

---

**LwGSM**

**Tilen MAJERLE**

**Nov 15, 2020**



# CONTENTS

<b>1</b>	<b>Features</b>	<b>3</b>
<b>2</b>	<b>Requirements</b>	<b>5</b>
<b>3</b>	<b>Contribute</b>	<b>7</b>
<b>4</b>	<b>License</b>	<b>9</b>
<b>5</b>	<b>Table of contents</b>	<b>11</b>
5.1	Getting started . . . . .	11
5.2	User manual . . . . .	13
5.3	API reference . . . . .	59
5.4	Examples and demos . . . . .	182
	<b>Index</b>	<b>185</b>



Welcome to the documentation for version 0.1.0.

LwGSM is generic, platform independent, library to control *SIM800/SIM900* or *SIM7000/SIM7020 (NB-Iot LTE)* devices from *SIMCom*. Its objective is to run on master system, while SIMCom device runs official AT commands firmware developed and maintained by *SIMCom*.

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



## 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 (C99)
- 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) 2020 Tilen MAJERLE

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

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

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



## TABLE OF CONTENTS

### 5.1 Getting started

#### 5.1.1 Download library

Library is primarily hosted on [Github](#).

- Download latest release from [releases area](#) on Github
- Clone *develop* branch for latest development

#### Download from releases

All releases are available on Github [releases area](#).

#### Clone from Github

##### First-time clone

- Download and install `git` if not already
- Open console and navigate to path in the system to clone repository to. Use command `cd your_path`
- Clone repository with one of available 3 options
  - Run `git clone --recurse-submodules https://github.com/MaJerle/lwgsn` command to clone entire repository, including submodules
  - Run `git clone --recurse-submodules --branch develop https://github.com/MaJerle/lwgsn` to clone *development* branch, including submodules
  - Run `git clone --recurse-submodules --branch master https://github.com/MaJerle/lwgsn` to clone *latest stable* branch, including submodules
- Navigate to `examples` directory and run favourite example

## Update cloned to latest version

- Open console and navigate to path in the system where your resources repository is. Use command `cd your_path`
- Run `git pull origin master --recurse-submodules` command to pull latest changes and to fetch latest changes from submodules
- Run `git submodule foreach git pull origin master` to update & merge all submodules

---

**Note:** This is preferred option to use when you want to evaluate library and run prepared examples. Repository consists of multiple submodules which can be automatically downloaded when cloning and pulling changes from root repository.

---

### 5.1.2 Add library to project

At this point it is assumed that you have successfully download library, either cloned it or from releases page.

- Copy `lwgsm` folder to your project
- Add `lwgsm/src/include` folder to *include path* of your toolchain
- Add port architecture `lwgsm/src/include/system/port/_arch_` folder to *include path* of your toolchain
- Add source files from `lwgsm/src/` folder to toolchain build
- Add source files from `lwgsm/src/system/` folder to toolchain build for arch port
- Copy `lwgsm/src/include/lwgsm/lwgsm_opts_template.h` to project folder and rename it to `lwgsm_opts.h`
- Build the project

### 5.1.3 Configuration file

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

---

**Note:** Default configuration template file location: `lwgsm/src/include/lwgsm/lwgsm_opts_template.h`. File must be renamed to `lwgsm_opts.h` first and then copied to the project directory (or simply renamed in-place) where compiler include paths have access to it by using `#include "lwgsm_opts.h"`.

---

---

**Tip:** Check [Configuration](#) section for possible configuration settings

---

Listing 1: Template options file

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

(continues on next page)



(continued from previous page)

```

6  /*
7   * Copyright (c) 2020 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 LwGSM - Lightweight GSM-AT library.
30  *
31  * Author:          Tilen MAJERLE <tilen@majerle.eu>
32  * Version:         v0.1.0
33  */
34 #ifndef LWGSM_HDR_OPTS_H
35 #define LWGSM_HDR_OPTS_H
36
37 /* Rename this file to "lwgsm_opts.h" for your application */
38
39 /*
40  * Open "include/lwgsm/lwgsm_opt.h" and
41  * copy & replace here settings you want to change values
42  */
43
44 #endif /* LWGSM_HDR_OPTS_H */

```

## 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.

*LwGSM* 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 *LwGSM* 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 *LwGSM* 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: LwGSM 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 LwGSM 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 *LWGSM\_LL*, is part, dedicated for communication between *LwGSM* middleware and *GSM* physical device. Application needs to implement output function to send necessary *AT command* instruction as well as implement *input module* to send received data from *GSM* device to *LwGSM* middleware.

Application must also assure memory assignment for *Memory manager* when default allocation is used.

---

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

---

## GSM physical device

### 5.2.3 Inter thread communication

LwGSM 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 `lwgsm_threads.c` file.

#### Processing thread

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

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

---

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

---

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

---

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

---

## Producing thread

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

- It checks if command is valid and if it has corresponding initial AT sequence, such as `AT+CIPSTART`
- It locks **sync\_sem** semaphore and waits for processing thread to unlock it
  - *Processing thread* is in charge to read 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 `LWGSM_CFG_INPUT_USE_PROCESS` is enabled and `lwgsm_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 *Event management* 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 `lwgsm_evt_register()` function to register a new, custom, event function.

---

**Tip:** Implementation of *Netconn API* leverages `lwgsm_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          lwgsm_netconn.c
3   * \brief         API functions for sequential calls
4   */
5
6  /*
7   * Copyright (c) 2020 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 LwGSM - Lightweight GSM-AT library.
30  *
31  * Author:          Tilen MAJERLE <tilen@majerle.eu>
32  * Version:         v0.1.0
33  */
34 #include "lwgsm/lwgsm_netconn.h"
35 #include "lwgsm/lwgsm_private.h"
36 #include "lwgsm/lwgsm_conn.h"
37 #include "lwgsm/lwgsm_mem.h"
38
39 #if LWGSM_CFG_NETCONN || __DOXYGEN__
40
41 /* Check conditions */
42 #if !LWGSM_CFG_CONN
43 #error "LWGSM_CFG_CONN must be enabled for NETCONN API!"
44 #endif /* !LWGSM_CFG_CONN */
45
46 #if LWGSM_CFG_NETCONN_RECEIVE_QUEUE_LEN < 2
47 #error "LWGSM_CFG_NETCONN_RECEIVE_QUEUE_LEN must be greater or equal to 2"
48 #endif /* LWGSM_CFG_NETCONN_RECEIVE_QUEUE_LEN < 2 */
49
50 /**
51  * \brief          Sequential API structure
52  */
53 typedef struct lwgsm_netconn {
54     struct lwgsm_netconn* next;                /*!< Linked list entry */
55
56     lwgsm_netconn_type_t type;                 /*!< Netconn type */
57
58     size_t rcv_packets;                        /*!< Number of received packets so_
↳ far on this connection */
59     lwgsm_conn_p conn;                        /*!< Pointer to actual connection */
60
61     lwgsm_sys_mbox_t mbox_receive;            /*!< Message queue for receive mbox */
62
63     lwgsm_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. */

```

(continues on next page)

(continued from previous page)

```

69
70 #if LWGSM_CFG_NETCONN_RECEIVE_TIMEOUT || __DOXYGEN__
71     uint32_t rcv_timeout;                /*!< Receive timeout in unit of_
↪ milliseconds */
72 #endif
73 } lwgsm_netconn_t;
74
75 static uint8_t rcv_closed = 0xFF;
76 static lwgsm_netconn_t* netconn_list;    /*!< Linked list of netconn entries */
77
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(lwgsm_netconn_t* nc, uint8_t protect) {
85     lwgsm_pbuf_p pbuf;
86     if (protect) {
87         lwgsm_core_lock();
88     }
89     if (lwgsm_sys_mbox_isvalid(&nc->mbox_receive)) {
90         while (lwgsm_sys_mbox_getnow(&nc->mbox_receive, (void**)&pbuf)) {
91             if (pbuf != NULL && (uint8_t*)pbuf != (uint8_t*)&rcv_closed) {
92                 lwgsm_pbuf_free(pbuf);    /* Free received data buffers */
93             }
94         }
95         lwgsm_sys_mbox_delete(&nc->mbox_receive); /* Delete message queue */
96         lwgsm_sys_mbox_invalid(&nc->mbox_receive); /* Invalid handle */
97     }
98     if (protect) {
99         lwgsm_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 lwgsmr_t enumeration
107  */
108 static lwgsmr_t
109 netconn_evt(lwgsm_evt_t* evt) {
110     lwgsm_conn_p conn;
111     lwgsm_netconn_t* nc = NULL;
112     uint8_t close = 0;
113
114     conn = lwgsm_conn_get_from_evt(evt);    /* Get connection from event */
115     switch (lwgsm_evt_get_type(evt)) {
116         /*
117          * A new connection has been active
118          * and should be handled by netconn API
119          */
120         case LWGSM_EVT_CONN_ACTIVE: {    /* A new connection active is active_
↪ */
121             if (lwgsm_conn_is_client(conn)) { /* Was connection started by us? */
122                 nc = lwgsm_conn_get_arg(conn); /* Argument should be already set */
123                 if (nc != NULL) {

```

(continues on next page)

(continued from previous page)

```

124         nc->conn = conn;                /* Save actual connection */
125     } else {
126         close = 1;                      /* Close this connection, invalid_
↪netconn */
127     }
128     } else {
129         LWGSM_DEBUGF(LWGSM_CFG_DBG_NETCONN | LWGSM_DBG_TYPE_TRACE | LWGSM_DBG_
↪LVL_WARNING,
130                     "[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             lwgsm_conn_set_arg(conn, NULL); /* Reset argument */
138             lwgsm_netconn_delete(nc); /* Free memory for API */
139         }
140         lwgsm_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 LWGSM_EVT_CONN_RECV: {
151     lwgsm_pbuf_p pbuf;
152
153     nc = lwgsm_conn_get_arg(conn); /* Get API from connection */
154     pbuf = lwgsm_evt_conn_recv_get_buff(evt); /* Get received buff */
155
156     lwgsm_conn_recved(conn, pbuf); /* Notify stack about received data */
157
158     lwgsm_pbuf_ref(pbuf); /* Increase reference counter */
159     if (nc == NULL || !lwgsm_sys_mbox_isvalid(&nc->mbox_receive)
160         || !lwgsm_sys_mbox_putnow(&nc->mbox_receive, pbuf)) {
161         LWGSM_DEBUGF(LWGSM_CFG_DBG_NETCONN,
162                     "[NETCONN] Ignoring more data for receive!\r\n");
163         lwgsm_pbuf_free(pbuf); /* Free pbuf */
164         return lwgsmOKIGNOREMORE; /* Return OK to free the memory and_
↪ignore further data */
165     }
166     ++nc->rcv_packets; /* Increase number of received_
↪packets */
167     LWGSM_DEBUGF(LWGSM_CFG_DBG_NETCONN | LWGSM_DBG_TYPE_TRACE,
168                 "[NETCONN] Received pbuf contains %d bytes. Handle written to_
↪receive mbox\r\n",
169                 (int) lwgsm_pbuf_length(pbuf, 0));
170     break;
171 }
172
173 /* Connection was just closed */

```

(continues on next page)



(continued from previous page)

```

174     case LWGSM_EVT_CONN_CLOSE: {
175         nc = lwgsm_conn_get_arg(conn);      /* Get API from connection */
176
177         /*
178          * In case we have a netconn available,
179          * simply write pointer to received variable to indicate closed state
180          */
181         if (nc != NULL && lwgsm_sys_mbox_isvalid(&nc->mbox_receive)) {
182             lwgsm_sys_mbox_putnow(&nc->mbox_receive, (void*)&recv_closed);
183         }
184
185         break;
186     }
187     default:
188         return lwgsmERR;
189 }
190 return lwgsmOK;
191 }
192
193 /**
194  * \brief          Global event callback function
195  * \param[in]      evt: Callback information and data
196  * \return         \ref lwgsmOK on success, member of \ref lwgsmr_t otherwise
197  */
198 static lwgsmr_t
199 lwgsm_evt(lwgsm_evt_t* evt) {
200     switch (lwgsm_evt_get_type(evt)) {
201         default:
202             break;
203     }
204     return lwgsmOK;
205 }
206
207 /**
208  * \brief          Create new netconn connection
209  * \param[in]      type: Netconn connection type
210  * \return         New netconn connection on success, `NULL` otherwise
211  */
212 lwgsm_netconn_p
213 lwgsm_netconn_new(lwgsm_netconn_type_t type) {
214     lwgsm_netconn_t* a;
215     static uint8_t first = 1;
216
217     /* Register only once! */
218     lwgsm_core_lock();
219     if (first) {
220         first = 0;
221         lwgsm_evt_register(lwgsm_evt);      /* Register global event function */
222     }
223     lwgsm_core_unlock();
224     a = lwgsm_mem_calloc(1, sizeof(*a));    /* Allocate memory for core object */
225     if (a != NULL) {
226         a->type = type;                    /* Save netconn type */
227         a->conn_timeout = 0;               /* Default connection timeout */
228         if (!lwgsm_sys_mbox_create(&a->mbox_receive, LWGSM_CFG_NETCONN_RECEIVE_QUEUE_
229     ↪ LEN)) { /* Allocate memory for receiving message box */
230             LWGSM_DEBUGF(LWGSM_CFG_DBG_NETCONN | LWGSM_DBG_TYPE_TRACE | LWGSM_DBG_LVL_
231     ↪ DANGER,

```

(continues on next page)

(continued from previous page)

```

230         "[NETCONN] Cannot create receive MBOX\r\n");
231     goto free_ret;
232 }
233 lwgsm_core_lock();
234 if (netconn_list == NULL) {           /* Add new netconn to the existing_
↪list */
235     netconn_list = a;
236 } else {
237     a->next = netconn_list;           /* Add it to beginning of the list */
238     netconn_list = a;
239 }
240 lwgsm_core_unlock();
241 }
242 return a;
243 free_ret:
244 if (lwgsm_sys_mbox_isvalid(&a->mbox_receive)) {
245     lwgsm_sys_mbox_delete(&a->mbox_receive);
246     lwgsm_sys_mbox_invalid(&a->mbox_receive);
247 }
248 if (a != NULL) {
249     lwgsm_mem_free_s((void**) &a);
250 }
251 return NULL;
252 }
253
254 /**
255  * \brief          Delete netconn connection
256  * \param[in]      nc: Netconn handle
257  * \return         \ref lwgsmOK on success, member of \ref lwgsmr_t enumeration_
↪otherwise
258  */
259 lwgsmr_t
260 lwgsm_netconn_delete(lwgsm_netconn_p nc) {
261     LWGSM_ASSERT("netconn != NULL", nc != NULL);
262
263     lwgsm_core_lock();
264     flush_mboxes(nc, 0);               /* Clear mboxes */
265
266     /* Remove netconn from linkedlist */
267     if (netconn_list == nc) {
268         netconn_list = netconn_list->next;   /* Remove first from linked list */
269     } else if (netconn_list != NULL) {
270         lwgsm_netconn_p tmp, prev;
271         /* Find element on the list */
272         for (prev = netconn_list, tmp = netconn_list->next;
273              tmp != NULL; prev = tmp, tmp = tmp->next) {
274             if (nc == tmp) {
275                 prev->next = tmp->next;       /* Remove tmp from linked list */
276                 break;
277             }
278         }
279     }
280     lwgsm_core_unlock();
281
282     lwgsm_mem_free_s((void**) &nc);
283     return lwgsmOK;
284 }

```

(continues on next page)

(continued from previous page)

```

285
286 /**
287  * \brief          Connect to server as client
288  * \param[in]      nc: Netconn handle
289  * \param[in]      host: Pointer to host, such as domain name or IP address in_
↳string format
290  * \param[in]      port: Target port to use
291  * \return         \ref lwgsmOK if successfully connected, member of \ref lwgsmr_t_
↳otherwise
292  */
293 lwgsmr_t
294 lwgsm_netconn_connect(lwgsm_netconn_p nc, const char* host, lwgsm_port_t port) {
295     lwgsmr_t res;
296
297     LWGSM_ASSERT("nc != NULL", nc != NULL);
298     LWGSM_ASSERT("host != NULL", host != NULL);
299     LWGSM_ASSERT("port > 0", 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 = lwgsm_conn_start(NULL, (lwgsm_conn_type_t)nc->type, host, port, nc, netconn_
↳evt, 1);
309     return res;
310 }
311
312 /**
313  * \brief          Write data to connection output buffers
314  * \note           This function may only be used on TCP or SSL connections
315  * \param[in]      nc: Netconn handle used to write data to
316  * \param[in]      data: Pointer to data to write
317  * \param[in]      btw: Number of bytes to write
318  * \return         \ref lwgsmOK on success, member of \ref lwgsmr_t enumeration_
↳otherwise
319  */
320 lwgsmr_t
321 lwgsm_netconn_write(lwgsm_netconn_p nc, const void* data, size_t btw) {
322     size_t len, sent;
323     const uint8_t* d = data;
324     lwgsmr_t res;
325
326     LWGSM_ASSERT("nc != NULL", nc != NULL);
327     LWGSM_ASSERT("nc->type must be TCP or SSL", nc->type == LWGSM_NETCONN_TYPE_TCP ||_
↳nc->type == LWGSM_NETCONN_TYPE_SSL);
328     LWGSM_ASSERT("nc->conn must be active", lwgsm_conn_is_active(nc->conn));
329
330     /*
331      * Several steps are done in write process
332      *
333      * 1. Check if buffer is set and check if there is something to write to it.
334      * 1. In case buffer will be full after copy, send it and free memory.
335      * 2. Check how many bytes we can write directly without need to copy
336      * 3. Try to allocate a new buffer and copy remaining input data to it

```

(continues on next page)

(continued from previous page)

```

337     * 4. In case buffer allocation fails, send data directly (may affect on speed_
↳and effectiveness)
338     */
339
340     /* Step 1 */
341     if (nc->buff.buff != NULL) {                                     /* Is there a write buffer ready to_
↳accept more data? */
342         len = LWGSM_MIN(nc->buff.len - nc->buff.ptr, btw); /* Get number of bytes we_
↳can write to buffer */
343         if (len > 0) {
344             LWGSM_MEMCPY(&nc->buff.buff[nc->buff.ptr], data, len); /* Copy memory to_
↳temporary write buffer */
345             d += len;
346             nc->buff.ptr += len;
347             btw -= len;
348         }
349
350         /* Step 1.1 */
351         if (nc->buff.ptr == nc->buff.len) {
352             res = lwgsm_conn_send(nc->conn, nc->buff.buff, nc->buff.len, &sent, 1);
353
354             lwgsm_mem_free_s((void**) &nc->buff.buff);
355             if (res != lwgsmOK) {
356                 return res;
357             }
358         } else {
359             return lwgsmOK;                                     /* Buffer is not yet full yet */
360         }
361     }
362
363     /* Step 2 */
364     if (btw >= LWGSM_CFG_CONN_MAX_DATA_LEN) {
365         size_t rem;
366         rem = btw % LWGSM_CFG_CONN_MAX_DATA_LEN; /* Get remaining bytes for max data_
↳length */
367         res = lwgsm_conn_send(nc->conn, d, btw - rem, &sent, 1); /* Write data_
↳directly */
368         if (res != lwgsmOK) {
369             return res;
370         }
371         d += sent;                                             /* Advance in data pointer */
372         btw -= sent;                                           /* Decrease remaining data to send */
373     }
374
375     if (btw == 0) {                                           /* Sent everything? */
376         return lwgsmOK;
377     }
378
379     /* Step 3 */
380     if (nc->buff.buff == NULL) {                               /* Check if we should allocate a new_
↳buffer */
381         nc->buff.buff = lwgsm_mem_malloc(sizeof(*nc->buff.buff) * LWGSM_CFG_CONN_MAX_
↳DATA_LEN);
382         nc->buff.len = LWGSM_CFG_CONN_MAX_DATA_LEN; /* Save buffer length */
383         nc->buff.ptr = 0;                                     /* Save buffer pointer */
384     }
385

```

(continues on next page)

(continued from previous page)

