
LwCELL

Tilen MAJERLE

Jul 20, 2023

CONTENTS

1	Features	3
2	Requirements	5
3	Contribute	7
4	License	9
5	Table of contents	11
5.1	Getting started	11
5.2	User manual	14
5.3	API reference	64
5.4	Examples and demos	104
5.5	Changelog	106
	Index	109

Welcome to the documentation for version .

LwCELL is lightweight, platform independent, cellular modem AT commands parser, targeting (as of today) communication 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](#)

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

REQUIREMENTS

- C compiler
- Supported GSM Physical device

CONTRIBUTE

Fresh contributions are always welcome. Simple instructions to proceed:

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

Alternatively you may:

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

LICENSE

MIT License

Copyright (c) 2023 Tilen MAJERLE

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

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

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

TABLE OF CONTENTS

5.1 Getting started

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 primarily 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

- 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 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"`.

List of configuration options are 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

```
1  /**
2   * \file          lwcell_opts_template.h
3   * \brief        Template config file
4   */
5
6  /**
```

(continues on next page)

(continued from previous page)

```

7  * Copyright (c) 2023 Tilen MAJERLE
8  *
9  * Permission is hereby granted, free of charge, to any person
10 * obtaining a copy of this software and associated documentation
11 * files (the "Software"), to deal in the Software without restriction,
12 * including without limitation the rights to use, copy, modify, merge,
13 * publish, distribute, sublicense, and/or sell copies of the Software,
14 * and to permit persons to whom the Software is furnished to do so,
15 * subject to the following conditions:
16 *
17 * The above copyright notice and this permission notice shall be
18 * included in all copies or substantial portions of the Software.
19 *
20 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
21 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
22 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
23 * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
24 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
25 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
26 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
27 * OTHER DEALINGS IN THE SOFTWARE.
28 *
29 * This file is part of LwCELL - Lightweight cellular modem AT library.
30 *
31 * Author:          Tilen MAJERLE <tilen@majerle.eu>
32 * Version:         v0.1.1
33 */
34 #ifndef LWCELL_OPTS_HDR_H
35 #define LWCELL_OPTS_HDR_H
36
37 /* Rename this file to "lwcell_opts.h" for your application */
38
39 /*
40 * Open "include/lwcell/lwcell_opt.h" and
41 * copy & replace here settings you want to change values
42 */
43
44 #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 successfully 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 *Ø*, *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 response 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 as well 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 excluding 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

```

1  /**
2   * \file          lwcell_netconn.c
3   * \brief         API functions for sequential calls
4   */
5
6  /**
7   * Copyright (c) 2023 Tilen MAJERLE
8   *
9   * Permission is hereby granted, free of charge, to any person
10  * obtaining a copy of this software and associated documentation
11  * files (the "Software"), to deal in the Software without restriction,
12  * including without limitation the rights to use, copy, modify, merge,
13  * publish, distribute, sublicense, and/or sell copies of the Software,
14  * and to permit persons to whom the Software is furnished to do so,
15  * subject to the following conditions:
16  *
17  * The above copyright notice and this permission notice shall be
18  * included in all copies or substantial portions of the Software.
19  *
20  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
21  * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
22  * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
23  * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
24  * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
25  * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
26  * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
27  * OTHER DEALINGS IN THE SOFTWARE.
28  */

```

(continues on next page)

(continued from previous page)

```

29  * This file is part of LwCELL - Lightweight cellular modem AT library.
30  *
31  * Author:          Tilen MAJERLE <tilen@majerle.eu>
32  * Version:         v0.1.1
33  */
34  #include "lwcell/lwcell_netconn.h"
35  #include "lwcell/lwcell_conn.h"
36  #include "lwcell/lwcell_mem.h"
37  #include "lwcell/lwcell_private.h"
38
39  #if LWCELL_CFG_NETCONN || __DOXYGEN__
40
41  /* Check conditions */
42  #if !LWCELL_CFG_CONN
43  #error "LWCELL_CFG_CONN must be enabled for NETCONN API!"
44  #endif /* !LWCELL_CFG_CONN */
45
46  #if LWCELL_CFG_NETCONN_RECEIVE_QUEUE_LEN < 2
47  #error "LWCELL_CFG_NETCONN_RECEIVE_QUEUE_LEN must be greater or equal to 2"
48  #endif /* LWCELL_CFG_NETCONN_RECEIVE_QUEUE_LEN < 2 */
49
50  /**
51   * \brief          Sequential API structure
52   */
53  typedef struct lwcell_netconn {
54      struct lwcell_netconn* next;    /*!< Linked list entry */
55
56      lwcell_netconn_type_t type;     /*!< Netconn type */
57
58      size_t rcv_packets;             /*!< Number of received packets so far on this_
↳ connection */
59      lwcell_conn_p conn;             /*!< Pointer to actual connection */
60
61      lwcell_sys_mbox_t mbox_receive; /*!< Message queue for receive mbox */
62
63      lwcell_linbuff_t buff;          /*!< Linear buffer structure */
64
65      uint16_t conn_timeout;          /*!< Connection timeout in units of seconds when
66                                       netconn is in server (listen) mode.
67                                       Connection will be automatically_
↳ closed if there is no
68                                       data exchange in time. Set to `0`_
↳ when timeout feature is disabled. */
69
70  #if LWCELL_CFG_NETCONN_RECEIVE_TIMEOUT || __DOXYGEN__
71      uint32_t rcv_timeout; /*!< Receive timeout in unit of milliseconds */
72  #endif
73  } lwcell_netconn_t;
74
75  static uint8_t rcv_closed = 0xFF;
76  static lwcell_netconn_t* netconn_list; /*!< Linked list of netconn entries */
77

```

(continues on next page)

(continued from previous page)

```

78 /**
79  * \brief          Flush all mboxs and clear possible used memories
80  * \param[in]      nc: Pointer to netconn to flush
81  * \param[in]      protect: Set to 1 to protect against multi-thread access
82  */
83 static void
84 flush_mboxes(lwcell_netconn_t* nc, uint8_t protect) {
85     lwcell_pbuf_p pbuf;
86     if (protect) {
87         lwcell_core_lock();
88     }
89     if (lwcell_sys_mbox_isvalid(&nc->mbox_receive)) {
90         while (lwcell_sys_mbox_getnow(&nc->mbox_receive, (void*)&pbuf)) {
91             if (pbuf != NULL && (uint8_t*)pbuf != (uint8_t*)&recv_closed) {
92                 lwcell_pbuf_free_s(&pbuf); /* Free received data buffers */
93             }
94         }
95         lwcell_sys_mbox_delete(&nc->mbox_receive); /* Delete message queue */
96         lwcell_sys_mbox_invalid(&nc->mbox_receive); /* Invalid handle */
97     }
98     if (protect) {
99         lwcell_core_unlock();
100     }
101 }
102
103 /**
104  * \brief          Callback function for every server connection
105  * \param[in]      evt: Pointer to callback structure
106  * \return         Member of \ref lwcellr_t enumeration
107  */
108 static lwcellr_t
109 netconn_evt(lwcell_evt_t* evt) {
110     lwcell_conn_p conn;
111     lwcell_netconn_t* nc = NULL;
112     uint8_t close = 0;
113
114     conn = lwcell_conn_get_from_evt(evt); /* Get connection from event */
115     switch (lwcell_evt_get_type(evt)) {
116         /*
117          * A new connection has been active
118          * and should be handled by netconn API
119          */
120         case LWCELL_EVT_CONN_ACTIVE: { /* A new connection active is active */
121             if (lwcell_conn_is_client(conn)) { /* Was connection started by us? */
122                 nc = lwcell_conn_get_arg(conn); /* Argument should be already set */
123                 if (nc != NULL) {
124                     nc->conn = conn; /* Save actual connection */
125                 } else {
126                     close = 1; /* Close this connection, invalid netconn_
127                                ↪ */
128                 }
129             } else {
130                 close = 1;
131             }
132         }
133     }

```

(continues on next page)

(continued from previous page)

```

129         LWCELL_DEBUGF(LWCELL_CFG_DBG_NETCONN | LWCELL_DBG_TYPE_TRACE | LWCELL_
↳DBG_LVL_WARNING,
130                     "[LWCELL NETCONN] Closing connection, it is not in client_
↳mode!\r\n");
131         close = 1; /* Close the connection at this point */
132     }
133
134     /* Decide if some events want to close the connection */
135     if (close) {
136         if (nc != NULL) {
137             lwcell_conn_set_arg(conn, NULL); /* Reset argument */
138             lwcell_netconn_delete(nc);      /* Free memory for API */
139         }
140         lwcell_conn_close(conn, 0);          /* Close the connection */
141         close = 0;
142     }
143     break;
144 }
145
146 /*
147  * We have a new data received which
148  * should have netconn structure as argument
149  */
150 case LWCELL_EVT_CONN_RECV: {
151     lwcell_pbuf_p pbuf;
152
153     nc = lwcell_conn_get_arg(conn);          /* Get API from connection */
154     pbuf = lwcell_evt_conn_recv_get_buff(evt); /* Get received buff */
155
156     lwcell_conn_recved(conn, pbuf);           /* Notify stack about received_
↳data */
157
158     lwcell_pbuf_ref(pbuf);                    /* Increase reference counter */
159     if (nc == NULL || !lwcell_sys_mbox_isvalid(&nc->mbox_receive)
160         || !lwcell_sys_mbox_putnow(&nc->mbox_receive, pbuf)) {
161         LWCELL_DEBUGF(LWCELL_CFG_DBG_NETCONN, "[LWCELL NETCONN] Ignoring more_
↳data for receive!\r\n");
162         lwcell_pbuf_free_s(&pbuf); /* Free pbuf */
163         return lwcellOKIGNOREMORE; /* Return OK to free the memory and ignore_
↳further data */
164     }
165     ++nc->rcv_packets; /* Increase number of received packets */
166     LWCELL_DEBUGF(LWCELL_CFG_DBG_NETCONN | LWCELL_DBG_TYPE_TRACE,
167                 "[LWCELL NETCONN] Received pbuf contains %d bytes. Handle_
↳written to receive mbox\r\n",
168                 (int)lwcell_pbuf_length(pbuf, 0));
169     break;
170 }
171
172 /* Connection was just closed */
173 case LWCELL_EVT_CONN_CLOSE: {
174     nc = lwcell_conn_get_arg(conn); /* Get API from connection */

```

(continues on next page)

(continued from previous page)

```

175
176     /*
177      * In case we have a netconn available,
178      * simply write pointer to received variable to indicate closed state
179      */
180     if (nc != NULL && lwcell_sys_mbox_isvalid(&nc->mbox_receive)) {
181         lwcell_sys_mbox_putnow(&nc->mbox_receive, (void*)&recv_closed);
182     }
183
184     break;
185 }
186 default: return lwcellERR;
187 }
188 return lwcellOK;
189 }
190
191 /**
192  * \brief          Global event callback function
193  * \param[in]      evt: Callback information and data
194  * \return         \ref lwcellOK on success, member of \ref lwcellr_t otherwise
195  */
196 static lwcellr_t
197 lwcell_evt(lwcell_evt_t* evt) {
198     switch (lwcell_evt_get_type(evt)) {
199         default: break;
200     }
201     return lwcellOK;
202 }
203
204 /**
205  * \brief          Create new netconn connection
206  * \param[in]      type: Netconn connection type
207  * \return         New netconn connection on success, `NULL` otherwise
208  */
209 lwcell_netconn_p
210 lwcell_netconn_new(lwcell_netconn_type_t type) {
211     lwcell_netconn_t* a;
212     static uint8_t first = 1;
213
214     /* Register only once! */
215     lwcell_core_lock();
216     if (first) {
217         first = 0;
218         lwcell_evt_register(lwcell_evt); /* Register global event function */
219     }
220     lwcell_core_unlock();
221     a = lwcell_mem_calloc(1, sizeof(*a)); /* Allocate memory for core object */
222     if (a != NULL) {
223         a->type = type; /* Save netconn type */
224         a->conn_timeout = 0; /* Default connection timeout */
225         if (!lwcell_sys_mbox_create(
226             &a->mbox_receive,

```

(continues on next page)

(continued from previous page)

```

227         LWCELL_CFG_NETCONN_RECEIVE_QUEUE_LEN)) { /* Allocate memory for
↳receiving message box */
228         LWCELL_DEBUGF(LWCELL_CFG_DBG_NETCONN | LWCELL_DBG_TYPE_TRACE | LWCELL_DBG_
↳LVL_DANGER,
229                     "[LWCELL NETCONN] Cannot create receive MBOX\r\n");
230         goto free_ret;
231     }
232     lwcell_core_lock();
233     if (netconn_list == NULL) { /* Add new netconn to the existing list */
234         netconn_list = a;
235     } else {
236         a->next = netconn_list; /* Add it to beginning of the list */
237         netconn_list = a;
238     }
239     lwcell_core_unlock();
240 }
241 return a;
242 free_ret:
243 if (lwcell_sys_mbox_isvalid(&a->mbox_receive)) {
244     lwcell_sys_mbox_delete(&a->mbox_receive);
245     lwcell_sys_mbox_invalid(&a->mbox_receive);
246 }
247 if (a != NULL) {
248     lwcell_mem_free_s((void*)&a);
249 }
250 return NULL;
251 }
252
253 /**
254  * \brief          Delete netconn connection
255  * \param[in]      nc: Netconn handle
256  * \return         \ref lwcellOK on success, member of \ref lwcellr_t enumeration
↳otherwise
257  */
258 lwcellr_t
259 lwcell_netconn_delete(lwcell_netconn_p nc) {
260     LWCELL_ASSERT(nc != NULL);
261
262     lwcell_core_lock();
263     flush_mboxes(nc, 0); /* Clear mboxes */
264
265     /* Remove netconn from linkedlist */
266     if (netconn_list == nc) {
267         netconn_list = netconn_list->next; /* Remove first from linked list */
268     } else if (netconn_list != NULL) {
269         lwcell_netconn_p tmp, prev;
270         /* Find element on the list */
271         for (prev = netconn_list, tmp = netconn_list->next; tmp != NULL; prev = tmp, tmp
↳= tmp->next) {
272             if (nc == tmp) {
273                 prev->next = tmp->next; /* Remove tmp from linked list */
274                 break;

```

(continues on next page)

(continued from previous page)

