex);
    return 1;
}

uint8_t
lwcell_sys_unprotect(void) {
    lwcell_sys_mutex_unlock(&sys_mutex);
    return 1;
}

uint8_t
lwcell_sys_mutex_create(lwcell_sys_mutex_t* p) {
    const osMutexAttr_t attr = {
        .attr_bits = osMutexRecursive,
        .name = "lwcell_mutex",
    };
    return (*p = osMutexNew(&attr)) != NULL;
}

uint8_t
lwcell_sys_mutex_delete(lwcell_sys_mutex_t* p) {
    return osMutexDelete(*p) == osOK;
}

uint8_t
lwcell_sys_mutex_lock(lwcell_sys_mutex_t* p) {
    return osMutexAcquire(*p, osWaitForever) == osOK;
}

uint8_t
lwcell_sys_mutex_unlock(lwcell_sys_mutex_t* p) {
    return osMutexRelease(*p) == osOK;
}

uint8_t
lwcell_sys_mutex_isvalid(lwcell_sys_mutex_t* p) {
    return p != NULL && *p != NULL;
}

```

(continues on next page)

(continued from previous page)

```
uint8_t
lwcell_sys_mutex_invalid(lwcell_sys_mutex_t* p) {
    *p = LWCELL_SYS_MUTEX_NULL;
    return 1;
}

uint8_t
lwcell_sys_sem_create(lwcell_sys_sem_t* p, uint8_t cnt) {
    const osSemaphoreAttr_t attr = {
        .name = "lwcell_sem",
    };
    return (*p = osSemaphoreNew(1, cnt > 0 ? 1 : 0, &attr)) != NULL;
}

uint8_t
lwcell_sys_sem_delete(lwcell_sys_sem_t* p) {
    return osSemaphoreDelete(*p) == osOK;
}

uint32_t
lwcell_sys_sem_wait(lwcell_sys_sem_t* p, uint32_t timeout) {
    uint32_t tick = osKernelSysTick();
    return (osSemaphoreAcquire(*p, timeout == 0 ? osWaitForever : timeout) == osOK) ?␣
↪(osKernelSysTick() - tick)
                                                                                    :␣
↪LWCELL_SYS_TIMEOUT;
}

uint8_t
lwcell_sys_sem_release(lwcell_sys_sem_t* p) {
    return osSemaphoreRelease(*p) == osOK;
}

uint8_t
lwcell_sys_sem_isvalid(lwcell_sys_sem_t* p) {
    return p != NULL && *p != NULL;
}

uint8_t
lwcell_sys_sem_invalid(lwcell_sys_sem_t* p) {
    *p = LWCELL_SYS_SEM_NULL;
    return 1;
}

uint8_t
lwcell_sys_mbox_create(lwcell_sys_mbox_t* b, size_t size) {
    const osMessageQueueAttr_t attr = {
        .name = "lwcell_mbox",
    };
    return (*b = osMessageQueueNew(size, sizeof(void*), &attr)) != NULL;
}

```

(continues on next page)

(continued from previous page)

```
uint8_t
lwcell_sys_mbox_delete(lwcell_sys_mbox_t* b) {
    if (osMessageQueueGetCount(*b) > 0) {
        return 0;
    }
    return osMessageQueueDelete(*b) == osOK;
}

uint32_t
lwcell_sys_mbox_put(lwcell_sys_mbox_t* b, void* m) {
    uint32_t tick = osKernelSysTick();
    return osMessageQueuePut(*b, &m, 0, osWaitForever) == osOK ? (osKernelSysTick() - tick) : LWCELL_SYS_TIMEOUT;
}

uint32_t
lwcell_sys_mbox_get(lwcell_sys_mbox_t* b, void** m, uint32_t timeout) {
    uint32_t tick = osKernelSysTick();
    return osMessageQueueGet(*b, m, NULL, timeout == 0 ? osWaitForever : timeout) == osOK ? (osKernelSysTick() - tick)
                                                                                          : LWCELL_SYS_TIMEOUT;
}

uint8_t
lwcell_sys_mbox_putnow(lwcell_sys_mbox_t* b, void* m) {
    return osMessageQueuePut(*b, &m, 0, 0) == osOK;
}

uint8_t
lwcell_sys_mbox_getnow(lwcell_sys_mbox_t* b, void** m) {
    return osMessageQueueGet(*b, m, NULL, 0) == osOK;
}

uint8_t
lwcell_sys_mbox_isvalid(lwcell_sys_mbox_t* b) {
    return b != NULL && *b != NULL;
}

uint8_t
lwcell_sys_mbox_invalid(lwcell_sys_mbox_t* b) {
    *b = LWCELL_SYS_MBOX_NULL;
    return 1;
}

uint8_t
lwcell_sys_thread_create(lwcell_sys_thread_t* t, const char* name, lwcell_sys_thread_fn thread_func, void* const arg,
                         size_t stack_size, lwcell_sys_thread_prio_t prio) {
    lwcell_sys_thread_t id;
    const osThreadAttr_t thread_attr = {.name = (char*)name,
                                        .priority = (osPriority)prio,
```

(continues on next page)

(continued from previous page)

```
                                        .stack_size = stack_size > 0 ? stack_size :␣
↪LWCELL_SYS_THREAD_SS};

    id = osThreadNew(thread_func, arg, &thread_attr);
    if (t != NULL) {
        *t = id;
    }
    return id != NULL;
}

uint8_t
lwcell_sys_thread_terminate(lwcell_sys_thread_t* t) {
    if (t != NULL) {
        osThreadTerminate(*t);
    } else {
        osThreadExit();
    }
    return 1;
}

uint8_t
lwcell_sys_thread_yield(void) {
    osThreadYield();
    return 1;
}

#endif /* !__DOXYGEN__ */
```

5.3 API reference

List of all the modules:

5.3.1 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 `lwcell_opts.h` file.

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

group **LWCELL_OPT**

LwCELL options.

Defines

LWCELL_CFG_OS

Enables 1 or disables 0 operating system support for GSM library.

Note: Value must be set to 1 in the current revision

Note: Check OS configuration group for more configuration related to operating system

LWCELL_CFG_MEM_CUSTOM

Enables 1 or disables 0 custom memory management functions.

When set to 1, Memory manager block must be provided manually. This includes implementation of functions lwcell_mem_malloc, lwcell_mem_calloc, lwcell_mem_realloc and lwcell_mem_free

Note: Function declaration follows standard C functions `malloc, calloc, realloc, free`. Declaration is available in `lwcell/lwcell_mem.h` file. Include this file to final implementation file

Note: When implementing custom memory allocation, it is necessary to take care of multiple threads accessing same resource for custom allocator

LWCELL_CFG_MEM_ALIGNMENT

Memory alignment for dynamic memory allocations.

Note: Some CPUs can work faster if memory is aligned, usually to 4 or 8 bytes. To speed up this possibilities, you can set memory alignment and library will try to allocate memory on aligned boundaries.

Note: Some CPUs such ARM Cortex-M0 don't support unaligned memory access. This CPUs must have set correct memory alignment value.

Note: This value must be power of 2

LWCELL_CFG_USE_API_FUNC_EVT

Enables 1 or disables 0 callback function and custom parameter for API functions.

When enabled, 2 additional parameters are available in API functions. When command is executed, callback function with its parameter could be called when not set to NULL.

LWCELL_CFG_AT_PORT_BAUDRATE

Default baudrate used for AT port.

Note: Later, user may call API function to change to desired baudrate if necessary

LWCELL_CFG_RCV_BUFF_SIZE

Buffer size for received data waiting to be processed.

Note: When server mode is active and a lot of connections are in queue this should be set high otherwise your buffer may overflow

Note: Buffer size also depends on TX user driver if it uses DMA or blocking mode In case of DMA (CPU can work other tasks), buffer may be smaller as CPU will have more time to process all the incoming bytes

Note: This parameter has no meaning when *LWCELL_CFG_INPUT_USE_PROCESS* is enabled

LWCELL_CFG_RESET_ON_INIT

Enables `1` or disables `0` reset sequence after lwcell_init call.

Note: When this functionality is disabled, user must manually call lwcell_reset to send reset sequence to GSM device.

LWCELL_CFG_RESET_ON_DEVICE_PRESENT

Enables `1` or disables `0` reset sequence after lwcell_device_set_present call.

Note: When this functionality is disabled, user must manually call lwcell_reset to send reset sequence to GSM device.

LWCELL_CFG_RESET_DELAY_DEFAULT

Default delay (milliseconds unit) before sending first AT command on reset sequence.

LWCELL_CFG_RESET_DELAY_AFTER

Default delay (milliseconds unit) after reset sequence.

LWCELL_CFG_KEEP_ALIVE

Enables `1` or disables `0` periodic keep-alive events to registered callbacks.

LWCELL_CFG_KEEP_ALIVE_TIMEOUT

Timeout periodic time to trigger keep alive events to registered callbacks.

Feature must be enabled with *LWCELL_CFG_KEEP_ALIVE*

LWCELL_CFG_CONN_POLL_INTERVAL

Poll interval for connections in units of milliseconds.

Value indicates interval time to call poll event on active connections.

Note: Single poll interval applies for all connections

Connection settings.

Defines

LWCELL_CFG_MAX_CONNS

Maximal number of connections AT software can support on GSM device.

LWCELL_CFG_CONN_MAX_DATA_LEN

Maximal number of bytes we can send at single command to GSM.

Note: Value can not exceed `1460` bytes or no data will be ever send

Note: This is limitation of GSM AT commands and on systems where RAM is not an issue, it should be set to maximal value (`1460`) to optimize data transfer speed performance

LWCELL_CFG_CONN_MIN_DATA_LEN

Minimal buffer in bytes for connection receive allocation.