```

386     /* Step 4 */
387     if (nc->buff.buff != NULL) {                                /* Memory available? */
388         LWGSM_MEMCPY(&nc->buff.buff[nc->buff.ptr], d, btw); /* Copy data to buffer */
389         nc->buff.ptr += btw;
390     } else {                                                    /* Still no memory available? */
391         return lwgsm_conn_send(nc->conn, data, btw, NULL, 1); /* Simply send_
↳directly blocking */
392     }
393     return lwgsmOK;
394 }
395
396 /**
397  * \brief          Flush buffered data on netconn \e TCP/SSL connection
398  * \note           This function may only be used on \e TCP/SSL connection
399  * \param[in]      nc: Netconn handle to flush data
400  * \return         \ref lwgsmOK on success, member of \ref lwgsmr_t enumeration_
↳otherwise
401  */
402 lwgsmr_t
403 lwgsm_netconn_flush(lwgsm_netconn_p nc) {
404     LWGSM_ASSERT("nc != NULL", nc != NULL);
405     LWGSM_ASSERT("nc->type must be TCP or SSL", nc->type == LWGSM_NETCONN_TYPE_TCP ||_
↳nc->type == LWGSM_NETCONN_TYPE_SSL);
406     LWGSM_ASSERT("nc->conn must be active", lwgsm_conn_is_active(nc->conn));
407
408     /*
409      * In case we have data in write buffer,
410      * flush them out to network
411      */
412     if (nc->buff.buff != NULL) {                                /* Check remaining data */
413         if (nc->buff.ptr > 0) {                                  /* Do we have data in current buffer?_
↳*/
414             lwgsm_conn_send(nc->conn, nc->buff.buff, nc->buff.ptr, NULL, 1); /* Send_
↳data */
415         }
416         lwgsm_mem_free_s((void**) &nc->buff.buff);
417     }
418     return lwgsmOK;
419 }
420
421 /**
422  * \brief          Send data on \e UDP connection to default IP and port
423  * \param[in]      nc: Netconn handle used to send
424  * \param[in]      data: Pointer to data to write
425  * \param[in]      btw: Number of bytes to write
426  * \return         \ref lwgsmOK on success, member of \ref lwgsmr_t enumeration_
↳otherwise
427  */
428 lwgsmr_t
429 lwgsm_netconn_send(lwgsm_netconn_p nc, const void* data, size_t btw) {
430     LWGSM_ASSERT("nc != NULL", nc != NULL);
431     LWGSM_ASSERT("nc->type must be UDP", nc->type == LWGSM_NETCONN_TYPE_UDP);
432     LWGSM_ASSERT("nc->conn must be active", lwgsm_conn_is_active(nc->conn));
433
434     return lwgsm_conn_send(nc->conn, data, btw, NULL, 1);
435 }
436

```

(continues on next page)

(continued from previous page)

```

437 /**
438  * \brief      Send data on \e UDP connection to specific IP and port
439  * \note       Use this function in case of UDP type netconn
440  * \param[in]  nc: Netconn handle used to send
441  * \param[in]  ip: Pointer to IP address
442  * \param[in]  port: Port number used to send data
443  * \param[in]  data: Pointer to data to write
444  * \param[in]  btw: Number of bytes to write
445  * \return     \ref lwgsmOK on success, member of \ref lwgsmr_t enumeration_
↳ otherwise
446  */
447 lwgsmr_t
448 lwgsm_netconn_sendto(lwgsm_netconn_p nc, const lwgsm_ip_t* ip, lwgsm_port_t port,
↳ const void* data, size_t btw) {
449     LWGSM_ASSERT("nc != NULL", nc != NULL);
450     LWGSM_ASSERT("nc->type must be UDP", nc->type == LWGSM_NETCONN_TYPE_UDP);
451     LWGSM_ASSERT("nc->conn must be active", lwgsm_conn_is_active(nc->conn));
452
453     return lwgsm_conn_sendto(nc->conn, ip, port, data, btw, NULL, 1);
454 }
455
456 /**
457  * \brief      Receive data from connection
458  * \param[in]  nc: Netconn handle used to receive from
459  * \param[in]  pbuf: Pointer to pointer to save new receive buffer to.
460  *             When function returns, user must check for valid pbuf value_
↳ `pbuf != NULL`
461  * \return     \ref lwgsmOK when new data ready,
462  * \return     \ref lwgsmCLOSED when connection closed by remote side,
463  * \return     \ref lwgsmTIMEOUT when receive timeout occurs
464  * \return     Any other member of \ref lwgsmr_t otherwise
465  */
466 lwgsmr_t
467 lwgsm_netconn_receive(lwgsm_netconn_p nc, lwgsm_pbuf_p* pbuf) {
468     LWGSM_ASSERT("nc != NULL", nc != NULL);
469     LWGSM_ASSERT("pbuf != NULL", pbuf != NULL);
470
471     *pbuf = NULL;
472 #if LWGSM_CFG_NETCONN_RECEIVE_TIMEOUT
473     /*
474      * Wait for new received data for up to specific timeout
475      * or throw error for timeout notification
476      */
477     if (lwgsm_sys_mbox_get(&nc->mbox_receive, (void**)pbuf, nc->rcv_timeout) == LWGSM_
↳ SYS_TIMEOUT) {
478         return lwgsmTIMEOUT;
479     }
480 #else /* LWGSM_CFG_NETCONN_RECEIVE_TIMEOUT */
481     /* Forever wait for new receive packet */
482     lwgsm_sys_mbox_get(&nc->mbox_receive, (void**)pbuf, 0);
483 #endif /* !LWGSM_CFG_NETCONN_RECEIVE_TIMEOUT */
484
485     /* Check if connection closed */
486     if ((uint8_t*)(*pbuf) == (uint8_t*)&recv_closed) {
487         *pbuf = NULL; /* Reset pbuf */
488         return lwgsmCLOSED;
489     }

```

(continues on next page)

(continued from previous page)

```

490     return lwgsmOK;                                /* We have data available */
491 }
492
493 /**
494  * \brief          Close a netconn connection
495  * \param[in]      nc: Netconn handle to close
496  * \return         \ref lwgsmOK on success, member of \ref lwgsmr_t enumeration_
497  * \otherwise
498  */
499 lwgsmr_t
500 lwgsm_netconn_close(lwgsm_netconn_p nc) {
501     lwgsm_conn_p conn;
502
503     LWGSM_ASSERT("nc != NULL", nc != NULL);
504     LWGSM_ASSERT("nc->conn != NULL", nc->conn != NULL);
505     LWGSM_ASSERT("nc->conn must be active", lwgsm_conn_is_active(nc->conn));
506
507     lwgsm_netconn_flush(nc);                          /* Flush data and ignore result */
508     conn = nc->conn;
509     nc->conn = NULL;
510
511     lwgsm_conn_set_arg(conn, NULL);                    /* Reset argument */
512     lwgsm_conn_close(conn, 1);                        /* Close the connection */
513     flush_mboxes(nc, 1);                              /* Flush message queues */
514     return lwgsmOK;
515 }
516
517 /**
518  * \brief          Get connection number used for netconn
519  * \param[in]      nc: Netconn handle
520  * \return         '-1' on failure, connection number between '0' and \ref LWGSM_CFG_
521  * \MAX_CONNS otherwise
522  */
523 int8_t
524 lwgsm_netconn_getconnnum(lwgsm_netconn_p nc) {
525     if (nc != NULL && nc->conn != NULL) {
526         return lwgsm_conn_getnum(nc->conn);
527     }
528     return -1;
529 }
530
531 #if LWGSM_CFG_NETCONN_RECEIVE_TIMEOUT || __DOXYGEN__
532
533 /**
534  * \brief          Set timeout value for receiving data.
535  *
536  * When enabled, \ref lwgsm_netconn_receive will only block for up to
537  * \e timeout value and will return if no new data within this time
538  *
539  * \param[in]      nc: Netconn handle
540  * \param[in]      timeout: Timeout in units of milliseconds.
541  * Set to '0' to disable timeout for \ref lwgsm_netconn_receive_
542  *
543  * \function
544  */
545 void
546 lwgsm_netconn_set_receive_timeout(lwgsm_netconn_p nc, uint32_t timeout) {
547     nc->rcv_timeout = timeout;

```

(continues on next page)

(continued from previous page)

```

544 }
545
546 /**
547  * \brief          Get netconn receive timeout value
548  * \param[in]      nc: Netconn handle
549  * \return         Timeout in units of milliseconds.
550  *                If value is `0`, timeout is disabled (wait forever)
551  */
552 uint32_t
553 lwgsm_netconn_get_receive_timeout(lwgsm_netconn_p nc) {
554     return nc->rcv_timeout;          /* Return receive timeout */
555 }
556
557 #endif /* LWGSM_CFG_NETCONN_RECEIVE_TIMEOUT || __DOXYGEN__ */
558
559 #endif /* LWGSM_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 `LWGSM_EVT_CONN_*`, such as `LWGSM_EVT_CONN_RECV`. Check [Event management](#) for list of all connection events.

---

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

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

```

1  #include "client.h"
2  #include "lwgsm/lwgsm.h"
3  #include "lwgsm/lwgsm_network_api.h"
4
5  /* Host parameter */
6  #define CONN_HOST          "example.com"
7  #define CONN_PORT          80
8
9  static lwgsmr_t    conn_callback_func(lwgsm_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  */

```

(continues on next page)



(continued from previous page)

```

24 void
25 client_connect(void) {
26     lwgsmr_t res;
27
28     /* Attach to GSM network */
29     lwgsm_network_request_attach();
30
31     /* Start a new connection as client in non-blocking mode */
32     if ((res = lwgsm_conn_start(NULL, LWGSM_CONN_TYPE_TCP, "example.com", 80, NULL,
↪conn_callback_func, 0)) == lwgsmOK) {
33         printf("Connection to " CONN_HOST " started...\r\n");
34     } else {
35         printf("Cannot start connection to " CONN_HOST "!\r\n");
36     }
37 }
38
39 /**
40  * \brief      Event callback function for connection-only
41  * \param[in]  evt: Event information with data
42  * \return     \ref lwgsmOK on success, member of \ref lwgsmr_t otherwise
43  */
44 static lwgsmr_t
45 conn_callback_func(lwgsm_evt_t* evt) {
46     lwgsm_conn_p conn;
47     lwgsmr_t res;
48     uint8_t conn_num;
49
50     conn = lwgsm_conn_get_from_evt(evt);          /* Get connection handle from event_
↪ */
51     if (conn == NULL) {
52         return lwgsmERR;
53     }
54     conn_num = lwgsm_conn_getnum(conn);           /* Get connection number for_
↪ identification */
55     switch (lwgsm_evt_get_type(evt)) {
56         case LWGSM_EVT_CONN_ACTIVE: {             /* Connection just active */
57             printf("Connection %d active!\r\n", (int) conn_num);
58             res = lwgsm_conn_send(conn, req_data, sizeof(req_data) - 1, NULL, 0); /*
↪ Start sending data in non-blocking mode */
59             if (res == lwgsmOK) {
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                 lwgsm_conn_close(conn, 0);         /* Close the connection */
64             }
65             break;
66         }
67         case LWGSM_EVT_CONN_CLOSE: {               /* Connection closed */
68             if (lwgsm_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 LWGSM_EVT_CONN_SEND: {               /* Data send event */

```

(continues on next page)

(continued from previous page)

```

76     lwgsmr_t res = lwgsm_evt_conn_send_get_result(evt);
77     if (res == lwgsmOK) {
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 LWGSM_EVT_CONN_RECV: { /* Data received from remote side */
85     lwgsm_pbuf_p pbuf = lwgsm_evt_conn_recv_get_buff(evt);
86     lwgsm_conn_recved(conn, pbuf); /* Notify stack about received pbuf_
↪*/
87     printf("Received %d bytes on connection %d...\r\n", (int)lwgsm_pbuf_
↪length(pbuf, 1), (int)conn_num);
88     break;
89 }
90 case LWGSM_EVT_CONN_ERROR: { /* Error connecting to server */
91     const char* host = lwgsm_evt_conn_error_get_host(evt);
92     lwgsm_port_t port = lwgsm_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 lwgsmOK;
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 (lwgsm_sms_send("+0123456789", "text", NULL, NULL, 1 /* 1 means blocking call */)_
↪== lwgsmOK) {
6      /* At this point we have valid result from device */
7      printf("SMS sent successfully\r\n");

```

(continues on next page)

(continued from previous page)

```

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 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 (lwgsm_sms_send("+0123456789", "text", NULL, NULL, 1 /* 1 means blocking call */)
    ↪ == lwgsmOK) {
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 lwgsm_sms_send() function finishes */
2  void
3  sms_send_fn(lwgsmt_t res, void* arg) {
4      /* Check actual result from device */
5      if (res == lwgsmOK) {
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 (lwgsm_sms_send("number", "text message", sms_send_fn, NULL, 0 /* 0 means non-
17    ↳blocking call */) == lwgsmOK) {
18     /* At this point we only know that command has been sent to queue */
19     printf("SMS send message command sent to queue.\r\n");
20 } else {
21     /* Error writing message to queue */
22     printf("Cannot send SMS send message command to queue. Maybe out of memory? Check
23    ↳result from function\r\n");

```

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

Example of a bad code:

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

```

1  /* Hostname event function, called when lwgsm_sms_send() function finishes */
2  void
3  sms_send_fn(lwgsmt_t res, void* arg) {
4      /* Check actual result from device */
5      if (res == lwgsmOK) {
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 (lwgsm_sms_send("number", message, sms_send_fn, NULL, 0 /* 0 means non-
    ↳blocking call */) == lwgsmOK) {

```

(continues on next page)

(continued from previous page)

```

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 *LwGSM* library is platform independent, written in ANSI C99, 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 `lwgsm_ll_init()` and `lwgsm_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 *Input module* 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 `lwgsm_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 `lwgsm/src/system/lwgsm_sys_template.c` to the same folder and rename it to application port, eg. `lwgsm_sys_win32.c`
- Open newly created file and implement all system functions
- Copy folder `lwgsm/src/include/system/port/template/*` to the same folder and rename *folder name* to application port, eg. `cmsis_os`
- Open `lwgsm_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 `lwgsm_input_process()` or `lwgsm_input()` functions, based on application configuration of `LWGSM_CFG_INPUT_USE_PROCESS` parameter.
  - When `LWGSM_CFG_INPUT_USE_PROCESS` is disabled, dedicated receive buffer is created by *LwGSM* library and `lwgsm_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 `LWGSM_CFG_INPUT_USE_PROCESS` is enabled, `lwgsm_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 `LWGSM_MEM_SIZE` size
- It sets *send* and *reset* callback functions for *LwGSM* library

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

```

1  /**
2   * \file          lwgsm_ll_win32.c
3   * \brief         Low-level communication with GSM device for WIN32
4   */
5
6  /**
7   * Copyright (c) 2020 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 LwGSM - Lightweight GSM-AT library.
30  *
31  * Author:          Tilen MAJERLE <tilen@majerle.eu>
32  * Version:         v0.1.0
33  */
34 #include "system/lwgsm_ll.h"
35 #include "lwgsm/lwgsm.h"
36 #include "lwgsm/lwgsm_mem.h"
37 #include "lwgsm/lwgsm_input.h"
38
39 #if !__DOXYGEN__
40
41 static uint8_t initialized = 0;
42 static HANDLE thread_handle;
43 static volatile HANDLE com_port;          /*!< COM port handle */
44 static uint8_t data_buffer[0x1000];      /*!< Received data array */
45
46 static void uart_thread(void* param);
47
48 /**
49  * \brief         Send data to GSM device, function called from GSM stack when we
50  * ↪ have data to send
51  * \param[in]    data: Pointer to data to send
52  * \param[in]    len: Number of bytes to send
53  * \return       Number of bytes sent
54  */
55 static size_t

```

(continues on next page)

(continued from previous page)

```

55 send_data(const void* data, size_t len) {
56     DWORD written;
57     if (com_port != NULL) {
58 #if !LWGSM_CFG_AT_ECHO
59         const uint8_t* d = data;
60         HANDLE hConsole;
61
62         hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
63         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
64         for (DWORD i = 0; i < len; ++i) {
65             printf("%c", d[i]);
66         }
67         SetConsoleTextAttribute(hConsole, FOREGROUND_RED | FOREGROUND_GREEN |
↪FOREGROUND_BLUE);
68 #endif /* !LWGSM_CFG_AT_ECHO */
69
70         /* Write data to AT port */
71         WriteFile(com_port, data, len, &written, NULL);
72         FlushFileBuffers(com_port);
73         return written;
74     }
75     return 0;
76 }
77
78 /**
79  * \brief          Configure UART (USB to UART)
80  */
81 static void
82 configure_uart(uint32_t baudrate) {
83     DCB dcb = { 0 };
84     dcb.DCBlength = sizeof(dcb);
85
86     /*
87      * On first call,
88      * create virtual file on selected COM port and open it
89      * as generic read and write
90      */
91     if (!initialized) {
92         static const LPCWSTR com_ports[] = {
93             L"\\\\.\\COM23",
94             L"\\\\.\\COM12",
95             L"\\\\.\\COM9",
96             L"\\\\.\\COM8",
97             L"\\\\.\\COM4"
98         };
99         for (size_t i = 0; i < sizeof(com_ports) / sizeof(com_ports[0]); ++i) {
100             com_port = CreateFile(com_ports[i],
101                                     GENERIC_READ | GENERIC_WRITE,
102                                     0,
103                                     0,
104                                     OPEN_EXISTING,
105                                     0,
106                                     NULL
107                                 );
108             if (GetCommState(com_port, &dcb)) {
109                 printf("COM PORT %s opened!\\r\\n", (const char*)com_ports[i]);
110                 break;

```

(continues on next page)



(continued from previous page)

```

111     }
112 }
113 }
114
115 /* Configure COM port parameters */
116 if (GetCommState(com_port, &dcb)) {
117     COMMTIMEOUTS timeouts;
118
119     dcb.BaudRate = baudrate;
120     dcb.ByteSize = 8;
121     dcb.Parity = NOPARITY;
122     dcb.StopBits = ONESTOPBIT;
123
124     if (!SetCommState(com_port, &dcb)) {
125         printf("Cannot set COM PORT info\r\n");
126     }
127     if (GetCommTimeouts(com_port, &timeouts)) {
128         /* Set timeout to return immediately from ReadFile function */
129         timeouts.ReadIntervalTimeout = MAXDWORD;
130         timeouts.ReadTotalTimeoutConstant = 0;
131         timeouts.ReadTotalTimeoutMultiplier = 0;
132         if (!SetCommTimeouts(com_port, &timeouts)) {
133             printf("Cannot set COM PORT timeouts\r\n");
134         }
135         GetCommTimeouts(com_port, &timeouts);
136     } else {
137         printf("Cannot get COM PORT timeouts\r\n");
138     }
139 } else {
140     printf("Cannot get COM PORT info\r\n");
141 }
142
143 /* On first function call, create a thread to read data from COM port */
144 if (!initialized) {
145     thread_handle = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)uart_thread, NULL,
146 ↪ 0, 0);
147 }
148
149 /**
150  * \brief          UART thread
151  */
152 static void
153 uart_thread(void* param) {
154     DWORD bytes_read;
155     lwgsm_sys_sem_t sem;
156     FILE* file = NULL;
157
158     lwgsm_sys_sem_create(&sem, 0);           /* Create semaphore for delay */
159 ↪ functions */
160
161     while (com_port == NULL) {
162         lwgsm_sys_sem_wait(&sem, 1);         /* Add some delay with yield */
163     }
164
165     fopen_s(&file, "log_file.txt", "w+");    /* Open debug file in write mode */
166     while (1) {

```

(continues on next page)

(continued from previous page)

```

166     /*
167     * Try to read data from COM port
168     * and send it to upper layer for processing
169     */
170     do {
171         ReadFile(com_port, data_buffer, sizeof(data_buffer), &bytes_read, NULL);
172         if (bytes_read > 0) {
173             HANDLE hConsole;
174             hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
175             SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
176             for (DWORD i = 0; i < bytes_read; ++i) {
177                 printf("%c", data_buffer[i]);
178             }
179             SetConsoleTextAttribute(hConsole, FOREGROUND_RED | FOREGROUND_GREEN |
180 ↪ FOREGROUND_BLUE);
181
182             /* Send received data to input processing module */
183             #if LWGSM_CFG_INPUT_USE_PROCESS
184                 lwgsm_input_process(data_buffer, (size_t)bytes_read);
185             #else /* LWGSM_CFG_INPUT_USE_PROCESS */
186                 lwgsm_input(data_buffer, (size_t)bytes_read);
187             #endif /* !LWGSM_CFG_INPUT_USE_PROCESS */
188
189             /* Write received data to output debug file */
190             if (file != NULL) {
191                 fwrite(data_buffer, 1, bytes_read, file);
192                 fflush(file);
193             }
194         } while (bytes_read == (DWORD)sizeof(data_buffer));
195
196         /* Implement delay to allow other tasks processing */
197         lwgsm_sys_sem_wait(&sem, 1);
198     }
199 }
200
201 /**
202  * \brief          Callback function called from initialization process
203  *
204  * \note           This function may be called multiple times if AT baudrate is
205 ↪ changed from application.
206  *                It is important that every configuration except AT baudrate is
207 ↪ configured only once!
208  *
209  * \note           This function may be called from different threads in GSM stack
210 ↪ when using OS.
211  *                When \ref LWGSM_CFG_INPUT_USE_PROCESS is set to 1, this function
212 ↪ may be called from user UART thread.
213  *
214  * \param[in,out]  ll: Pointer to \ref lwgsm_ll_t structure to fill data for
215 ↪ communication functions
216  * \return          \ref lwgsmOK on success, member of \ref lwgsmr_t enumeration
217 ↪ otherwise
218  */
219 lwgsmr_t
220 lwgsm_ll_init(lwgsm_ll_t* ll) {
221     #if !LWGSM_CFG_MEM_CUSTOM

```

(continues on next page)

(continued from previous page)