```

275     }
276 }
277 }
278 lwcell_core_unlock();
279
280 lwcell_mem_free_s((void**)&nc);
281 return lwcellOK;
282 }
283
284 /**
285  * \brief      Connect to server as client
286  * \param[in]  nc: Netconn handle
287  * \param[in]  host: Pointer to host, such as domain name or IP address in string
288  * \format
289  * \param[in]  port: Target port to use
290  * \return     \ref lwcellOK if successfully connected, member of \ref lwcellr_t
291  * \otherwise
292  */
293 lwcellr_t
294 lwcell_netconn_connect(lwcell_netconn_p nc, const char* host, lwcell_port_t port) {
295     lwcellr_t res;
296
297     LWCELL_ASSERT(nc != NULL);
298     LWCELL_ASSERT(host != NULL);
299     LWCELL_ASSERT(port > 0);
300
301     /*
302      * Start a new connection as client and:
303      *
304      * - Set current netconn structure as argument
305      * - Set netconn callback function for connection management
306      * - Start connection in blocking mode
307      */
308     res = lwcell_conn_start(NULL, (lwcell_conn_type_t)nc->type, host, port, nc, netconn_
309     ↪ evt, 1);
310     return res;
311 }
312
313 /**
314  * \brief      Write data to connection output buffers
315  * \note       This function may only be used on TCP or SSL connections
316  * \param[in]  nc: Netconn handle used to write data to
317  * \param[in]  data: Pointer to data to write
318  * \param[in]  btw: Number of bytes to write
319  * \return     \ref lwcellOK on success, member of \ref lwcellr_t enumeration
320  * \otherwise
321  */
322 lwcellr_t
323 lwcell_netconn_write(lwcell_netconn_p nc, const void* data, size_t btw) {
324     size_t len, sent;
325     const uint8_t* d = data;
326     lwcellr_t res;

```

(continues on next page)

(continued from previous page)

```

323     LWCELL_ASSERT(nc != NULL);
324     LWCELL_ASSERT(nc->type == LWCELL_NETCONN_TYPE_TCP || nc->type == LWCELL_NETCONN_TYPE_
325     ↪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
336     ↪effectiveness)
337     */
338
339     /* Step 1 */
340     if (nc->buff.buff != NULL) {                                /* Is there a write buffer
341     ↪ready to accept more data? */
342         len = LWCELL_MIN(nc->buff.len - nc->buff.ptr, btw); /* Get number of bytes we
343     ↪can write to buffer */
344         if (len > 0) {
345             LWCELL_MEMCPY(&nc->buff.buff[nc->buff.ptr], data, len); /* Copy memory to
346     ↪temporary write buffer */
347             d += len;
348             nc->buff.ptr += len;
349             btw -= len;
350         }
351
352         /* Step 1.1 */
353         if (nc->buff.ptr == nc->buff.len) {
354             res = lwcell_conn_send(nc->conn, nc->buff.buff, nc->buff.len, &sent, 1);
355
356             lwcell_mem_free_s((void**)&nc->buff.buff);
357             if (res != lwcellOK) {
358                 return res;
359             }
360         } else {
361             return lwcellOK; /* Buffer is not yet full yet */
362         }
363     }
364
365     /* Step 2 */
366     if (btw >= LWCELL_CFG_CONN_MAX_DATA_LEN) {
367         size_t rem;
368         rem = btw % LWCELL_CFG_CONN_MAX_DATA_LEN;                                /* Get remaining bytes
369     ↪for max data length */
370         res = lwcell_conn_send(nc->conn, d, btw - rem, &sent, 1); /* Write data directly
371     ↪*/
372         if (res != lwcellOK) {
373             return res;
374         }
375     }

```

(continues on next page)

(continued from previous page)

```

368     }
369     d += sent; /* Advance in data pointer */
370     btw -= sent; /* Decrease remaining data to send */
371 }
372
373 if (btw == 0) { /* Sent everything? */
374     return lwcellOK;
375 }
376
377 /* Step 3 */
378 if (nc->buff.buff == NULL) { /* Check if we should allocate a new
↪buffer */
379     nc->buff.buff = lwcell_mem_malloc(sizeof(*nc->buff.buff) * LWCELL_CFG_CONN_MAX_
↪DATA_LEN);
380     nc->buff.len = LWCELL_CFG_CONN_MAX_DATA_LEN; /* Save buffer length */
381     nc->buff.ptr = 0; /* Save buffer pointer */
382 }
383
384 /* Step 4 */
385 if (nc->buff.buff != NULL) { /* Memory available? */
386     LWCELL_MEMCPY(&nc->buff.buff[nc->buff.ptr], d, btw); /* Copy data to buffer */
387     nc->buff.ptr += btw;
388 } else { /* Still no memory
↪available? */
389     return lwcell_conn_send(nc->conn, data, btw, NULL, 1); /* Simply send directly
↪blocking */
390 }
391 return lwcellOK;
392 }
393
394 /**
395  * \brief Extended version of \ref lwcell_netconn_write with additional
396  * option to set custom flags.
397  *
398  * \note It is recommended to use this for full features support
399  *
400  * \param[in] nc: Netconn handle used to write data to
401  * \param[in] data: Pointer to data to write
402  * \param[in] btw: Number of bytes to write
403  * \param flags: Bitwise-ORed set of flags for netconn.
404  *               Flags start with \ref LWCELL_NETCONN_FLAG_XXX
405  * \return \ref lwcellOK on success, member of \ref lwcellr_t enumeration
↪otherwise
406  */
407 lwcellr_t
408 lwcell_netconn_write_ex(lwcell_netconn_p nc, const void* data, size_t btw, uint16_t
↪flags) {
409     lwcellr_t res = lwcell_netconn_write(nc, data, btw);
410     if (res == lwcellOK) {
411         if (flags & LWCELL_NETCONN_FLAG_FLUSH) {
412             res = lwcell_netconn_flush(nc);
413         }

```

(continues on next page)

(continued from previous page)

```

414     }
415     return res;
416 }
417
418 /**
419  * \brief      Flush buffered data on netconn \e TCP/SSL connection
420  * \note       This function may only be used on \e TCP/SSL connection
421  * \param[in]  nc: Netconn handle to flush data
422  * \return     \ref lwcellOK on success, member of \ref lwcellr_t enumeration
423  * \otherwise
424  */
425 lwcellr_t
426 lwcell_netconn_flush(lwcell_netconn_p nc) {
427     LWCELL_ASSERT(nc != NULL);
428     LWCELL_ASSERT(nc->type == LWCELL_NETCONN_TYPE_TCP || nc->type == LWCELL_NETCONN_TYPE_
429     ↪ SSL);
430     LWCELL_ASSERT(lwcell_conn_is_active(nc->conn));
431
432     /*
433      * In case we have data in write buffer,
434      * flush them out to network
435      */
436     if (nc->buff.buff != NULL) {                                     /* Check
437     ↪ remaining data */
438         if (nc->buff.ptr > 0) {                                       /* Do we
439     ↪ have data in current buffer? */
440             lwcell_conn_send(nc->conn, nc->buff.buff, nc->buff.ptr, NULL, 1); /* Send
441     ↪ data */
442         }
443         lwcell_mem_free_s((void*)&nc->buff.buff);
444     }
445     return lwcellOK;
446 }
447
448 /**
449  * \brief      Send data on \e UDP connection to default IP and port
450  * \param[in]  nc: Netconn handle used to send
451  * \param[in]  data: Pointer to data to write
452  * \param[in]  btw: Number of bytes to write
453  * \return     \ref lwcellOK on success, member of \ref lwcellr_t enumeration
454  * \otherwise
455  */
456 lwcellr_t
457 lwcell_netconn_send(lwcell_netconn_p nc, const void* data, size_t btw) {
458     LWCELL_ASSERT(nc != NULL);
459     LWCELL_ASSERT(nc->type == LWCELL_NETCONN_TYPE_UDP);
460     LWCELL_ASSERT(lwcell_conn_is_active(nc->conn));
461
462     return lwcell_conn_send(nc->conn, data, btw, NULL, 1);
463 }
464
465 /**

```

(continues on next page)

(continued from previous page)

```

460  * \brief          Send data on \e UDP connection to specific IP and port
461  * \note           Use this function in case of UDP type netconn
462  * \param[in]      nc: Netconn handle used to send
463  * \param[in]      ip: Pointer to IP address
464  * \param[in]      port: Port number used to send data
465  * \param[in]      data: Pointer to data to write
466  * \param[in]      btw: Number of bytes to write
467  * \return          \ref lwcellOK on success, member of \ref lwcellr_t enumeration
↳ otherwise
468  */
469  lwcellr_t
470  lwcell_netconn_sendto(lwcell_netconn_p nc, const lwcell_ip_t* ip, lwcell_port_t port,
↳ const void* data, size_t btw) {
471      LWCELL_ASSERT(nc != NULL);
472      LWCELL_ASSERT(nc->type == LWCELL_NETCONN_TYPE_UDP);
473      LWCELL_ASSERT(lwcell_conn_is_active(nc->conn));
474
475      return lwcell_conn_sendto(nc->conn, ip, port, data, btw, NULL, 1);
476  }
477
478  /**
479  * \brief          Receive data from connection
480  * \param[in]      nc: Netconn handle used to receive from
481  * \param[in]      pbuf: Pointer to pointer to save new receive buffer to.
482  *                When function returns, user must check for valid pbuf value `pbuf`
↳ != NULL`
483  * \return          \ref lwcellOK when new data ready,
484  * \return          \ref lwcellCLOSED when connection closed by remote side,
485  * \return          \ref lwcellTIMEOUT when receive timeout occurs
486  * \return          Any other member of \ref lwcellr_t otherwise
487  */
488  lwcellr_t
489  lwcell_netconn_receive(lwcell_netconn_p nc, lwcell_pbuf_p* pbuf) {
490      LWCELL_ASSERT(nc != NULL);
491      LWCELL_ASSERT(pbuf != NULL);
492
493      *pbuf = NULL;
494      #if LWCELL_CFG_NETCONN_RECEIVE_TIMEOUT
495      /*
496       * Wait for new received data for up to specific timeout
497       * or throw error for timeout notification
498       */
499      if (nc->rcv_timeout == LWCELL_NETCONN_RECEIVE_NO_WAIT) {
500          if (!lwcell_sys_mbox_getnow(&nc->mbox_receive, (void**)pbuf)) {
501              return lwcellTIMEOUT;
502          }
503      } else if (lwcell_sys_mbox_get(&nc->mbox_receive, (void**)pbuf, nc->rcv_timeout) ==
↳ LWCELL_SYS_TIMEOUT) {
504          return lwcellTIMEOUT;
505      }
506      #else /* LWCELL_CFG_NETCONN_RECEIVE_TIMEOUT */
507      /* Forever wait for new receive packet */

```

(continues on next page)

(continued from previous page)

```

508     lwcell_sys_mbox_get(&nc->mbox_receive, (void**)pbuf, 0);
509 #endif /* !LWCELL_CFG_NETCONN_RECEIVE_TIMEOUT */
510
511     /* Check if connection closed */
512     if ((uint8_t*)(*pbuf) == (uint8_t*)&recv_closed) {
513         *pbuf = NULL; /* Reset pbuf */
514         return lwcellCLOSED;
515     }
516     return lwcellOK; /* We have data available */
517 }
518
519 /**
520  * \brief          Close a netconn connection
521  * \param[in]      nc: Netconn handle to close
522  * \return         \ref lwcellOK on success, member of \ref lwcellr_t enumeration
523  * \otherwise
524  */
525 lwcellr_t
526 lwcell_netconn_close(lwcell_netconn_p nc) {
527     lwcell_conn_p conn;
528
529     LWCELL_ASSERT(nc != NULL);
530     LWCELL_ASSERT(nc->conn != NULL);
531     LWCELL_ASSERT(lwcell_conn_is_active(nc->conn));
532
533     lwcell_netconn_flush(nc); /* Flush data and ignore result */
534     conn = nc->conn;
535     nc->conn = NULL;
536
537     lwcell_conn_set_arg(conn, NULL); /* Reset argument */
538     lwcell_conn_close(conn, 1);      /* Close the connection */
539     flush_mboxes(nc, 1);             /* Flush message queues */
540     return lwcellOK;
541 }
542
543 /**
544  * \brief          Get connection number used for netconn
545  * \param[in]      nc: Netconn handle
546  * \return         `-1` on failure, connection number between `0` and \ref LWCELL_CFG_
547  * \MAX_CONNS otherwise
548  */
549 int8_t
550 lwcell_netconn_getconnnum(lwcell_netconn_p nc) {
551     if (nc != NULL && nc->conn != NULL) {
552         return lwcell_conn_getnum(nc->conn);
553     }
554     return -1;
555 }
556
557 #if LWCELL_CFG_NETCONN_RECEIVE_TIMEOUT || __DOXYGEN__

```

(continues on next page)

(continued from previous page)

```

558  * \brief          Set timeout value for receiving data.
559  *
560  * When enabled, \ref lwcell_netconn_receive will only block for up to
561  * \e timeout value and will return if no new data within this time
562  *
563  * \param[in]      nc: Netconn handle
564  * \param[in]      timeout: Timeout in units of milliseconds.
565  *                  Set to `0` to disable timeout feature. Function blocks until data_
↪ receive or connection closed
566  *                  Set to `> 0` to set maximum milliseconds to wait before timeout
567  *                  Set to \ref LWCELL_NETCONN_RECEIVE_NO_WAIT to enable non-
↪ blocking receive
568  */
569  void
570  lwcell_netconn_set_receive_timeout(lwcell_netconn_p nc, uint32_t timeout) {
571      nc->rcv_timeout = timeout;
572  }
573
574  /**
575  * \brief          Get netconn receive timeout value
576  * \param[in]      nc: Netconn handle
577  * \return         Timeout in units of milliseconds.
578  *                  If value is `0`, timeout is disabled (wait forever)
579  */
580  uint32_t
581  lwcell_netconn_get_receive_timeout(lwcell_netconn_p nc) {
582      return nc->rcv_timeout; /* Return receive timeout */
583  }
584
585  #endif /* LWCELL_CFG_NETCONN_RECEIVE_TIMEOUT || __DOXYGEN__ */
586
587  #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

```

1  #include "client.h"
2  #include "lwcell/lwcell.h"
3  #include "lwcell/lwcell_network_api.h"
4

```

(continues on next page)