```
            Allocation will always start with (up to) \ref LWCELL_CFG_CONN_MAX_
↪DATA_LEN
            and will continue with trial down to this setting up until␣
↪allocating is successful.
```

LWCELL_CFG_MAX_SEND_RETRIES

Set number of retries for send data command.

Sometimes it may happen that `AT+SEND` command fails due to different problems. Trying to send the same data multiple times can raise chances we are successful.

Debugging configurations.

Defines

LWCELL_CFG_DBG

Set global debug support.

Possible values are LWCELL_DBG_ON or LWCELL_DBG_OFF

Note: Set to LWCELL_DBG_OFF to globally disable all debugs

LWCELL_CFG_DBG_OUT(fmt, ...)

Debugging output function.

```
Called with format and optional parameters for printf style debug
```

LWCELL_CFG_DBG_LVL_MIN

Minimal debug level.

```
Check \ref LWCELL_DBG_LVL for possible values
```

LWCELL_CFG_DBG_TYPES_ON

Enabled debug types.

When debug is globally enabled with *LWCELL_CFG_DBG* parameter, user must enable debug types such as TRACE or STATE messages.

Check LWCELL_DBG_TYPE for possible options. Separate values with `bitwise OR` operator

LWCELL_CFG_DBG_INIT

Set debug level for init function.

Possible values are LWCELL_DBG_ON or LWCELL_DBG_OFF

LWCELL_CFG_DBG_MEM

Set debug level for memory manager.

Possible values are LWCELL_DBG_ON or LWCELL_DBG_OFF

LWCELL_CFG_DBG_INPUT

Set debug level for input module.

Possible values are LWCELL_DBG_ON or LWCELL_DBG_OFF

LWCELL_CFG_DBG_THREAD

Set debug level for GSM threads.

Possible values are LWCELL_DBG_ON or LWCELL_DBG_OFF

LWCELL_CFG_DBG_ASSERT

Set debug level for asserting of input variables.

Possible values are LWCELL_DBG_ON or LWCELL_DBG_OFF

LWCELL_CFG_DBG_IPD

Set debug level for incoming data received from device.

Possible values are LWCELL_DBG_ON or LWCELL_DBG_OFF

LWCELL_CFG_DBG_PBUF

Set debug level for packet buffer manager.

Possible values are LWCELL_DBG_ON or LWCELL_DBG_OFF

LWCELL_CFG_DBG_CONN

Set debug level for connections.

Possible values are LWCELL_DBG_ON or LWCELL_DBG_OFF

LWCELL_CFG_DBG_VAR

Set debug level for dynamic variable allocations.

Possible values are LWCELL_DBG_ON or LWCELL_DBG_OFF

LWCELL_CFG_DBG_NETCONN

Set debug level for netconn sequential API.

Possible values are LWCELL_DBG_ON or LWCELL_DBG_OFF

LWCELL_CFG_AT_ECHO

Enables `1` or disables `0` echo mode on AT commands sent to GSM device.

Note: This mode is useful when debugging GSM communication

Operating system dependant configuration.

Defines

LWCELL_CFG_THREAD_PRODUCER_MBOX_SIZE

Set number of message queue entries for procuder thread.

Message queue is used for storing memory address to command data

LWCELL_CFG_THREAD_PROCESS_MBOX_SIZE

Set number of message queue entries for processing thread.

Message queue is used to notify processing thread about new received data on AT port

LWCELL_CFG_INPUT_USE_PROCESS

Enables `1` or disables `0` direct support for processing input data.

When this mode is enabled, no overhead is included for copying data to receive buffer because bytes are processed directly.

Note: This mode can only be used when *LWCELL_CFG_OS* is enabled

Note: When using this mode, separate thread must be dedicated only for reading data on AT port

Note: Best case for using this mode is if DMA receive is supported by host device

LWCELL_THREAD_PRODUCER_HOOK()

Producer thread hook, called each time thread wakes-up and does the processing.

It can be used to check if thread is alive.

LWCELL_THREAD_PROCESS_HOOK()

Process thread hook, called each time thread wakes-up and does the processing.

It can be used to check if thread is alive.

LWCELL_CFG_THREADX_CUSTOM_MEM_BYTE_POOL

Enables `1` or disables `0` custom memory byte pool extension for ThreadX port.

When enabled, user must manually set byte pool at run-time, before lwcell_init is called

LWCELL_CFG_THREADX_IDLE_THREAD_EXTENSION

Enables `1` or disables `0` idle thread extensions feature of ThreadX.

When enabled, user must manually configure idle thread and setup additional thread handle extension fields. By default ThreadX doesn't support self-thread cleanup when thread memory is dynamically allocated & thread terminated, hence another thread is mandatory to do the cleanup process instead.

This configuration does not create idle-thread, rather only sets additional TX_THREAD fields, indicating thread handle and thread stack are dynamically allocated.

Have a look at System-ThreadX port for implementation

Configuration of specific modules.

Defines

LWCELL_CFG_NETWORK

Enables `1` or disables `0` network functionality used for TCP/IP communication.

Network must be enabled to use all GPRS/LTE functions such as connection API, FTP, HTTP, etc.

LWCELL_CFG_NETWORK_IGNORE_CGACT_RESULT

Ignores `1` or not `0` result from `AT+CGACT` command.

Note: This may be used for data-only SIM cards where command might fail when trying to attach to network for data transfer

LWCELL_CFG_CONN

Enables 1 or disables 0 connection API.

Note: *LWCELL_CFG_NETWORK* must be enabled to use connection feature

LWCELL_CFG_SMS

Enables 1 or disables 0 SMS API.

LWCELL_CFG_CALL

Enables 1 or disables 0 call API.

LWCELL_CFG_PHONEBOOK

Enables 1 or disables 0 phonebook API.

LWCELL_CFG_HTTP

Enables 1 or disables 0 HTTP API.

Note: *LWCELL_CFG_NETWORK* must be enabled to use connection feature

LWCELL_CFG_FTP

Enables 1 or disables 0 FTP API.

Note: *LWCELL_CFG_NETWORK* must be enabled to use connection feature

LWCELL_CFG_PING

Enables 1 or disables 0 PING API.

Note: *LWCELL_CFG_NETWORK* must be enabled to use connection feature

LWCELL_CFG_USSD

Enables 1 or disables 0 USSD API.

Configuration of netconn API module.

Defines

LWCELL_CFG_NETCONN

Enables 1 or disables 0 NETCONN sequential API support for OS systems.

See also:

LWCELL_CFG_OS

Note: To use this feature, OS support is mandatory.

LWCELL_CFG_NETCONN_RECEIVE_TIMEOUT

Enables 1 or disables 0 receive timeout feature.

When this option is enabled, user will get an option to set timeout value for receive data on netconn, before function returns timeout error.

Note: Even if this option is enabled, user must still manually set timeout, by default time will be set to 0 which means no timeout.

LWCELL_CFG_NETCONN_ACCEPT_QUEUE_LEN

Accept queue length for new client when netconn server is used.

Defines number of maximal clients waiting in accept queue of server connection

LWCELL_CFG_NETCONN_RECEIVE_QUEUE_LEN

Receive queue length for pbuf entries.

Defines maximal number of pbuf data packet references for receive

Configuration of MQTT and MQTT API client modules.

Defines

LWCELL_CFG_MQTT_MAX_REQUESTS

Maximal number of open MQTT requests at a time.

LWCELL_CFG_MQTT_API_MBOX_SIZE

Size of MQTT API message queue for received messages.

LWCELL_CFG_DBG_MQTT

Set debug level for MQTT client module.

Possible values are LWCELL_DBG_ON or LWCELL_DBG_OFF

LWCELL_CFG_DBG_MQTT_API

Set debug level for MQTT API client module.

Possible values are LWCELL_DBG_ON or LWCELL_DBG_OFF

Standard C library configuration.

Configuration allows you to overwrite default C language function in case of better implementation with hardware (for example DMA for data copy).

Defines

LWCELL_MEMCPY(dst, src, len)

Memory copy function declaration.

User is able to change the memory function, in case hardware supports copy operation, it may implement its own

Function prototype must be similar to:

```
void *  my_memcpy(void* dst, const void* src, size_t len);
```

Parameters

- **dst** – **[in]** Destination memory start address
- **src** – **[in]** Source memory start address
- **len** – **[in]** Number of bytes to copy

Returns

Destination memory start address

LWCELL_MEMSET(dst, b, len)

Memory set function declaration.

Function prototype must be similar to:

```
void *  my_memset(void* dst, int b, size_t len);
```

Parameters

- **dst** – **[in]** Destination memory start address
- **b** – **[in]** Value (byte) to set in memory
- **len** – **[in]** Number of bytes to set

Returns

Destination memory start address

5.3.2 Platform specific

List of all the modules:

Low-Level functions

Low-level module consists of callback-only functions, which are called by middleware and must be implemented by final application.

Tip: Check *Porting guide* for actual implementation

group **LWCELL_LL**

Low-level communication functions.

Typedefs

typedef size_t (***lwcell_ll_send_fn**)(const void *data, size_t len)

Function prototype for AT output data.

Param data
[in] Pointer to data to send. This parameter can be set to `NULL`, indicating to the low-level that (if used) DMA could be started to transmit data to the device

Param len
[in] Number of bytes to send. This parameter can be set to `0` to indicate that internal buffer can be flushed to stream. This is implementation defined and feature might be ignored

Return
Number of bytes sent

typedef uint8_t (***lwcell_ll_reset_fn**)(uint8_t state)

Function prototype for hardware reset of GSM device.

Param state
[in] State indicating reset. When set to `1`, reset must be active (usually pin active low), or set to `0` when reset is cleared

Return
`1` on successful action, `0` otherwise

Functions

lwcellr_t **lwcell_ll_init**(*lwcell_ll_t* *ll)

Callback function called from initialization process.

Note: This function may be called multiple times if AT baudrate is changed from application. It is important that every configuration except AT baudrate is configured only once!

Note: This function may be called from different threads in GSM stack when using OS. When *LWCELL_CFG_INPUT_USE_PROCESS* is set to 1, this function may be called from user UART thread.

Parameters
ll – **[inout]** Pointer to *lwcell_ll_t* structure to fill data for communication functions

Returns
lwcellOK on success, member of lwcellr_t enumeration otherwise

lwcellr_t **lwcell_ll_deinit**(*lwcell_ll_t* *ll)

Callback function to de-init low-level communication part.

Parameters
ll – **[inout]** Pointer to *lwcell_ll_t* structure to fill data for communication functions

Returns
lwcellOK on success, member of lwcellr_t enumeration otherwise

struct **lwcell_ll_t**

#include <lwcell_types.h> Low level user specific functions.

Public Members

lwcell_ll_send_fn **send_fn**

Callback function to transmit data

lwcell_ll_reset_fn **reset_fn**

Reset callback function

uint32_t **baudrate**

UART baudrate value

struct *lwcell_ll_t*::[anonymous] **uart**

UART communication parameters

System functions

System functions are bridge between operating system system calls and middleware system calls. Middleware is tightly coupled with operating system features hence it is important to include OS features directly.

It includes support for:

- Thread management, to start/stop threads
- Mutex management for recursive mutexes
- Semaphore management for binary-only semaphores
- Message queues for thread-safe data exchange between threads
- Core system protection for mutual exclusion to access shared resources

Tip: Check *Porting guide* for actual implementation guidelines.

group **LWCELL_SYS**

System based function for OS management, timings, etc.

Main

uint8_t **lwcell_sys_init**(void)

Init system dependant parameters.

After this function is called, all other system functions must be fully ready.

Returns
1 on success, 0 otherwise

uint32_t **lwcell_sys_now**(void)

Get current time in units of milliseconds.

Returns
Current time in units of milliseconds

uint8_t **lwcell_sys_protect**(void)

Protect middleware core.

Stack protection must support recursive mode. This function may be called multiple times, even if access has been granted before.

Note: Most operating systems support recursive mutexes.

Returns
1 on success, 0 otherwise

uint8_t **lwcell_sys_unprotect**(void)

Unprotect middleware core.

This function must follow number of calls of *lwcell_sys_protect* and unlock access only when counter reached back zero.

Note: Most operating systems support recursive mutexes.

Returns
1 on success, 0 otherwise

Mutex

uint8_t **lwcell_sys_mutex_create**(lwcell_sys_mutex_t *p)

Create new recursive mutex.

Note: Recursive mutex has to be created as it may be locked multiple times before unlocked

Parameters
p – **[out]** Pointer to mutex structure to allocate

Returns
1 on success, 0 otherwise

uint8_t **lwcell_sys_mutex_delete**(lwcell_sys_mutex_t *p)

Delete recursive mutex from system.

Parameters
p – **[in]** Pointer to mutex structure

Returns
1 on success, 0 otherwise

uint8_t **lwcell_sys_mutex_lock**(lwcell_sys_mutex_t *p)

Lock recursive mutex, wait forever to lock.

Parameters
p – **[in]** Pointer to mutex structure

Returns
1 on success, 0 otherwise

uint8_t **lwcell_sys_mutex_unlock**(lwcell_sys_mutex_t *p)

Unlock recursive mutex.

Parameters
p – **[in]** Pointer to mutex structure

Returns
1 on success, 0 otherwise

uint8_t **lwcell_sys_mutex_isvalid**(lwcell_sys_mutex_t *p)

Check if mutex structure is valid system.

Parameters
p – **[in]** Pointer to mutex structure

Returns
1 on success, 0 otherwise

uint8_t **lwcell_sys_mutex_invalid**(lwcell_sys_mutex_t *p)

Set recursive mutex structure as invalid.

Parameters
p – **[in]** Pointer to mutex structure

Returns
1 on success, 0 otherwise

Semaphores

uint8_t **lwcell_sys_sem_create**(*lwcell_sys_sem_t* *p, uint8_t cnt)

Create a new binary semaphore and set initial state.

Note: Semaphore may only have 1 token available

Parameters

- **p** – **[out]** Pointer to semaphore structure to fill with result
- **cnt** – **[in]** Count indicating default semaphore state: 0: Take semaphore token immediately 1: Keep token available

Returns

1 on success, 0 otherwise

uint8_t **lwcell_sys_sem_delete**(*lwcell_sys_sem_t* *p)

Delete binary semaphore.

Parameters

p – **[in]** Pointer to semaphore structure

Returns

1 on success, 0 otherwise

uint32_t **lwcell_sys_sem_wait**(*lwcell_sys_sem_t* *p, uint32_t timeout)

Wait for semaphore to be available.

Parameters

- **p** – **[in]** Pointer to semaphore structure
- **timeout** – **[in]** Timeout to wait in milliseconds. When 0 is applied, wait forever

Returns

Number of milliseconds waited for semaphore to become available or *LWCELL_SYS_TIMEOUT* if not available within given time

uint8_t **lwcell_sys_sem_release**(*lwcell_sys_sem_t* *p)

Release semaphore.

Parameters

p – **[in]** Pointer to semaphore structure

Returns

1 on success, 0 otherwise

uint8_t **lwcell_sys_sem_isvalid**(*lwcell_sys_sem_t* *p)

Check if semaphore is valid.

Parameters

p – **[in]** Pointer to semaphore structure

Returns

1 on success, 0 otherwise

uint8_t **lwcell_sys_sem_invalid**(*lwcell_sys_sem_t* *p)

Invalid semaphore.

Parameters
p – **[in]** Pointer to semaphore structure

Returns
1 on success, 0 otherwise

Message queues

uint8_t **lwcell_sys_mbox_create**(*lwcell_sys_mbox_t* *b, size_t size)

Create a new message queue with entry type of `void *`

Parameters

- **b** – **[out]** Pointer to message queue structure
- **size** – **[in]** Number of entries for message queue to hold

Returns
1 on success, 0 otherwise

uint8_t **lwcell_sys_mbox_delete**(*lwcell_sys_mbox_t* *b)

Delete message queue.

Parameters
b – **[in]** Pointer to message queue structure

Returns
1 on success, 0 otherwise

uint32_t **lwcell_sys_mbox_put**(*lwcell_sys_mbox_t* *b, void *m)

Put a new entry to message queue and wait until memory available.

Parameters

- **b** – **[in]** Pointer to message queue structure
- **m** – **[in]** Pointer to entry to insert to message queue

Returns
Time in units of milliseconds needed to put a message to queue

uint32_t **lwcell_sys_mbox_get**(*lwcell_sys_mbox_t* *b, void **m, uint32_t timeout)

Get a new entry from message queue with timeout.

Parameters

- **b** – **[in]** Pointer to message queue structure
- **m** – **[in]** Pointer to pointer to result to save value from message queue to
- **timeout** – **[in]** Maximal timeout to wait for new message. When 0 is applied, wait for unlimited time