```

216  /* Step 1: Configure memory for dynamic allocations */
217  static uint8_t memory[0x10000];          /* Create memory for dynamic_
↳ allocations with specific size */
218
219  /*
220   * Create memory region(s) of memory.
221   * If device has internal/external memory available,
222   * multiple memories may be used
223   */
224  lwgsm_mem_region_t mem_regions[] = {
225      { memory, sizeof(memory) }
226  };
227  if (!initialized) {
228      lwgsm_mem_assignmemory(mem_regions, LWGSM_ARRAYSIZE(mem_regions)); /* Assign_
↳ memory for allocations to GSM library */
229  }
230  #endif /* !LWGSM_CFG_MEM_CUSTOM */
231
232  /* Step 2: Set AT port send function to use when we have data to transmit */
233  if (!initialized) {
234      ll->send_fn = send_data;              /* Set callback function to send data_
↳ */
235  }
236
237  /* Step 3: Configure AT port to be able to send/receive data to/from GSM device */
238  configure_uart(ll->uart.baudrate);        /* Initialize UART for communication_
↳ */
239  initialized = 1;
240  return lwgsmOK;
241  }
242
243  #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 `lwgsm_input_process()` function to directly process received data without using intermediate receive buffer
- Memory manager has been assigned to 1 region of `LWGSM_MEM_SIZE` size
- It sets *send* and *reset* callback functions for *LwGSM* library

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

```

1  /**
2   * \file          lwgsm_ll_stm32.c
3   * \brief         Generic STM32 driver, included in various STM32 driver variants
4   */
5
6  /*
7   * Copyright (c) 2020 Tilen MAJERLE

```

(continues on next page)

(continued from previous page)

```

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 LwGSM - Lightweight GSM-AT library.
30 *
31 * Author:          Tilen MAJERLE <tilen@majerle.eu>
32 * Version:         v0.1.0
33 */
34
35 /*
36  * How it works
37  *
38  * On first call to \ref lwgsm_ll_init, new thread is created and processed in usart_
39  * ll_thread function.
40  * USART is configured in RX DMA mode and any incoming bytes are processed inside_
41  * thread function.
42  * DMA and USART implement interrupt handlers to notify main thread about new data_
43  * ready to send to upper layer.
44  *
45  * More about UART + RX DMA: https://github.com/MaJerle/stm32-usart-dma-rx-tx
46  *
47  * \ref LWGSM_CFG_INPUT_USE_PROCESS must be enabled in `lwgsm_config.h` to use this_
48  * driver.
49  */
50
51 #include "lwgsm/lwgsm.h"
52 #include "lwgsm/lwgsm_mem.h"
53 #include "lwgsm/lwgsm_input.h"
54 #include "system/lwgsm_ll.h"
55
56 #if !__DOXYGEN__
57
58 #if !LWGSM_CFG_INPUT_USE_PROCESS
59 #error "LWGSM_CFG_INPUT_USE_PROCESS must be enabled in `lwgsm_config.h` to use this_
60 driver."
61 #endif /* LWGSM_CFG_INPUT_USE_PROCESS */
62
63 #if !defined(LWGSM_USART_DMA_RX_BUFF_SIZE)
64 #define LWGSM_USART_DMA_RX_BUFF_SIZE 0x1000
65 #endif /* !defined(LWGSM_USART_DMA_RX_BUFF_SIZE) */

```

(continues on next page)

(continued from previous page)

```

60
61 #if !defined(LWGSM_MEM_SIZE)
62 #define LWGSM_MEM_SIZE                0x1000
63 #endif /* !defined(LWGSM_MEM_SIZE) */
64
65 #if !defined(LWGSM_USART_RDR_NAME)
66 #define LWGSM_USART_RDR_NAME          RDR
67 #endif /* !defined(LWGSM_USART_RDR_NAME) */
68
69 /* USART memory */
70 static uint8_t    usart_mem[LWGSM_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     LWGSM_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(LWGSM_USART_DMA_RX_STREAM)
97         pos = sizeof(usart_mem) - LL_DMA_GetDataLength(LWGSM_USART_DMA, LWGSM_USART_
98 ↪DMA_RX_STREAM);
99 #else
100         pos = sizeof(usart_mem) - LL_DMA_GetDataLength(LWGSM_USART_DMA, LWGSM_USART_
101 ↪DMA_RX_CH);
102 #endif /* defined(LWGSM_USART_DMA_RX_STREAM) */
103         if (pos != old_pos && is_running) {
104             if (pos > old_pos) {
105                 lwgsm_input_process(&usart_mem[old_pos], pos - old_pos);
106             } else {
107                 lwgsm_input_process(&usart_mem[old_pos], sizeof(usart_mem) - old_pos);
108                 if (pos > 0) {
109                     lwgsm_input_process(&usart_mem[0], pos);
110                 }
111             }
112             old_pos = pos;
113             if (old_pos == sizeof(usart_mem)) {
114                 old_pos = 0;
115             }
116         }
117     }
118 }

```

(continues on next page)

(continued from previous page)

```

115     }
116 }
117
118 /**
119  * \brief          Configure UART using DMA for receive in double buffer mode and
120  * ↪ IDLE 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         LWGSM_USART_CLK;
131         LWGSM_USART_DMA_CLK;
132         LWGSM_USART_TX_PORT_CLK;
133         LWGSM_USART_RX_PORT_CLK;
134
135 #if defined(LWGSM_RESET_PIN)
136         LWGSM_RESET_PORT_CLK;
137 #endif /* defined(LWGSM_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(LWGSM_RESET_PIN)
147         /* Configure RESET pin */
148         gpio_init.Pin = LWGSM_RESET_PIN;
149         LL_GPIO_Init(LWGSM_RESET_PORT, &gpio_init);
150 #endif /* defined(LWGSM_RESET_PIN) */
151
152         /* Configure USART pins */
153         gpio_init.Mode = LL_GPIO_MODE_ALTERNATE;
154
155         /* TX PIN */
156         gpio_init.Alternate = LWGSM_USART_TX_PIN_AF;
157         gpio_init.Pin = LWGSM_USART_TX_PIN;
158         LL_GPIO_Init(LWGSM_USART_TX_PORT, &gpio_init);
159
160         /* RX PIN */
161         gpio_init.Alternate = LWGSM_USART_RX_PIN_AF;
162         gpio_init.Pin = LWGSM_USART_RX_PIN;
163         LL_GPIO_Init(LWGSM_USART_RX_PORT, &gpio_init);
164
165         /* Configure UART */
166         LL_USART_DeInit(LWGSM_USART);
167         LL_USART_StructInit(&usart_init);
168         usart_init.BaudRate = baudrate;
169         usart_init.DataWidth = LL_USART_DATAWIDTH_8B;
170         usart_init.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
171         usart_init.OverSampling = LL_USART_OVERSAMPLING_16;

```

(continues on next page)

(continued from previous page)

```

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(LWGSM_USART, &usart_init);
175
176     /* Enable USART interrupts and DMA request */
177     LL_USART_EnableIT_IDLE(LWGSM_USART);
178     LL_USART_EnableIT_PE(LWGSM_USART);
179     LL_USART_EnableIT_ERROR(LWGSM_USART);
180     LL_USART_EnableDMAReq_RX(LWGSM_USART);
181
182     /* Enable USART interrupts */
183     NVIC_SetPriority(LWGSM_USART_IRQ, NVIC_EncodePriority(NVIC_
↪GetPriorityGrouping(), 0x07, 0x00));
184     NVIC_EnableIRQ(LWGSM_USART_IRQ);
185
186     /* Configure DMA */
187     is_running = 0;
188     #if defined(LWGSM_USART_DMA_RX_STREAM)
189         LL_DMA_DeInit(LWGSM_USART_DMA, LWGSM_USART_DMA_RX_STREAM);
190         dma_init.Channel = LWGSM_USART_DMA_RX_CH;
191     #else
192         LL_DMA_DeInit(LWGSM_USART_DMA, LWGSM_USART_DMA_RX_CH);
193         dma_init.PeriphRequest = LWGSM_USART_DMA_RX_REQ_NUM;
194     #endif /* defined(LWGSM_USART_DMA_RX_STREAM) */
195     dma_init.PeriphOrM2MSrcAddress = (uint32_t)&LWGSM_USART->LWGSM_USART_RDR_NAME;
196     dma_init.MemoryOrM2MDstAddress = (uint32_t)usart_mem;
197     dma_init.Direction = LL_DMA_DIRECTION_PERIPH_TO_MEMORY;
198     dma_init.Mode = LL_DMA_MODE_CIRCULAR;
199     dma_init.PeriphOrM2MSrcIncMode = LL_DMA_PERIPH_NOINCREMENT;
200     dma_init.MemoryOrM2MDstIncMode = LL_DMA_MEMORY_INCREMENT;
201     dma_init.PeriphOrM2MSrcDataSize = LL_DMA_PDATAALIGN_BYTE;
202     dma_init.MemoryOrM2MDstDataSize = LL_DMA_MDATAALIGN_BYTE;
203     dma_init.NbData = sizeof(usart_mem);
204     dma_init.Priority = LL_DMA_PRIORITY_MEDIUM;
205     #if defined(LWGSM_USART_DMA_RX_STREAM)
206         LL_DMA_Init(LWGSM_USART_DMA, LWGSM_USART_DMA_RX_STREAM, &dma_init);
207     #else
208         LL_DMA_Init(LWGSM_USART_DMA, LWGSM_USART_DMA_RX_CH, &dma_init);
209     #endif /* defined(LWGSM_USART_DMA_RX_STREAM) */
210
211     /* Enable DMA interrupts */
212     #if defined(LWGSM_USART_DMA_RX_STREAM)
213         LL_DMA_EnableIT_HT(LWGSM_USART_DMA, LWGSM_USART_DMA_RX_STREAM);
214         LL_DMA_EnableIT_TC(LWGSM_USART_DMA, LWGSM_USART_DMA_RX_STREAM);
215         LL_DMA_EnableIT_TE(LWGSM_USART_DMA, LWGSM_USART_DMA_RX_STREAM);
216         LL_DMA_EnableIT_FE(LWGSM_USART_DMA, LWGSM_USART_DMA_RX_STREAM);
217         LL_DMA_EnableIT_DME(LWGSM_USART_DMA, LWGSM_USART_DMA_RX_STREAM);
218     #else
219         LL_DMA_EnableIT_HT(LWGSM_USART_DMA, LWGSM_USART_DMA_RX_CH);
220         LL_DMA_EnableIT_TC(LWGSM_USART_DMA, LWGSM_USART_DMA_RX_CH);
221         LL_DMA_EnableIT_TE(LWGSM_USART_DMA, LWGSM_USART_DMA_RX_CH);
222     #endif /* defined(LWGSM_USART_DMA_RX_STREAM) */
223
224     /* Enable DMA interrupts */
225     NVIC_SetPriority(LWGSM_USART_DMA_RX_IRQ, NVIC_EncodePriority(NVIC_
↪GetPriorityGrouping(), 0x07, 0x00));

```

(continues on next page)

(continued from previous page)

```

226     NVIC_EnableIRQ(LWGSM_USART_DMA_RX_IRQ);
227
228     old_pos = 0;
229     is_running = 1;
230
231     /* Start DMA and USART */
232 #if defined(LWGSM_USART_DMA_RX_STREAM)
233     LL_DMA_EnableStream(LWGSM_USART_DMA, LWGSM_USART_DMA_RX_STREAM);
234 #else
235     LL_DMA_EnableChannel(LWGSM_USART_DMA, LWGSM_USART_DMA_RX_CH);
236 #endif /* defined(LWGSM_USART_DMA_RX_STREAM) */
237     LL_USART_Enable(LWGSM_USART);
238 } else {
239     osDelay(10);
240     LL_USART_Disable(LWGSM_USART);
241     usart_init.BaudRate = baudrate;
242     LL_USART_Init(LWGSM_USART, &usart_init);
243     LL_USART_Enable(LWGSM_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 = {
252         .stack_size = 1024
253     };
254     usart_ll_thread_id = osThreadNew(usart_ll_thread, usart_ll_mbox_id, &attr);
255 }
256 }
257
258 #if defined(LWGSM_RESET_PIN)
259 /**
260  * \brief      Hardware reset callback
261  */
262 static uint8_t
263 reset_device(uint8_t state) {
264     if (state) { /* Activate reset line */
265         LL_GPIO_ResetOutputPin(LWGSM_RESET_PORT, LWGSM_RESET_PIN);
266     } else {
267         LL_GPIO_SetOutputPin(LWGSM_RESET_PORT, LWGSM_RESET_PIN);
268     }
269     return 1;
270 }
271 #endif /* defined(LWGSM_RESET_PIN) */
272
273 /**
274  * \brief      Send data to GSM device
275  * \param[in]  data: Pointer to data to send
276  * \param[in]  len: Number of bytes to send
277  * \return     Number of bytes sent
278  */
279 static size_t
280 send_data(const void* data, size_t len) {
281     const uint8_t* d = data;
282

```

(continues on next page)



(continued from previous page)

```

283     for (size_t i = 0; i < len; ++i, ++d) {
284         LL_USART_TransmitData8(LWGSM_USART, *d);
285         while (!LL_USART_IsActiveFlag_TXE(LWGSM_USART)) {}
286     }
287     return len;
288 }
289
290 /**
291  * \brief          Callback function called from initialization process
292  * \note           This function may be called multiple times if AT baudrate is
↳ changed from application
293  * \param[in,out]  ll: Pointer to \ref lwgsm_ll_t structure to fill data for
↳ communication functions
294  * \param[in]      baudrate: Baudrate to use on AT port
295  * \return         Member of \ref lwgsmr_t enumeration
296  */
297 lwgsmr_t
298 lwgsm_ll_init(lwgsm_ll_t* ll) {
299     #if !LWGSM_CFG_MEM_CUSTOM
300         static uint8_t memory[LWGSM_MEM_SIZE];
301         lwgsm_mem_region_t mem_regions[] = {
302             { memory, sizeof(memory) }
303         };
304
305         if (!initialized) {
306             lwgsm_mem_assignmemory(mem_regions, LWGSM_ARRAYSIZE(mem_regions)); /* Assign
↳ memory for allocations */
307         }
308     #endif /* !LWGSM_CFG_MEM_CUSTOM */
309
310     if (!initialized) {
311         ll->send_fn = send_data; /* Set callback function to send data
↳ */
312     #if defined(LWGSM_RESET_PIN)
313         ll->reset_fn = reset_device; /* Set callback for hardware reset */
314     #endif /* defined(LWGSM_RESET_PIN) */
315     }
316
317     configure_uart(ll->uart.baudrate); /* Initialize UART for communication
↳ */
318     initialized = 1;
319     return lwgsmOK;
320 }
321
322 /**
323  * \brief          Callback function to de-init low-level communication part
324  * \param[in,out]  ll: Pointer to \ref lwgsm_ll_t structure to fill data for
↳ communication functions
325  * \return         \ref lwgsmOK on success, member of \ref lwgsmr_t enumeration
↳ otherwise
326  */
327 lwgsmr_t
328 lwgsm_ll_deinit(lwgsm_ll_t* ll) {
329     if (usart_ll_mbox_id != NULL) {
330         osMessageQueueId_t tmp = usart_ll_mbox_id;
331         usart_ll_mbox_id = NULL;
332         osMessageQueueDelete(tmp);

```

(continues on next page)

(continued from previous page)

```

333     }
334     if (usart_ll_thread_id != NULL) {
335         osThreadId_t tmp = usart_ll_thread_id;
336         usart_ll_thread_id = NULL;
337         osThreadTerminate(tmp);
338     }
339     initialized = 0;
340     LWGSM_UNUSED(ll);
341     return lwgsmOK;
342 }
343
344 /**
345  * \brief          UART global interrupt handler
346  */
347 void
348 LWGSM_USART_IRQHANDLER(void) {
349     LL_USART_ClearFlag_IDLE(LWGSM_USART);
350     LL_USART_ClearFlag_PE(LWGSM_USART);
351     LL_USART_ClearFlag_FE(LWGSM_USART);
352     LL_USART_ClearFlag_ORE(LWGSM_USART);
353     LL_USART_ClearFlag_NE(LWGSM_USART);
354
355     if (usart_ll_mbox_id != NULL) {
356         void* d = (void*)1;
357         osMessageQueuePut(usart_ll_mbox_id, &d, 0, 0);
358     }
359 }
360
361 /**
362  * \brief          UART DMA stream/channel handler
363  */
364 void
365 LWGSM_USART_DMA_RX_IRQHANDLER(void) {
366     LWGSM_USART_DMA_RX_CLEAR_TC;
367     LWGSM_USART_DMA_RX_CLEAR_HT;
368
369     if (usart_ll_mbox_id != NULL) {
370         void* d = (void*)1;
371         osMessageQueuePut(usart_ll_mbox_id, &d, 0, 0);
372     }
373 }
374
375 #endif /* !__DOXYGEN__ */

```

### Example: System functions for WIN32

Listing 10: Actual header implementation of system functions for WIN32

```

1  /**
2   * \file          lwgsm_sys_port.h
3   * \brief          WIN32 based system file implementation
4   */
5
6  /**

```

(continues on next page)

(continued from previous page)

```

7  * Copyright (c) 2020 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 LwGSM - Lightweight GSM-AT library.
30 *
31 * Author:          Tilen MAJERLE <tilen@majerle.eu>
32 * Version:         v0.1.0
33 */
34 #ifndef LWGSM_HDR_SYSTEM_PORT_H
35 #define LWGSM_HDR_SYSTEM_PORT_H
36
37 #include <stdint.h>
38 #include <stdlib.h>
39 #include "lwgsm/lwgsm_opt.h"
40 #include "windows.h"
41
42 #ifdef __cplusplus
43 extern "C" {
44 #endif /* __cplusplus */
45
46 #if LWGSM_CFG_OS && !__DOXYGEN__
47
48 typedef HANDLE          lwgsm_sys_mutex_t;
49 typedef HANDLE          lwgsm_sys_sem_t;
50 typedef HANDLE          lwgsm_sys_mbox_t;
51 typedef HANDLE          lwgsm_sys_thread_t;
52 typedef int             lwgsm_sys_thread_prio_t;
53
54 #define LWGSM_SYS_MUTEX_NULL      ((HANDLE)0)
55 #define LWGSM_SYS_SEM_NULL        ((HANDLE)0)
56 #define LWGSM_SYS_MBOX_NULL       ((HANDLE)0)
57 #define LWGSM_SYS_TIMEOUT         (INFINITE)
58 #define LWGSM_SYS_THREAD_PRIO     (0)
59 #define LWGSM_SYS_THREAD_SS       (4096)
60
61 #endif /* LWGSM_CFG_OS && !__DOXYGEN__ */
62
63 #ifdef __cplusplus

```

(continues on next page)

(continued from previous page)

```

64 }
65 #endif /* __cplusplus */
66
67 #endif /* LWGSM_HDR_SYSTEM_PORT_H */

```

Listing 11: Actual implementation of system functions for WIN32

```

1  /**
2   * \file          lwgsm_sys_win32.c
3   * \brief         System dependant functions for WIN32
4   */
5
6  /**
7   * Copyright (c) 2020 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 LwGSM - Lightweight GSM-AT library.
30  *
31  * Author:          Tilen MAJERLE <tilen@majerle.eu>
32  * Version:         v0.1.0
33  */
34 #include <string.h>
35 #include <stdlib.h>
36 #include "system/lwgsm_sys.h"
37
38 #if !__DOXYGEN__
39
40 /**
41  * \brief         Custom message queue implementation for WIN32
42  */
43 #ifndef struct {
44     lwgsm_sys_sem_t sem_not_empty;          /*!< Semaphore indicates not empty */
45     lwgsm_sys_sem_t sem_not_full;          /*!< Semaphore indicates not full */
46     lwgsm_sys_sem_t sem;                   /*!< Semaphore to lock access */
47     size_t in, out, size;
48     void* entries[1];
49 } win32_mbox_t;