(continued from previous page)

```

5  /* Host parameter */
6  #define CONN_HOST          "example.com"
7  #define CONN_PORT          80
8
9  static lwcellr_t  conn_callback_func(lwcell_evt_t* evt);
10
11 /**
12  * \brief          Request data for connection
13  */
14 static const
15 uint8_t req_data[] = ""
16             "GET / HTTP/1.1\r\n"
17             "Host: " CONN_HOST "\r\n"
18             "Connection: close\r\n"
19             "\r\n";
20
21 /**
22  * \brief          Start a new connection(s) as client
23  */
24 void
25 client_connect(void) {
26     lwcellr_t res;
27
28     /* Attach to GSM network */
29     lwcell_network_request_attach();
30
31     /* Start a new connection as client in non-blocking mode */
32     if ((res = lwcell_conn_start(NULL, LWCELL_CONN_TYPE_TCP, "example.com", 80, NULL,
33     ↪ conn_callback_func, 0)) == lwcellOK) {
34         printf("Connection to " CONN_HOST " started...\r\n");
35     } else {
36         printf("Cannot start connection to " CONN_HOST "!\r\n");
37     }
38 }
39
40 /**
41  * \brief          Event callback function for connection-only
42  * \param[in]      evt: Event information with data
43  * \return         \ref lwcellOK on success, member of \ref lwcellr_t otherwise
44  */
45 static lwcellr_t
46 conn_callback_func(lwcell_evt_t* evt) {
47     lwcell_conn_p conn;
48     lwcellr_t res;
49     uint8_t conn_num;
50
51     conn = lwcell_conn_get_from_evt(evt);          /* Get connection handle from event */
52     if (conn == NULL) {
53         return lwcellERR;
54     }
55     conn_num = lwcell_conn_getnum(conn);           /* Get connection number for
56     ↪ identification */

```

(continues on next page)

```

55     switch (lwcell_evt_get_type(evt)) {
56         case LWCELL_EVT_CONN_ACTIVE: {           /* Connection just active */
57             printf("Connection %d active!\r\n", (int)conn_num);
58             res = lwcell_conn_send(conn, req_data, sizeof(req_data) - 1, NULL, 0); /*
↳ Start sending data in non-blocking mode */
59             if (res == lwcellOK) {
60                 printf("Sending request data to server...\r\n");
61             } else {
62                 printf("Cannot send request data to server. Closing connection manually..
↳ .\r\n");
63                 lwcell_conn_close(conn, 0);          /* Close the connection */
64             }
65             break;
66         }
67         case LWCELL_EVT_CONN_CLOSE: {             /* Connection closed */
68             if (lwcell_evt_conn_close_is_forced(evt)) {
69                 printf("Connection %d closed by client!\r\n", (int)conn_num);
70             } else {
71                 printf("Connection %d closed by remote side!\r\n", (int)conn_num);
72             }
73             break;
74         }
75         case LWCELL_EVT_CONN_SEND: {              /* Data send event */
76             lwcellr_t res = lwcell_evt_conn_send_get_result(evt);
77             if (res == lwcellOK) {
78                 printf("Data sent successfully on connection %d...waiting to receive
↳ data from remote side...\r\n", (int)conn_num);
79             } else {
80                 printf("Error while sending data on connection %d!\r\n", (int)conn_num);
81             }
82             break;
83         }
84         case LWCELL_EVT_CONN_RECV: {              /* Data received from remote side */
85             lwcell_pbuf_p pbuf = lwcell_evt_conn_recv_get_buff(evt);
86             lwcell_conn_recved(conn, pbuf);        /* Notify stack about received pbuf */
87             printf("Received %d bytes on connection %d...\r\n", (int)lwcell_pbuf_
↳ length(pbuf, 1), (int)conn_num);
88             break;
89         }
90         case LWCELL_EVT_CONN_ERROR: {             /* Error connecting to server */
91             const char* host = lwcell_evt_conn_error_get_host(evt);
92             lwcell_port_t port = lwcell_evt_conn_error_get_port(evt);
93             printf("Error connecting to %s:%d\r\n", host, (int)port);
94             break;
95         }
96         default:
97             break;
98     }
99     return lwcellOK;
100 }

```

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

```

1  /* Somewhere in thread function */
2
3  /* Get device hostname in blocking mode */
4  /* Function returns actual result */
5  if (lwcell_sms_send("+0123456789", "text", NULL, NULL, 1 /* 1 means blocking call */) ==
    ↪ lwcellOK) {
6      /* At this point we have valid result from device */
7      printf("SMS sent successfully\r\n");
8  } else {
9      printf("Error trying to send SMS.\r\n");
10 }

```

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

```

1  /* Somewhere in thread function */
2
3  /* Get device hostname in blocking mode */
4  /* Function returns actual result */
5  if (lwcell_sms_send("+0123456789", "text", NULL, NULL, 1 /* 1 means blocking call */) ==
    ↪ lwcellOK) {
6      /* At this point we have valid result from device */
7      printf("SMS sent successfully\r\n");
8  } else {
9      printf("Error trying to send SMS.\r\n");
10 }

```

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

```

1  /* Hostname event function, called when lwcell_sms_send() function finishes */
2  void
3  sms_send_fn(lwcellr_t res, void* arg) {
4      /* Check actual result from device */
5      if (res == lwcellOK) {
6          printf("SMS sent successfully\r\n");
7      } else {
8          printf("Error trying to send SMS\r\n");
9      }
10 }
11
12 /* Somewhere in thread and/or other GSM event function */
13
14 /* Send SMS in non-blocking mode */
15 /* Function now returns if command has been sent to internal message queue */
16 if (lwcell_sms_send("number", "text message", sms_send_fn, NULL, 0 /* 0 means non-
    ↪ blocking call */) == lwcellOK) {
17     /* At this point we only know that command has been sent to queue */
18     printf("SMS send message command sent to queue.\r\n");
19 } else {
20     /* Error writing message to queue */
21     printf("Cannot send SMS send message command to queue. Maybe out of memory? Check
    ↪ result from function\r\n");
22 }

```

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

```

1  /* Hostname event function, called when lwcell_sms_send() function finishes */
2  void
3  sms_send_fn(lwcellr_t res, void* arg) {
4      /* Check actual result from device */
5      if (res == lwcellOK) {
6          printf("SMS sent successfully\r\n");
7      } else {
8          printf("Error trying to send SMS\r\n");
9      }
10 }
11
12 /* Check hostname */
13 void
14 check_hostname(void) {
15     char message[] = "text message";
16
17     /* Send SMS in non-blocking mode */
18     /* Function now returns if command has been sent to internal message queue */
19     /* It uses pointer to local data but w/o blocking command */
20     if (lwcell_sms_send("number", message, sms_send_fn, NULL, 0 /* 0 means non-blocking_
↪call */) == lwcellOK) {
21         /* At this point we only know that command has been sent to queue */
22         printf("SMS send message command sent to queue.\r\n");
23     } else {
24         /* Error writing message to queue */
25         printf("Cannot send SMS send message command to queue. Maybe out of memory?_
↪Check result from function\r\n");
26     }
27 }

```

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

```

1  /**
2   * \file          lwcell_ll_win32.c
3   * \brief         Low-level communication with GSM device for WIN32
4   */
5
6  /**
7   * Copyright (c) 2023 Tilen MAJERLE
8   *
9   * Permission is hereby granted, free of charge, to any person
10  * obtaining a copy of this software and associated documentation
11  * files (the "Software"), to deal in the Software without restriction,
12  * including without limitation the rights to use, copy, modify, merge,
13  * publish, distribute, sublicense, and/or sell copies of the Software,
14  * and to permit persons to whom the Software is furnished to do so,
15  * subject to the following conditions:
16  *

```

(continues on next page)

(continued from previous page)

```

17  * The above copyright notice and this permission notice shall be
18  * included in all copies or substantial portions of the Software.
19  *
20  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
21  * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
22  * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
23  * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
24  * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
25  * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
26  * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
27  * OTHER DEALINGS IN THE SOFTWARE.
28  *
29  * This file is part of LwCELL - Lightweight cellular modem AT library.
30  *
31  * Author:          Tilen MAJERLE <tilen@majerle.eu>
32  * Version:         v0.1.1
33  */
34 #include "lwcell/lwcell_input.h"
35 #include "lwcell/lwcell_mem.h"
36 #include "lwcell/lwcell_types.h"
37 #include "lwcell/lwcell_utils.h"
38 #include "system/lwcell_ll.h"
39 #include "system/lwcell_sys.h"
40 #include "windows.h"
41
42 #if !__DOXYGEN__
43
44 static uint8_t initialized = 0;
45 static HANDLE thread_handle;
46 static volatile HANDLE com_port;    /*!< COM port handle */
47 static uint8_t data_buffer[0x1000]; /*!< Received data array */
48
49 static void uart_thread(void* param);
50
51 /**
52  * \brief          Send data to GSM device, function called from GSM stack when we have
53  * ↪ data to send
54  * \param[in]      data: Pointer to data to send
55  * \param[in]      len: Number of bytes to send
56  * \return         Number of bytes sent
57  */
58 static size_t
59 send_data(const void* data, size_t len) {
60     DWORD written;
61     if (com_port != NULL) {
62 #if !LWCELL_CFG_AT_ECHO
63         const uint8_t* d = data;
64         HANDLE hConsole;
65
66         hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
67         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
68         for (DWORD i = 0; i < len; ++i) {

```

(continues on next page)

(continued from previous page)

```

68         printf("%c", d[i]);
69     }
70     SetConsoleTextAttribute(hConsole, FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_
↪BLUE);
71 #endif /* !LWCELL_CFG_AT_ECHO */
72
73     /* Write data to AT port */
74     WriteFile(com_port, data, len, &written, NULL);
75     FlushFileBuffers(com_port);
76     return written;
77 }
78 return 0;
79 }
80
81 /**
82  * \brief          Configure UART (USB to UART)
83  */
84 static uint8_t
85 configure_uart(uint32_t baudrate) {
86     DCB dcb = {0};
87     dcb.DCBlength = sizeof(dcb);
88
89     /*
90      * On first call,
91      * create virtual file on selected COM port and open it
92      * as generic read and write
93      */
94     if (!initialized) {
95         static const char* com_ports[] = {"\\\\.\\COM23", "\\\\.\\COM12", "\\\\.\\COM9",
↪"\\\\.\\COM8", "\\\\.\\COM4"};
96         for (size_t i = 0; i < sizeof(com_ports) / sizeof(com_ports[0]); ++i) {
97             com_port = CreateFileA(com_ports[i], GENERIC_READ | GENERIC_WRITE, 0, 0,
↪OPEN_EXISTING, 0, NULL);
98             if (GetCommState(com_port, &dcb)) {
99                 printf("COM PORT %s opened!\r\n", (const char*)com_ports[i]);
100                 break;
101             }
102         }
103     }
104
105     /* Configure COM port parameters */
106     if (GetCommState(com_port, &dcb)) {
107         COMMTIMEOUTS timeouts;
108
109         dcb.BaudRate = baudrate;
110         dcb.ByteSize = 8;
111         dcb.Parity = NOPARITY;
112         dcb.StopBits = ONESTOPBIT;
113
114         if (!SetCommState(com_port, &dcb)) {
115             printf("Cannot set COM PORT info\r\n");
116             return 0;

```

(continues on next page)

(continued from previous page)

```

117     }
118     if (GetCommTimeouts(com_port, &timeouts)) {
119         /* Set timeout to return immediately from ReadFile function */
120         timeouts.ReadIntervalTimeout = MAXDWORD;
121         timeouts.ReadTotalTimeoutConstant = 0;
122         timeouts.ReadTotalTimeoutMultiplier = 0;
123         if (!SetCommTimeouts(com_port, &timeouts)) {
124             printf("Cannot set COM PORT timeouts\r\n");
125         }
126         GetCommTimeouts(com_port, &timeouts);
127     } else {
128         printf("Cannot get COM PORT timeouts\r\n");
129         return 0;
130     }
131 } else {
132     printf("Cannot get COM PORT info\r\n");
133     return 0;
134 }
135
136 /* On first function call, create a thread to read data from COM port */
137 if (!initialized) {
138     lwcell_sys_thread_create(&thread_handle, "lwcell_ll_thread", uart_thread, NULL,
139 ↪ 0, 0);
140 }
141 return 1;
142 }
143
144 /**
145  * \brief          UART thread
146  */
147 static void
148 uart_thread(void* param) {
149     DWORD bytes_read;
150     lwcell_sys_sem_t sem;
151     FILE* file = NULL;
152
153     LWCELL_UNUSED(param);
154
155     lwcell_sys_sem_create(&sem, 0); /* Create semaphore for delay functions */
156
157     while (com_port == NULL) {
158         lwcell_sys_sem_wait(&sem, 1); /* Add some delay with yield */
159     }
160
161     fopen_s(&file, "log_file.txt", "w+"); /* Open debug file in write mode */
162     while (1) {
163         /*
164          * Try to read data from COM port
165          * and send it to upper layer for processing
166          */
167         do {
168             ReadFile(com_port, data_buffer, sizeof(data_buffer), &bytes_read, NULL);

```

(continues on next page)

(continued from previous page)

```

168     if (bytes_read > 0) {
169         HANDLE hConsole;
170         hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
171         SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
172         for (DWORD i = 0; i < bytes_read; ++i) {
173             printf("%c", data_buffer[i]);
174         }
175         SetConsoleTextAttribute(hConsole, FOREGROUND_RED | FOREGROUND_GREEN |
↪ FOREGROUND_BLUE);
176
177         /* Send received data to input processing module */
178     #if LWCELL_CFG_INPUT_USE_PROCESS
179         lwcell_input_process(data_buffer, (size_t)bytes_read);
180     #else /* LWCELL_CFG_INPUT_USE_PROCESS */
181         lwcell_input(data_buffer, (size_t)bytes_read);
182     #endif /* !LWCELL_CFG_INPUT_USE_PROCESS */
183
184         /* Write received data to output debug file */
185         if (file != NULL) {
186             fwrite(data_buffer, 1, bytes_read, file);
187             fflush(file);
188         }
189     }
190     } while (bytes_read == (DWORD)sizeof(data_buffer));
191
192     /* Implement delay to allow other tasks processing */
193     lwcell_sys_sem_wait(&sem, 1);
194 }
195 }
196
197 /**
198  * \brief          Callback function called from initialization process
199  *
200  * \note           This function may be called multiple times if AT baudrate is changed
↪ from application.
201  *                It is important that every configuration except AT baudrate is
↪ configured only once!
202  *
203  * \note           This function may be called from different threads in GSM stack when
↪ using OS.
204  *                When \ref LWCELL_CFG_INPUT_USE_PROCESS is set to 1, this function
↪ may be called from user UART thread.
205  *
206  * \param[in,out]  ll: Pointer to \ref lwcell_ll_t structure to fill data for
↪ communication functions
207  * \return         \ref lwcellOK on success, member of \ref lwcellr_t enumeration
↪ otherwise
208  */
209 lwcellr_t
210 lwcell_ll_init(lwcell_ll_t* ll) {
211     #if !LWCELL_CFG_MEM_CUSTOM
212         /* Step 1: Configure memory for dynamic allocations */

```

(continues on next page)