Returns
Time in units of milliseconds needed to put a message to queue or *LWCELL_SYS_TIMEOUT* if it was not successful

uint8_t **lwcell_sys_mbox_putnow**(*lwcell_sys_mbox_t* *b, void *m)

Put a new entry to message queue without timeout (now or fail)

Parameters

- **b** – **[in]** Pointer to message queue structure
- **m** – **[in]** Pointer to message to save to queue

Returns

1 on success, 0 otherwise

uint8_t **lwcell_sys_mbox_getnow**(*lwcell_sys_mbox_t* *b, void **m)

Get an entry from message queue immediately.

Parameters

- **b** – **[in]** Pointer to message queue structure
- **m** – **[in]** Pointer to pointer to result to save value from message queue to

Returns

1 on success, 0 otherwise

uint8_t **lwcell_sys_mbox_isvalid**(*lwcell_sys_mbox_t* *b)

Check if message queue is valid.

Parameters

b – **[in]** Pointer to message queue structure

Returns

1 on success, 0 otherwise

uint8_t **lwcell_sys_mbox_invalid**(*lwcell_sys_mbox_t* *b)

Invalid message queue.

Parameters

b – **[in]** Pointer to message queue structure

Returns

1 on success, 0 otherwise

Threads

uint8_t **lwcell_sys_thread_create**(*lwcell_sys_thread_t* *t, const char *name, *lwcell_sys_thread_fn* thread_func, void *const arg, size_t stack_size, *lwcell_sys_thread_prio_t* prio)

Create a new thread.

Parameters

- **t** – **[out]** Pointer to thread identifier if create was successful. It may be set to NULL
- **name** – **[in]** Name of a new thread
- **thread_func** – **[in]** Thread function to use as thread body
- **arg** – **[in]** Thread function argument
- **stack_size** – **[in]** Size of thread stack in uints of bytes. If set to 0, reserve default stack size

- **prio** – **[in]** Thread priority

Returns
1 on success, 0 otherwise

uint8_t **lwcell_sys_thread_terminate**(*lwcell_sys_thread_t* *t)

Terminate thread (shut it down and remove)

Parameters
t – **[in]** Pointer to thread handle to terminate. If set to NULL, terminate current thread (thread from where function is called)

Returns
1 on success, 0 otherwise

uint8_t **lwcell_sys_thread_yield**(void)

Yield current thread.

Returns
1 on success, 0 otherwise

Defines

LWCELL_SYS_MUTEX_NULL

Mutex invalid value.

Value assigned to lwcell_sys_mutex_t type when it is not valid.

LWCELL_SYS_SEM_NULL

Semaphore invalid value.

Value assigned to *lwcell_sys_sem_t* type when it is not valid.

LWCELL_SYS_MBOX_NULL

Message box invalid value.

Value assigned to *lwcell_sys_mbox_t* type when it is not valid.

LWCELL_SYS_TIMEOUT

OS timeout value.

Value returned by operating system functions (mutex wait, sem wait, mbox wait) when it returns timeout and does not give valid value to application

LWCELL_SYS_THREAD_PRIO

Default thread priority value used by middleware to start built-in threads.

Threads can well operate with normal (default) priority and do not require any special feature in terms of priority for proper operation.

lwcell_sys_THREAD_SS

Stack size in units of bytes for system threads.

It is used as default stack size for all built-in threads.

Typedefs

typedef void (*lwcell_sys_thread_fn)(void*)

Thread function prototype.

typedef void *lwcell_sys_lwcell_sys_mutex_t

System mutex type.

It is used by middleware as base type of mutex.

typedef void *lwcell_sys_sem_t

System semaphore type.

It is used by middleware as base type of mutex.

typedef void *lwcell_sys_mbox_t

System message queue type.

It is used by middleware as base type of mutex.

typedef void *lwcell_sys_thread_t

System thread ID type.

typedef int lwcell_sys_thread_prio_t

System thread priority type.

It is used as priority type for system function, to start new threads by middleware.

5.3.3 Applications

MQTT Client

MQTT client v3.1.1 implementation, based on callback (non-netconn) connection API.

group LWCELL_APP_MQTT_CLIENT

MQTT client.

Typedefs

typedef struct lwcell_mqtt_client *lwcell_mqtt_client_p

Pointer to lwcell_mqtt_client_t structure.

typedef void (*lwcell_mqtt_evt_fn)(*lwcell_mqtt_client_p* client, *lwcell_mqtt_evt_t* *evt)

MQTT event callback function.

Param client
[in] MQTT client

Param evt
[in] MQTT event with type and related data

Enums

enum **lwcell_mqtt_qos_t**

Quality of service enumeration.

Values:

enumerator **LWCELL_MQTT_QOS_AT_MOST_ONCE** = 0x00

Delivery is not guaranteed to arrive, but can arrive `up to 1 time` = non-critical packets where losses are allowed

enumerator **LWCELL_MQTT_QOS_AT_LEAST_ONCE** = 0x01

Delivery is quaranteed `at least once`, but it may be delivered multiple times with the same content

enumerator **LWCELL_MQTT_QOS_EXACTLY_ONCE** = 0x02

Delivery is quaranteed `exactly once` = very critical packets such as billing informations or similar

enum **lwcell_mqtt_state_t**

State of MQTT client.

Values:

enumerator **LWCELL_MQTT_CONN_DISCONNECTED** = 0x00

Connection with server is not established

enumerator **LWCELL_MQTT_CONN_CONNECTING**

Client is connecting to server

enumerator **LWCELL_MQTT_CONN_DISCONNECTING**

Client connection is disconnecting from server

enumerator **LWCELL_MQTT_CONNECTING**

MQTT client is connecting… CONNECT command has been sent to server

enumerator **LWCELL_MQTT_CONNECTED**

MQTT is fully connected and ready to send data on topics

enum **lwcell_mqtt_evt_type_t**

MQTT event types.

Values:

enumerator **LWCELL_MQTT_EVT_CONNECT**

MQTT client connect event

enumerator **LWCELL_MQTT_EVT_SUBSCRIBE**

MQTT client subscribed to specific topic

enumerator **LWCELL_MQTT_EVT_UNSUBSCRIBE**

MQTT client unsubscribed from specific topic

enumerator **LWCELL_MQTT_EVT_PUBLISH**

MQTT client publish message to server event.

Note: When publishing packet with quality of service *LWCELL_MQTT_QOS_AT_MOST_ONCE*, you may not receive event, even if packet was successfully sent, thus do not rely on this event for packet with `qos = LWCELL_MQTT_QOS_AT_MOST_ONCE`

---

enumerator **LWCELL_MQTT_EVT_PUBLISH_RECV**

MQTT client received a publish message from server

enumerator **LWCELL_MQTT_EVT_DISCONNECT**

MQTT client disconnected from MQTT server

enumerator **LWCELL_MQTT_EVT_KEEP_ALIVE**

MQTT keep-alive sent to server and reply received

enum **lwcell_mqtt_conn_status_t**

List of possible results from MQTT server when executing connect command.

Values:

enumerator **LWCELL_MQTT_CONN_STATUS_ACCEPTED** = 0x00

Connection accepted and ready to use

enumerator **LWCELL_MQTT_CONN_STATUS_REFUSED_PROTOCOL_VERSION** = 0x01

Connection Refused, unacceptable protocol version

enumerator **LWCELL_MQTT_CONN_STATUS_REFUSED_ID** = 0x02

Connection refused, identifier rejected

enumerator **LWCELL_MQTT_CONN_STATUS_REFUSED_SERVER** = 0x03

Connection refused, server unavailable

enumerator **LWCELL_MQTT_CONN_STATUS_REFUSED_USER_PASS** = 0x04

Connection refused, bad user name or password

enumerator **LWCELL_MQTT_CONN_STATUS_REFUSED_NOT_AUTHORIZED** = 0x05

Connection refused, not authorized

enumerator **LWCELL_MQTT_CONN_STATUS_TCP_FAILED** = 0x100

TCP connection to server was not successful

Functions

lwcell_mqtt_client_p **lwcell_mqtt_client_new**(size_t tx_buff_len, size_t rx_buff_len)

Allocate a new MQTT client structure.

Parameters

- **tx_buff_len** – **[in]** Length of raw data output buffer
- **rx_buff_len** – **[in]** Length of raw data input buffer

Returns

Pointer to new allocated MQTT client structure or `NULL` on failure

void **lwcell_mqtt_client_delete**(*lwcell_mqtt_client_p* client)

Delete MQTT client structure.

Note: MQTT client must be disconnected first

Parameters

client – **[in]** MQTT client

lwcellr_t **lwcell_mqtt_client_connect**(*lwcell_mqtt_client_p* client, const char *host, lwcell_port_t port, *lwcell_mqtt_evt_fn* evt_fn, const *lwcell_mqtt_client_info_t* *info)

Connect to MQTT server.

Note: After TCP connection is established, CONNECT packet is automatically sent to server

Parameters

- **client** – **[in]** MQTT client
- **host** – **[in]** Host address for server
- **port** – **[in]** Host port number
- **evt_fn** – **[in]** Callback function for all events on this MQTT client
- **info** – **[in]** Information structure for connection

Returns

lwcellOK on success, member of lwcellr_t enumeration otherwise

lwcellr_t **lwcell_mqtt_client_disconnect**(*lwcell_mqtt_client_p* client)

Disconnect from MQTT server.

Parameters

client – **[in]** MQTT client

Returns

lwcellOK if request sent to queue or member of lwcellr_t otherwise

uint8_t **lwcell_mqtt_client_is_connected**(*lwcell_mqtt_client_p* client)