```

(continues on next page)

(continued from previous page)

```

50
51 static LARGE_INTEGER freq, sys_start_time;
52 static lwgsm_sys_mutex_t sys_mutex;           /* Mutex ID for main protection */
53
54 static uint8_t
55 mbox_is_full(win32_mbox_t* m) {
56     size_t size = 0;
57     if (m->in > m->out) {
58         size = (m->in - m->out);
59     } else if (m->out > m->in) {
60         size = m->size - m->out + m->in;
61     }
62     return size == m->size - 1;
63 }
64
65 static uint8_t
66 mbox_is_empty(win32_mbox_t* m) {
67     return m->in == m->out;
68 }
69
70 static uint32_t
71 osKernelSysTick(void) {
72     LONGLONG ret;
73     LARGE_INTEGER now;
74
75     QueryPerformanceFrequency(&freq);           /* Get frequency */
76     QueryPerformanceCounter(&now);              /* Get current time */
77     ret = now.QuadPart - sys_start_time.QuadPart;
78     return (uint32_t)((ret) * 1000) / freq.QuadPart;
79 }
80
81 uint8_t
82 lwgsm_sys_init(void) {
83     QueryPerformanceFrequency(&freq);
84     QueryPerformanceCounter(&sys_start_time);
85
86     lwgsm_sys_mutex_create(&sys_mutex);
87     return 1;
88 }
89
90 uint32_t
91 lwgsm_sys_now(void) {
92     return osKernelSysTick();
93 }
94
95 uint8_t
96 lwgsm_sys_protect(void) {
97     lwgsm_sys_mutex_lock(&sys_mutex);
98     return 1;
99 }
100
101 uint8_t
102 lwgsm_sys_unprotect(void) {
103     lwgsm_sys_mutex_unlock(&sys_mutex);
104     return 1;
105 }
106

```

(continues on next page)

(continued from previous page)

```

107 uint8_t
108 lwgsm_sys_mutex_create(lwgsm_sys_mutex_t* p) {
109     *p = CreateMutex(NULL, FALSE, NULL);
110     return *p != NULL;
111 }
112
113 uint8_t
114 lwgsm_sys_mutex_delete(lwgsm_sys_mutex_t* p) {
115     return CloseHandle(*p);
116 }
117
118 uint8_t
119 lwgsm_sys_mutex_lock(lwgsm_sys_mutex_t* p) {
120     DWORD ret;
121     ret = WaitForSingleObject(*p, INFINITE);
122     if (ret != WAIT_OBJECT_0) {
123         return 0;
124     }
125     return 1;
126 }
127
128 uint8_t
129 lwgsm_sys_mutex_unlock(lwgsm_sys_mutex_t* p) {
130     return (uint8_t)ReleaseMutex(*p);
131 }
132
133 uint8_t
134 lwgsm_sys_mutex_isvalid(lwgsm_sys_mutex_t* p) {
135     return p != NULL && *p != NULL;
136 }
137
138 uint8_t
139 lwgsm_sys_mutex_invalid(lwgsm_sys_mutex_t* p) {
140     *p = LWGSM_SYS_MUTEX_NULL;
141     return 1;
142 }
143
144 uint8_t
145 lwgsm_sys_sem_create(lwgsm_sys_sem_t* p, uint8_t cnt) {
146     HANDLE h;
147     h = CreateSemaphore(NULL, !!cnt, 1, NULL);
148     *p = h;
149     return *p != NULL;
150 }
151
152 uint8_t
153 lwgsm_sys_sem_delete(lwgsm_sys_sem_t* p) {
154     return CloseHandle(*p);
155 }
156
157 uint32_t
158 lwgsm_sys_sem_wait(lwgsm_sys_sem_t* p, uint32_t timeout) {
159     DWORD ret;
160     uint32_t tick = osKernelSysTick();
161
162     if (timeout == 0) {
163         ret = WaitForSingleObject(*p, INFINITE);

```

(continues on next page)

(continued from previous page)

```

164     return 1;
165 } else {
166     ret = WaitForSingleObject(*p, timeout);
167     if (ret == WAIT_OBJECT_0) {
168         return 1;
169     } else {
170         return LWGSM_SYS_TIMEOUT;
171     }
172 }
173 }
174
175 uint8_t
176 lwgsm_sys_sem_release(lwgsm_sys_sem_t* p) {
177     return ReleaseSemaphore(*p, 1, NULL);
178 }
179
180 uint8_t
181 lwgsm_sys_sem_isvalid(lwgsm_sys_sem_t* p) {
182     return p != NULL && *p != NULL;
183 }
184
185 uint8_t
186 lwgsm_sys_sem_invalid(lwgsm_sys_sem_t* p) {
187     *p = LWGSM_SYS_SEM_NULL;
188     return 1;
189 }
190
191 uint8_t
192 lwgsm_sys_mbox_create(lwgsm_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         lwgsm_sys_sem_create(&mbox->sem, 1);
202         lwgsm_sys_sem_create(&mbox->sem_not_empty, 0);
203         lwgsm_sys_sem_create(&mbox->sem_not_full, 0);
204         *b = mbox;
205     }
206     return *b != NULL;
207 }
208
209 uint8_t
210 lwgsm_sys_mbox_delete(lwgsm_sys_mbox_t* b) {
211     win32_mbox_t* mbox = *b;
212     lwgsm_sys_sem_delete(&mbox->sem);
213     lwgsm_sys_sem_delete(&mbox->sem_not_full);
214     lwgsm_sys_sem_delete(&mbox->sem_not_empty);
215     free(mbox);
216     return 1;
217 }
218
219 uint32_t

```

(continues on next page)

(continued from previous page)

```

220 lwgsm_sys_mbox_put(lwgsm_sys_mbox_t* b, void* m) {
221     win32_mbox_t* mbox = *b;
222     uint32_t time = osKernelSysTick();          /* Get start time */
223
224     lwgsm_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         lwgsm_sys_sem_release(&mbox->sem);      /* Release semaphore */
233         lwgsm_sys_sem_wait(&mbox->sem_not_full, 0); /* Wait for semaphore indicating_
↳not full */
234         lwgsm_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     lwgsm_sys_sem_release(&mbox->sem_not_empty); /* Signal non-empty state */
241     lwgsm_sys_sem_release(&mbox->sem);          /* Release access for other threads */
242     return osKernelSysTick() - time;
243 }
244
245 uint32_t
246 lwgsm_sys_mbox_get(lwgsm_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 (lwgsm_sys_sem_wait(&mbox->sem, timeout) == LWGSM_SYS_TIMEOUT) {
254         return LWGSM_SYS_TIMEOUT;
255     }
256     while (mbox_is_empty(mbox)) {
257         lwgsm_sys_sem_release(&mbox->sem);
258         if (lwgsm_sys_sem_wait(&mbox->sem_not_empty, timeout) == LWGSM_SYS_TIMEOUT) {
259             return LWGSM_SYS_TIMEOUT;
260         }
261         lwgsm_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     lwgsm_sys_sem_release(&mbox->sem_not_full);
268     lwgsm_sys_sem_release(&mbox->sem);
269
270     return osKernelSysTick() - time;
271 }
272
273 uint8_t
274 lwgsm_sys_mbox_putnow(lwgsm_sys_mbox_t* b, void* m) {
275     win32_mbox_t* mbox = *b;

```

(continues on next page)



(continued from previous page)

```

276     lwgsm_sys_sem_wait(&mbox->sem, 0);
277     if (mbox_is_full(mbox)) {
278         lwgsm_sys_sem_release(&mbox->sem);
279         return 0;
280     }
281     mbox->entries[mbox->in] = m;
282     if (mbox->in == mbox->out) {
283         lwgsm_sys_sem_release(&mbox->sem_not_empty);
284     }
285     if (++mbox->in >= mbox->size) {
286         mbox->in = 0;
287     }
288     lwgsm_sys_sem_release(&mbox->sem);
289     return 1;
290 }
291
292
293 uint8_t
294 lwgsm_sys_mbox_getnow(lwgsm_sys_mbox_t* b, void** m) {
295     win32_mbox_t* mbox = *b;
296
297     lwgsm_sys_sem_wait(&mbox->sem, 0);           /* Wait exclusive access */
298     if (mbox->in == mbox->out) {
299         lwgsm_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     lwgsm_sys_sem_release(&mbox->sem_not_full); /* Queue not full anymore */
308     lwgsm_sys_sem_release(&mbox->sem);         /* Release semaphore */
309     return 1;
310 }
311
312 uint8_t
313 lwgsm_sys_mbox_isvalid(lwgsm_sys_mbox_t* b) {
314     return b != NULL && *b != NULL;           /* Return status if message box is_
315     ↪ valid */
316 }
317
318 uint8_t
319 lwgsm_sys_mbox_invalid(lwgsm_sys_mbox_t* b) {
320     *b = LWGSM_SYS_MBOX_NULL;                /* Invalidate message box */
321     return 1;
322 }
323
324 uint8_t
325 lwgsm_sys_thread_create(lwgsm_sys_thread_t* t, const char* name, lwgsm_sys_thread_fn_
326     ↪ thread_func, void* const arg, size_t stack_size, lwgsm_sys_thread_prio_t prio) {
327     HANDLE h;
328     DWORD id;
329     h = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)thread_func, arg, 0, &id);
330     if (t != NULL) {
331         *t = h;
332     }
333 }

```

(continues on next page)

(continued from previous page)

```

331     return h != NULL;
332 }
333
334 uint8_t
335 lwgsm_sys_thread_terminate(lwgsm_sys_thread_t* t) {
336     HANDLE h = NULL;
337
338     if (t == NULL) {                /* Shall we terminate ourself? */
339         h = GetCurrentThread();      /* Get current thread handle */
340     } else {                        /* We have known thread, find handle_
    ↳by looking at ID */
341         h = *t;
342     }
343     TerminateThread(h, 0);
344     return 1;
345 }
346
347 uint8_t
348 lwgsm_sys_thread_yield(void) {
349     /* Not implemented */
350     return 1;
351 }
352
353 #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          lwgsm_sys_port.h
3   * \brief        System dependent functions for CMSIS-OS based operating system
4   */
5
6  /**
7   * Copyright (c) 2020 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

```

(continues on next page)

(continued from previous page)

```

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 LwGSM - Lightweight GSM-AT library.
30  *
31  * Author:          Tilen MAJERLE <tilen@majerle.eu>
32  * Version:         v0.1.0
33  */
34  #ifndef LWGSM_HDR_SYSTEM_PORT_H
35  #define LWGSM_HDR_SYSTEM_PORT_H
36
37  #include <stdint.h>
38  #include <stdlib.h>
39  #include "lwgsm/lwgsm_opt.h"
40  #include "cmsis_os.h"
41
42  #ifdef __cplusplus
43  extern "C" {
44  #endif /* __cplusplus */
45
46  #if LWGSM_CFG_OS && !__DOXYGEN__
47
48  typedef osMutexId_t          lwgsm_sys_mutex_t;
49  typedef osSemaphoreId_t      lwgsm_sys_sem_t;
50  typedef osMessageQueueId_t   lwgsm_sys_mbox_t;
51  typedef osThreadId_t         lwgsm_sys_thread_t;
52  typedef osPriority_t          lwgsm_sys_thread_prio_t;
53
54  #define LWGSM_SYS_MUTEX_NULL      ((lwgsm_sys_mutex_t)0)
55  #define LWGSM_SYS_SEM_NULL        ((lwgsm_sys_sem_t)0)
56  #define LWGSM_SYS_MBOX_NULL       ((lwgsm_sys_mbox_t)0)
57  #define LWGSM_SYS_TIMEOUT         ((uint32_t)osWaitForever)
58  #define LWGSM_SYS_THREAD_PRIO     (osPriorityNormal)
59  #define LWGSM_SYS_THREAD_SS       (512)
60
61  #endif /* LWGSM_CFG_OS && !__DOXYGEN__ */
62
63  #ifdef __cplusplus
64  }
65  #endif /* __cplusplus */
66
67  #endif /* LWGSM_HDR_SYSTEM_PORT_H */

```

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

```

1  /**
2   * \file          lwgsm_sys_cmsis_os.c
3   * \brief         System dependent functions for CMSIS-OS based operating system
4   */
5
6  /**
7   * Copyright (c) 2020 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 LwGSM - Lightweight GSM-AT library.
30  *
31  * Author:          Tilen MAJERLE <tilen@majerle.eu>
32  * Version:         v0.1.0
33  */
34 #include "system/lwgsmsys.h"
35 #include "cmsis_os.h"
36
37 #if !__DOXYGEN__
38
39 static osMutexId_t sys_mutex;
40
41 uint8_t
42 lwgsmsys_init(void) {
43     lwgsmsys_mutex_create(&sys_mutex);
44     return 1;
45 }
46
47 uint32_t
48 lwgsmsys_now(void) {
49     return osKernelSysTick();
50 }
51
52 uint8_t
53 lwgsmsys_protect(void) {
54     lwgsmsys_mutex_lock(&sys_mutex);
55     return 1;
56 }
57
58 uint8_t
59 lwgsmsys_unprotect(void) {
60     lwgsmsys_mutex_unlock(&sys_mutex);
61     return 1;
62 }
63
64 uint8_t
65 lwgsmsys_mutex_create(lwgsmsys_mutex_t* p) {
66     const osMutexAttr_t attr = {
67         .attr_bits = osMutexRecursive

```

(continues on next page)

(continued from previous page)

```

68     };
69     *p = osMutexNew(&attr);
70     return *p != NULL;
71 }
72
73 uint8_t
74 lwgsm_sys_mutex_delete(lwgsm_sys_mutex_t* p) {
75     return osMutexDelete(*p) == osOK;
76 }
77
78 uint8_t
79 lwgsm_sys_mutex_lock(lwgsm_sys_mutex_t* p) {
80     return osMutexAcquire(*p, osWaitForever) == osOK;
81 }
82
83 uint8_t
84 lwgsm_sys_mutex_unlock(lwgsm_sys_mutex_t* p) {
85     return osMutexRelease(*p) == osOK;
86 }
87
88 uint8_t
89 lwgsm_sys_mutex_isvalid(lwgsm_sys_mutex_t* p) {
90     return p != NULL && *p != NULL;
91 }
92
93 uint8_t
94 lwgsm_sys_mutex_invalid(lwgsm_sys_mutex_t* p) {
95     *p = LWGSM_SYS_MUTEX_NULL;
96     return 1;
97 }
98
99 uint8_t
100 lwgsm_sys_sem_create(lwgsm_sys_sem_t* p, uint8_t cnt) {
101     return (*p = osSemaphoreNew(1, cnt > 0 ? 1 : 0, NULL)) != NULL;
102 }
103
104 uint8_t
105 lwgsm_sys_sem_delete(lwgsm_sys_sem_t* p) {
106     return osSemaphoreDelete(*p) == osOK;
107 }
108
109 uint32_t
110 lwgsm_sys_sem_wait(lwgsm_sys_sem_t* p, uint32_t timeout) {
111     uint32_t tick = osKernelSysTick();
112     return (osSemaphoreAcquire(*p, timeout == 0 ? osWaitForever : timeout) == osOK) ?
113     ↪ (osKernelSysTick() - tick) : LWGSM_SYS_TIMEOUT;
114 }
115
116 uint8_t
117 lwgsm_sys_sem_release(lwgsm_sys_sem_t* p) {
118     return osSemaphoreRelease(*p) == osOK;
119 }
120
121 uint8_t
122 lwgsm_sys_sem_isvalid(lwgsm_sys_sem_t* p) {
123     return p != NULL && *p != NULL;
124 }

```

(continues on next page)

(continued from previous page)

```

124
125 uint8_t
126 lwgsm_sys_sem_invalid(lwgsm_sys_sem_t* p) {
127     *p = LWGSM_SYS_SEM_NULL;
128     return 1;
129 }
130
131 uint8_t
132 lwgsm_sys_mbox_create(lwgsm_sys_mbox_t* b, size_t size) {
133     return (*b = osMessageQueueNew(size, sizeof(void*), NULL)) != NULL;
134 }
135
136 uint8_t
137 lwgsm_sys_mbox_delete(lwgsm_sys_mbox_t* b) {
138     if (osMessageQueueGetCount(*b) > 0) {
139         return 0;
140     }
141     return osMessageQueueDelete(*b) == osOK;
142 }
143
144 uint32_t
145 lwgsm_sys_mbox_put(lwgsm_sys_mbox_t* b, void* m) {
146     uint32_t tick = osKernelSysTick();
147     return osMessageQueuePut(*b, &m, 0, osWaitForever) == osOK ? (osKernelSysTick() -
148 ↪tick) : LWGSM_SYS_TIMEOUT;
149 }
150
151 uint32_t
152 lwgsm_sys_mbox_get(lwgsm_sys_mbox_t* b, void** m, uint32_t timeout) {
153     uint32_t tick = osKernelSysTick();
154     return osMessageQueueGet(*b, m, NULL, timeout == 0 ? osWaitForever : timeout) ==
155 ↪osOK ? (osKernelSysTick() - tick) : LWGSM_SYS_TIMEOUT;
156 }
157
158 uint8_t
159 lwgsm_sys_mbox_putnow(lwgsm_sys_mbox_t* b, void* m) {
160     return osMessageQueuePut(*b, &m, 0, 0) == osOK;
161 }
162
163 uint8_t
164 lwgsm_sys_mbox_getnow(lwgsm_sys_mbox_t* b, void** m) {
165     return osMessageQueueGet(*b, m, NULL, 0) == osOK;
166 }
167
168 uint8_t
169 lwgsm_sys_mbox_isvalid(lwgsm_sys_mbox_t* b) {
170     return b != NULL && *b != NULL;
171 }
172
173 uint8_t
174 lwgsm_sys_mbox_invalid(lwgsm_sys_mbox_t* b) {
175     *b = LWGSM_SYS_MBOX_NULL;
176     return 1;
177 }
178
179 uint8_t
180 lwgsm_sys_thread_create(lwgsm_sys_thread_t* t, const char* name, lwgsm_sys_thread_fn_
181 ↪thread_func, void* const arg, size_t stack_size, lwgsm_sys_thread_prio_t prio) {

```

(continues on next page)

(continued from previous page)

```

179     lwgsm_sys_thread_t id;
180     const osThreadAttr_t thread_attr = {
181         .name = (char*)name,
182         .priority = (osPriority)prio,
183         .stack_size = stack_size > 0 ? stack_size : LWGSM_SYS_THREAD_SS
184     };
185
186     id = osThreadNew(thread_func, arg, &thread_attr);
187     if (t != NULL) {
188         *t = id;
189     }
190     return id != NULL;
191 }
192
193 uint8_t
194 lwgsm_sys_thread_terminate(lwgsm_sys_thread_t* t) {
195     if (t != NULL) {
196         osThreadTerminate(*t);
197     } else {
198         osThreadExit();
199     }
200     return 1;
201 }
202
203 uint8_t
204 lwgsm_sys_thread_yield(void) {
205     osThreadYield();
206     return 1;
207 }
208
209 #endif /* !__DOXYGEN__ */

```

## 5.3 API reference

List of all the modules:

### 5.3.1 LwGSM

#### Ring buffer

*group* **LWGSM\_BUFF**  
Generic ring buffer.

## Defines

### **BUF\_PREF** (*x*)

Buffer function/typedef prefix string.

It is used to change function names in zero time to easily re-use same library between applications.

Use `#define BUF_PREF(x) my_prefix_ ## x` to change all function names to (for example) `my_prefix_buff_init`

**Note** Modification of this macro must be done in header and source file aswell

## Functions

`uint8_t lwgsm_buff_init (lwgsm_buff_t *buff, size_t size)`

Initialize buffer.

**Return** 1 on success, 0 otherwise

### Parameters

- [in] buff: Pointer to buffer structure
- [in] size: Size of buffer in units of bytes

`void lwgsm_buff_free (lwgsm_buff_t *buff)`

Free dynamic allocation if used on memory.

### Parameters

- [in] buff: Pointer to buffer structure

`void lwgsm_buff_reset (lwgsm_buff_t *buff)`

Resets buffer to default values. Buffer size is not modified.

### Parameters

- [in] buff: Buffer handle

`size_t lwgsm_buff_write (lwgsm_buff_t *buff, const void *data, size_t btw)`

Write data to buffer Copies data from data array to buffer and marks buffer as full for maximum count number of bytes.