(continued from previous page)

```

213 static uint8_t memory[0x10000]; /* Create memory for dynamic allocations with
↳specific size */
214
215 /*
216  * Create memory region(s) of memory.
217  * If device has internal/external memory available,
218  * multiple memories may be used
219  */
220 lwcell_mem_region_t mem_regions[] = {{memory, sizeof(memory)}};
221 if (!initialized) {
222     lwcell_mem_assignmemory(mem_regions,
223                             LWCELL_ARRAYSIZE(mem_regions)); /* Assign memory for
↳allocations to GSM library */
224 }
225 #endif /* !LWCELL_CFG_MEM_CUSTOM */
226
227 /* Step 2: Set AT port send function to use when we have data to transmit */
228 if (!initialized) {
229     ll->send_fn = send_data; /* Set callback function to send data */
230 }
231
232 /* Step 3: Configure AT port to be able to send/receive data to/from GSM device */
233 if (!configure_uart(ll->uart.baudrate)) { /* Initialize UART for communication */
234     return lwcellERR;
235 }
236 initialized = 1;
237 return lwcellOK;
238 }
239
240 #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

```

1 /**
2  * \file          lwcell_ll_stm32.c
3  * \brief         Generic STM32 driver, included in various STM32 driver variants
4  */
5
6 /**

```

(continues on next page)

(continued from previous page)

```

7  * Copyright (c) 2023 Tilen MAJERLE
8  *
9  * Permission is hereby granted, free of charge, to any person
10 * obtaining a copy of this software and associated documentation
11 * files (the "Software"), to deal in the Software without restriction,
12 * including without limitation the rights to use, copy, modify, merge,
13 * publish, distribute, sublicense, and/or sell copies of the Software,
14 * and to permit persons to whom the Software is furnished to do so,
15 * subject to the following conditions:
16 *
17 * The above copyright notice and this permission notice shall be
18 * included in all copies or substantial portions of the Software.
19 *
20 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
21 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
22 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
23 * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
24 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
25 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
26 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
27 * OTHER DEALINGS IN THE SOFTWARE.
28 *
29 * This file is part of LwCELL - Lightweight cellular modem AT library.
30 *
31 * Author:          Tilen MAJERLE <tilen@majerle.eu>
32 * Version:         v0.1.1
33 */
34
35 /*
36  * How it works
37  *
38  * On first call to \ref lwcell_ll_init, new thread is created and processed in usart_ll_
↳ thread function.
39  * USART is configured in RX DMA mode and any incoming bytes are processed inside thread_
↳ function.
40  * DMA and USART implement interrupt handlers to notify main thread about new data ready_
↳ to send to upper layer.
41  *
42  * More about UART + RX DMA: https://github.com/MaJerle/stm32-usart-dma-rx-tx
43  *
44  * \ref LWCELL_CFG_INPUT_USE_PROCESS must be enabled in `lwcell_config.h` to use this_
↳ driver.
45  */
46 #include "lwcell/lwcell_input.h"
47 #include "lwcell/lwcell_mem.h"
48 #include "lwcell/lwcell_types.h"
49 #include "system/lwcell_ll.h"
50
51 #if !__DOXYGEN__
52
53 #if !LWCELL_CFG_INPUT_USE_PROCESS
54 #error "LWCELL_CFG_INPUT_USE_PROCESS must be enabled in `lwcell_config.h` to use this_
↳ driver."

```

(continues on next page)

(continued from previous page)

```

55 #endif /* LWCELL_CFG_INPUT_USE_PROCESS */
56
57 #if !defined(LWCELL_USART_DMA_RX_BUFF_SIZE)
58 #define LWCELL_USART_DMA_RX_BUFF_SIZE 0x1000
59 #endif /* !defined(LWCELL_USART_DMA_RX_BUFF_SIZE) */
60
61 #if !defined(LWCELL_MEM_SIZE)
62 #define LWCELL_MEM_SIZE 0x1000
63 #endif /* !defined(LWCELL_MEM_SIZE) */
64
65 #if !defined(LWCELL_USART_RDR_NAME)
66 #define LWCELL_USART_RDR_NAME RDR
67 #endif /* !defined(LWCELL_USART_RDR_NAME) */
68
69 /* USART memory */
70 static uint8_t usart_mem[LWCELL_USART_DMA_RX_BUFF_SIZE];
71 static uint8_t is_running, initialized;
72 static size_t old_pos;
73
74 /* USART thread */
75 static void usart_ll_thread(void* arg);
76 static osThreadId_t usart_ll_thread_id;
77
78 /* Message queue */
79 static osMessageQueueId_t usart_ll_mbox_id;
80
81 /**
82  * \brief      USART data processing
83  */
84 static void
85 usart_ll_thread(void* arg) {
86     size_t pos;
87
88     LWCELL_UNUSED(arg);
89
90     while (1) {
91         void* d;
92         /* Wait for the event message from DMA or USART */
93         osMessageQueueGet(usart_ll_mbox_id, &d, NULL, osWaitForever);
94
95         /* Read data */
96 #if defined(LWCELL_USART_DMA_RX_STREAM)
97         pos = sizeof(usart_mem) - LL_DMA_GetDataLength(LWCELL_USART_DMA, LWCELL_USART_
98 ↪DMA_RX_STREAM);
99 #else
100         pos = sizeof(usart_mem) - LL_DMA_GetDataLength(LWCELL_USART_DMA, LWCELL_USART_
101 ↪DMA_RX_CH);
102 #endif /* defined(LWCELL_USART_DMA_RX_STREAM) */
103         if (pos != old_pos && is_running) {
104             if (pos > old_pos) {
105                 lwcell_input_process(&usart_mem[old_pos], pos - old_pos);
106             } else {

```

(continues on next page)

(continued from previous page)

```

105         lwcell_input_process(&usart_mem[old_pos], sizeof(usart_mem) - old_pos);
106         if (pos > 0) {
107             lwcell_input_process(&usart_mem[0], pos);
108         }
109     }
110     old_pos = pos;
111     if (old_pos == sizeof(usart_mem)) {
112         old_pos = 0;
113     }
114 }
115 }
116 }
117
118 /**
119  * \brief          Configure UART using DMA for receive in double buffer mode and IDLE_
120  * \line detection
121  */
122 static void
123 configure_uart(uint32_t baudrate) {
124     static LL_USART_InitTypeDef usart_init;
125     static LL_DMA_InitTypeDef dma_init;
126     LL_GPIO_InitTypeDef gpio_init;
127
128     if (!initialized) {
129         /* Enable peripheral clocks */
130         LWCELL_USART_CLK;
131         LWCELL_USART_DMA_CLK;
132         LWCELL_USART_TX_PORT_CLK;
133         LWCELL_USART_RX_PORT_CLK;
134
135         #if defined(LWCELL_RESET_PIN)
136             LWCELL_RESET_PORT_CLK;
137         #endif /* defined(LWCELL_RESET_PIN) */
138
139         /* Global pin configuration */
140         LL_GPIO_StructInit(&gpio_init);
141         gpio_init.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
142         gpio_init.Pull = LL_GPIO_PULL_UP;
143         gpio_init.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
144         gpio_init.Mode = LL_GPIO_MODE_OUTPUT;
145
146         #if defined(LWCELL_RESET_PIN)
147             /* Configure RESET pin */
148             gpio_init.Pin = LWCELL_RESET_PIN;
149             LL_GPIO_Init(LWCELL_RESET_PORT, &gpio_init);
150         #endif /* defined(LWCELL_RESET_PIN) */
151
152         /* Configure USART pins */
153         gpio_init.Mode = LL_GPIO_MODE_ALTERNATE;
154
155         /* TX PIN */
156         gpio_init.Alternate = LWCELL_USART_TX_PIN_AF;

```

(continues on next page)

(continued from previous page)

```

156     gpio_init.Pin = LWCELL_USART_TX_PIN;
157     LL_GPIO_Init(LWCELL_USART_TX_PORT, &gpio_init);
158
159     /* RX PIN */
160     gpio_init.Alternate = LWCELL_USART_RX_PIN_AF;
161     gpio_init.Pin = LWCELL_USART_RX_PIN;
162     LL_GPIO_Init(LWCELL_USART_RX_PORT, &gpio_init);
163
164     /* Configure UART */
165     LL_USART_DeInit(LWCELL_USART);
166     LL_USART_StructInit(&usart_init);
167     usart_init.BaudRate = baudrate;
168     usart_init.DataWidth = LL_USART_DATAWIDTH_8B;
169     usart_init.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
170     usart_init.OverSampling = LL_USART_OVERSAMPLING_16;
171     usart_init.Parity = LL_USART_PARITY_NONE;
172     usart_init.StopBits = LL_USART_STOPBITS_1;
173     usart_init.TransferDirection = LL_USART_DIRECTION_TX_RX;
174     LL_USART_Init(LWCELL_USART, &usart_init);
175
176     /* Enable USART interrupts and DMA request */
177     LL_USART_EnableIT_IDLE(LWCELL_USART);
178     LL_USART_EnableIT_PE(LWCELL_USART);
179     LL_USART_EnableIT_ERROR(LWCELL_USART);
180     LL_USART_EnableDMAReq_RX(LWCELL_USART);
181
182     /* Enable USART interrupts */
183     NVIC_SetPriority(LWCELL_USART_IRQ, NVIC_EncodePriority(NVIC_
184     ↳GetPriorityGrouping(), 0x07, 0x00));
185     NVIC_EnableIRQ(LWCELL_USART_IRQ);
186
187     /* Configure DMA */
188     is_running = 0;
189     #if defined(LWCELL_USART_DMA_RX_STREAM)
190     LL_DMA_DeInit(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM);
191     dma_init.Channel = LWCELL_USART_DMA_RX_CH;
192 #else
193     LL_DMA_DeInit(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_CH);
194     dma_init.PeriphRequest = LWCELL_USART_DMA_RX_REQ_NUM;
195 #endif /* defined(LWCELL_USART_DMA_RX_STREAM) */
196     dma_init.PeriphOrM2MSrcAddress = (uint32_t)&LWCELL_USART->LWCELL_USART_RDR_NAME;
197     dma_init.MemoryOrM2MDstAddress = (uint32_t)usart_mem;
198     dma_init.Direction = LL_DMA_DIRECTION_PERIPH_TO_MEMORY;
199     dma_init.Mode = LL_DMA_MODE_CIRCULAR;
200     dma_init.PeriphOrM2MSrcIncMode = LL_DMA_PERIPH_NOINCREMENT;
201     dma_init.MemoryOrM2MDstIncMode = LL_DMA_MEMORY_INCREMENT;
202     dma_init.PeriphOrM2MSrcDataSize = LL_DMA_PDATAALIGN_BYTE;
203     dma_init.MemoryOrM2MDstDataSize = LL_DMA_MDATAALIGN_BYTE;
204     dma_init.NbData = sizeof(usart_mem);
205     dma_init.Priority = LL_DMA_PRIORITY_MEDIUM;
206     #if defined(LWCELL_USART_DMA_RX_STREAM)
207     LL_DMA_Init(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM, &dma_init);

```

(continues on next page)

(continued from previous page)

```

207 #else
208     LL_DMA_Init(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_CH, &dma_init);
209 #endif /* defined(LWCELL_USART_DMA_RX_STREAM) */
210
211     /* Enable DMA interrupts */
212 #if defined(LWCELL_USART_DMA_RX_STREAM)
213     LL_DMA_EnableIT_HT(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM);
214     LL_DMA_EnableIT_TC(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM);
215     LL_DMA_EnableIT_TE(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM);
216     LL_DMA_EnableIT_FE(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM);
217     LL_DMA_EnableIT_DME(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM);
218 #else
219     LL_DMA_EnableIT_HT(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_CH);
220     LL_DMA_EnableIT_TC(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_CH);
221     LL_DMA_EnableIT_TE(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_CH);
222 #endif /* defined(LWCELL_USART_DMA_RX_STREAM) */
223
224     /* Enable DMA interrupts */
225     NVIC_SetPriority(LWCELL_USART_DMA_RX_IRQ, NVIC_EncodePriority(NVIC_
↳ GetPriorityGrouping(), 0x07, 0x00));
226     NVIC_EnableIRQ(LWCELL_USART_DMA_RX_IRQ);
227
228     old_pos = 0;
229     is_running = 1;
230
231     /* Start DMA and USART */
232 #if defined(LWCELL_USART_DMA_RX_STREAM)
233     LL_DMA_EnableStream(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_STREAM);
234 #else
235     LL_DMA_EnableChannel(LWCELL_USART_DMA, LWCELL_USART_DMA_RX_CH);
236 #endif /* defined(LWCELL_USART_DMA_RX_STREAM) */
237     LL_USART_Enable(LWCELL_USART);
238 } else {
239     osDelay(10);
240     LL_USART_Disable(LWCELL_USART);
241     usart_init.BaudRate = baudrate;
242     LL_USART_Init(LWCELL_USART, &usart_init);
243     LL_USART_Enable(LWCELL_USART);
244 }
245
246 /* Create mbox and start thread */
247 if (usart_ll_mbox_id == NULL) {
248     usart_ll_mbox_id = osMessageQueueNew(10, sizeof(void*), NULL);
249 }
250 if (usart_ll_thread_id == NULL) {
251     const osThreadAttr_t attr = {.stack_size = 1024};
252     usart_ll_thread_id = osThreadNew(usart_ll_thread, usart_ll_mbox_id, &attr);
253 }
254 }
255
256 #if defined(LWCELL_RESET_PIN)
257 /**

```

(continues on next page)

(continued from previous page)

```

258  * \brief          Hardware reset callback
259  */
260  static uint8_t
261  reset_device(uint8_t state) {
262      if (state) { /* Activate reset line */
263          LL_GPIO_ResetOutputPin(LWCELL_RESET_PORT, LWCELL_RESET_PIN);
264      } else {
265          LL_GPIO_SetOutputPin(LWCELL_RESET_PORT, LWCELL_RESET_PIN);
266      }
267      return 1;
268  }
269  #endif /* defined(LWCELL_RESET_PIN) */
270
271  /**
272   * \brief          Send data to GSM device
273   * \param[in]      data: Pointer to data to send
274   * \param[in]      len: Number of bytes to send
275   * \return         Number of bytes sent
276   */
277  static size_t
278  send_data(const void* data, size_t len) {
279      const uint8_t* d = data;
280
281      for (size_t i = 0; i < len; ++i, ++d) {
282          LL_USART_TransmitData8(LWCELL_USART, *d);
283          while (!LL_USART_IsActiveFlag_TXE(LWCELL_USART)) {}
284      }
285      return len;
286  }
287
288  /**
289   * \brief          Callback function called from initialization process
290   * \note           This function may be called multiple times if AT baudrate is changed.
291   * \param[in,out]  ll: Pointer to \ref lwcell_ll_t structure to fill data for.
292   * \param[in]      baudrate: Baudrate to use on AT port
293   * \return         Member of \ref lwcellr_t enumeration
294   */
295  lwcellr_t
296  lwcell_ll_init(lwcell_ll_t* ll) {
297      #if !LWCELL_CFG_MEM_CUSTOM
298          static uint8_t memory[LWCELL_MEM_SIZE];
299          lwcell_mem_region_t mem_regions[] = {{memory, sizeof(memory)}};
300
301          if (!initialized) {
302              lwcell_mem_assignmemory(mem_regions, LWCELL_ARRAYSIZE(mem_regions)); /* Assign
303   * memory for allocations */
304          }
305      #endif /* !LWCELL_CFG_MEM_CUSTOM */
306
307      if (!initialized) {

```

(continues on next page)

(continued from previous page)