Test if client is connected to server and accepted to MQTT protocol.

Note: Function will return error if TCP is connected but MQTT not accepted

Parameters
`client` – **[in]** MQTT client

Returns
1 on success, `0` otherwise

lwcellr_t **lwcell_mqtt_client_subscribe**(*lwcell_mqtt_client_p* client, const char *topic, *lwcell_mqtt_qos_t* qos, void *arg)

Subscribe to MQTT topic.

Parameters

- `client` – **[in]** MQTT client
- `topic` – **[in]** Topic name to subscribe to
- `qos` – **[in]** Quality of service. This parameter can be a value of *lwcell_mqtt_qos_t*
- `arg` – **[in]** User custom argument used in callback

Returns
lwcellOK on success, member of lwcellr_t enumeration otherwise

lwcellr_t **lwcell_mqtt_client_unsubscribe**(*lwcell_mqtt_client_p* client, const char *topic, void *arg)

Unsubscribe from MQTT topic.

Parameters

- `client` – **[in]** MQTT client
- `topic` – **[in]** Topic name to unsubscribe from
- `arg` – **[in]** User custom argument used in callback

Returns
lwcellOK on success, member of lwcellr_t enumeration otherwise

lwcellr_t **lwcell_mqtt_client_publish**(*lwcell_mqtt_client_p* client, const char *topic, const void *payload, uint16_t len, *lwcell_mqtt_qos_t* qos, uint8_t retain, void *arg)

Publish a new message on specific topic.

Parameters

- `client` – **[in]** MQTT client
- `topic` – **[in]** Topic to send message to
- `payload` – **[in]** Message data
- `payload_len` – **[in]** Length of payload data
- `qos` – **[in]** Quality of service. This parameter can be a value of *lwcell_mqtt_qos_t* enumeration
- `retain` – **[in]** Retian parameter value
- `arg` – **[in]** User custom argument used in callback

Returns
lwcellOK on success, member of lwcellr_t enumeration otherwise

void *lwcell_mqtt_client_get_arg(*lwcell_mqtt_client_p* client)

Get user argument on client.

Parameters
client – **[in]** MQTT client handle

Returns
User argument

void lwcell_mqtt_client_set_arg(*lwcell_mqtt_client_p* client, void *arg)

Set user argument on client.

Parameters

- **client** – **[in]** MQTT client handle
- **arg** – **[in]** User argument

struct lwcell_mqtt_client_info_t

#include <*lwcell_mqtt_client.h*> MQTT client information structure.

Public Members

const char *id

Client unique identifier. It is required and must be set by user

const char *user

Authentication username. Set to NULL if not required

const char *pass

Authentication password, set to NULL if not required

uint16_t keep_alive

Keep-alive parameter in units of seconds. When set to 0, functionality is disabled (not recommended)

const char *will_topic

Will topic

const char *will_message

Will message

lwcell_mqtt_qos_t will_qos

Will topic quality of service

struct lwcell_mqtt_request_t

#include <*lwcell_mqtt_client.h*> MQTT request object.

Public Members

uint8_t **status**

Entry status flag for in use or pending bit

uint16_t **packet_id**

Packet ID generated by client on publish

void ***arg**

User defined argument

uint32_t **expected_sent_len**

Number of total bytes which must be sent on connection before we can say "packet was sent".

uint32_t **timeout_start_time**

Timeout start time in units of milliseconds

struct **lwcell_mqtt_evt_t**

#include <lwcell_mqtt_client.h> MQTT event structure for callback function.

Public Members

lwcell_mqtt_evt_type_t **type**

Event type

lwcell_mqtt_conn_status_t **status**

Connection status with MQTT

struct *lwcell_mqtt_evt_t*::[anonymous]::[anonymous] **connect**

Event for connecting to server

uint8_t **is_accepted**

Status if client was accepted to MQTT prior disconnect event

struct *lwcell_mqtt_evt_t*::[anonymous]::[anonymous] **disconnect**

Event for disconnecting from server

void ***arg**

User argument for callback function

lwcellr_t **res**

Response status

struct *lwcell_mqtt_evt_t*::[anonymous]::[anonymous] **sub_unsub_scribed**

Event for (un)subscribe to/from topics

struct *lwcell_mqtt_evt_t*::[anonymous]::[anonymous] **publish**

Published event

const uint8_t ***topic**

Pointer to topic identifier

size_t **topic_len**

Length of topic

const void ***payload**

Topic payload

size_t **payload_len**

Length of topic payload

uint8_t **dup**

Duplicate flag if message was sent again

lwcell_mqtt_qos_t **qos**

Received packet quality of service

struct *lwcell_mqtt_evt_t*::[anonymous]::[anonymous] **publish_recv**

Publish received event

union *lwcell_mqtt_evt_t*::[anonymous] **evt**

Event data parameters

group **LWCELL_APP_MQTT_CLIENT_EVT**

Event helper functions.

Connect event

Note: Use these functions on *LWCELL_MQTT_EVT_CONNECT* event

lwcell_mqtt_client_evt_connect_get_status(client, evt)

Get connection status.

Parameters

- **client** – **[in]** MQTT client
- **evt** – **[in]** Event handle

Returns

Connection status. Member of *lwcell_mqtt_conn_status_t*

Disconnect event

Note: Use these functions on *LWCELL_MQTT_EVT_DISCONNECT* event

`lwcell_mqtt_client_evt_disconnect_is_accepted`(client, evt)

Check if MQTT client was accepted by server when disconnect event occurred.

Parameters

- **client** – **[in]** MQTT client
- **evt** – **[in]** Event handle

Returns

1 on success, `0` otherwise

Subscribe/unsubscribe event

Note: Use these functions on *LWCELL_MQTT_EVT_SUBSCRIBE* or *LWCELL_MQTT_EVT_UNSUBSCRIBE* events

`lwcell_mqtt_client_evt_subscribe_get_argument`(client, evt)

Get user argument used on *lwcell_mqtt_client_subscribe*.

Parameters

- **client** – **[in]** MQTT client
- **evt** – **[in]** Event handle

Returns

User argument

`lwcell_mqtt_client_evt_subscribe_get_result`(client, evt)

Get result of subscribe event.

Parameters

- **client** – **[in]** MQTT client
- **evt** – **[in]** Event handle

Returns

lwcellOK on success, member of lwcellr_t otherwise

`lwcell_mqtt_client_evt_unsubscribe_get_argument`(client, evt)

Get user argument used on *lwcell_mqtt_client_unsubscribe*.

Parameters

- **client** – **[in]** MQTT client

- **evt** – **[in]** Event handle

Returns
User argument

`lwcell_mqtt_client_evt_unsubscribe_get_result`(client, evt)

Get result of unsubscribe event.

Parameters

- **client** – **[in]** MQTT client
- **evt** – **[in]** Event handle

Returns
lwcellOK on success, member of lwcellr_t otherwise

Publish receive event

Note: Use these functions on *LWCELL_MQTT_EVT_PUBLISH_RECV* event

`lwcell_mqtt_client_evt_publish_recv_get_topic`(client, evt)

Get topic from received publish packet.

Parameters

- **client** – **[in]** MQTT client
- **evt** – **[in]** Event handle

Returns
Topic name

`lwcell_mqtt_client_evt_publish_recv_get_topic_len`(client, evt)

Get topic length from received publish packet.

Parameters

- **client** – **[in]** MQTT client
- **evt** – **[in]** Event handle

Returns
Topic length

`lwcell_mqtt_client_evt_publish_recv_get_payload`(client, evt)

Get payload from received publish packet.

Parameters

- **client** – **[in]** MQTT client
- **evt** – **[in]** Event handle

Returns
Packet payload

`lwcell_mqtt_client_evt_publish_recv_get_payload_len`(client, evt)

Get payload length from received publish packet.

Parameters

- **client** – **[in]** MQTT client
- **evt** – **[in]** Event handle

Returns
Payload length

`lwcell_mqtt_client_evt_publish_recv_is_duplicate`(client, evt)

Check if packet is duplicated.

Parameters

- **client** – **[in]** MQTT client
- **evt** – **[in]** Event handle

Returns
1 if duplicated, 0 otherwise

`lwcell_mqtt_client_evt_publish_recv_get_qos`(client, evt)

Get received quality of service.

Parameters

- **client** – **[in]** MQTT client
- **evt** – **[in]** Event handle

Returns
Member of *lwcell_mqtt_qos_t* enumeration

Publish event

Note: Use these functions on *LWCELL_MQTT_EVT_PUBLISH* event

`lwcell_mqtt_client_evt_publish_get_argument`(client, evt)

Get user argument used on *lwcell_mqtt_client_publish*.

Parameters

- **client** – **[in]** MQTT client
- **evt** – **[in]** Event handle

Returns
User argument

`lwcell_mqtt_client_evt_publish_get_result`(client, evt)

Get result of publish event.

Parameters

- **client** – **[in]** MQTT client

- **evt** – **[in]** Event handle

Returns
lwcellOK on success, member of lwcellr_t otherwise

Defines

lwcell_mqtt_client_evt_get_type(client, evt)

Get MQTT event type.

Parameters

- **client** – **[in]** MQTT client
- **evt** – **[in]** Event handle

Returns
MQTT Event type, value of *lwcell_mqtt_evt_type_t* enumeration

MQTT Client API

MQTT Client API provides sequential API built on top of *MQTT Client*.

Listing 14: MQTT API application example code