**Return** Number of bytes written to buffer. When returned value is less than btw, there was no enough memory available to copy full data array

### Parameters

- [in] buff: Buffer handle
- [in] data: Pointer to data to write into buffer
- [in] btw: Number of bytes to write

`size_t lwgsm_buff_read (lwgsm_buff_t *buff, void *data, size_t btr)`

Read data from buffer Copies data from buffer to data array and marks buffer as free for maximum btr number of bytes.

**Return** Number of bytes read and copied to data array



**Parameters**

- [in] buff: Buffer handle
- [out] data: Pointer to output memory to copy buffer data to
- [in] btr: Number of bytes to read

size\_t **lwgsm\_buff\_peek** (*lwgsm\_buff\_t* \*buff, size\_t skip\_count, void \*data, size\_t btp)  
Read from buffer without changing read pointer (peek only)

**Return** Number of bytes peeked and written to output array

**Parameters**

- [in] buff: Buffer handle
- [in] skip\_count: Number of bytes to skip before reading data
- [out] data: Pointer to output memory to copy buffer data to
- [in] btp: Number of bytes to peek

size\_t **lwgsm\_buff\_get\_free** (*lwgsm\_buff\_t* \*buff)  
Get number of bytes in buffer available to write.

**Return** Number of free bytes in memory

**Parameters**

- [in] buff: Buffer handle

size\_t **lwgsm\_buff\_get\_full** (*lwgsm\_buff\_t* \*buff)  
Get number of bytes in buffer available to read.

**Return** Number of bytes ready to be read

**Parameters**

- [in] buff: Buffer handle

void \***lwgsm\_buff\_get\_linear\_block\_read\_address** (*lwgsm\_buff\_t* \*buff)  
Get linear address for buffer for fast read.

**Return** Linear buffer start address

**Parameters**

- [in] buff: Buffer handle

size\_t **lwgsm\_buff\_get\_linear\_block\_read\_length** (*lwgsm\_buff\_t* \*buff)  
Get length of linear block address before it overflows for read operation.

**Return** Linear buffer size in units of bytes for read operation

**Parameters**

- [in] buff: Buffer handle

size\_t **lwgsm\_buff\_skip** (*lwgsm\_buff\_t* \*buff, size\_t len)  
Skip (ignore; advance read pointer) buffer data Marks data as read in the buffer and increases free memory for up to len bytes.

**Note** Useful at the end of streaming transfer such as DMA

**Return** Number of bytes skipped

**Parameters**

- [in] `buff`: Buffer handle
- [in] `len`: Number of bytes to skip and mark as read

`void *lwgsm_buff_get_linear_block_write_address (lwgsm_buff_t *buff)`  
Get linear address for buffer for fast read.

**Return** Linear buffer start address

**Parameters**

- [in] `buff`: Buffer handle

`size_t lwgsm_buff_get_linear_block_write_length (lwgsm_buff_t *buff)`  
Get length of linear block address before it overflows for write operation.

**Return** Linear buffer size in units of bytes for write operation

**Parameters**

- [in] `buff`: Buffer handle

`size_t lwgsm_buff_advance (lwgsm_buff_t *buff, size_t len)`  
Advance write pointer in the buffer. Similar to skip function but modifies write pointer instead of read.

**Note** Useful when hardware is writing to buffer and application needs to increase number of bytes written to buffer by hardware

**Return** Number of bytes advanced for write operation

**Parameters**

- [in] `buff`: Buffer handle
- [in] `len`: Number of bytes to advance

`struct lwgsm_buff_t`  
*#include <lwgsm\_typedefs.h>* Buffer structure.

**Public Members**

`uint8_t *buff`  
Pointer to buffer data. Buffer is considered initialized when `buff != NULL`

`size_t size`  
Size of buffer data. Size of actual buffer is 1 byte less than this value

`size_t r`  
Next read pointer. Buffer is considered empty when `r == w` and full when `w == r - 1`

`size_t w`  
Next write pointer. Buffer is considered empty when `r == w` and full when `w == r - 1`

## Connections

Connections are essential feature of WiFi device and middleware. It is developed with strong focus on its performance and since it may interact with huge amount of data, it tries to use zero-copy (when available) feature, to decrease processing time.

*GSM AT Firmware* by default supports up to 5 connections being active at the same time and supports:

- Up to 5 TCP connections active at the same time
- Up to 5 UDP connections active at the same time
- Up to 1 SSL connection active at a time

---

**Note:** Client or server connections are available. Same API function call are used to send/receive data or close connection.

---

Architecture of the connection API is using callback event functions. This allows maximal optimization in terms of responsiveness on different kind of events.

Example below shows *bare minimum* implementation to:

- Start a new connection to remote host
- Send *HTTP GET* request to remote host
- Process received data in event and print number of received bytes

Listing 14: Client connection minimum example

```

1  #include "client.h"
2  #include "lwgsm/lwgsm.h"
3  #include "lwgsm/lwgsm_network_api.h"
4
5  /* Host parameter */
6  #define CONN_HOST          "example.com"
7  #define CONN_PORT          80
8
9  static lwgsmr_t    conn_callback_func(lwgsm_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     lwgsmr_t res;
27
28     /* Attach to GSM network */
29     lwgsm_network_request_attach();

```

(continues on next page)

(continued from previous page)

```

30
31  /* Start a new connection as client in non-blocking mode */
32  if ((res = lwgsm_conn_start(NULL, LWGSM_CONN_TYPE_TCP, "example.com", 80, NULL,
↪conn_callback_func, 0)) == lwgsmOK) {
33      printf("Connection to " CONN_HOST " started...\r\n");
34  } else {
35      printf("Cannot start connection to " CONN_HOST "!\r\n");
36  }
37  }
38
39  /**
40   * \brief          Event callback function for connection-only
41   * \param[in]      evt: Event information with data
42   * \return         \ref lwgsmOK on success, member of \ref lwgsmr_t otherwise
43   */
44  static lwgsmr_t
45  conn_callback_func(lwgsm_evt_t* evt) {
46      lwgsm_conn_p conn;
47      lwgsmr_t res;
48      uint8_t conn_num;
49
50      conn = lwgsm_conn_get_from_evt(evt);          /* Get connection handle from event_
↪ */
51      if (conn == NULL) {
52          return lwgsmERR;
53      }
54      conn_num = lwgsm_conn_getnum(conn);          /* Get connection number for_
↪ identification */
55      switch (lwgsm_evt_get_type(evt)) {
56          case LWGSM_EVT_CONN_ACTIVE: {            /* Connection just active */
57              printf("Connection %d active!\r\n", (int) conn_num);
58              res = lwgsm_conn_send(conn, req_data, sizeof(req_data) - 1, NULL, 0); /*_
↪ Start sending data in non-blocking mode */
59              if (res == lwgsmOK) {
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                  lwgsm_conn_close(conn, 0);        /* Close the connection */
64              }
65              break;
66          }
67          case LWGSM_EVT_CONN_CLOSE: {              /* Connection closed */
68              if (lwgsm_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 LWGSM_EVT_CONN_SEND: {              /* Data send event */
76              lwgsmr_t res = lwgsm_evt_conn_send_get_result(evt);
77              if (res == lwgsmOK) {
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);

```

(continues on next page)

(continued from previous page)

```

81         }
82         break;
83     }
84     case LWGSM_EVT_CONN_RECV: { /* Data received from remote side */
85         lwgsm_pbuf_p pbuf = lwgsm_evt_conn_recv_get_buff(evt);
86         lwgsm_conn_recved(conn, pbuf); /* Notify stack about received pbuf_
87     ↪ */
88         printf("Received %d bytes on connection %d..\r\n", (int)lwgsm_pbuf_
89     ↪ length(pbuf, 1), (int)conn_num);
90         break;
91     }
92     case LWGSM_EVT_CONN_ERROR: { /* Error connecting to server */
93         const char* host = lwgsm_evt_conn_error_get_host(evt);
94         lwgsm_port_t port = lwgsm_evt_conn_error_get_port(evt);
95         printf("Error connecting to %s:%d\r\n", host, (int)port);
96         break;
97     }
98     default:
99         break;
100 }
return lwgsmOK;
}

```

## Sending data

Receiving data flow is always the same. Whenever new data packet arrives, corresponding event is called to notify application layer. When it comes to sending data, application may decide between 2 options (\*this is valid only for non-UDP connections):

- Write data to temporary transmit buffer
- Execute *send command* for every API function call

## Temporary transmit buffer

By calling `lwgsm_conn_write()` on active connection, temporary buffer is allocated and input data are copied to it. There is always up to 1 internal buffer active. When it is full (or if input data length is longer than maximal size), data are immediately send out and are not written to buffer.

*GSM AT Firmware* allows (current revision) to transmit up to 2048 bytes at a time with single command. When trying to send more than this, application would need to issue multiple *send commands* on *AT commands level*.

Write option is used mostly when application needs to write many different small chunks of data. Temporary buffer hence prevents many *send command* instructions as it is faster to send single command with big buffer, than many of them with smaller chunks of bytes.

## Transmit packet manually

In some cases it is not possible to use temporary buffers, mostly because of memory constraints. Application can directly start *send data* instructions on AT level by using `lwgsm_conn_send()` or `lwgsm_conn_sendto()` functions.

group **LWGSM\_CONN**

Connection API functions.

## Typedefs

**typedef struct** lwgsm\_conn \*lwgsm\_conn\_p

Pointer to `lwgsm_conn_t` structure.

## Enums

**enum** lwgsm\_conn\_type\_t

List of possible connection types.

*Values:*

**enumerator** LWGSM\_CONN\_TYPE\_TCP

Connection type is TCP

**enumerator** LWGSM\_CONN\_TYPE\_UDP

Connection type is UDP

**enumerator** LWGSM\_CONN\_TYPE\_SSL

Connection type is TCP over SSL

## Functions

*lwgsmt\_t* **lwgsm\_conn\_start** (*lwgsm\_conn\_p* \*conn, *lwgsm\_conn\_type\_t* type, **const** char \***const** host, *lwgsm\_port\_t* port, void \***const** arg, *lwgsm\_evt\_fn* conn\_evt\_fn, **const** uint32\_t blocking)

Start a new connection of specific type.

**Return** *lwgsmtOK* on success, member of *lwgsmt\_t* enumeration otherwise

### Parameters

- [out] conn: Pointer to connection handle to set new connection reference in case of successful connection
- [in] type: Connection type. This parameter can be a value of *lwgsm\_conn\_type\_t* enumeration
- [in] host: Connection host. In case of IP, write it as string, ex. "192.168.1.1"
- [in] port: Connection port
- [in] arg: Pointer to user argument passed to connection if successfully connected
- [in] conn\_evt\_fn: Callback function for this connection
- [in] blocking: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsmr\_conn\_close** (*lwgsmr\_conn\_p* conn, **const** uint32\_t blocking)

Close specific or all connections.

**Return** *lwgsmrOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] conn: Connection handle to close. Set to NULL if you want to close all connections.
- [in] blocking: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsmr\_conn\_send** (*lwgsmr\_conn\_p* conn, **const** void \*data, size\_t btw, size\_t \***const** bw, **const** uint32\_t blocking)

Send data on already active connection either as client or server.

**Return** *lwgsmrOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] conn: Connection handle to send data
- [in] data: Data to send
- [in] btw: Number of bytes to send
- [out] bw: Pointer to output variable to save number of sent data when successfully sent. Parameter value might not be accurate if you combine *lwgsmr\_conn\_write* and *lwgsmr\_conn\_send* functions
- [in] blocking: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsmr\_conn\_sendto** (*lwgsmr\_conn\_p* conn, **const** *lwgsmr\_ip\_t* \***const** ip, *lwgsmr\_port\_t* port, **const** void \*data, size\_t btw, size\_t \*bw, **const** uint32\_t blocking)

Send data on active connection of type UDP to specific remote IP and port.

**Note** In case IP and port values are not set, it will behave as normal send function (suitable for TCP too)

**Return** *lwgsmrOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] conn: Connection handle to send data
- [in] ip: Remote IP address for UDP connection
- [in] port: Remote port connection
- [in] data: Pointer to data to send
- [in] btw: Number of bytes to send
- [out] bw: Pointer to output variable to save number of sent data when successfully sent
- [in] blocking: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsmr\_conn\_set\_arg** (*lwgsmr\_conn\_p* conn, void \***const** arg)

Set argument variable for connection.

**Return** *lwgsmrOK* on success, member of *lwgsmr\_t* enumeration otherwise

See *lwgsmr\_conn\_get\_arg*

**Parameters**

- [in] conn: Connection handle to set argument
- [in] arg: Pointer to argument

void **\*lwgsm\_conn\_get\_arg** (*lwgsm\_conn\_p* conn)  
Get user defined connection argument.

**Return** User argument

**See** *lwgsm\_conn\_set\_arg*

**Parameters**

- [in] conn: Connection handle to get argument

uint8\_t **lwgsm\_conn\_is\_client** (*lwgsm\_conn\_p* conn)  
Check if connection type is client.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] conn: Pointer to connection to check for status

uint8\_t **lwgsm\_conn\_is\_active** (*lwgsm\_conn\_p* conn)  
Check if connection is active.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] conn: Pointer to connection to check for status

uint8\_t **lwgsm\_conn\_is\_closed** (*lwgsm\_conn\_p* conn)  
Check if connection is closed.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] conn: Pointer to connection to check for status

int8\_t **lwgsm\_conn\_getnum** (*lwgsm\_conn\_p* conn)  
Get the number from connection.

**Return** Connection number in case of success or -1 on failure

**Parameters**

- [in] conn: Connection pointer

*lwgsmr\_t* **lwgsm\_get\_conns\_status** (const uint32\_t blocking)  
Gets connections status.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] blocking: Status whether command should be blocking or not



*lwgsm\_conn\_p* **lwgsm\_conn\_get\_from\_evt** (*lwgsm\_evt\_t* \**evt*)

Get connection from connection based event.

**Return** Connection pointer on success, NULL otherwise

**Parameters**

- [in] *evt*: Event which happened for connection

*lwgsmr\_t* **lwgsm\_conn\_write** (*lwgsm\_conn\_p* *conn*, **const** void \**data*, size\_t *btw*, uint8\_t *flush*, size\_t \***const** *mem\_available*)

Write data to connection buffer and if it is full, send it non-blocking way.

**Note** This function may only be called from core (connection callbacks)

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] *conn*: Connection to write
- [in] *data*: Data to copy to write buffer
- [in] *btw*: Number of bytes to write
- [in] *flush*: Flush flag. Set to 1 if you want to send data immediately after copying
- [out] *mem\_available*: Available memory size available in current write buffer. When the buffer length is reached, current one is sent and a new one is automatically created. If function returns *lwgsmOK* and \**mem\_available* = 0, there was a problem allocating a new buffer for next operation

*lwgsmr\_t* **lwgsm\_conn\_recved** (*lwgsm\_conn\_p* *conn*, *lwgsm\_pbuf\_p* *pbuf*)

Notify connection about received data which means connection is ready to accept more data.

Once data reception is confirmed, stack will try to send more data to user.

**Note** Since this feature is not supported yet by AT commands, function is only prototype and should be used in connection callback when data are received

**Note** Function is not thread safe and may only be called from connection event function

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] *conn*: Connection handle
- [in] *pbuf*: Packet buffer received on connection

size\_t **lwgsm\_conn\_get\_total\_recved\_count** (*lwgsm\_conn\_p* *conn*)

Get total number of bytes ever received on connection and sent to user.

**Return** Count of received bytes on connection

**Parameters**

- [in] *conn*: Connection handle

uint8\_t **lwgsm\_conn\_get\_remote\_ip** (*lwgsm\_conn\_p* *conn*, *lwgsm\_ip\_t* \**ip*)

Get connection remote IP address.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] conn: Connection handle
- [out] ip: Pointer to IP output handle

*lwgsm\_port\_t* **lwgsm\_conn\_get\_remote\_port** (*lwgsm\_conn\_p* conn)

Get connection remote port number.

**Return** Port number on success, 0 otherwise

**Parameters**

- [in] conn: Connection handle

*lwgsm\_port\_t* **lwgsm\_conn\_get\_local\_port** (*lwgsm\_conn\_p* conn)

Get connection local port number.

**Return** Port number on success, 0 otherwise

**Parameters**

- [in] conn: Connection handle

## Debug support

Middleware has extended debugging capabilities. These consist of different debugging levels and types of debug messages, allowing to track and catch different types of warnings, severe problems or simply output messages program flow messages (trace messages).

Module is highly configurable using library configuration methods. Application must enable some options to decide what type of messages and for which modules it would like to output messages.

With default configuration, `printf` is used as output function. This behavior can be changed with `LWGSM_CFG_DBG_OUT` configuration.

For successful debugging, application must:

- Enable global debugging by setting `LWGSM_CFG_DBG` to `LWGSM_DBG_ON`
- Configure which types of messages to output
- Configure debugging level, from all messages to severe only
- Enable specific modules to debug, by setting its configuration value to `LWGSM_DBG_ON`

---

**Tip:** Check *Configuration* for all modules with debug implementation.

---

An example code with config and latter usage:

Listing 15: Debug configuration setup

```
1  /* Modifications of lwgsm_opts.h file for configuration */
2
3  /* Enable global debug */
4  #define LWGSM_CFG_DBG                LWGSM_DBG_ON
5
6  /*
7  * Enable debug types.
```

(continues on next page)

(continued from previous page)

```

8  * Application may use bitwise OR | to use multiple types:
9  *   LWGSM_DBG_TYPE_TRACE | LWGSM_DBG_TYPE_STATE
10 *
11 #define LWGSM_CFG_DBG_TYPES_ON      LWGSM_DBG_TYPE_TRACE
12
13 /* Enable debug on custom module */
14 #define MY_DBG_MODULE                LWGSM_DBG_ON

```

Listing 16: Debug usage within middleware

```

1  #include "lwgsm/lwgsm_debug.h"
2
3  /*
4   * Print debug message to the screen
5   * Trace message will be printed as it is enabled in types
6   * while state message will not be printed.
7   */
8  LWGSM_DEBUGF(MY_DBG_MODULE | LWGSM_DBG_TYPE_TRACE, "This is trace message on my_
↳program\r\n");
9  LWGSM_DEBUGF(MY_DBG_MODULE | LWGSM_DBG_TYPE_STATE, "This is state message on my_
↳program\r\n");

```

**group LWGSM\_DEBUG**

Debugging support module to track stack.

**Debug levels**

List of debug levels

**LWGSM\_DBG\_LVL\_ALL**

Print all messages of all types

**LWGSM\_DBG\_LVL\_WARNING**

Print warning and upper messages

**LWGSM\_DBG\_LVL\_DANGER**

Print danger errors

**LWGSM\_DBG\_LVL\_SEVERE**

Print severe problems affecting program flow

**LWGSM\_DBG\_LVL\_MASK**

Mask for getting debug level

**Debug types**

List of possible debugging types

**LWGSM\_DBG\_TYPE\_TRACE**

Debug trace messages for program flow

**LWGSM\_DBG\_TYPE\_STATE**

Debug state messages (such as state machines)

**LWGSM\_DBG\_TYPE\_ALL**

All debug types

## Defines

### **LWGSM\_DBG\_ON**

Indicates debug is enabled

### **LWGSM\_DBG\_OFF**

Indicates debug is disabled

### **LWGSM\_DEBUGF** (*c, fmt, ...*)

Print message to the debug “window” if enabled.

#### Parameters

- [in] *c*: Condition if debug of specific type is enabled
- [in] *fmt*: Formatted string for debug
- [in] *...*: Variable parameters for formatted string

### **LWGSM\_DEBUGW** (*c, cond, fmt, ...*)

Print message to the debug “window” if enabled when specific condition is met.

#### Parameters

- [in] *c*: Condition if debug of specific type is enabled
- [in] *cond*: Debug only if this condition is true
- [in] *fmt*: Formatted string for debug
- [in] *...*: Variable parameters for formatted string

## Device info

### *group* **LWGSM\_DEVICE\_INFO**

Basic device information.

## Functions

*lwgsmr\_t* **lwgsm\_device\_get\_manufacturer** (*char \*manuf*, *size\_t len*, **const** *lwgsmr\_api\_cmd\_evt\_fn* *evt\_fn*, **void \*const** *evt\_arg*, **const** *uint32\_t blocking*)

Get device manufacturer.