```

307     ll->send_fn = send_data; /* Set callback function to send data */
308 #if defined(LWCELL_RESET_PIN)
309     ll->reset_fn = reset_device; /* Set callback for hardware reset */
310 #endif /* defined(LWCELL_RESET_PIN) */
311 }
312
313 configure_uart(ll->uart.baudrate); /* Initialize UART for communication */
314 initialized = 1;
315 return lwcellOK;
316 }
317
318 /**
319  * \brief      Callback function to de-init low-level communication part
320  * \param[in,out] ll: Pointer to \ref lwcell_ll_t structure to fill data for.
321  * \return      \ref lwcellOK on success, member of \ref lwcellr_t enumeration.
322  * \otherwise
323  */
324 lwcellr_t
325 lwcell_ll_deinit(lwcell_ll_t* ll) {
326     if (usart_ll_mbox_id != NULL) {
327         osMessageQueueId_t tmp = usart_ll_mbox_id;
328         usart_ll_mbox_id = NULL;
329         osMessageQueueDelete(tmp);
330     }
331     if (usart_ll_thread_id != NULL) {
332         osThreadId_t tmp = usart_ll_thread_id;
333         usart_ll_thread_id = NULL;
334         osThreadTerminate(tmp);
335     }
336     initialized = 0;
337     LWCELL_UNUSED(ll);
338     return lwcellOK;
339 }
340
341 /**
342  * \brief      UART global interrupt handler
343  */
344 void
345 LWCELL_USART_IRQHANDLER(void) {
346     LL_USART_ClearFlag_IDLE(LWCELL_USART);
347     LL_USART_ClearFlag_PE(LWCELL_USART);
348     LL_USART_ClearFlag_FE(LWCELL_USART);
349     LL_USART_ClearFlag_ORE(LWCELL_USART);
350     LL_USART_ClearFlag_NE(LWCELL_USART);
351
352     if (usart_ll_mbox_id != NULL) {
353         void* d = (void*)1;
354         osMessageQueuePut(usart_ll_mbox_id, &d, 0, 0);
355     }
356 }

```

(continues on next page)

(continued from previous page)

```

357 /**
358  * \brief          UART DMA stream/channel handler
359  */
360 void
361 LWCELL_USART_DMA_RX_IRQHANDLER(void) {
362     LWCELL_USART_DMA_RX_CLEAR_TC;
363     LWCELL_USART_DMA_RX_CLEAR_HT;
364
365     if (usart_ll_mbox_id != NULL) {
366         void* d = (void*)1;
367         osMessageQueuePut(usart_ll_mbox_id, &d, 0, 0);
368     }
369 }
370
371 #endif /* !__DOXYGEN__ */

```

Example: System functions for WIN32

Listing 10: Actual header implementation of system functions for WIN32

```

1  /**
2  * \file           lwcell_sys_port.h
3  * \brief          WIN32 based system file implementation
4  */
5
6  /**
7  * Copyright (c) 2023 Tilen MAJERLE
8  *
9  * Permission is hereby granted, free of charge, to any person
10 * obtaining a copy of this software and associated documentation
11 * files (the "Software"), to deal in the Software without restriction,
12 * including without limitation the rights to use, copy, modify, merge,
13 * publish, distribute, sublicense, and/or sell copies of the Software,
14 * and to permit persons to whom the Software is furnished to do so,
15 * subject to the following conditions:
16 *
17 * The above copyright notice and this permission notice shall be
18 * included in all copies or substantial portions of the Software.
19 *
20 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
21 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
22 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
23 * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
24 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
25 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
26 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
27 * OTHER DEALINGS IN THE SOFTWARE.
28 *
29 * This file is part of LwCELL - Lightweight cellular modem AT library.
30 *
31 * Author:          Tilen MAJERLE <tilen@majerle.eu>

```

(continues on next page)

(continued from previous page)

```

32  * Version:          v0.1.1
33  */
34  #ifndef LWCELL_SYSTEM_PORT_HDR_H
35  #define LWCELL_SYSTEM_PORT_HDR_H
36
37  #include <stdint.h>
38  #include <stdlib.h>
39  #include "lwcell/lwcell_opt.h"
40  #include "windows.h"
41
42  #ifndef __cplusplus
43  extern "C" {
44  #endif /* __cplusplus */
45
46  #if LWCELL_CFG_OS && !__DOXYGEN__
47
48  typedef HANDLE lwcell_sys_mutex_t;
49  typedef HANDLE lwcell_sys_sem_t;
50  typedef HANDLE lwcell_sys_mbox_t;
51  typedef HANDLE lwcell_sys_thread_t;
52  typedef int lwcell_sys_thread_prio_t;
53
54  #define LWCELL_SYS_MUTEX_NULL ((HANDLE)0)
55  #define LWCELL_SYS_SEM_NULL ((HANDLE)0)
56  #define LWCELL_SYS_MBOX_NULL ((HANDLE)0)
57  #define LWCELL_SYS_TIMEOUT (INFINITE)
58  #define LWCELL_SYS_THREAD_PRIO (0)
59  #define LWCELL_SYS_THREAD_SS (4096)
60
61  #endif /* LWCELL_CFG_OS && !__DOXYGEN__ */
62
63  #ifndef __cplusplus
64  }
65  #endif /* __cplusplus */
66
67  #endif /* LWCELL_SYSTEM_PORT_HDR_H */

```

Listing 11: Actual implementation of system functions for WIN32

```

1  /**
2   * \file          lwcell_sys_win32.c
3   * \brief        System dependant functions for WIN32
4   */
5
6  /**
7   * Copyright (c) 2023 Tilen MAJERLE
8   *
9   * Permission is hereby granted, free of charge, to any person
10  * obtaining a copy of this software and associated documentation
11  * files (the "Software"), to deal in the Software without restriction,
12  * including without limitation the rights to use, copy, modify, merge,
13  * publish, distribute, sublicense, and/or sell copies of the Software,

```

(continues on next page)

(continued from previous page)

```

14  * and to permit persons to whom the Software is furnished to do so,
15  * subject to the following conditions:
16  *
17  * The above copyright notice and this permission notice shall be
18  * included in all copies or substantial portions of the Software.
19  *
20  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
21  * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
22  * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
23  * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
24  * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
25  * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
26  * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
27  * OTHER DEALINGS IN THE SOFTWARE.
28  *
29  * This file is part of LwCELL - Lightweight cellular modem AT library.
30  *
31  * Author:          Tilen MAJERLE <tilen@majerle.eu>
32  * Version:         v0.1.1
33  */
34 #include <stdlib.h>
35 #include <string.h>
36 #include "lwcell/lwcell_private.h"
37 #include "system/lwcell_sys.h"
38
39 #if !__DOXYGEN__
40
41 /**
42  * \brief          Custom message queue implementation for WIN32
43  */
44 typedef struct {
45     lwcell_sys_sem_t sem_not_empty; /*!< Semaphore indicates not empty */
46     lwcell_sys_sem_t sem_not_full;  /*!< Semaphore indicates not full */
47     lwcell_sys_sem_t sem;           /*!< Semaphore to lock access */
48     size_t in, out, size;
49     void* entries[1];
50 } win32_mbox_t;
51
52 static LARGE_INTEGER freq, sys_start_time;
53 static lwcell_sys_mutex_t sys_mutex; /* Mutex ID for main protection */
54
55 static uint8_t
56 mbox_is_full(win32_mbox_t* m) {
57     size_t size = 0;
58     if (m->in > m->out) {
59         size = (m->in - m->out);
60     } else if (m->out > m->in) {
61         size = m->size - m->out + m->in;
62     }
63     return size == m->size - 1;
64 }
65

```

(continues on next page)

(continued from previous page)

```

66 static uint8_t
67 mbox_is_empty(win32_mbox_t* m) {
68     return m->in == m->out;
69 }
70
71 static uint32_t
72 osKernelSysTick(void) {
73     LONGLONG ret;
74     LARGE_INTEGER now;
75
76     QueryPerformanceFrequency(&freq); /* Get frequency */
77     QueryPerformanceCounter(&now);    /* Get current time */
78     ret = now.QuadPart - sys_start_time.QuadPart;
79     return (uint32_t)(((ret)*1000) / freq.QuadPart);
80 }
81
82 uint8_t
83 lwcell_sys_init(void) {
84     QueryPerformanceFrequency(&freq);
85     QueryPerformanceCounter(&sys_start_time);
86
87     lwcell_sys_mutex_create(&sys_mutex);
88     return 1;
89 }
90
91 uint32_t
92 lwcell_sys_now(void) {
93     return osKernelSysTick();
94 }
95
96 uint8_t
97 lwcell_sys_protect(void) {
98     lwcell_sys_mutex_lock(&sys_mutex);
99     return 1;
100 }
101
102 uint8_t
103 lwcell_sys_unprotect(void) {
104     lwcell_sys_mutex_unlock(&sys_mutex);
105     return 1;
106 }
107
108 uint8_t
109 lwcell_sys_mutex_create(lwcell_sys_mutex_t* p) {
110     *p = CreateMutex(NULL, FALSE, NULL);
111     return *p != NULL;
112 }
113
114 uint8_t
115 lwcell_sys_mutex_delete(lwcell_sys_mutex_t* p) {
116     return CloseHandle(*p);
117 }

```

(continues on next page)

```

118
119 uint8_t
120 lwcell_sys_mutex_lock(lwcell_sys_mutex_t* p) {
121     DWORD ret;
122     ret = WaitForSingleObject(*p, INFINITE);
123     if (ret != WAIT_OBJECT_0) {
124         return 0;
125     }
126     return 1;
127 }
128
129 uint8_t
130 lwcell_sys_mutex_unlock(lwcell_sys_mutex_t* p) {
131     return (uint8_t)ReleaseMutex(*p);
132 }
133
134 uint8_t
135 lwcell_sys_mutex_isvalid(lwcell_sys_mutex_t* p) {
136     return p != NULL && *p != NULL;
137 }
138
139 uint8_t
140 lwcell_sys_mutex_invalid(lwcell_sys_mutex_t* p) {
141     *p = LWCELL_SYS_MUTEX_NULL;
142     return 1;
143 }
144
145 uint8_t
146 lwcell_sys_sem_create(lwcell_sys_sem_t* p, uint8_t cnt) {
147     HANDLE h;
148     h = CreateSemaphore(NULL, !!cnt, 1, NULL);
149     *p = h;
150     return *p != NULL;
151 }
152
153 uint8_t
154 lwcell_sys_sem_delete(lwcell_sys_sem_t* p) {
155     return CloseHandle(*p);
156 }
157
158 uint32_t
159 lwcell_sys_sem_wait(lwcell_sys_sem_t* p, uint32_t timeout) {
160     DWORD ret;
161
162     if (timeout == 0) {
163         ret = WaitForSingleObject(*p, INFINITE);
164         return 1;
165     } else {
166         ret = WaitForSingleObject(*p, timeout);
167         if (ret == WAIT_OBJECT_0) {
168             return 1;
169         } else {

```

(continues on next page)

(continued from previous page)

```

170         return LWCELL_SYS_TIMEOUT;
171     }
172 }
173
174
175 uint8_t
176 lwcell_sys_sem_release(lwcell_sys_sem_t* p) {
177     return ReleaseSemaphore(*p, 1, NULL);
178 }
179
180 uint8_t
181 lwcell_sys_sem_isvalid(lwcell_sys_sem_t* p) {
182     return p != NULL && *p != NULL;
183 }
184
185 uint8_t
186 lwcell_sys_sem_invalid(lwcell_sys_sem_t* p) {
187     *p = LWCELL_SYS_SEM_NULL;
188     return 1;
189 }
190
191 uint8_t
192 lwcell_sys_mbox_create(lwcell_sys_mbox_t* b, size_t size) {
193     win32_mbox_t* mbox;
194
195     *b = NULL;
196
197     mbox = malloc(sizeof(*mbox) + size * sizeof(void*));
198     if (mbox != NULL) {
199         memset(mbox, 0x00, sizeof(*mbox));
200         mbox->size = size + 1; /* Set it to 1 more as cyclic buffer has only one less_
↳ than size */
201         lwcell_sys_sem_create(&mbox->sem, 1);
202         lwcell_sys_sem_create(&mbox->sem_not_empty, 0);
203         lwcell_sys_sem_create(&mbox->sem_not_full, 0);
204         *b = mbox;
205     }
206     return *b != NULL;
207 }
208
209 uint8_t
210 lwcell_sys_mbox_delete(lwcell_sys_mbox_t* b) {
211     win32_mbox_t* mbox = *b;
212     lwcell_sys_sem_delete(&mbox->sem);
213     lwcell_sys_sem_delete(&mbox->sem_not_full);
214     lwcell_sys_sem_delete(&mbox->sem_not_empty);
215     free(mbox);
216     return 1;
217 }
218
219 uint32_t
220 lwcell_sys_mbox_put(lwcell_sys_mbox_t* b, void* m) {

```

(continues on next page)

(continued from previous page)