```
/*
 * MQTT client API example with GSM device.
 *
 * Once device is connected to network,
 * it will try to connect to mosquitto test server and start the MQTT.
 *
 * If successfully connected, it will publish data to "lwcell_mqtt_topic" topic every x seconds.
 *
 * To check if data are sent, you can use mqtt-spy PC software to inspect
 * test.mosquitto.org server and subscribe to publishing topic
 */

#include "mqtt_client_api.h"
#include "lwcell/apps/lwcell_mqtt_client_api.h"
#include "lwcell/lwcell.h"
#include "lwcell/lwcell_mem.h"
#include "lwcell/lwcell_network_api.h"

/**
 * \brief           Connection information for MQTT CONNECT packet
 */
static const lwcell_mqtt_client_info_t mqtt_client_info = {
    .keep_alive = 10,

    /* Server login data */
    .user = "8a215f70-a644-11e8-ac49-e932ed599553",
    .pass = "26aa943f702e5e780f015cd048a91e8fb54cca28",

```

(continues on next page)

(continued from previous page)

```
    /* Device identifier address */
    .id = "2c3573a0-0176-11e9-a056-c5cffe7f75f9",
};

/**
 * \brief           Memory for temporary topic
 */
static char mqtt_topic_str[256];

/**
 * \brief           Generate random number and write it to string
 * \param[out]      str: Output string with new number
 */
static void
generate_random(char* str) {
    static uint32_t random_beg = 0x8916;
    random_beg = random_beg * 0x00123455 + 0x85654321;
    sprintf(str, "%u", (unsigned)((random_beg >> 8) & 0xFFFF));
}

/**
 * \brief           MQTT client API thread
 */
void
lwcell_mqtt_client_api_thread(void const* arg) {
    lwcell_mqtt_client_api_p client;
    lwcell_mqtt_conn_status_t conn_status;
    lwcell_mqtt_client_api_buf_p buf;
    lwcellr_t res;
    char random_str[10];

    LWCELL_UNUSED(arg);

    /* Request network attach */
    while (lwcell_network_request_attach() != lwcellOK) {
        lwcell_delay(1000);
    }

    /* Create new MQTT API */
    if ((client = lwcell_mqtt_client_api_new(256, 128)) == NULL) {
        goto terminate;
    }

    while (1) {
        /* Make a connection */
        printf("Joining MQTT server\r\n");

        /* Try to join */
        conn_status = lwcell_mqtt_client_api_connect(client, "mqtt.mydevices.com", 1883,␣
↪&mqtt_client_info);
        if (conn_status == LWCELL_MQTT_CONN_STATUS_ACCEPTED) {
            printf("Connected and accepted!\r\n");
```

(continues on next page)

(continued from previous page)

```
            printf("Client is ready to subscribe and publish to new messages\r\n");
        } else {
            printf("Connect API response: %d\r\n", (int)conn_status);
            lwcell_delay(5000);
            continue;
        }

        /* Subscribe to topics */
        sprintf(mqtt_topic_str, "v1/%s/things/%s/cmd/#", mqtt_client_info.user, mqtt_
↪client_info.id);
        if (lwcell_mqtt_client_api_subscribe(client, mqtt_topic_str, LWCELL_MQTT_QOS_AT_
↪LEAST_ONCE) == lwcellOK) {
            printf("Subscribed to topic\r\n");
        } else {
            printf("Problem subscribing to topic!\r\n");
        }

        while (1) {
            /* Receive MQTT packet with timeout */
            if ((res = lwcell_mqtt_client_api_receive(client, &buf, 5000)) == lwcellOK) {
                if (buf != NULL) {
                    printf("Publish received!\r\n");
                    printf("Topic: %s, payload: %s\r\n", buf->topic, buf->payload);
                    lwcell_mqtt_client_api_buf_free(buf);
                    buf = NULL;
                }
            } else if (res == lwcellCLOSED) {
                printf("MQTT connection closed!\r\n");
                break;
            } else if (res == lwcellTIMEOUT) {
                printf("Timeout on MQTT receive function. Manually publishing.\r\n");

                /* Publish data on channel 1 */
                generate_random(random_str);
                sprintf(mqtt_topic_str, "v1/%s/things/%s/data/1", mqtt_client_info.user,␣
↪mqtt_client_info.id);
                lwcell_mqtt_client_api_publish(client, mqtt_topic_str, random_str,␣
↪strlen(random_str),
                                               LWCELL_MQTT_QOS_AT_LEAST_ONCE, 0);
            }
        }
        goto terminate;
    }

terminate:
    lwcell_mqtt_client_api_delete(client);
    lwcell_network_request_detach();
    printf("MQTT client thread terminate\r\n");
    lwcell_sys_thread_terminate(NULL);
}
```

group **LWCELL_APP_MQTT_CLIENT_API**

Sequential, single thread MQTT client API.

Typedefs

typedef struct lwcell_mqtt_client_api_buf *__lwcell_mqtt_client_api_buf_p__

Pointer to *lwcell_mqtt_client_api_buf_t* structure.

Functions

lwcell_mqtt_client_api_p __lwcell_mqtt_client_api_new__(size_t tx_buff_len, size_t rx_buff_len)

Create new MQTT client API.

Parameters

- **tx_buff_len** – **[in]** Maximal TX buffer for maximal packet length
- **rx_buff_len** – **[in]** Maximal RX buffer

Returns

Client handle on success, `NULL` otherwise

void __lwcell_mqtt_client_api_delete__(lwcell_mqtt_client_api_p client)

Delete client from memory.

Parameters

client – **[in]** MQTT API client handle

lwcell_mqtt_conn_status_t __lwcell_mqtt_client_api_connect__(lwcell_mqtt_client_api_p client, const char *host, lwcell_port_t port, const *lwcell_mqtt_client_info_t* *info)

Connect to MQTT broker.

Parameters

- **client** – **[in]** MQTT API client handle
- **host** – **[in]** TCP host
- **port** – **[in]** TCP port
- **info** – **[in]** MQTT client info

Returns

LWCELL_MQTT_CONN_STATUS_ACCEPTED on success, member of *lwcell_mqtt_conn_status_t* otherwise

lwcellr_t __lwcell_mqtt_client_api_close__(lwcell_mqtt_client_api_p client)

Close MQTT connection.

Parameters

client – **[in]** MQTT API client handle

Returns

lwcellOK on success, member of lwcellr_t otherwise

lwcellr_t __lwcell_mqtt_client_api_subscribe__(lwcell_mqtt_client_api_p client, const char *topic, *lwcell_mqtt_qos_t* qos)

Subscribe to topic.

Parameters

- **client** – **[in]** MQTT API client handle
- **topic** – **[in]** Topic to subscribe on
- **qos** – **[in]** Quality of service. This parameter can be a value of *lwcell_mqtt_qos_t*

Returns
lwcellOK on success, member of lwcellr_t otherwise

lwcellr_t **lwcell_mqtt_client_api_unsubscribe**(lwcell_mqtt_client_api_p client, const char *topic)

Unsubscribe from topic.

Parameters

- **client** – **[in]** MQTT API client handle
- **topic** – **[in]** Topic to unsubscribe from

Returns
lwcellOK on success, member of lwcellr_t otherwise

lwcellr_t **lwcell_mqtt_client_api_publish**(lwcell_mqtt_client_api_p client, const char *topic, const void *data, size_t btw, *lwcell_mqtt_qos_t* qos, uint8_t retain)

Publish new packet to MQTT network.

Parameters

- **client** – **[in]** MQTT API client handle
- **topic** – **[in]** Topic to publish on
- **data** – **[in]** Data to send
- **btw** – **[in]** Number of bytes to send for data parameter
- **qos** – **[in]** Quality of service. This parameter can be a value of *lwcell_mqtt_qos_t*
- **retain** – **[in]** Set to 1 for retain flag, 0 otherwise

Returns
lwcellOK on success, member of lwcellr_t otherwise

uint8_t **lwcell_mqtt_client_api_is_connected**(lwcell_mqtt_client_api_p client)

Check if client MQTT connection is active.

Parameters
client – **[in]** MQTT API client handle

Returns
1 on success, 0 otherwise

lwcellr_t **lwcell_mqtt_client_api_receive**(lwcell_mqtt_client_api_p client, *lwcell_mqtt_client_api_buf_p* *p, uint32_t timeout)

Receive next packet in specific timeout time.

Note: This function can be called from separate thread than the rest of API function, which allows you to handle receive data separated with custom timeout

---

Parameters

- **client** – **[in]** MQTT API client handle

- **p** – **[in]** Pointer to output buffer
- **timeout** – **[in]** Maximal time to wait before function returns timeout

Returns
lwcellOK on success, lwcellCLOSED if MQTT is closed, lwcellTIMEOUT on timeout

void **lwcell_mqtt_client_api_buf_free**(*lwcell_mqtt_client_api_buf_p* p)

Free buffer memory after usage.

Parameters
p – **[in]** Buffer to free

struct **lwcell_mqtt_client_api_buf_t**

#include <lwcell_mqtt_client_api.h> MQTT API RX buffer.

Public Members

char ***topic**

Topic data

size_t **topic_len**

Topic length

uint8_t ***payload**

Payload data

size_t **payload_len**

Payload length

lwcell_mqtt_qos_t **qos**

Quality of service

Netconn API

Netconn API is addon on top of existing connection module and allows sending and receiving data with sequential API calls, similar to *POSIX socket* API.

It can operate in client mode and uses operating system features, such as message queues and semaphore to link non-blocking callback API for connections with sequential API for application thread.

Note: Connection API does not directly allow receiving data with sequential and linear code execution. All is based on connection event system. Netconn adds this functionality as it is implemented on top of regular connection API.

Warning: Netconn API are designed to be called from application threads ONLY. It is not allowed to call any of *netconn API* functions from within interrupt or callback event functions.

Netconn client

Fig. 3: Netconn API client block diagram

Above block diagram shows basic architecture of netconn client application. There is always one application thread (in green) which calls *netconn API* functions to interact with connection API in synchronous mode.

Every netconn connection uses dedicated structure to handle message queue for data received packet buffers. Each time new packet is received (red block, *data received event*), reference to it is written to message queue of netconn structure, while application thread reads new entries from the same queue to get packets.

Listing 15: Netconn client example