**Return** *lwgsmrOK* on success, member of *lwgsmr\_t* enumeration otherwise

#### Parameters

- [in] *manuf*: Pointer to output string array to save manufacturer info
- [in] *len*: Length of string array including NULL termination
- [in] *evt\_fn*: Callback function called when command has finished. Set to NULL when not used
- [in] *evt\_arg*: Custom argument for event callback function
- [in] *blocking*: Status whether command should be blocking or not

---

```
lwgsmr_t lwgsm_device_get_model(char *model, size_t len, const lwgsm_api_cmd_evt_fn
                                evt_fn, void *const evt_arg, const uint32_t blocking)
```

Get device model name.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] *model*: Pointer to output string array to save model info
- [in] *len*: Length of string array including NULL termination
- [in] *evt\_fn*: Callback function called when command has finished. Set to NULL when not used
- [in] *evt\_arg*: Custom argument for event callback function
- [in] *blocking*: Status whether command should be blocking or not

```
lwgsmr_t lwgsm_device_get_revision(char *rev, size_t len, const lwgsm_api_cmd_evt_fn
                                    evt_fn, void *const evt_arg, const uint32_t blocking)
```

Get device revision.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] *rev*: Pointer to output string array to save revision info
- [in] *len*: Length of string array including NULL termination
- [in] *evt\_fn*: Callback function called when command has finished. Set to NULL when not used
- [in] *evt\_arg*: Custom argument for event callback function
- [in] *blocking*: Status whether command should be blocking or not

```
lwgsmr_t lwgsm_device_get_serial_number(char *serial, size_t len, const
                                         lwgsm_api_cmd_evt_fn evt_fn, void *const
                                         evt_arg, const uint32_t blocking)
```

Get device serial number.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] *serial*: Pointer to output string array to save serial number info
- [in] *len*: Length of string array including NULL termination
- [in] *evt\_fn*: Callback function called when command has finished. Set to NULL when not used
- [in] *evt\_arg*: Custom argument for event callback function
- [in] *blocking*: Status whether command should be blocking or not

### Event management

#### group **LWGSM\_EVT**

Event helper functions.

#### Reset event

Event helper functions for LWGSM\_EVT\_RESET event

*lwgsmr\_t* **lwgsmr\_evt\_reset\_get\_result** (*lwgsmr\_evt\_t* \*cc)

Get reset sequence operation status.

**Return** Member of *lwgsmr\_t* enumeration

##### Parameters

- [in] cc: Event data

#### Restore event

Event helper functions for LWGSM\_EVT\_RESTORE event

*lwgsmr\_t* **lwgsmr\_evt\_restore\_get\_result** (*lwgsmr\_evt\_t* \*cc)

Get restore sequence operation status.

**Return** Member of *lwgsmr\_t* enumeration

##### Parameters

- [in] cc: Event data

#### Current network operator

Event helper functions for LWGSM\_EVT\_NETWORK\_OPERATOR\_CURRENT event

**const** *lwgsmr\_operator\_curr\_t* \***lwgsmr\_evt\_network\_operator\_get\_current** (*lwgsmr\_evt\_t* \*cc)

Get current operator data from event.

**Return** Current operator handle

##### Parameters

- [in] cc: Event data

#### Connection data received

Event helper functions for LWGSM\_EVT\_CONN\_RECV event

*lwgsmr\_pbuf\_p* **lwgsmr\_evt\_conn\_recv\_get\_buff** (*lwgsmr\_evt\_t* \*cc)

Get buffer from received data.

**Return** Buffer handle

##### Parameters

- [in] cc: Event handle

*lwgsm\_conn\_p* **lwgsm\_evt\_conn\_recv\_get\_conn** (*lwgsm\_evt\_t* \*cc)

Get connection handle for receive.

**Return** Connection handle

**Parameters**

- [in] cc: Event handle

### Connection data send

Event helper functions for LWGSM\_EVT\_CONN\_SEND event

*lwgsm\_conn\_p* **lwgsm\_evt\_conn\_send\_get\_conn** (*lwgsm\_evt\_t* \*cc)

Get connection handle for data sent event.

**Return** Connection handle

**Parameters**

- [in] cc: Event handle

*size\_t* **lwgsm\_evt\_conn\_send\_get\_length** (*lwgsm\_evt\_t* \*cc)

Get number of bytes sent on connection.

**Return** Number of bytes sent

**Parameters**

- [in] cc: Event handle

*lwgsmr\_t* **lwgsm\_evt\_conn\_send\_get\_result** (*lwgsm\_evt\_t* \*cc)

Check if connection send was successful.

**Return** Member of *lwgsmr\_t* enumeration

**Parameters**

- [in] cc: Event handle

### Connection active

Event helper functions for LWGSM\_EVT\_CONN\_ACTIVE event

*lwgsm\_conn\_p* **lwgsm\_evt\_conn\_active\_get\_conn** (*lwgsm\_evt\_t* \*cc)

Get connection handle.

**Return** Connection handle

**Parameters**

- [in] cc: Event handle

*uint8\_t* **lwgsm\_evt\_conn\_active\_is\_client** (*lwgsm\_evt\_t* \*cc)

Check if new connection is client.

**Return** 1 if client, 0 otherwise

**Parameters**

- [in] cc: Event handle

### Connection close event

Event helper functions for LWGSM\_EVT\_CONN\_CLOSE event

*lwgsm\_conn\_p* **lwgsm\_evt\_conn\_close\_get\_conn** (*lwgsm\_evt\_t* \*cc)  
Get connection handle.

**Return** Connection handle

**Parameters**

- [in] cc: Event handle

uint8\_t **lwgsm\_evt\_conn\_close\_is\_client** (*lwgsm\_evt\_t* \*cc)  
Check if close connection was client.

**Return** 1 if client, 0 otherwise

**Parameters**

- [in] cc: Event handle

uint8\_t **lwgsm\_evt\_conn\_close\_is\_forced** (*lwgsm\_evt\_t* \*cc)  
Check if connection close even was forced by user.

**Return** 1 if forced, 0 otherwise

**Parameters**

- [in] cc: Event handle

*lwgsmr\_t* **lwgsm\_evt\_conn\_close\_get\_result** (*lwgsm\_evt\_t* \*cc)  
Get connection close event result.

**Return** Member of *lwgsmr\_t* enumeration

**Parameters**

- [in] cc: Event handle

### Connection poll

Event helper functions for LWGSM\_EVT\_CONN\_POLL event

*lwgsm\_conn\_p* **lwgsm\_evt\_conn\_poll\_get\_conn** (*lwgsm\_evt\_t* \*cc)  
Get connection handle.

**Return** Connection handle

**Parameters**

- [in] cc: Event handle



## Connection error

Event helper functions for LWGSM\_EVT\_CONN\_ERROR event

*lwgsmr\_t* **lwgsm\_evt\_conn\_error\_get\_error** (*lwgsm\_evt\_t* \*cc)

Get connection error type.

**Return** Member of *lwgsmr\_t* enumeration

**Parameters**

- [in] cc: Event handle

*lwgsm\_conn\_type\_t* **lwgsm\_evt\_conn\_error\_get\_type** (*lwgsm\_evt\_t* \*cc)

Get connection type.

**Return** Member of *lwgsmr\_t* enumeration

**Parameters**

- [in] cc: Event handle

**const** char \***lwgsm\_evt\_conn\_error\_get\_host** (*lwgsm\_evt\_t* \*cc)

Get connection host.

**Return** Host name for connection

**Parameters**

- [in] cc: Event handle

*lwgsm\_port\_t* **lwgsm\_evt\_conn\_error\_get\_port** (*lwgsm\_evt\_t* \*cc)

Get connection port.

**Return** Host port number

**Parameters**

- [in] cc: Event handle

**void** \***lwgsm\_evt\_conn\_error\_get\_arg** (*lwgsm\_evt\_t* \*cc)

Get user argument.

**Return** User argument

**Parameters**

- [in] cc: Event handle

## Signal strength

Event helper functions for LWGSM\_EVT\_CONN\_RECV event

`int16_t lwgsm_evt_signal_strength_get_rssi (lwgsm_evt_t *cc)`  
Get RSSI from CSQ command.

**Return** RSSI value in units of dBm

**Parameters**

- [in] cc: Event data

## SMS received

Event helper functions for LWGSM\_EVT\_SMS\_RECV event

`size_t lwgsm_evt_sms_recv_get_pos (lwgsm_evt_t *cc)`  
Get SMS position in memory which has been saved on receive.

**Return** SMS position in memory

**Parameters**

- [in] cc: Event handle

`lwgsm_mem_t lwgsm_evt_sms_recv_get_mem (lwgsm_evt_t *cc)`  
Get SMS memory used to save SMS on receive.

**Return** SMS memory location

**Parameters**

- [in] cc: Event handle

## SMS content read

Event helper functions for LWGSM\_EVT\_SMS\_READ event

`lwgsm_sms_entry_t *lwgsm_evt_sms_read_get_entry (lwgsm_evt_t *cc)`  
Get SMS entry after successful read.

**Return** SMS entry

**Parameters**

- [in] cc: Event handle

`lwgsmr_t lwgsm_evt_sms_read_get_result (lwgsm_evt_t *cc)`  
Get SMS read operation result.

**Return** SMS entry

**Parameters**

- [in] cc: Event handle

## SMS send

Event helper functions for LWGSM\_EVT\_SMS\_SEND event

*lwgsmr\_t* **lwgsm\_evt\_sms\_send\_get\_result** (*lwgsm\_evt\_t* \*cc)

Get SMS send result status.

**Return** Member of *lwgsmr\_t* enumeration

**Parameters**

- [in] cc: Event handle

size\_t **lwgsm\_evt\_sms\_send\_get\_pos** (*lwgsm\_evt\_t* \*cc)

Get SMS send position in memory.

**Note** Use only if SMS sent successfully

**Return** Position in memory

**Parameters**

- [in] cc: Event handle

## SMS delete

Event helper functions for LWGSM\_EVT\_SMS\_DELETE event

*lwgsmr\_t* **lwgsm\_evt\_sms\_delete\_get\_result** (*lwgsm\_evt\_t* \*cc)

Get SMS delete result status.

**Return** Member of *lwgsmr\_t* enumeration

**Parameters**

- [in] cc: Event handle

size\_t **lwgsm\_evt\_sms\_delete\_get\_pos** (*lwgsm\_evt\_t* \*cc)

Get SMS delete memory position.

**Return** Deleted position in memory

**Parameters**

- [in] cc: Event handle

*lwgsm\_mem\_t* **lwgsm\_evt\_sms\_delete\_get\_mem** (*lwgsm\_evt\_t* \*cc)

Get SMS delete memory.

**Return** SMS memory for delete operation

**Parameters**

- [in] cc: Event handle

## Call status changed

Event helper functions for LWGSM\_EVT\_CALL\_CHANGED event

**const** lwgsm\_call\_t \*lwgsm\_evt\_call\_changed\_get\_call (lwgsm\_evt\_t \*cc)  
Get call information from changed event.

**Return** Position in memory

**Parameters**

- [in] cc: Event handle

## Operator scan

Event helper functions for LWGSM\_EVT\_OPERATOR\_SCAN event

*lwgsmr\_t* lwgsm\_evt\_operator\_scan\_get\_result (lwgsm\_evt\_t \*cc)  
Get operator scan operation status.

**Return** Member of *lwgsmr\_t* enumeration

**Parameters**

- [in] cc: Event data

*lwgsm\_operator\_t* \*lwgsm\_evt\_operator\_scan\_get\_entries (lwgsm\_evt\_t \*cc)  
Get operator entries from scan.

**Return** Pointer to array of operator entries

**Parameters**

- [in] cc: Event data

size\_t lwgsm\_evt\_operator\_scan\_get\_length (lwgsm\_evt\_t \*cc)  
Get length of operators scanned.

**Return** Number of operators scanned

**Parameters**

- [in] cc: Event data

## Typedefs

**typedef** *lwgsmr\_t* (\*lwgsm\_evt\_fn) (struct lwgsm\_evt \*evt)  
Event function prototype.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* otherwise

**Parameters**

- [in] evt: Callback event data

## Enums

### **enum lwgsm\_evt\_type\_t**

List of possible callback types received to user.

*Values:*

#### **enumerator LWGSM\_EVT\_INIT\_FINISH**

Initialization has been finished at this point

#### **enumerator LWGSM\_EVT\_RESET**

Device reset operation finished

#### **enumerator LWGSM\_EVT\_RESTORE**

Device restore operation finished

#### **enumerator LWGSM\_EVT\_CMD\_TIMEOUT**

Timeout on command. When application receives this event, it may reset system as there was (maybe) a problem in device

#### **enumerator LWGSM\_EVT\_DEVICE\_PRESENT**

Notification when device present status changes

#### **enumerator LWGSM\_EVT\_DEVICE\_IDENTIFIED**

Device identified event

#### **enumerator LWGSM\_EVT\_SIGNAL\_STRENGTH**

Signal strength event

#### **enumerator LWGSM\_EVT\_SIM\_STATE\_CHANGED**

SIM card state changed

#### **enumerator LWGSM\_EVT\_OPERATOR\_SCAN**

Operator scan finished event

#### **enumerator LWGSM\_EVT\_NETWORK\_OPERATOR\_CURRENT**

Current operator event

#### **enumerator LWGSM\_EVT\_NETWORK\_REG\_CHANGED**

Network registration changed. Available even when [LWGSM\\_CFG\\_NETWORK](#) is disabled

#### **enumerator LWGSM\_EVT\_NETWORK\_ATTACHED**

Attached to network, PDP context active and ready for TCP/IP application

#### **enumerator LWGSM\_EVT\_NETWORK\_DETACHED**

Detached from network, PDP context not active anymore

#### **enumerator LWGSM\_EVT\_CONN\_RECV**

Connection data received

#### **enumerator LWGSM\_EVT\_CONN\_SEND**

Connection data send

#### **enumerator LWGSM\_EVT\_CONN\_ACTIVE**

Connection just became active

#### **enumerator LWGSM\_EVT\_CONN\_ERROR**

Client connection start was not successful

#### **enumerator LWGSM\_EVT\_CONN\_CLOSE**

Connection close event. Check status if successful

#### **enumerator LWGSM\_EVT\_CONN\_POLL**

Poll for connection if there are any changes

**enumerator LWGSM\_EVT\_SMS\_ENABLE**  
SMS enable event

**enumerator LWGSM\_EVT\_SMS\_READY**  
SMS ready event

**enumerator LWGSM\_EVT\_SMS\_SEND**  
SMS send event

**enumerator LWGSM\_EVT\_SMS\_RECV**  
SMS received

**enumerator LWGSM\_EVT\_SMS\_READ**  
SMS read

**enumerator LWGSM\_EVT\_SMS\_DELETE**  
SMS delete

**enumerator LWGSM\_EVT\_SMS\_LIST**  
SMS list

**enumerator LWGSM\_EVT\_CALL\_ENABLE**  
Call enable event

**enumerator LWGSM\_EVT\_CALL\_READY**  
Call ready event

**enumerator LWGSM\_EVT\_CALL\_CHANGED**  
Call info changed, +CLCK statement received

**enumerator LWGSM\_EVT\_CALL\_RING**  
Call is ringing event

**enumerator LWGSM\_EVT\_CALL\_BUSY**  
Call is busy

**enumerator LWGSM\_EVT\_CALL\_NO\_CARRIER**  
No carrier to make a call

**enumerator LWGSM\_EVT\_PB\_ENABLE**  
Phonebook enable event

**enumerator LWGSM\_EVT\_PB\_LIST**  
Phonebook list event

**enumerator LWGSM\_EVT\_PB\_SEARCH**  
Phonebook search event

## Functions

*lwgsmr\_t* **lwgsmr\_evt\_register** (*lwgsmr\_evt\_fn* fn)

Register callback function for global (non-connection based) events.

**Return** *lwgsmrOK* on success, member of *lwgsmr\_t* enumeration otherwise

### Parameters

- [in] fn: Callback function to call on specific event

*lwgsmr\_t* **lwgsmr\_evt\_unregister** (*lwgsmr\_evt\_fn* fn)

Unregister callback function for global (non-connection based) events.

**Note** Function must be first registered using *lwgsm\_evt\_register*

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] *fn*: Callback function to remove from event list

*lwgsm\_evt\_type\_t* **lwgsm\_evt\_get\_type** (*lwgsm\_evt\_t* \**cc*)

Get event type.

**Return** Event type. Member of *lwgsm\_evt\_type\_t* enumeration

**Parameters**

- [in] *cc*: Event handle

**struct** *lwgsm\_evt\_t*

*#include <lwgsm\_typedefs.h>* Global callback structure to pass as parameter to callback function.

**Public Members**

*lwgsm\_evt\_type\_t* **type**

Callback type

*lwgsmr\_t* **res**

Reset operation result

Restore operation result

Scan operation result

Send data result

Result of close event. Set to *lwgsmOK* on success.

SMS send result information

SMS read result information

Operation success

Result on command

Enable status

**struct** *lwgsm\_evt\_t*::[anonymous]::[anonymous] **reset**

Reset sequence finish. Use with LWGSM\_EVT\_RESET event

**struct** *lwgsm\_evt\_t*::[anonymous]::[anonymous] **restore**

Restore sequence finish. Use with LWGSM\_EVT\_RESTORE event

*lwgsm\_sim\_state\_t* **state**

SIM state

**struct** *lwgsm\_evt\_t*::[anonymous]::[anonymous] **cpin**

CPIN event

**const** *lwgsm\_operator\_curr\_t* \***operator\_current**

Current operator info

**struct** *lwgsm\_evt\_t*::[anonymous]::[anonymous] **operator\_current**

Current operator event. Use with LWGSM\_EVT\_NETWORK\_OPERATOR\_CURRENT event

*lwgsm\_operator\_t* \***ops**

Pointer to operators

size\_t **opf**

Number of operators found

**struct** *lwgsm\_evt\_t*::[anonymous]::[anonymous] **operator\_scan**

Operator scan event. Use with LWGSM\_EVT\_OPERATOR\_SCAN event

int16\_t **rssi**

Strength in units of dBm

**struct** *lwgsm\_evt\_t*::[anonymous]::[anonymous] **rssi**

Signal strength event. Use with LWGSM\_EVT\_SIGNAL\_STRENGTH event

*lwgsm\_conn\_p* **conn**

Connection where data were received

Connection where data were sent

Pointer to connection

Set connection pointer

*lwgsm\_pbuf\_p* **buff**

Pointer to received data

**struct** *lwgsm\_evt\_t*::[anonymous]::[anonymous] **conn\_data\_recv**

Network data received. Use with LWGSM\_EVT\_CONN\_RECV event

size\_t **sent**

Number of bytes sent on connection

**struct** *lwgsm\_evt\_t*::[anonymous]::[anonymous] **conn\_data\_send**

Data successfully sent. Use with LWGSM\_EVT\_CONN\_SEND event

**const** char \***host**

Host to use for connection

*lwgsm\_port\_t* **port**

Remote port used for connection

*lwgsm\_conn\_type\_t* **type**

Connection type

void \***arg**

Connection argument used on connection

*lwgsnr\_t* **err**

Error value

**struct** *lwgsm\_evt\_t*::[anonymous]::[anonymous] **conn\_error**

Client connection start error. Use with LWGSM\_EVT\_CONN\_ERROR event

uint8\_t **client**

Set to 1 if connection is/was client mode

uint8\_t **forced**

Set to 1 if connection action was forced

**struct** *lwgsm\_evt\_t*::[anonymous]::[anonymous] **conn\_active\_close**

Process active and closed statuses at the same time. Use with LWGSM\_EVT\_CONN\_ACTIVE or LWGSM\_EVT\_CONN\_CLOSE events



```

struct lwgsm_evt_t::[anonymous]::[anonymous] conn_poll
    Polling active connection to check for timeouts. Use with LWGSM_EVT_CONN_POLL event

lwgsmr_t status
    Enable status

struct lwgsm_evt_t::[anonymous]::[anonymous] sms_enable
    SMS enable event. Use with LWGSM_EVT_SMS_ENABLE event

size_t pos
    Position in memory

    Received position in memory for sent SMS

    Deleted position in memory for sent SMS

struct lwgsm_evt_t::[anonymous]::[anonymous] sms_send
    SMS sent info. Use with LWGSM_EVT_SMS_SEND event

lwgsm_mem_t mem
    Memory of received message

    Memory of deleted message

    Memory used for scan

struct lwgsm_evt_t::[anonymous]::[anonymous] sms_recv
    SMS received info. Use with LWGSM_EVT_SMS_RECV event

lwgsm_sms_entry_t *entry
    SMS entry

struct lwgsm_evt_t::[anonymous]::[anonymous] sms_read
    SMS read. Use with LWGSM_EVT_SMS_READ event

struct lwgsm_evt_t::[anonymous]::[anonymous] sms_delete
    SMS delete. Use with LWGSM_EVT_SMS_DELETE event

lwgsm_sms_entry_t *entries
    Pointer to entries

size_t size
    Number of valid entries

struct lwgsm_evt_t::[anonymous]::[anonymous] sms_list
    SMS list. Use with LWGSM_EVT_SMS_LIST event

struct lwgsm_evt_t::[anonymous]::[anonymous] call_enable
    Call enable event. Use with LWGSM_EVT_CALL_ENABLE event

const lwgsm_call_t *call
    Call information

struct lwgsm_evt_t::[anonymous]::[anonymous] call_changed
    Call changed info. Use with LWGSM_EVT_CALL_CHANGED event

struct lwgsm_evt_t::[anonymous]::[anonymous] pb_enable
    Phonebook enable event. Use with LWGSM_EVT_PB_ENABLE event

lwgsm_pb_entry_t *entries
    Pointer to entries

struct lwgsm_evt_t::[anonymous]::[anonymous] pb_list
    Phonebok list. Use with LWGSM_EVT_PB_LIST event

```

```
const char *search
    Search string

struct lwgsm_evt_t::[anonymous]::[anonymous] pb_search
    Phonebok search list. Use with LWGSM_EVT_PB_SEARCH event

union lwgsm_evt_t::[anonymous] evt
    Callback event union
```

### File Transfer Protocol

*group* **LWGSM\_FTP**  
File Transfer Protocol (FTP) manager.  
Currently it is under development

### HTTP

*group* **LWGSM\_HTTP**  
Hyper Text Transfer Protocol (HTTP) manager.  
Currently it is under development

### Input module

Input module is used to input received data from *GSM* device to *LwGSM-Lib* middleware part. 2 processing options are possible:

- Indirect processing with *lwgsm\_input()* (default mode)
- Direct processing with *lwgsm\_input\_process()*

---

**Tip:** Direct or indirect processing mode is select by setting *LWGSM\_CFG\_INPUT\_USE\_PROCESS* configuration value.

---

### Indirect processing

With indirect processing mode, every received character from *GSM* physical device is written to intermediate buffer between low-level driver and *processing* thread.