```

221 win32_mbox_t* mbox = *b;
222 uint32_t time = osKernelSysTick(); /* Get start time */
223
224 lwcell_sys_sem_wait(&mbox->sem, 0); /* Wait for access */
225
226 /*
227  * Since function is blocking until ready to write something to queue,
228  * wait and release the semaphores to allow other threads
229  * to process the queue before we can write new value.
230  */
231 while (mbox_is_full(mbox)) {
232     lwcell_sys_sem_release(&mbox->sem); /* Release semaphore */
233     lwcell_sys_sem_wait(&mbox->sem_not_full, 0); /* Wait for semaphore indicating
↳not full */
234     lwcell_sys_sem_wait(&mbox->sem, 0); /* Wait availability again */
235 }
236 mbox->entries[mbox->in] = m;
237 if (++mbox->in >= mbox->size) {
238     mbox->in = 0;
239 }
240 lwcell_sys_sem_release(&mbox->sem_not_empty); /* Signal non-empty state */
241 lwcell_sys_sem_release(&mbox->sem); /* Release access for other threads */
242 return osKernelSysTick() - time;
243 }
244
245 uint32_t
246 lwcell_sys_mbox_get(lwcell_sys_mbox_t* b, void** m, uint32_t timeout) {
247     win32_mbox_t* mbox = *b;
248     uint32_t time;
249
250     time = osKernelSysTick();
251
252     /* Get exclusive access to message queue */
253     if (lwcell_sys_sem_wait(&mbox->sem, timeout) == LWCELL_SYS_TIMEOUT) {
254         return LWCELL_SYS_TIMEOUT;
255     }
256     while (mbox_is_empty(mbox)) {
257         lwcell_sys_sem_release(&mbox->sem);
258         if (lwcell_sys_sem_wait(&mbox->sem_not_empty, timeout) == LWCELL_SYS_TIMEOUT) {
259             return LWCELL_SYS_TIMEOUT;
260         }
261         lwcell_sys_sem_wait(&mbox->sem, timeout);
262     }
263     *m = mbox->entries[mbox->out];
264     if (++mbox->out >= mbox->size) {
265         mbox->out = 0;
266     }
267     lwcell_sys_sem_release(&mbox->sem_not_full);
268     lwcell_sys_sem_release(&mbox->sem);
269
270     return osKernelSysTick() - time;
271 }

```

(continues on next page)

(continued from previous page)

```

272
273 uint8_t
274 lwcell_sys_mbox_putnow(lwcell_sys_mbox_t* b, void* m) {
275     win32_mbox_t* mbox = *b;
276
277     lwcell_sys_sem_wait(&mbox->sem, 0);
278     if (mbox_is_full(mbox)) {
279         lwcell_sys_sem_release(&mbox->sem);
280         return 0;
281     }
282     mbox->entries[mbox->in] = m;
283     if (mbox->in == mbox->out) {
284         lwcell_sys_sem_release(&mbox->sem_not_empty);
285     }
286     if (++mbox->in >= mbox->size) {
287         mbox->in = 0;
288     }
289     lwcell_sys_sem_release(&mbox->sem);
290     return 1;
291 }
292
293 uint8_t
294 lwcell_sys_mbox_getnow(lwcell_sys_mbox_t* b, void** m) {
295     win32_mbox_t* mbox = *b;
296
297     lwcell_sys_sem_wait(&mbox->sem, 0); /* Wait exclusive access */
298     if (mbox->in == mbox->out) {
299         lwcell_sys_sem_release(&mbox->sem); /* Release access */
300         return 0;
301     }
302
303     *m = mbox->entries[mbox->out];
304     if (++mbox->out >= mbox->size) {
305         mbox->out = 0;
306     }
307     lwcell_sys_sem_release(&mbox->sem_not_full); /* Queue not full anymore */
308     lwcell_sys_sem_release(&mbox->sem); /* Release semaphore */
309     return 1;
310 }
311
312 uint8_t
313 lwcell_sys_mbox_isvalid(lwcell_sys_mbox_t* b) {
314     return b != NULL && *b != NULL; /* Return status if message box is valid */
315 }
316
317 uint8_t
318 lwcell_sys_mbox_invalid(lwcell_sys_mbox_t* b) {
319     *b = LWCELL_SYS_MBOX_NULL; /* Invalidate message box */
320     return 1;
321 }
322
323 uint8_t

```

(continues on next page)

(continued from previous page)

```

324 lwcell_sys_thread_create(lwcell_sys_thread_t* t, const char* name, lwcell_sys_thread_fn_
↳ thread_func, void* const arg,
325                          size_t stack_size, lwcell_sys_thread_prio_t prio) {
326     HANDLE h;
327     DWORD id;
328
329     LWCELL_UNUSED(name);
330     LWCELL_UNUSED(stack_size);
331     LWCELL_UNUSED(prio);
332
333     h = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)thread_func, arg, 0, &id);
334     if (t != NULL) {
335         *t = h;
336     }
337     return h != NULL;
338 }
339
340 uint8_t
341 lwcell_sys_thread_terminate(lwcell_sys_thread_t* t) {
342     if (t == NULL) { /* Shall we terminate ourself? */
343         ExitThread(0);
344     } else {
345         /* We have known thread, find handle by looking at ID */
346         TerminateThread(*t, 0);
347     }
348     return 1;
349 }
350
351 uint8_t
352 lwcell_sys_thread_yield(void) {
353     /* Not implemented */
354     return 1;
355 }
356
357 #endif /* !__DOXYGEN__ */

```

Example: System functions for CMSIS-OS

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

```

1  /**
2   * \file          lwcell_sys_port.h
3   * \brief        System dependent functions for CMSIS-OS based operating system
4   */
5
6  /**
7   * Copyright (c) 2023 Tilen MAJERLE
8   *
9   * Permission is hereby granted, free of charge, to any person
10  * obtaining a copy of this software and associated documentation

```

(continues on next page)

(continued from previous page)

```

11  * files (the "Software"), to deal in the Software without restriction,
12  * including without limitation the rights to use, copy, modify, merge,
13  * publish, distribute, sublicense, and/or sell copies of the Software,
14  * and to permit persons to whom the Software is furnished to do so,
15  * subject to the following conditions:
16  *
17  * The above copyright notice and this permission notice shall be
18  * included in all copies or substantial portions of the Software.
19  *
20  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
21  * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
22  * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
23  * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
24  * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
25  * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
26  * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
27  * OTHER DEALINGS IN THE SOFTWARE.
28  *
29  * This file is part of LwCELL - Lightweight cellular modem AT library.
30  *
31  * Author:          Tilen MAJERLE <tilen@majerle.eu>
32  * Version:         v0.1.1
33  */
34 #ifndef LWCELL_SYSTEM_PORT_HDR_H
35 #define LWCELL_SYSTEM_PORT_HDR_H
36
37 #include <stdint.h>
38 #include <stdlib.h>
39 #include "cmsis_os.h"
40 #include "lwcell/lwcell_opt.h"
41
42 #ifdef __cplusplus
43 extern "C" {
44 #endif /* __cplusplus */
45
46 #if LWCELL_CFG_OS && !__DOXYGEN__
47
48 typedef osMutexId_t lwcell_sys_mutex_t;
49 typedef osSemaphoreId_t lwcell_sys_sem_t;
50 typedef osMessageQueueId_t lwcell_sys_mbox_t;
51 typedef osThreadId_t lwcell_sys_thread_t;
52 typedef osPriority_t lwcell_sys_thread_prio_t;
53
54 #define LWCELL_SYS_MUTEX_NULL ((lwcell_sys_mutex_t)0)
55 #define LWCELL_SYS_SEM_NULL ((lwcell_sys_sem_t)0)
56 #define LWCELL_SYS_MBOX_NULL ((lwcell_sys_mbox_t)0)
57 #define LWCELL_SYS_TIMEOUT ((uint32_t)osWaitForever)
58 #define LWCELL_SYS_THREAD_PRIO (osPriorityNormal)
59 #define LWCELL_SYS_THREAD_SS (512)
60
61 #endif /* LWCELL_CFG_OS && !__DOXYGEN__ */
62

```

(continues on next page)

(continued from previous page)

```

63 #ifdef __cplusplus
64 }
65 #endif /* __cplusplus */
66
67 #endif /* LWCELL_SYSTEM_PORT_HDR_H */

```

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

```

1  /**
2   * \file      lwcell_sys_cmsis_os.c
3   * \brief     System dependent functions for CMSIS-OS based operating system
4   */
5
6  /**
7   * Copyright (c) 2023 Tilen MAJERLE
8   *
9   * Permission is hereby granted, free of charge, to any person
10  * obtaining a copy of this software and associated documentation
11  * files (the "Software"), to deal in the Software without restriction,
12  * including without limitation the rights to use, copy, modify, merge,
13  * publish, distribute, sublicense, and/or sell copies of the Software,
14  * and to permit persons to whom the Software is furnished to do so,
15  * subject to the following conditions:
16  *
17  * The above copyright notice and this permission notice shall be
18  * included in all copies or substantial portions of the Software.
19  *
20  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
21  * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
22  * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
23  * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
24  * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
25  * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
26  * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
27  * OTHER DEALINGS IN THE SOFTWARE.
28  *
29  * This file is part of LwCELL - Lightweight cellular modem AT library.
30  *
31  * Author:      Tilen MAJERLE <tilen@majerle.eu>
32  * Version:     v0.1.1
33  */
34 #include "cmsis_os.h"
35 #include "system/lwcell_sys.h"
36
37 #if !__DOXYGEN__
38
39 static osMutexId_t sys_mutex;
40
41 uint8_t
42 lwcell_sys_init(void) {
43     lwcell_sys_mutex_create(&sys_mutex);

```

(continues on next page)

(continued from previous page)

```

44     return 1;
45 }
46
47 uint32_t
48 lwcell_sys_now(void) {
49     return osKernelGetTickCount();
50 }
51
52 uint8_t
53 lwcell_sys_protect(void) {
54     lwcell_sys_mutex_lock(&sys_mutex);
55     return 1;
56 }
57
58 uint8_t
59 lwcell_sys_unprotect(void) {
60     lwcell_sys_mutex_unlock(&sys_mutex);
61     return 1;
62 }
63
64 uint8_t
65 lwcell_sys_mutex_create(lwcell_sys_mutex_t* p) {
66     const osMutexAttr_t attr = {
67         .attr_bits = osMutexRecursive,
68         .name = "lwcell_mutex",
69     };
70     return (*p = osMutexNew(&attr)) != NULL;
71 }
72
73 uint8_t
74 lwcell_sys_mutex_delete(lwcell_sys_mutex_t* p) {
75     return osMutexDelete(*p) == osOK;
76 }
77
78 uint8_t
79 lwcell_sys_mutex_lock(lwcell_sys_mutex_t* p) {
80     return osMutexAcquire(*p, osWaitForever) == osOK;
81 }
82
83 uint8_t
84 lwcell_sys_mutex_unlock(lwcell_sys_mutex_t* p) {
85     return osMutexRelease(*p) == osOK;
86 }
87
88 uint8_t
89 lwcell_sys_mutex_isvalid(lwcell_sys_mutex_t* p) {
90     return p != NULL && *p != NULL;
91 }
92
93 uint8_t
94 lwcell_sys_mutex_invalid(lwcell_sys_mutex_t* p) {
95     *p = LWCELL_SYS_MUTEX_NULL;

```

(continues on next page)

(continued from previous page)

```

96     return 1;
97 }
98
99 uint8_t
100 lwcell_sys_sem_create(lwcell_sys_sem_t* p, uint8_t cnt) {
101     const osSemaphoreAttr_t attr = {
102         .name = "lwcell_sem",
103     };
104     return (*p = osSemaphoreNew(1, cnt > 0 ? 1 : 0, &attr)) != NULL;
105 }
106
107 uint8_t
108 lwcell_sys_sem_delete(lwcell_sys_sem_t* p) {
109     return osSemaphoreDelete(*p) == osOK;
110 }
111
112 uint32_t
113 lwcell_sys_sem_wait(lwcell_sys_sem_t* p, uint32_t timeout) {
114     uint32_t tick = osKernelSysTick();
115     return (osSemaphoreAcquire(*p, timeout == 0 ? osWaitForever : timeout) == osOK) ?
116         ↪(osKernelSysTick() - tick)
117         :
118         ↪LWCELL_SYS_TIMEOUT;
119 }
120
121 uint8_t
122 lwcell_sys_sem_release(lwcell_sys_sem_t* p) {
123     return osSemaphoreRelease(*p) == osOK;
124 }
125
126 uint8_t
127 lwcell_sys_sem_isvalid(lwcell_sys_sem_t* p) {
128     return p != NULL && *p != NULL;
129 }
130
131 uint8_t
132 lwcell_sys_sem_invalid(lwcell_sys_sem_t* p) {
133     *p = LWCELL_SYS_SEM_NULL;
134     return 1;
135 }
136
137 uint8_t
138 lwcell_sys_mbox_create(lwcell_sys_mbox_t* b, size_t size) {
139     const osMessageQueueAttr_t attr = {
140         .name = "lwcell_mbox",
141     };
142     return (*b = osMessageQueueNew(size, sizeof(void*), &attr)) != NULL;
143 }
144
145 uint8_t
146 lwcell_sys_mbox_delete(lwcell_sys_mbox_t* b) {
147     if (osMessageQueueGetCount(*b) > 0) {

```

(continues on next page)

(continued from previous page)

```

146     return 0;
147 }
148 return osMessageQueueDelete(*b) == osOK;
149 }
150
151 uint32_t
152 lwcell_sys_mbox_put(lwcell_sys_mbox_t* b, void* m) {
153     uint32_t tick = osKernelSysTick();
154     return osMessageQueuePut(*b, &m, 0, osWaitForever) == osOK ? (osKernelSysTick() -
↪ tick) : LWCELL_SYS_TIMEOUT;
155 }
156
157 uint32_t
158 lwcell_sys_mbox_get(lwcell_sys_mbox_t* b, void** m, uint32_t timeout) {
159     uint32_t tick = osKernelSysTick();
160     return osMessageQueueGet(*b, m, NULL, timeout == 0 ? osWaitForever : timeout) ==
↪ osOK ? (osKernelSysTick() - tick)
161
↪ : LWCELL_SYS_TIMEOUT;
162 }
163
164 uint8_t
165 lwcell_sys_mbox_putnow(lwcell_sys_mbox_t* b, void* m) {
166     return osMessageQueuePut(*b, &m, 0, 0) == osOK;
167 }
168
169 uint8_t
170 lwcell_sys_mbox_getnow(lwcell_sys_mbox_t* b, void** m) {
171     return osMessageQueueGet(*b, m, NULL, 0) == osOK;
172 }
173
174 uint8_t
175 lwcell_sys_mbox_isvalid(lwcell_sys_mbox_t* b) {
176     return b != NULL && *b != NULL;
177 }
178
179 uint8_t
180 lwcell_sys_mbox_invalid(lwcell_sys_mbox_t* b) {
181     *b = LWCELL_SYS_MBOX_NULL;
182     return 1;
183 }
184
185 uint8_t
186 lwcell_sys_thread_create(lwcell_sys_thread_t* t, const char* name, lwcell_sys_thread_fn_
↪ thread_func, void* const arg,
187                          size_t stack_size, lwcell_sys_thread_prio_t prio) {
188     lwcell_sys_thread_t id;
189     const osThreadAttr_t thread_attr = {.name = (char*)name,
190                                         .priority = (osPriority)prio,
191                                         .stack_size = stack_size > 0 ? stack_size :
↪ LWCELL_SYS_THREAD_SS};
192

```

(continues on next page)

(continued from previous page)