```
#include "netconn_client.h"
#include "lwcell/lwcell.h"
#include "lwcell/lwcell_network_api.h"

#if LWCELL_CFG_NETCONN

/**
 * \brief           Host and port settings
 */
#define NETCONN_HOST "example.com"
#define NETCONN_PORT 80

/**
 * \brief           Request header to send on successful connection
 */
static const char request_header[] = ""
                                     "GET / HTTP/1.1\r\n"
                                     "Host: " NETCONN_HOST "\r\n"
                                     "Connection: close\r\n"
                                     "\r\n";

/**
 * \brief           Netconn client thread implementation
 * \param[in]       arg: User argument
 */
void
netconn_client_thread(void const* arg) {
    lwcellr_t res;
    lwcell_pbuf_p pbuf;
    lwcell_netconn_p client;
    lwcell_sys_sem_t* sem = (void*)arg;

    /* Request attach to network */
    while (lwcell_network_request_attach() != lwcellOK) {
        lwcell_delay(1000);
    }

    /*
     * First create a new instance of netconn
     * connection and initialize system message boxes
```

(continues on next page)

(continued from previous page)

```
     * to accept received packet buffers
     */
    if ((client = lwcell_netconn_new(LWCELL_NETCONN_TYPE_TCP)) != NULL) {
        /*
         * Connect to external server as client
         * with custom NETCONN_CONN_HOST and CONN_PORT values
         *
         * Function will block thread until we are successfully connected (or not) to server
         */
        if ((res = lwcell_netconn_connect(client, NETCONN_HOST, NETCONN_PORT)) == lwcellOK) {
            printf("Connected to %s\r\n", NETCONN_HOST);

            /* Send data to server */
            if ((res = lwcell_netconn_write(client, request_header, sizeof(request_header) - 1)) == lwcellOK) {
                res = lwcell_netconn_flush(client); /* Flush data to output */
            }
            if (res == lwcellOK) { /* Were data sent? */
                printf("Data were successfully sent to server\r\n");

                /*
                 * Since we sent HTTP request,
                 * we are expecting some data from server
                 * or at least forced connection close from remote side
                 */
                do {
                    /*
                     * Receive single packet of data
                     *
                     * Function will block thread until new packet
                     * is ready to be read from remote side
                     *
                     * After function returns, don't forgot the check value.
                     * Returned status will give you info in case connection
                     * was closed too early from remote side
                     */
                    res = lwcell_netconn_receive(client, &pbuf);
                    if (res
                        == lwcellCLOSED) { /* Was the connection closed? This can be checked by return status of receive function */
                        printf("Connection closed by remote side...\r\n");
                        break;
                    } else if (res == lwcellTIMEOUT) {
                        printf("Netconn timeout while receiving data. You may try multiple readings before deciding to "
                               "close manually\r\n");
                    }

                    if (res == lwcellOK && pbuf != NULL) { /* Make sure we have valid packet buffer */
```

(continues on next page)

(continued from previous page)

```
                        /*
                         * At this point read and manipulate
                         * with received buffer and check if you expect more data
                         *
                         * After you are done using it, it is important
                         * you free the memory otherwise memory leaks will appear
                         */
                        printf("Received new data packet of %d bytes\r\n", (int)lwcell_
↪pbuf_length(pbuf, 1));
                        lwcell_pbuf_free_s(&pbuf); /* Free the memory after usage */
                    }
                } while (1);
            } else {
                printf("Error writing data to remote host!\r\n");
            }

            /*
             * Check if connection was closed by remote server
             * and in case it wasn't, close it manually
             */
            if (res != lwcellCLOSED) {
                lwcell_netconn_close(client);
            }
        } else {
            printf("Cannot connect to remote host %s:%d!\r\n", NETCONN_HOST, NETCONN_
↪PORT);
        }
        lwcell_netconn_delete(client); /* Delete netconn structure */
    }
    lwcell_network_request_detach(); /* Detach from network */

    if (lwcell_sys_sem_isvalid(sem)) {
        lwcell_sys_sem_release(sem);
    }
    lwcell_sys_thread_terminate(NULL); /* Terminate current thread */
}