Function *lwgsm\_input()* is used to write data to buffer, which is later processed by *processing* thread.

Indirect processing mode allows embedded systems to write received data to buffer from interrupt context (outside threads). As a drawback, its performance is decreased as it involves copying every receive character to intermediate buffer, and may also introduce RAM memory footprint increase.

## Direct processing

Direct processing is targeting more advanced host controllers, like STM32 or WIN32 implementation use. It is developed with DMA support in mind, allowing low-level drivers to skip intermediate data buffer and process input bytes directly.

---

**Note:** When using this mode, function `lwgsm_input_process()` must be used and it may only be called from thread context. Processing of input bytes is done in low-level input thread, started by application.

---



---

**Tip:** Check *Porting guide* for implementation examples.

---

group **LWGSM\_INPUT**

Input function for received data.

## Functions

`lwgsmr_t lwgsm_input (const void *data, size_t len)`

Write data to input buffer.

**Note** `LWGSM_CFG_INPUT_USE_PROCESS` must be disabled to use this function

**Return** `lwgsmOK` on success, member of `lwgsmr_t` enumeration otherwise

### Parameters

- [in] data: Pointer to data to write
- [in] len: Number of data elements in units of bytes

`lwgsmr_t lwgsm_input_process (const void *data, size_t len)`

Process input data directly without writing it to input buffer.

**Note** This function may only be used when in OS mode, where single thread is dedicated for input read of AT receive

**Note** `LWGSM_CFG_INPUT_USE_PROCESS` must be enabled to use this function

**Return** `lwgsmOK` on success, member of `lwgsmr_t` enumeration otherwise

### Parameters

- [in] data: Pointer to received data to be processed
- [in] len: Length of data to process in units of bytes

## Memory manager

group **LWGSM\_MEM**

Dynamic memory manager.

### Functions

uint8\_t **lwgsm\_mem\_assignmemory** (const *lwgsm\_mem\_region\_t* \*regions, size\_t size)

Assign memory region(s) for allocation functions.

**Note** You can allocate multiple regions by assigning start address and region size in units of bytes

**Return** 1 on success, 0 otherwise

**Note** Function is not available when *LWGSM\_CFG\_MEM\_CUSTOM* is 1

#### Parameters

- [in] regions: Pointer to list of regions to use for allocations
- [in] len: Number of regions to use

void \***lwgsm\_mem\_malloc** (size\_t size)

Allocate memory of specific size.

**Return** Memory address on success, NULL otherwise

**Note** Function is not available when *LWGSM\_CFG\_MEM\_CUSTOM* is 1 and must be implemented by user

#### Parameters

- [in] size: Number of bytes to allocate

void \***lwgsm\_mem\_realloc** (void \*ptr, size\_t size)

Reallocate memory to specific size.

**Note** After new memory is allocated, content of old one is copied to new memory

**Return** Memory address on success, NULL otherwise

**Note** Function is not available when *LWGSM\_CFG\_MEM\_CUSTOM* is 1 and must be implemented by user

#### Parameters

- [in] ptr: Pointer to current allocated memory to resize, returned using *lwgsm\_mem\_malloc*, *lwgsm\_mem\_calloc* or *lwgsm\_mem\_realloc* functions
- [in] size: Number of bytes to allocate on new memory

void \***lwgsm\_mem\_calloc** (size\_t num, size\_t size)

Allocate memory of specific size and set memory to zero.

**Return** Memory address on success, NULL otherwise

**Note** Function is not available when *LWGSM\_CFG\_MEM\_CUSTOM* is 1 and must be implemented by user

#### Parameters

- [in] `num`: Number of elements to allocate
- [in] `size`: Size of each element

void **lwgsm\_mem\_free** (void \**ptr*)  
Free memory.

**Note** Function is not available when `LWGSM_CFG_MEM_CUSTOM` is 1 and must be implemented by user

#### Parameters

- [in] `ptr`: Pointer to memory previously returned using `lwgsm_mem_malloc`, `lwgsm_mem_calloc` or `lwgsm_mem_realloc` functions

uint8\_t **lwgsm\_mem\_free\_s** (void \*\**ptr*)  
Free memory in safe way by invalidating pointer after freeing.

**Return** 1 on success, 0 otherwise

#### Parameters

- [in] `ptr`: Pointer to pointer to allocated memory to free

struct **lwgsm\_mem\_region\_t**  
*#include <lwgsm\_mem.h>* Single memory region descriptor.

#### Public Members

void \***start\_addr**  
Start address of region

size\_t **size**  
Size in units of bytes of region

## Network

group **LWGSM\_NETWORK**  
Network manager.

#### Enums

enum **lwgsm\_network\_reg\_status\_t**  
Network Registration status.

*Values:*

enumerator **LWGSM\_NETWORK\_REG\_STATUS\_SIM\_ERR**  
SIM card error

enumerator **LWGSM\_NETWORK\_REG\_STATUS\_CONNECTED**  
Device is connected to network

enumerator **LWGSM\_NETWORK\_REG\_STATUS\_SEARCHING**  
Network search is in progress

**enumerator LWGSM\_NETWORK\_REG\_STATUS\_DENIED**

Registration denied

**enumerator LWGSM\_NETWORK\_REG\_STATUS\_CONNECTED\_ROAMING**

Device is connected and is roaming

**enumerator LWGSM\_NETWORK\_REG\_STATUS\_CONNECTED\_SMS\_ONLY**

Device is connected to home network in SMS-only mode

**enumerator LWGSM\_NETWORK\_REG\_STATUS\_CONNECTED\_ROAMING\_SMS\_ONLY**

Device is roaming in SMS-only mode

## Functions

*lwgsmr\_t* **lwgsm\_network\_rssi** (int16\_t \*rssi, const *lwgsm\_api\_cmd\_evt\_fn* evt\_fn, void \*const evt\_arg, const uint32\_t blocking)

Read RSSI signal from network operator.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

### Parameters

- [out] rssi: RSSI output variable. When set to 0, RSSI is not valid
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not used
- [in] evt\_arg: Custom argument for event callback function
- [in] blocking: Status whether command should be blocking or not

*lwgsm\_network\_reg\_status\_t* **lwgsm\_network\_get\_reg\_status** (void)

Get network registration status.

**Return** Member of *lwgsm\_network\_reg\_status\_t* enumeration

*lwgsmr\_t* **lwgsm\_network\_attach** (const char \*apn, const char \*user, const char \*pass, const *lwgsm\_api\_cmd\_evt\_fn* evt\_fn, void \*const evt\_arg, const uint32\_t blocking)

Attach to network and active PDP context.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

### Parameters

- [in] apn: APN name
- [in] user: User name to attach. Set to NULL if not used
- [in] pass: User password to attach. Set to NULL if not used
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not used
- [in] evt\_arg: Custom argument for event callback function
- [in] blocking: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsm\_network\_detach** (const *lwgsm\_api\_cmd\_evt\_fn* evt\_fn, void \*const evt\_arg, const uint32\_t blocking)

Detach from network.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] *evt\_fn*: Callback function called when command has finished. Set to NULL when not used
- [in] *evt\_arg*: Custom argument for event callback function
- [in] *blocking*: Status whether command should be blocking or not

`uint8_t lwgsm_network_is_attached (void)`

Check if device is attached to network and PDP context is active.

**Return** 1 on success, 0 otherwise

*lwgsmr\_t* **lwgsm\_network\_copy\_ip** (*lwgsm\_ip\_t* \**ip*)

Copy IP address from internal value to user variable.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [out] *ip*: Pointer to output IP variable

*lwgsmr\_t* **lwgsm\_network\_check\_status** (**const** *lwgsm\_api\_cmd\_evt\_fn* *evt\_fn*, void \***const** *evt\_arg*, **const** *uint32\_t* *blocking*)

Check network PDP status.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] *evt\_fn*: Callback function called when command has finished. Set to NULL when not used
- [in] *evt\_arg*: Custom argument for event callback function
- [in] *blocking*: Status whether command should be blocking or not

## Network API

Network API provides functions for multi-thread application network management. It allows multiple threads to request to join to network (internet access).

Network API module controls when network connection shall be active or can be closed.

*group* **LWGSMT\_NETWORK\_API**

Network API functions for multi-thread operations.

## Functions

*lwgsmr\_t* **lwgsml\_network\_set\_credentials** (**const** char \**apn*, **const** char \**user*, **const** char \**pass*)

Set system network credentials before asking for attach.

**Return** *lwgsmlOK* on success, member of *lwgsmr\_t* otherwise

### Parameters

- [in] *apn*: APN domain. Set to NULL if not used
- [in] *user*: APN username. Set to NULL if not used
- [in] *pass*: APN password. Set to NULL if not used

*lwgsmr\_t* **lwgsml\_network\_request\_attach** (void)

Request manager to attach to network.

**Note** This function is blocking and cannot be called from event functions

**Return** *lwgsmlOK* on success (when attached), member of *lwgsmr\_t* otherwise

*lwgsmr\_t* **lwgsml\_network\_request\_detach** (void)

Request manager to detach from network.

If other threads use network, manager will not disconnect from network otherwise it will disable network access

**Note** This function is blocking and cannot be called from event functions

**Return** *lwgsmlOK* on success (when attached), member of *lwgsmr\_t* otherwise

## Network operator

*group* **LWGSML\_OPERATOR**  
network operator API

## Enums

**enum** **lwgsml\_operator\_status\_t**

Operator status value.

*Values:*

**enumerator** **LWGSML\_OPERATOR\_STATUS\_UNKNOWN**

Unknown operator

**enumerator** **LWGSML\_OPERATOR\_STATUS\_AVAILABLE**

Operator is available

**enumerator** **LWGSML\_OPERATOR\_STATUS\_CURRENT**

Operator is currently active

**enumerator** **LWGSML\_OPERATOR\_STATUS\_FORBIDDEN**

Operator is forbidden



```
enum lwgsm_operator_mode_t
    Operator selection mode.

    Values:

    enumerator LWGSM_OPERATOR_MODE_AUTO
        Operator automatic mode

    enumerator LWGSM_OPERATOR_MODE_MANUAL
        Operator manual mode

    enumerator LWGSM_OPERATOR_MODE_DEREGISTER
        Operator deregistered from network

    enumerator LWGSM_OPERATOR_MODE_MANUAL_AUTO
        Operator manual mode first. If fails, auto mode enabled

enum lwgsm_operator_format_t
    Operator data format.

    Values:

    enumerator LWGSM_OPERATOR_FORMAT_LONG_NAME
        COPS command returned long name

    enumerator LWGSM_OPERATOR_FORMAT_SHORT_NAME
        COPS command returned short name

    enumerator LWGSM_OPERATOR_FORMAT_NUMBER
        COPS command returned number

    enumerator LWGSM_OPERATOR_FORMAT_INVALID
        Unknown format
```

## Functions

```
lwgsmr_t lwgsm_operator_get (lwgsm_operator_curr_t *curr, const lwgsm_api_cmd_evt_fn
    evt_fn, void *const evt_arg, const uint32_t blocking)
```

Get current operator.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

### Parameters

- [out] curr: Pointer to output variable to save info about current operator
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not used
- [in] evt\_arg: Custom argument for event callback function
- [in] blocking: Status whether command should be blocking or not

```
lwgsmr_t lwgsm_operator_set (lwgsm_operator_mode_t mode, lwgsm_operator_format_t format,
    const char *name, uint32_t num, const lwgsm_api_cmd_evt_fn
    evt_fn, void *const evt_arg, const uint32_t blocking)
```

Set current operator.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

### Parameters

- [in] mode: Operator mode. This parameter can be a value of *lwgsm\_operator\_mode\_t* enumeration
- [in] format: Operator data format. This parameter can be a value of *lwgsm\_operator\_format\_t* enumeration
- [in] name: Operator name. This parameter must be valid according to format parameter
- [in] num: Operator number. This parameter must be valid according to format parameter
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not used
- [in] evt\_arg: Custom argument for event callback function
- [in] blocking: Status whether command should be blocking or not

```
lwgsmr_t lwgsm_operator_scan(lwgsm_operator_t *ops, size_t opsl, size_t *opf, const  
                             lwgsm_api_cmd_evt_fn evt_fn, void *const evt_arg, const  
                             uint32_t blocking)
```

Scan for available operators.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

#### Parameters

- [in] ops: Pointer to array to write found operators
- [in] opsl: Length of input array in units of elements
- [out] opf: Pointer to output variable to save number of operators found
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not used
- [in] evt\_arg: Custom argument for event callback function
- [in] blocking: Status whether command should be blocking or not

```
struct lwgsm_operator_t  
#include <lwgsm_typedefs.h> Operator details for scan.
```

#### Public Members

```
lwgsm_operator_status_t stat  
Operator status
```

```
char long_name[20]  
Operator long name
```

```
char short_name[20]  
Operator short name
```

```
uint32_t num  
Operator numeric value
```

```
struct lwgsm_operator_curr_t  
#include <lwgsm_typedefs.h> Current operator info.
```

## Public Members

*lwgsm\_operator\_mode\_t* **mode**  
Operator mode

*lwgsm\_operator\_format\_t* **format**  
Data format

char **long\_name**[20]  
Long name format

char **short\_name**[20]  
Short name format

uint32\_t **num**  
Number format

**union** *lwgsm\_operator\_curr\_t::*[anonymous] **data**  
Operator data union

## Packet buffer

Packet buffer (or *pbuf*) is buffer manager to handle received data from any connection. It is optimized to construct big buffer of smaller chunks of fragmented data as received bytes are not always coming as single packet.

## Pbuf block diagram

Fig. 3: Block diagram of pbuf chain

Image above shows structure of *pbuf* chain. Each *pbuf* consists of:

- Pointer to next *pbuf*, or NULL when it is last in chain
- Length of current packet length
- Length of current packet and all next in chain
  - If *pbuf* is last in chain, total length is the same as current packet length
- Reference counter, indicating how many pointers point to current *pbuf*
- Actual buffer data

Top image shows 3 pbufs connected to single chain. There are 2 custom pointer variables to point at different *pbuf* structures. Second *pbuf* has reference counter set to 2, as 2 variables point to it:

- *next* of *pbuf 1* is the first one
- *User variable 2* is the second one

Table 1: Block structure

Block number	Next pbuf	Block size	Total size in chain	Reference counter
Block 1	<i>Block 2</i>	150	550	1
Block 2	<i>Block 3</i>	130	400	2
Block 3	NULL	270	270	1

## Reference counter

Reference counter holds number of references (or variables) pointing to this block. It is used to properly handle memory free operation, especially when *pbuf* is used by lib core and application layer.

---

**Note:** If there would be no reference counter information and application would free memory while another part of library still uses its reference, application would invoke *undefined behavior* and system could crash instantly.

---

When application tries to free pbuf chain as on first image, it would normally call `lwgsm_pbuf_free()` function. That would:

- Decrease reference counter by 1
- If reference counter == 0, it removes it from chain list and frees packet buffer memory
- If reference counter != 0 after decrease, it stops free procedure
- Go to next pbuf in chain and repeat steps

As per first example, result of freeing from *user variable 1* would look similar to image and table below. First block (blue) had reference counter set to 1 prior freeing operation. It was successfully removed as *user variable 1* was the only one pointing to it, while second (green) block had reference counter set to 2, preventing free operation.

Fig. 4: Block diagram of pbuf chain after free from *user variable 1*

Table 2: Block diagram of pbuf chain after free from *user variable 1*

Block number	Next pbuf	Block size	Total size in chain	Reference counter
Block 2	Block 3	130	400	1
Block 3	NULL	270	270	1

---

**Note:** *Block 1* has been successfully freed, but since *block 2* had reference counter set to 2 before, it was only decreased by 1 to a new value 1 and free operation stopped instead. *User variable 2* is still using *pbuf* starting at *block 2* and must manually call `lwgsm_pbuf_free()` to free it.

---

## Concatenating vs chaining

This section will explain difference between *concat* and *chain* operations. Both operations link 2 pbufs together in a chain of pbufs, difference is that *chain* operation increases *reference counter* to linked pbuf, while *concat* keeps *reference counter* at its current status.

Fig. 5: Different pbufs, each pointed to by its own variable

## Concat operation

Concat operation shall be used when 2 pbufs are linked together and reference to *second* is no longer used.

Fig. 6: Structure after pbuf concat

After concating 2 *pbufs* together, reference counter of *second* is still set to 1, however we can see that 2 pointers point to *second pbuf*.

---

**Note:** After application calls `lwgsm_pbuf_cat()`, it must not use pointer which points to *second pbuf*. This would invoke *undefined behavior* if one pointer tries to free memory while *second* still points to it.

---

An example code showing proper usage of concat operation:

Listing 17: Packet buffer concat example

```

1  lwgsm_pbuf_p a, b;
2
3  /* Create 2 pbufs of different sizes */
4  a = lwgsm_pbuf_new(10);
5  b = lwgsm_pbuf_new(20);
6
7  /* Link them together with concat operation */
8  /* Reference on b will stay as is, won't be increased */
9  lwgsm_pbuf_cat(a, b);
10
11 /*
12  * Operating with b variable has from now on undefined behavior,
13  * application shall stop using variable b to access pbuf.
14  *
15  * The best way would be to set b reference to NULL
16  */
17 b = NULL;
18
19 /*
20  * When application doesn't need pbufs anymore,
21  * free a and it will also free b
22  */
23 lwgsm_pbuf_free(a);

```

## Chain operation

Chain operation shall be used when 2 pbufs are linked together and reference to *second* is still required.

Fig. 7: Structure after pbuf chain

After chainin 2 *pbufs* together, reference counter of *second* is increased by 1, which allows application to reference *second pbuf* separately.

---

**Note:** After application calls `lwgsm_pbuf_chain()`, it also has to manually free its reference using `lwgsm_pbuf_free()` function. Forgetting to free pbuf invokes memory leak

---

An example code showing proper usage of chain operation:

Listing 18: Packet buffer chain example

```
1 lwgsm_pbuf_p a, b;
2
3 /* Create 2 pbufs of different sizes */
4 a = lwgsm_pbuf_new(10);
5 b = lwgsm_pbuf_new(20);
6
7 /* Chain both pbufs together */
8 /* This will increase reference on b as 2 variables now point to it */
9 lwgsm_pbuf_chain(a, b);
10
11 /*
12  * When application does not need a anymore, it may free it
13
14  * This will free only pbuf a, as pbuf b has now 2 references:
15  * - one from pbuf a
16  * - one from variable b
17  */
18
19 /* If application calls this, it will free only first pbuf */
20 /* As there is link to b pbuf somewhere */
21 lwgsm_pbuf_free(a);
22
23 /* Reset a variable, not used anymore */
24 a = NULL;
25
26 /*
27  * At this point, b is still valid memory block,
28  * but when application doesn't need it anymore,
29  * it should free it, otherwise memory leak appears
30  */
31 lwgsm_pbuf_free(b);
32
33 /* Reset b variable */
34 b = NULL;
```

## Extract pbuf data

Each *pbuf* holds some amount of data bytes. When multiple *pbufs* are linked together (either chained or concated), blocks of raw data are not linked to contiguous memory block. It is necessary to process block by block manually.