```
193     id = osThreadNew(thread_func, arg, &thread_attr);
194     if (t != NULL) {
195         *t = id;
196     }
197     return id != NULL;
198 }
199
200 uint8_t
201 lwcell_sys_thread_terminate(lwcell_sys_thread_t* t) {
202     if (t != NULL) {
203         osThreadTerminate(*t);
204     } else {
205         osThreadExit();
206     }
207     return 1;
208 }
209
210 uint8_t
211 lwcell_sys_thread_yield(void) {
212     osThreadYield();
213     return 1;
214 }
215
216 #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) [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 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 producer 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 *LW-CELL_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 *LW-CELL_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 guaranteed at least once, but it may be delivered multiple times with the same content

enumerator **LWCELL_MQTT_QOS_EXACTLY_ONCE** = 0x02

Delivery is guaranteed 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_rcv**
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

```

1  /*
2   * MQTT client API example with GSM device.
3   *
4   * Once device is connected to network,
5   * it will try to connect to mosquitto test server and start the MQTT.
6   *
7   * If successfully connected, it will publish data to "lwcell_mqtt_topic" topic every x_
  ↪ seconds.
8   *
9   * To check if data are sent, you can use mqtt-spy PC software to inspect
10  * test.mosquitto.org server and subscribe to publishing topic
11  */
12
13  #include "mqtt_client_api.h"
14  #include "lwcell/apps/lwcell_mqtt_client_api.h"
15  #include "lwcell/lwcell.h"
16  #include "lwcell/lwcell_mem.h"
17  #include "lwcell/lwcell_network_api.h"
18
19  /**
20   * \brief      Connection information for MQTT CONNECT packet
21   */
22  static const lwcell_mqtt_client_info_t mqtt_client_info = {
23      .keep_alive = 10,
24
25      /* Server login data */
26      .user = "8a215f70-a644-11e8-ac49-e932ed599553",
27      .pass = "26aa943f702e5e780f015cd048a91e8fb54cca28",
28
29      /* Device identifier address */
30      .id = "2c3573a0-0176-11e9-a056-c5cffe7f75f9",
31  };
32
33  /**
34   * \brief      Memory for temporary topic
35   */
36  static char mqtt_topic_str[256];
37
38  /**
39   * \brief      Generate random number and write it to string
40   * \param[out] str: Output string with new number
41   */
42  void
43  generate_random(char* str) {
44      static uint32_t random_beg = 0x8916;
45      random_beg = random_beg * 0x00123455 + 0x85654321;
46      sprintf(str, "%u", (unsigned)((random_beg >> 8) & 0xFFFF));
  
```

(continues on next page)

(continued from previous page)

```

47 }
48
49 /**
50  * \brief      MQTT client API thread
51  */
52 void
53 mqtt_client_api_thread(void const* arg) {
54     lwcell_mqtt_client_api_p client;
55     lwcell_mqtt_conn_status_t conn_status;
56     lwcell_mqtt_client_api_buf_p buf;
57     lwcellr_t res;
58     char random_str[10];
59
60     LWCELL_UNUSED(arg);
61
62     /* Request network attach */
63     while (lwcell_network_request_attach() != lwcellOK) {
64         lwcell_delay(1000);
65     }
66
67     /* Create new MQTT API */
68     if ((client = lwcell_mqtt_client_api_new(256, 128)) == NULL) {
69         goto terminate;
70     }
71
72     while (1) {
73         /* Make a connection */
74         printf("Joining MQTT server\r\n");
75
76         /* Try to join */
77         conn_status = lwcell_mqtt_client_api_connect(client, "mqtt.mydevices.com", 1883,
↪ &mqtt_client_info);
78         if (conn_status == LWCELL_MQTT_CONN_STATUS_ACCEPTED) {
79             printf("Connected and accepted!\r\n");
80             printf("Client is ready to subscribe and publish to new messages\r\n");
81         } else {
82             printf("Connect API response: %d\r\n", (int)conn_status);
83             lwcell_delay(5000);
84             continue;
85         }
86
87         /* Subscribe to topics */
88         sprintf(mqtt_topic_str, "v1/%s/things/%s/cmd/#", mqtt_client_info.user, mqtt_
↪ client_info.id);
89         if (lwcell_mqtt_client_api_subscribe(client, mqtt_topic_str, LWCELL_MQTT_QOS_AT_
↪ LEAST_ONCE) == lwcellOK) {
90             printf("Subscribed to topic\r\n");
91         } else {
92             printf("Problem subscribing to topic!\r\n");
93         }
94
95         while (1) {

```

(continues on next page)

(continued from previous page)

```

96      /* Receive MQTT packet with timeout */
97      if ((res = lwcell_mqtt_client_api_receive(client, &buf, 5000)) == lwcellOK) {
98          if (buf != NULL) {
99              printf("Publish received!\r\n");
100              printf("Topic: %s, payload: %s\r\n", buf->topic, buf->payload);
101              lwcell_mqtt_client_api_buf_free(buf);
102              buf = NULL;
103          }
104      } else if (res == lwcellCLOSED) {
105          printf("MQTT connection closed!\r\n");
106          break;
107      } else if (res == lwcellTIMEOUT) {
108          printf("Timeout on MQTT receive function. Manually publishing.\r\n");
109
110          /* Publish data on channel 1 */
111          generate_random(random_str);
112          sprintf(mqtt_topic_str, "v1/%s/things/%s/data/1", mqtt_client_info.user,
↪ mqtt_client_info.id);
113          lwcell_mqtt_client_api_publish(client, mqtt_topic_str, random_str,
↪ strlen(random_str),
114                                         LWCELL_MQTT_QOS_AT_LEAST_ONCE, 0);
115      }
116  }
117  goto terminate;
118 }
119
120 terminate:
121     lwcell_mqtt_client_api_delete(client);
122     lwcell_network_request_detach();
123     printf("MQTT client thread terminate\r\n");
124     lwcell_sys_thread_terminate(NULL);
125 }

```

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 add-on 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

```

1  #include "netconn_client.h"
2  #include "lwcell/lwcell.h"
3  #include "lwcell/lwcell_network_api.h"
4
5  #if LWCELL_CFG_NETCONN
6
7  /**
8   * \brief      Host and port settings
9   */
10 #define NETCONN_HOST "example.com"
11 #define NETCONN_PORT 80
12
13 /**
14  * \brief      Request header to send on successful connection
15  */
16 static const char request_header[] = ""
17                                     "GET / HTTP/1.1\r\n"
18                                     "Host: " NETCONN_HOST "\r\n"
19                                     "Connection: close\r\n"
20                                     "\r\n";
21
22 /**
23  * \brief      Netconn client thread implementation
24  * \param[in]  arg: User argument
25  */
26 void
27 netconn_client_thread(void const* arg) {
28     lwcellr_t res;
29     lwcell_pbuf_p pbuf;
30     lwcell_netconn_p client;
31     lwcell_sys_sem_t* sem = (void*)arg;
32
33     /* Request attach to network */
34     while (lwcell_network_request_attach() != lwcellOK) {
35         lwcell_delay(1000);
36     }
37
38     /*
39      * First create a new instance of netconn
40      * connection and initialize system message boxes
41      * to accept received packet buffers
42      */
43     if ((client = lwcell_netconn_new(LWCELL_NETCONN_TYPE_TCP)) != NULL) {
44         /*
45          * Connect to external server as client
46          * with custom NETCONN_HOST and NETCONN_PORT values
47          *
48          * Function will block thread until we are successfully connected (or not) to
49          ↪ server
49          */
50         if ((res = lwcell_netconn_connect(client, NETCONN_HOST, NETCONN_PORT)) == ↪
51             ↪ lwcellOK) {

```

(continues on next page)

(continued from previous page)

```

51     printf("Connected to %s\r\n", NETCONN_HOST);
52
53     /* Send data to server */
54     if ((res = lwcell_netconn_write(client, request_header, sizeof(request_
↪header) - 1)) == lwcellOK) {
55         res = lwcell_netconn_flush(client); /* Flush data to output */
56     }
57     if (res == lwcellOK) { /* Were data sent? */
58         printf("Data were successfully sent to server\r\n");
59
60         /*
61          * Since we sent HTTP request,
62          * we are expecting some data from server
63          * or at least forced connection close from remote side
64          */
65         do {
66             /*
67              * Receive single packet of data
68              *
69              * Function will block thread until new packet
70              * is ready to be read from remote side
71              *
72              * After function returns, don't forgot the check value.
73              * Returned status will give you info in case connection
74              * was closed too early from remote side
75              */
76             res = lwcell_netconn_receive(client, &pbuf);
77             if (res
78                 == lwcellCLOSED) { /* Was the connection closed? This can be
↪checked by return status of receive function */
79                 printf("Connection closed by remote side...\r\n");
80                 break;
81             } else if (res == lwcellTIMEOUT) {
82                 printf("Netconn timeout while receiving data. You may try
↪multiple readings before deciding to "
83                     "close manually\r\n");
84             }
85
86             if (res == lwcellOK && pbuf != NULL) { /* Make sure we have valid
↪packet buffer */
87                 /*
88                  * At this point read and manipulate
89                  * with received buffer and check if you expect more data
90                  *
91                  * After you are done using it, it is important
92                  * you free the memory otherwise memory leaks will appear
93                  */
94                 printf("Received new data packet of %d bytes\r\n", (int)lwcell_
↪pbuf_length(pbuf, 1));
95                 lwcell_pbuf_free_s(&pbuf); /* Free the memory after usage */
96             }
97         } while (1);

```

(continues on next page)

```

98     } else {
99         printf("Error writing data to remote host!\r\n");
100     }
101
102     /*
103      * Check if connection was closed by remote server
104      * and in case it wasn't, close it manually
105      */
106     if (res != lwcellCLOSED) {
107         lwcell_netconn_close(client);
108     }
109     } else {
110         printf("Cannot connect to remote host %s:%d!\r\n", NETCONN_HOST, NETCONN_
↪PORT);
111     }
112     lwcell_netconn_delete(client); /* Delete netconn structure */
113 }
114 lwcell_network_request_detach(); /* Detach from network */
115
116 if (lwcell_sys_sem_isvalid(sem)) {
117     lwcell_sys_sem_release(sem);
118 }
119 lwcell_sys_thread_terminate(NULL); /* Terminate current thread */
120 }
121
122 #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 LW-CELL_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
```

(continues on next page)