#endif /* LWCELL_CFG_NETCONN */
```

Non-blocking receive

By default, netconn API is written to only work in separate application thread, dedicated for network connection processing. Because of that, by default every function is fully blocking. It will wait until result is ready to be used by application.

It is, however, possible to enable timeout feature for receiving data only. When this feature is enabled, *lwcell_netconn_receive()* will block for maximal timeout set with *lwcell_netconn_set_receive_timeout()* function.

When enabled, if there is no received data for timeout amount of time, function will return with timeout status and application needs to process it accordingly.

Tip: *LWCELL_CFG_NETCONN_RECEIVE_TIMEOUT* must be set to 1 to use this feature.

group **LWCELL_NETCONN**

Network connection.

Defines

LWCELL_NETCONN_RECEIVE_NO_WAIT

Receive data with no timeout.

Note: Used with *lwcell_netconn_set_receive_timeout* function

LWCELL_NETCONN_FLAG_FLUSH

Immediate flush after netconn write

Typedefs

typedef struct lwcell_netconn ***lwcell_netconn_p**

Netconn object structure.

Enums

enum **lwcell_netconn_type_t**

Netconn connection type.

Values:

enumerator **LWCELL_NETCONN_TYPE_TCP** = LWCELL_CONN_TYPE_TCP

TCP connection

enumerator **LWCELL_NETCONN_TYPE_UDP** = LWCELL_CONN_TYPE_UDP

UDP connection

enumerator **LWCELL_NETCONN_TYPE_SSL** = LWCELL_CONN_TYPE_SSL

TCP connection over SSL

Functions

lwcell_netconn_p **lwcell_netconn_new**(*lwcell_netconn_type_t* type)

Create new netconn connection.

Parameters
type – **[in]** Netconn connection type

Returns
New netconn connection on success, NULL otherwise

lwcellr_t **lwcell_netconn_delete**(*lwcell_netconn_p* nc)

Delete netconn connection.

Parameters
nc – **[in]** Netconn handle

Returns
lwcellOK on success, member of lwcellr_t enumeration otherwise

lwcellr_t **lwcell_netconn_connect**(*lwcell_netconn_p* nc, const char *host, lwcell_port_t port)

Connect to server as client.

Parameters

- **nc** – **[in]** Netconn handle
- **host** – **[in]** Pointer to host, such as domain name or IP address in string format
- **port** – **[in]** Target port to use

Returns
lwcellOK if successfully connected, member of lwcellr_t otherwise

lwcellr_t **lwcell_netconn_receive**(*lwcell_netconn_p* nc, lwcell_pbuf_p *pbuf)

Receive data from connection.

Parameters

- **nc** – **[in]** Netconn handle used to receive from
- **pbuf** – **[in]** Pointer to pointer to save new receive buffer to. When function returns, user must check for valid pbuf value `pbuf != NULL`

Returns
lwcellOK when new data ready,

Returns
lwcellCLOSED when connection closed by remote side,

Returns
lwcellTIMEOUT when receive timeout occurs

Returns
Any other member of lwcellr_t otherwise

lwcellr_t **lwcell_netconn_close**(*lwcell_netconn_p* nc)

Close a netconn connection.

Parameters
nc – **[in]** Netconn handle to close

Returns
lwcellOK on success, member of lwcellr_t enumeration otherwise

int8_t **lwcell_netconn_getconnnum**(*lwcell_netconn_p* nc)

Get connection number used for netconn.

Parameters
nc – **[in]** Netconn handle

Returns
-1 on failure, connection number between 0 and *LWCELL_CFG_MAX_CONNS* otherwise

void **lwcell_netconn_set_receive_timeout**(*lwcell_netconn_p* nc, uint32_t timeout)

Set timeout value for receiving data.

When enabled, *lwcell_netconn_receive* will only block for up to *timeout* value and will return if no new data within this time

Parameters

- **nc** – **[in]** Netconn handle
- **timeout** – **[in]** Timeout in units of milliseconds. Set to 0 to disable timeout feature. Function blocks until data receive or connection closed Set to > 0 to set maximum milliseconds to wait before timeout Set to *LWCELL_NETCONN_RECEIVE_NO_WAIT* to enable non-blocking receive

uint32_t **lwcell_netconn_get_receive_timeout**(*lwcell_netconn_p* nc)

Get netconn receive timeout value.

Parameters
nc – **[in]** Netconn handle

Returns
Timeout in units of milliseconds. If value is 0, timeout is disabled (wait forever)

lwcellr_t **lwcell_netconn_write**(*lwcell_netconn_p* nc, const void *data, size_t btw)

Write data to connection output buffers.

Note: This function may only be used on TCP or SSL connections

Parameters

- **nc** – **[in]** Netconn handle used to write data to
- **data** – **[in]** Pointer to data to write
- **btw** – **[in]** Number of bytes to write

Returns
lwcellOK on success, member of lwcellr_t enumeration otherwise

lwcellr_t **lwcell_netconn_write_ex**(*lwcell_netconn_p* nc, const void *data, size_t btw, uint16_t flags)

Extended version of *lwcell_netconn_write* with additional option to set custom flags.

Note: It is recommended to use this for full features support

Parameters

- **nc** – **[in]** Netconn handle used to write data to

- **data** – **[in]** Pointer to data to write
- **btw** – **[in]** Number of bytes to write
- **flags** – Bitwise-ORed set of flags for netconn. Flags start with LWCELL_NETCONN_FLAG_xxx

Returns
lwcellOK on success, member of lwcellr_t enumeration otherwise

lwcellr_t **lwcell_netconn_flush**(*lwcell_netconn_p* nc)

Flush buffered data on netconn *TCP/SSL* connection.

Note: This function may only be used on *TCP/SSL* connection

Parameters
nc – **[in]** Netconn handle to flush data

Returns
lwcellOK on success, member of lwcellr_t enumeration otherwise

lwcellr_t **lwcell_netconn_send**(*lwcell_netconn_p* nc, const void *data, size_t btw)

Send data on *UDP* connection to default IP and port.

Parameters

- **nc** – **[in]** Netconn handle used to send
- **data** – **[in]** Pointer to data to write
- **btw** – **[in]** Number of bytes to write

Returns
lwcellOK on success, member of lwcellr_t enumeration otherwise

lwcellr_t **lwcell_netconn_sendto**(*lwcell_netconn_p* nc, const lwcell_ip_t *ip, lwcell_port_t port, const void *data, size_t btw)

Send data on *UDP* connection to specific IP and port.

Note: Use this function in case of UDP type netconn

Parameters

- **nc** – **[in]** Netconn handle used to send
- **ip** – **[in]** Pointer to IP address
- **port** – **[in]** Port number used to send data
- **data** – **[in]** Pointer to data to write
- **btw** – **[in]** Number of bytes to write

Returns
lwcellOK on success, member of lwcellr_t enumeration otherwise

5.4 Examples and demos

Various examples are provided for fast library evaluation on embedded systems. These are prepared and maintained for 2 platforms, but could be easily extended to more platforms:

- WIN32 examples, prepared as *CMake* projects, ready for *MSYS2 GCC compiler*
- ARM Cortex-M examples for STM32, prepared as STM32CubeIDE GCC projects. These are also supported in *Visual Studio Code* through *CMake* and *ninja* build system. Dedicated tutorial is available to get started in *VSCode*.

Note: Library is platform agnostic and can be used on many different products

5.4.1 Example architectures

There are many platforms available today on a market, however supporting them all would be tough task for single person. Therefore it has been decided to support (for purpose of examples) 2 platforms only, *WIN32* and *STM32*.

WIN32

Examples for *WIN32* are CMake-ready and *VSCode*-ready. It utilizes CMake-presets feature to let you select the example and compile it directly.

- Make sure you have installed GCC compiler and is in env path (you can get it through MSYS2 packet manager)
- Install ninja and cmake and make them available in the path (you can get all through MSYS2 packet manager)
- Go to *examples win32* folder, open vscode there or run cmd: `cmake --preset <project name>` to configure cmake and later `cmake --build --preset <project name>` to compile the project

Application opens *COM* port, set in the low-level driver. External USB to UART converter (FTDI-like device) is necessary in order to connect to *GSM* device.

Note: *GSM* device is connected with *USB to UART converter* only by *RX* and *TX* pins.

Device driver is located in `/lwcell/src/system/lwcell_ll_win32.c`

STM32

Embedded market is supported by many vendors and STMicroelectronics is, with their STM32 series of microcontrollers, one of the most important players. There are numerous amount of examples and topics related to this architecture.

Examples for *STM32* are natively supported with STM32CubeIDE, an official development IDE from STMicroelectronics.

You can run examples on one of official development boards, available in repository examples.

Table 1: Supported development boards

Board name	GSM settings				Debug settings		
	UART	MTX	MRX	RST	UART	MDTX	MDRX
STM32F429ZI-Nucleo	USART6	PC6	PC7	PC5	USART3	PD8	PD9

Pins to connect with GSM device:

- *MTX*: MCU TX pin, connected to GSM RX pin
- *MRX*: MCU RX pin, connected to GSM TX pin
- *RST*: MCU output pin to control reset state of GSM device

Other pins are for your information and are used for debugging purposes on board.

- MDTX: MCU Debug TX pin, connected via on-board ST-Link to PC
- MDRX: MCU Debug RX pin, connected via on-board ST-Link to PC
- Baudrate is always set to `921600` bauds

5.4.2 Examples list

Here is a list of all examples coming with this library.

Tip: Examples are located in `/examples/` folder in downloaded package. Check *Download library* section to get your package.

Tip: Do not forget to set PIN & PUK codes of your SIM card before running any of examples. Open `/snippets/sim_manager.c` and update `pin_code` and `puk_code` variables.

Device info

Simple example which prints basic device information:

- Device Manufacturer
- Device Model
- Device serial number
- Device revision number

MQTT Client API

Similar to *MQTT Client* examples, but it uses separate thread to process events in blocking mode. Application does not use events to process data, rather it uses blocking API to receive packets

Netconn client

Netconn client is based on sequential API. It starts connection to server, sends initial request and then waits to receive data.

Processing is in separate thread and fully sequential, no callbacks or events.

Call

Call example answers received call. If GSM device supports calls and has microphone/speaker connected to module itself, it can simply communicate over voice.

Call & SMS

This example shows how to receive a call and send reply with SMS. When application receives call, it hangs-up immediately and sends back SMS asking caller to send SMS instead.

When application receives SMS, it will send same SMS content back to the sender's number.

SMS Send receive

It demonstrates sending and receiving SMS either in events or using thread processing.

5.5 Changelog

```
# Changelog

## Develop

- Split CMakeLists.txt files between library and executable
- Change license year to 2022
- Timeout: Module returns ERRMEM if no memory to allocate block
- MQTT: update client to be consistent with client from LwESP library
- Port: Add ThreadX port aligned with LwESP library
- Add optional keep-alive system period event
- Add `LWGSM ` prefix for debug messages
- Update code style with astyle
- Add `.clang-format` draft
- Delete `lwgsm_datetime_t` and use generic `struct tm` instead
- Rename project from `lwgsm` to `lwcell`, indicating cellular

## v0.1.1

- Update CMSIS OS driver to support FreeRTOS aware kernel
- Update to support library.json for Platform.IO

## v0.1.0

- First release
- Support for SIM800/SIM900 for the moment
- Added AT commands as per documentation
- Sequential API for network supported
```

5.6 Authors

List of authors and contributors to the library

```
Tilen Majerle <tilen.majerle@gmail.com>
Tilen Majerle <tilen@majerle.eu>
yoyo <yonglanyouyou@gmail.com>
Tomas Strand <tomas@fik1.net>
vsanisimova <70365668+vsanisimova@users.noreply.github.com>
Ilya Kargapolov <d3vil.st@gmail.com>
Arthur Blomnik <arthur@blomnik.org.ua>
sa100 <sergio.rudenko@gmail.com>
```

INDEX

L