(continued from previous page)

```
- Add optional keep-alive system period event
- Add `LW GSM` prefix for debug messages
- Update code style with astyle
- Add `.clang-format` draft
- Delete `lwgsmdatetime_t` and use generic `struct tm` instead
- Rename project from `lwgsmd` to `lwcell`, indicating cellular

## v0.1.1

- Update CMSIS OS driver to support FreeRTOS aware kernel
- Update to support library.json for Platform.I0

## v0.1.0

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


INDEX

L

- LWCELL_CFG_AT_ECHO (*C macro*), 69
- LWCELL_CFG_AT_PORT_BAUDRATE (*C macro*), 65
- LWCELL_CFG_CALL (*C macro*), 70
- LWCELL_CFG_CONN (*C macro*), 70
- LWCELL_CFG_CONN_MAX_DATA_LEN (*C macro*), 67
- LWCELL_CFG_CONN_MIN_DATA_LEN (*C macro*), 67
- LWCELL_CFG_CONN_POLL_INTERVAL (*C macro*), 66
- LWCELL_CFG_DBG (*C macro*), 67
- LWCELL_CFG_DBG_ASSERT (*C macro*), 68
- LWCELL_CFG_DBG_CONN (*C macro*), 68
- LWCELL_CFG_DBG_INIT (*C macro*), 68
- LWCELL_CFG_DBG_INPUT (*C macro*), 68
- LWCELL_CFG_DBG_IPD (*C macro*), 68
- LWCELL_CFG_DBG_LVL_MIN (*C macro*), 68
- LWCELL_CFG_DBG_MEM (*C macro*), 68
- LWCELL_CFG_DBG_MQTT (*C macro*), 72
- LWCELL_CFG_DBG_MQTT_API (*C macro*), 72
- LWCELL_CFG_DBG_NETCONN (*C macro*), 69
- LWCELL_CFG_DBG_OUT (*C macro*), 67
- LWCELL_CFG_DBG_PBUF (*C macro*), 68
- LWCELL_CFG_DBG_THREAD (*C macro*), 68
- LWCELL_CFG_DBG_TYPES_ON (*C macro*), 68
- LWCELL_CFG_DBG_VAR (*C macro*), 69
- LWCELL_CFG_FTP (*C macro*), 71
- LWCELL_CFG_HTTP (*C macro*), 71
- LWCELL_CFG_INPUT_USE_PROCESS (*C macro*), 69
- LWCELL_CFG_KEEP_ALIVE (*C macro*), 66
- LWCELL_CFG_KEEP_ALIVE_TIMEOUT (*C macro*), 66
- LWCELL_CFG_MAX_CONNS (*C macro*), 67
- LWCELL_CFG_MAX_SEND_RETRIES (*C macro*), 67
- LWCELL_CFG_MEM_ALIGNMENT (*C macro*), 65
- LWCELL_CFG_MEM_CUSTOM (*C macro*), 65
- LWCELL_CFG_MQTT_API_MBOX_SIZE (*C macro*), 72
- LWCELL_CFG_MQTT_MAX_REQUESTS (*C macro*), 72
- LWCELL_CFG_NETCONN (*C macro*), 71
- LWCELL_CFG_NETCONN_ACCEPT_QUEUE_LEN (*C macro*), 72
- LWCELL_CFG_NETCONN_RECEIVE_QUEUE_LEN (*C macro*), 72
- LWCELL_CFG_NETCONN_RECEIVE_TIMEOUT (*C macro*), 71
- LWCELL_CFG_NETWORK (*C macro*), 70
- LWCELL_CFG_NETWORK_IGNORE_CGACT_RESULT (*C macro*), 70
- LWCELL_CFG_OS (*C macro*), 65
- LWCELL_CFG_PHONEBOOK (*C macro*), 71
- LWCELL_CFG_PING (*C macro*), 71
- LWCELL_CFG_RCV_BUFF_SIZE (*C macro*), 66
- LWCELL_CFG_RESET_DELAY_AFTER (*C macro*), 66
- LWCELL_CFG_RESET_DELAY_DEFAULT (*C macro*), 66
- LWCELL_CFG_RESET_ON_DEVICE_PRESENT (*C macro*), 66
- LWCELL_CFG_RESET_ON_INIT (*C macro*), 66
- LWCELL_CFG_SMS (*C macro*), 70
- LWCELL_CFG_THREAD_PROCESS_MBOX_SIZE (*C macro*), 69
- LWCELL_CFG_THREAD_PRODUCER_MBOX_SIZE (*C macro*), 69
- LWCELL_CFG_THREADX_CUSTOM_MEM_BYTE_POOL (*C macro*), 70
- LWCELL_CFG_THREADX_IDLE_THREAD_EXTENSION (*C macro*), 70
- LWCELL_CFG_USE_API_FUNC_EVT (*C macro*), 65
- LWCELL_CFG_USSD (*C macro*), 71
- lwcell_ll_deinit (*C++ function*), 74
- lwcell_ll_init (*C++ function*), 74
- lwcell_ll_reset_fn (*C++ type*), 73
- lwcell_ll_send_fn (*C++ type*), 73
- lwcell_ll_t (*C++ struct*), 74
- lwcell_ll_t::baudrate (*C++ member*), 74
- lwcell_ll_t::reset_fn (*C++ member*), 74
- lwcell_ll_t::send_fn (*C++ member*), 74
- lwcell_ll_t::uart (*C++ member*), 74
- LWCELL_MEMCPY (*C macro*), 72
- LWCELL_MEMSET (*C macro*), 73
- lwcell_mqtt_client_api_buf_free (*C++ function*), 96
- lwcell_mqtt_client_api_buf_p (*C++ type*), 94
- lwcell_mqtt_client_api_buf_t (*C++ struct*), 96
- lwcell_mqtt_client_api_buf_t::payload (*C++ member*), 97
- lwcell_mqtt_client_api_buf_t::payload_len (*C++ member*), 97

lwcell_mqtt_client_api_buf_t::qos (C++ member), 97	lwcell_mqtt_client_info_t (C++ struct), 85
lwcell_mqtt_client_api_buf_t::topic (C++ member), 97	lwcell_mqtt_client_info_t::id (C++ member), 85
lwcell_mqtt_client_api_buf_t::topic_len (C++ member), 97	lwcell_mqtt_client_info_t::keep_alive (C++ member), 86
lwcell_mqtt_client_api_close (C++ function), 95	lwcell_mqtt_client_info_t::pass (C++ member), 85
lwcell_mqtt_client_api_connect (C++ function), 95	lwcell_mqtt_client_info_t::user (C++ member), 85
lwcell_mqtt_client_api_delete (C++ function), 95	lwcell_mqtt_client_info_t::will_message (C++ member), 86
lwcell_mqtt_client_api_is_connected (C++ function), 96	lwcell_mqtt_client_info_t::will_qos (C++ member), 86
lwcell_mqtt_client_api_new (C++ function), 95	lwcell_mqtt_client_info_t::will_topic (C++ member), 86
lwcell_mqtt_client_api_publish (C++ function), 96	lwcell_mqtt_client_is_connected (C++ function), 84
lwcell_mqtt_client_api_receive (C++ function), 96	lwcell_mqtt_client_new (C++ function), 83
lwcell_mqtt_client_api_subscribe (C++ function), 95	lwcell_mqtt_client_p (C++ type), 81
lwcell_mqtt_client_api_unsubscribe (C++ function), 95	lwcell_mqtt_client_publish (C++ function), 85
lwcell_mqtt_client_connect (C++ function), 84	lwcell_mqtt_client_set_arg (C++ function), 85
lwcell_mqtt_client_delete (C++ function), 83	lwcell_mqtt_client_subscribe (C++ function), 84
lwcell_mqtt_client_disconnect (C++ function), 84	lwcell_mqtt_client_unsubscribe (C++ function), 84
lwcell_mqtt_client_evt_connect_get_status (C macro), 88	lwcell_mqtt_conn_status_t (C++ enum), 83
lwcell_mqtt_client_evt_disconnect_is_accepted (C macro), 88	lwcell_mqtt_conn_status_t::LWCELL_MQTT_CONN_STATUS_ACCEPTED (C++ enumerator), 83
lwcell_mqtt_client_evt_get_type (C macro), 91	lwcell_mqtt_conn_status_t::LWCELL_MQTT_CONN_STATUS_REFUSED (C++ enumerator), 83
lwcell_mqtt_client_evt_publish_get_argument (C macro), 91	lwcell_mqtt_conn_status_t::LWCELL_MQTT_CONN_STATUS_REFUSED (C++ enumerator), 83
lwcell_mqtt_client_evt_publish_get_result (C macro), 91	lwcell_mqtt_conn_status_t::LWCELL_MQTT_CONN_STATUS_REFUSED (C++ enumerator), 83
lwcell_mqtt_client_evt_publish_rcv_get_payload (C macro), 90	lwcell_mqtt_conn_status_t::LWCELL_MQTT_CONN_STATUS_REFUSED (C++ enumerator), 83
lwcell_mqtt_client_evt_publish_rcv_get_payload_len (C macro), 90	lwcell_mqtt_conn_status_t::LWCELL_MQTT_CONN_STATUS_REFUSED (C++ enumerator), 83
lwcell_mqtt_client_evt_publish_rcv_get_qos (C macro), 90	lwcell_mqtt_conn_status_t::LWCELL_MQTT_CONN_STATUS_TCP_FAIL (C++ enumerator), 83
lwcell_mqtt_client_evt_publish_rcv_get_topic (C macro), 90	lwcell_mqtt_evt_fn (C++ type), 81
lwcell_mqtt_client_evt_publish_rcv_get_topic_len (C macro), 90	lwcell_mqtt_evt_t (C++ struct), 86
lwcell_mqtt_client_evt_publish_rcv_is_duplicate (C macro), 90	lwcell_mqtt_evt_t::arg (C++ member), 87
lwcell_mqtt_client_evt_subscribe_get_argument (C macro), 89	lwcell_mqtt_evt_t::connect (C++ member), 87
lwcell_mqtt_client_evt_subscribe_get_result (C macro), 89	lwcell_mqtt_evt_t::disconnect (C++ member), 87
lwcell_mqtt_client_evt_unsubscribe_get_argument (C macro), 89	lwcell_mqtt_evt_t::dup (C++ member), 87
lwcell_mqtt_client_evt_unsubscribe_get_result (C macro), 89	lwcell_mqtt_evt_t::evt (C++ member), 88
lwcell_mqtt_client_get_arg (C++ function), 85	lwcell_mqtt_evt_t::is_accepted (C++ member), 87
	lwcell_mqtt_evt_t::payload (C++ member), 87
	lwcell_mqtt_evt_t::payload_len (C++ member), 87
	lwcell_mqtt_evt_t::publish (C++ member), 87
	lwcell_mqtt_evt_t::publish_rcv (C++ member), 87
	lwcell_mqtt_evt_t::qos (C++ member), 87

<code>lwcell_mqtt_evt_t::res</code> (C++ member), 87	<code>lwcell_netconn_get_receive_timeout</code> (C++ function), 102
<code>lwcell_mqtt_evt_t::status</code> (C++ member), 87	<code>lwcell_netconn_getconnnum</code> (C++ function), 102
<code>lwcell_mqtt_evt_t::sub_unsub_scribed</code> (C++ member), 87	<code>lwcell_netconn_new</code> (C++ function), 101
<code>lwcell_mqtt_evt_t::topic</code> (C++ member), 87	<code>lwcell_netconn_p</code> (C++ type), 101
<code>lwcell_mqtt_evt_t::topic_len</code> (C++ member), 87	<code>lwcell_netconn_receive</code> (C++ function), 102
<code>lwcell_mqtt_evt_t::type</code> (C++ member), 87	<code>LWCELL_NETCONN_RECEIVE_NO_WAIT</code> (C macro), 101
<code>lwcell_mqtt_evt_type_t</code> (C++ enum), 82	<code>lwcell_netconn_send</code> (C++ function), 103
<code>lwcell_mqtt_evt_type_t::LWCELL_MQTT_EVT_CONNECTED</code> (C++ enumerator), 82	<code>lwcell_netconn_sendto</code> (C++ function), 103
<code>lwcell_mqtt_evt_type_t::LWCELL_MQTT_EVT_DISCONNECT</code> (C++ enumerator), 82	<code>lwcell_netconn_set_receive_timeout</code> (C++ function), 102
<code>lwcell_mqtt_evt_type_t::LWCELL_MQTT_EVT_KEEP_ALIVE</code> (C++ enumerator), 83	<code>lwcell_netconn_type_t</code> (C++ enum), 101
<code>lwcell_mqtt_evt_type_t::LWCELL_MQTT_EVT_PUBLISH</code> (C++ enumerator), 82	<code>lwcell_netconn_type_t::LWCELL_NETCONN_TYPE_SSL</code> (C++ enumerator), 101
<code>lwcell_mqtt_evt_type_t::LWCELL_MQTT_EVT_PUBLISH_RET</code> (C++ enumerator), 82	<code>lwcell_netconn_type_t::LWCELL_NETCONN_TYPE_TCP</code> (C++ enumerator), 101
<code>lwcell_mqtt_evt_type_t::LWCELL_MQTT_EVT_PUBLISH_RET_C</code> (C++ enumerator), 82	<code>lwcell_netconn_type_t::LWCELL_NETCONN_TYPE_UDP</code> (C++ enumerator), 101
<code>lwcell_mqtt_evt_type_t::LWCELL_MQTT_EVT_SUBSCRIBE</code> (C++ enumerator), 82	<code>lwcell_netconn_write</code> (C++ function), 103
<code>lwcell_mqtt_evt_type_t::LWCELL_MQTT_EVT_UNSUBSCRIBE</code> (C++ enumerator), 82	<code>lwcell_netconn_write_ex</code> (C++ function), 103
<code>lwcell_mqtt_qos_t</code> (C++ enum), 81	<code>lwcell_sys_init</code> (C++ function), 75
<code>lwcell_mqtt_qos_t::LWCELL_MQTT_QOS_AT_LEAST_ONCE</code> (C++ enumerator), 81	<code>lwcell_sys_lwcell_sys_mutex_t</code> (C++ type), 80
<code>lwcell_mqtt_qos_t::LWCELL_MQTT_QOS_AT_MOST_ONCE</code> (C++ enumerator), 81	<code>lwcell_sys_mbox_create</code> (C++ function), 78
<code>lwcell_mqtt_qos_t::LWCELL_MQTT_QOS_EXACTLY_ONCE</code> (C++ enumerator), 81	<code>lwcell_sys_mbox_delete</code> (C++ function), 78
<code>lwcell_mqtt_request_t</code> (C++ struct), 86	<code>lwcell_sys_mbox_get</code> (C++ function), 78
<code>lwcell_mqtt_request_t::arg</code> (C++ member), 86	<code>lwcell_sys_mbox_getnow</code> (C++ function), 78
<code>lwcell_mqtt_request_t::expected_sent_len</code> (C++ member), 86	<code>lwcell_sys_mbox_invalid</code> (C++ function), 79
<code>lwcell_mqtt_request_t::packet_id</code> (C++ member), 86	<code>lwcell_sys_mbox_isvalid</code> (C++ function), 79
<code>lwcell_mqtt_request_t::status</code> (C++ member), 86	<code>LWCELL_SYS_MBOX_NULL</code> (C macro), 80
<code>lwcell_mqtt_request_t::timeout_start_time</code> (C++ member), 86	<code>lwcell_sys_mbox_put</code> (C++ function), 78
<code>lwcell_mqtt_state_t</code> (C++ enum), 82	<code>lwcell_sys_mbox_putnow</code> (C++ function), 78
<code>lwcell_mqtt_state_t::LWCELL_MQTT_CONN_CONNECTING</code> (C++ enumerator), 82	<code>lwcell_sys_mbox_t</code> (C++ type), 80
<code>lwcell_mqtt_state_t::LWCELL_MQTT_CONN_DISCONNECTED</code> (C++ enumerator), 82	<code>lwcell_sys_mutex_create</code> (C++ function), 76
<code>lwcell_mqtt_state_t::LWCELL_MQTT_CONN_DISCONNECTING</code> (C++ enumerator), 82	<code>lwcell_sys_mutex_delete</code> (C++ function), 76
<code>lwcell_mqtt_state_t::LWCELL_MQTT_CONNECTED</code> (C++ enumerator), 82	<code>lwcell_sys_mutex_invalid</code> (C++ function), 76
<code>lwcell_mqtt_state_t::LWCELL_MQTT_CONNECTING</code> (C++ enumerator), 82	<code>lwcell_sys_mutex_isvalid</code> (C++ function), 76
<code>lwcell_netconn_close</code> (C++ function), 102	<code>lwcell_sys_mutex_lock</code> (C++ function), 76
<code>lwcell_netconn_connect</code> (C++ function), 101	<code>LWCELL_SYS_MUTEX_NULL</code> (C macro), 80
<code>lwcell_netconn_delete</code> (C++ function), 101	<code>lwcell_sys_mutex_unlock</code> (C++ function), 76
<code>LWCELL_NETCONN_FLAG_FLUSH</code> (C macro), 101	<code>lwcell_sys_now</code> (C++ function), 75
<code>lwcell_netconn_flush</code> (C++ function), 103	<code>lwcell_sys_protect</code> (C++ function), 75
	<code>lwcell_sys_sem_create</code> (C++ function), 77
	<code>lwcell_sys_sem_delete</code> (C++ function), 77
	<code>lwcell_sys_sem_invalid</code> (C++ function), 77
	<code>lwcell_sys_sem_isvalid</code> (C++ function), 77
	<code>LWCELL_SYS_SEM_NULL</code> (C macro), 80
	<code>lwcell_sys_sem_release</code> (C++ function), 77
	<code>lwcell_sys_sem_t</code> (C++ type), 80
	<code>lwcell_sys_sem_wait</code> (C++ function), 77
	<code>lwcell_sys_thread_create</code> (C++ function), 79
	<code>lwcell_sys_thread_fn</code> (C++ type), 80
	<code>LWCELL_SYS_THREAD_PRIO</code> (C macro), 80
	<code>lwcell_sys_thread_prio_t</code> (C++ type), 81
	<code>lwcell_sys_THREAD_SS</code> (C macro), 80

lwcell_sys_thread_t (C++ *type*), 81
lwcell_sys_thread_terminate (C++ *function*), 79
lwcell_sys_thread_yield (C++ *function*), 79
LWCELL_SYS_TIMEOUT (C *macro*), 80
lwcell_sys_unprotect (C++ *function*), 75
LWCELL_THREAD_PROCESS_HOOK (C *macro*), 70
LWCELL_THREAD_PRODUCER_HOOK (C *macro*), 69