An example code showing proper reading of any *pbuf*:

Listing 19: Packet buffer data extraction

```
1 const void* data;
2 size_t pos, len;
3 lwgsm_pbuf_p a, b, c;
4
5 const char str_a[] = "This is one long";
6 const char str_a[] = "string. We want to save";
7 const char str_a[] = "chain of pbufs to file";
8
```

(continues on next page)

(continued from previous page)

```

9  /* Create pbufs to hold these strings */
10 a = lwgsm_pbuf_new(strlen(str_a));
11 b = lwgsm_pbuf_new(strlen(str_b));
12 c = lwgsm_pbuf_new(strlen(str_c));
13
14 /* Write data to pbufs */
15 lwgsm_pbuf_take(a, str_a, strlen(str_a), 0);
16 lwgsm_pbuf_take(b, str_b, strlen(str_b), 0);
17 lwgsm_pbuf_take(c, str_c, strlen(str_c), 0);
18
19 /* Connect pbufs together */
20 lwgsm_pbuf_chain(a, b);
21 lwgsm_pbuf_chain(a, c);
22
23 /*
24  * pbuf a now contains chain of b and c together
25  * and at this point application wants to print (or save) data from chained pbuf
26  *
27  * Process pbuf by pbuf with code below
28  */
29
30 /*
31  * Get linear address of current pbuf at specific offset
32  * Function will return pointer to memory address at specific position
33  * and `len` will hold length of data block
34  */
35 pos = 0;
36 while ((data = lwgsm_pbuf_get_linear_addr(a, pos, &len)) != NULL) {
37     /* Custom process function... */
38     /* Process data with data pointer and block length */
39     process_data(data, len);
40     printf("Str: %.*s", len, data);
41
42     /* Increase offset position for next block */
43     pos += len;
44 }
45
46 /* Call free only on a pbuf. Since it is chained, b and c will be freed too */
47 lwgsm_pbuf_free(a);

```

group **LWGSMPBUF**  
 Packet buffer manager.

## Typedefs

**typedef struct** lwgsm\_pbuf \***lwgsm\_pbuf\_p**  
Pointer to *lwgsm\_pbuf\_t* structure.

## Functions

*lwgsm\_pbuf\_p* **lwgsm\_pbuf\_new** (size\_t len)  
Allocate packet buffer for network data of specific size.

**Return** Pointer to allocated memory, NULL otherwise

### Parameters

- [in] len: Length of payload memory to allocate

size\_t **lwgsm\_pbuf\_free** (*lwgsm\_pbuf\_p* pbuf)  
Free previously allocated packet buffer.

**Return** Number of freed pbufs from head

### Parameters

- [in] pbuf: Packet buffer to free

void \***lwgsm\_pbuf\_data** (const *lwgsm\_pbuf\_p* pbuf)  
Get data pointer from packet buffer.

**Return** Pointer to data buffer on success, NULL otherwise

### Parameters

- [in] pbuf: Packet buffer

size\_t **lwgsm\_pbuf\_length** (const *lwgsm\_pbuf\_p* pbuf, uint8\_t tot)  
Get length of packet buffer.

**Return** Length of data in units of bytes

### Parameters

- [in] pbuf: Packet buffer to get length for
- [in] tot: Set to 1 to return total packet chain length or 0 to get only first packet length

*lwgsnr\_t* **lwgsm\_pbuf\_take** (*lwgsm\_pbuf\_p* pbuf, const void \*data, size\_t len, size\_t offset)  
Copy user data to chain of pbufs.

**Return** *lwgsnrOK* on success, member of *lwgsnr\_t* enumeration otherwise

### Parameters

- [in] pbuf: First pbuf in chain to start copying to
- [in] data: Input data to copy to pbuf memory
- [in] len: Length of input data to copy
- [in] offset: Start offset in pbuf where to start copying



`size_t lwgsm_pbuf_copy (lwgsm_pbuf_p pbuf, void *data, size_t len, size_t offset)`  
Copy memory from pbuf to user linear memory.

**Return** Number of bytes copied

**Parameters**

- [in] pbuf: Pbuf to copy from
- [out] data: User linear memory to copy to
- [in] len: Length of data in units of bytes
- [in] offset: Possible start offset in pbuf

`lwgsmr_t lwgsm_pbuf_cat (lwgsm_pbuf_p head, const lwgsm_pbuf_p tail)`  
Concatenate 2 packet buffers together to one big packet.

**Note** After `tail` pbuf has been added to `head` pbuf chain, it must not be referenced by user anymore as it is now completely controlled by `head` pbuf. In simple words, when user calls this function, it should not call `lwgsm_pbuf_free` function anymore, as it might make memory undefined for `head` pbuf.

**Return** `lwgsmOK` on success, member of `lwgsmr_t` enumeration otherwise

**See** `lwgsm_pbuf_chain`

**Parameters**

- [in] head: Head packet buffer to append new pbuf to
- [in] tail: Tail packet buffer to append to head pbuf

`lwgsmr_t lwgsm_pbuf_chain (lwgsm_pbuf_p head, lwgsm_pbuf_p tail)`

Chain 2 pbufs together. Similar to `lwgsm_pbuf_cat` but now new reference is done from head pbuf to tail pbuf.

**Note** After this function call, user must call `lwgsm_pbuf_free` to remove its reference to tail pbuf and allow control to head pbuf: `lwgsm_pbuf_free (tail)`

**Return** `lwgsmOK` on success, member of `lwgsmr_t` enumeration otherwise

**See** `lwgsm_pbuf_cat`

**Parameters**

- [in] head: Head packet buffer to append new pbuf to
- [in] tail: Tail packet buffer to append to head pbuf

`lwgsmr_t lwgsm_pbuf_ref (lwgsm_pbuf_p pbuf)`

Increment reference count on pbuf.

**Return** `lwgsmOK` on success, member of `lwgsmr_t` enumeration otherwise

**Parameters**

- [in] pbuf: pbuf to increase reference

`uint8_t lwgsm_pbuf_get_at (const lwgsm_pbuf_p pbuf, size_t pos, uint8_t *el)`  
Get value from pbuf at specific position.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] pbuf: Pbuf used to get data from
- [in] pos: Position at which to get element
- [out] el: Output variable to save element value at desired position

`size_t lwgsm_pbuf_memcmp (const lwgsm_pbuf_p pbuf, const void *data, size_t len, size_t offset)`  
Compare pbuf memory with memory from data.

**Note** Compare is done on entire pbuf chain

**Return** 0 if equal, LWGSM\_SIZET\_MAX if memory/offset too big or anything between if not equal

**See** *lwgsm\_pbuf\_strcmp*

**Parameters**

- [in] pbuf: Pbuf used to compare with data memory
- [in] data: Actual data to compare with
- [in] len: Length of input data in units of bytes
- [in] offset: Start offset to use when comparing data

`size_t lwgsm_pbuf_strcmp (const lwgsm_pbuf_p pbuf, const char *str, size_t offset)`  
Compare pbuf memory with input string.

**Note** Compare is done on entire pbuf chain

**Return** 0 if equal, LWGSM\_SIZET\_MAX if memory/offset too big or anything between if not equal

**See** *lwgsm\_pbuf\_memcmp*

**Parameters**

- [in] pbuf: Pbuf used to compare with data memory
- [in] str: String to be compared with pbuf
- [in] offset: Start memory offset in pbuf

`size_t lwgsm_pbuf_memfind (const lwgsm_pbuf_p pbuf, const void *data, size_t len, size_t off)`  
Find desired needle in a haystack.

**Return** LWGSM\_SIZET\_MAX if no match or position where in pbuf we have a match

**See** *lwgsm\_pbuf\_strfind*

**Parameters**

- [in] pbuf: Pbuf used as haystack
- [in] needle: Data memory used as needle
- [in] len: Length of needle memory
- [in] off: Starting offset in pbuf memory

`size_t lwgsm_pbuf_strfind (const lwgsm_pbuf_p pbuf, const char *str, size_t off)`  
Find desired needle (str) in a haystack (pbuf)

**Return** LWGSM\_SIZE\_MAX if no match or position where in pbuf we have a match

**See** [lwgsm\\_pbuf\\_memfind](#)

**Parameters**

- [in] pbuf: Pbuf used as haystack
- [in] str: String to search for in pbuf
- [in] off: Starting offset in pbuf memory

uint8\_t **lwgsm\_pbuf\_advance** ([lwgsm\\_pbuf\\_p](#) pbuf, int len)

Advance pbuf payload pointer by number of len bytes. It can only advance single pbuf in a chain.

**Note** When other pbuffers are referencing current one, they are not adjusted in length and total length

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] pbuf: Pbuf to advance
- [in] len: Number of bytes to advance. when negative is used, buffer size is increased only if it was decreased before

[lwgsm\\_pbuf\\_p](#) **lwgsm\_pbuf\_skip** ([lwgsm\\_pbuf\\_p](#) pbuf, size\_t offset, size\_t \*new\_offset)

Skip a list of pbuffers for desired offset.

**Note** Reference is not changed after return and user must not free the memory of new pbuf directly

**Return** New pbuf on success, NULL otherwise

**Parameters**

- [in] pbuf: Start of pbuf chain
- [in] offset: Offset in units of bytes to skip
- [out] new\_offset: Pointer to output variable to save new offset in returned pbuf

void \***lwgsm\_pbuf\_get\_linear\_addr** (const [lwgsm\\_pbuf\\_p](#) pbuf, size\_t offset, size\_t \*new\_len)

Get linear offset address for pbuf from specific offset.

**Note** Since pbuf memory can be fragmented in chain, you may need to call function multiple times to get memory for entire pbuf chain

**Return** Pointer to memory on success, NULL otherwise

**Parameters**

- [in] pbuf: Pbuf to get linear address
- [in] offset: Start offset from where to start
- [out] new\_len: Length of memory returned by function

void **lwgsm\_pbuf\_set\_ip** ([lwgsm\\_pbuf\\_p](#) pbuf, const [lwgsm\\_ip\\_t](#) \*ip, [lwgsm\\_port\\_t](#) port)

Set IP address and port number for received data.

**Parameters**

- [in] pbuf: Packet buffer

- [in] `ip`: IP to assign to packet buffer
- [in] `port`: Port number to assign to packet buffer

**struct lwgsm\_pbuf\_t**  
*#include <lwgsm\_private.h>* Packet buffer structure.

### Public Members

**struct lwgsm\_pbuf \*next**  
Next pbuf in chain list

**size\_t tot\_len**  
Total length of pbuf chain

**size\_t len**  
Length of payload

**size\_t ref**  
Number of references to this structure

**uint8\_t \*payload**  
Pointer to payload memory

*lwgsm\_ip\_t* **ip**  
Remote address for received IPD data

*lwgsm\_port\_t* **port**  
Remote port for received IPD data

### Phonebook

*group* **LWGSM\_PHONEBOOK**  
Phonebook manager.

### Functions

*lwgsmr\_t* **lwgsm\_pb\_enable**(**const** *lwgsm\_api\_cmd\_evt\_fn* *evt\_fn*, void \***const** *evt\_arg*, **const** *uint32\_t* *blocking*)  
Enable phonebook functionality.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* otherwise

#### Parameters

- [in] `evt_fn`: Callback function called when command has finished. Set to NULL when not used
- [in] `evt_arg`: Custom argument for event callback function
- [in] `blocking`: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsm\_pb\_disable**(**const** *lwgsm\_api\_cmd\_evt\_fn* *evt\_fn*, void \***const** *evt\_arg*, **const** *uint32\_t* *blocking*)  
Disable phonebook functionality.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* otherwise

**Parameters**

- [in] `evt_fn`: Callback function called when command has finished. Set to `NULL` when not used
- [in] `evt_arg`: Custom argument for event callback function
- [in] `blocking`: Status whether command should be blocking or not

```
lwgsmr_t lwgsmr_pb_add(lwgsmr_mem_t mem, const char *name, const char *num,
                        lwgsmr_number_type_t type, const lwgsmr_api_cmd_evt_fn evt_fn,
                        void *const evt_arg, const uint32_t blocking)
```

Add new phonebook entry to desired memory.

**Return** *lwgsmrOK* on success, member of *lwgsmr\_t* otherwise

**Parameters**

- [in] `mem`: Memory to use to save entry. Use *LWGSMMEM\_CURRENT* to use current memory
- [in] `name`: Entry name
- [in] `num`: Entry phone number
- [in] `type`: Entry phone number type
- [in] `evt_fn`: Callback function called when command has finished. Set to `NULL` when not used
- [in] `evt_arg`: Custom argument for event callback function
- [in] `blocking`: Status whether command should be blocking or not

```
lwgsmr_t lwgsmr_pb_edit(lwgsmr_mem_t mem, size_t pos, const char *name, const char *num,
                          lwgsmr_number_type_t type, const lwgsmr_api_cmd_evt_fn evt_fn, void
                          *const evt_arg, const uint32_t blocking)
```

Edit or overwrite phonebook entry at desired memory and position.

**Return** *lwgsmrOK* on success, member of *lwgsmr\_t* otherwise

**Parameters**

- [in] `mem`: Memory to use to save entry. Use *LWGSMMEM\_CURRENT* to use current memory
- [in] `pos`: Entry position in memory to edit
- [in] `name`: New entry name
- [in] `num`: New entry phone number
- [in] `type`: New entry phone number type
- [in] `evt_fn`: Callback function called when command has finished. Set to `NULL` when not used
- [in] `evt_arg`: Custom argument for event callback function
- [in] `blocking`: Status whether command should be blocking or not

```
lwgsmr_t lwgsmr_pb_delete(lwgsmr_mem_t mem, size_t pos, const lwgsmr_api_cmd_evt_fn evt_fn,
                           void *const evt_arg, const uint32_t blocking)
```

Delete phonebook entry at desired memory and position.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* otherwise

**Parameters**

- [in] mem: Memory to use to save entry. Use *LWGSM\_MEM\_CURRENT* to use current memory
- [in] pos: Entry position in memory to delete
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not used
- [in] evt\_arg: Custom argument for event callback function
- [in] blocking: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsm\_pb\_read**(*lwgsm\_mem\_t* mem, size\_t pos, lwgsm\_pb\_entry\_t \*entry, **const** *lwgsm\_api\_cmd\_evt\_fn* evt\_fn, void \***const** evt\_arg, **const** uint32\_t blocking)

Read single phonebook entry.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* otherwise

**Parameters**

- [in] mem: Memory to use to save entry. Use *LWGSM\_MEM\_CURRENT* to use current memory
- [in] pos: Entry position in memory to read
- [out] entry: Pointer to entry variable to save data
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not used
- [in] evt\_arg: Custom argument for event callback function
- [in] blocking: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsm\_pb\_list**(*lwgsm\_mem\_t* mem, size\_t start\_index, lwgsm\_pb\_entry\_t \*entries, size\_t etr, size\_t \*er, **const** *lwgsm\_api\_cmd\_evt\_fn* evt\_fn, void \***const** evt\_arg, **const** uint32\_t blocking)

List entires from specific memory.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* otherwise

**Parameters**

- [in] mem: Memory to use to save entry. Use *LWGSM\_MEM\_CURRENT* to use current memory
- [in] start\_index: Start position in memory to list
- [out] entries: Pointer to array to save entries
- [in] etr: Number of entries to read
- [out] er: Pointer to output variable to save entries listed
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not used
- [in] evt\_arg: Custom argument for event callback function
- [in] blocking: Status whether command should be blocking or not

```
lwgsmr_t lwgsm_pb_search(lwgsm_mem_t mem, const char *search, lwgsm_pb_entry_t *en-
    tries, size_t etr, size_t *er, const lwgsm_api_cmd_evt_fn evt_fn, void
    *const evt_arg, const uint32_t blocking)
```

Search for entires with specific name from specific memory.

**Note** Search works by entry name only. Phone number search is not available

**Return** *lwgsmsOK* on success, member of *lwgsmr\_t* otherwise

#### Parameters

- [in] mem: Memory to use to save entry. Use *LWGSM\_MEM\_CURRENT* to use current mem-  
ory
- [in] search: String to search for
- [out] entries: Pointer to array to save entries
- [in] etr: Number of entries to read
- [out] er: Pointer to output variable to save entries found
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not  
used
- [in] evt\_arg: Custom argument for event callback function
- [in] blocking: Status whether command should be blocking or not

## Ping support

*group* **LWGSM\_PING**  
PING manager.

## SIM card

*group* **LWGSM\_SIM**  
SIM card manager.

## Enums

**enum lwgsm\_sim\_state\_t**  
SIM state.

*Values:*

**enumerator LWGSM\_SIM\_STATE\_NOT\_INSERTED**  
SIM is not inserted in socket

**enumerator LWGSM\_SIM\_STATE\_READY**  
SIM is ready for operations

**enumerator LWGSM\_SIM\_STATE\_NOT\_READY**  
SIM is not ready for any operation

**enumerator LWGSM\_SIM\_STATE\_PIN**  
SIM is waiting for SIM to be given

**enumerator LWGSM\_SIM\_STATE\_PUK**  
SIM is waiting for PUT to be given

enumerator **LWGSM\_SIM\_STATE\_PH\_PIN**

enumerator **LWGSM\_SIM\_STATE\_PH\_PUK**

## Functions

*lwgsm\_sim\_state\_t* **lwgsm\_sim\_get\_current\_state** (void)

Get current cached SIM state from stack.

**Note** Information is always valid, starting after successful device reset using *lwgsm\_reset* function call

**Return** Member of *lwgsm\_sim\_state\_t* enumeration

*lwgsmr\_t* **lwgsm\_sim\_pin\_enter** (const char \*pin, const *lwgsm\_api\_cmd\_evt\_fn* evt\_fn, void \*const evt\_arg, const uint32\_t blocking)

Enter pin code to unlock SIM.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

### Parameters

- [in] pin: Pin code in string format
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not used
- [in] evt\_arg: Custom argument for event callback function
- [in] blocking: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsm\_sim\_pin\_add** (const char \*pin, const *lwgsm\_api\_cmd\_evt\_fn* evt\_fn, void \*const evt\_arg, const uint32\_t blocking)

Add pin number to open SIM card.

**Note** Use this function only if your SIM card doesn't have PIN code. If you wish to change current pin, use *lwgsm\_sim\_pin\_change* instead

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

### Parameters

- [in] pin: Current SIM pin code
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not used
- [in] evt\_arg: Custom argument for event callback function
- [in] blocking: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsm\_sim\_pin\_remove** (const char \*pin, const *lwgsm\_api\_cmd\_evt\_fn* evt\_fn, void \*const evt\_arg, const uint32\_t blocking)

Remove pin code from SIM.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

### Parameters

- [in] pin: Current pin code
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not used



- [in] `evt_arg`: Custom argument for event callback function
- [in] `blocking`: Status whether command should be blocking or not

```
lwgsmr_t lwgsm_sim_pin_change(const char *pin, const char *new_pin, const
                                lwgsm_api_cmd_evt_fn evt_fn, void *const evt_arg, const
                                uint32_t blocking)
```

Change current pin code.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

#### Parameters

- [in] `pin`: Current pin code
- [in] `new_pin`: New pin code
- [in] `evt_fn`: Callback function called when command has finished. Set to NULL when not used
- [in] `evt_arg`: Custom argument for event callback function
- [in] `blocking`: Status whether command should be blocking or not

```
lwgsmr_t lwgsm_sim_puk_enter(const char *puk, const char *new_pin, const
                                lwgsm_api_cmd_evt_fn evt_fn, void *const evt_arg, const
                                uint32_t blocking)
```

Enter PUK code and new PIN to unlock SIM card.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

#### Parameters

- [in] `puk`: PUK code associated with SIM card
- [in] `new_pin`: New PIN code to use
- [in] `evt_fn`: Callback function called when command has finished. Set to NULL when not used
- [in] `evt_arg`: Custom argument for event callback function
- [in] `blocking`: Status whether command should be blocking or not

## SMS

*group* **LWGSM\_SMS**  
SMS manager.

### Enums

```
enum lwgsm_sms_status_t
    SMS status in current memory.
```

*Values:*

```
enumerator LWGSM_SMS_STATUS_ALL
    Process all SMS, used for mass delete or SMS list
```

```
enumerator LWGSM_SMS_STATUS_READ
    SMS status is read
```

**enumerator** **LWGSM\_SMS\_STATUS\_UNREAD**

SMS status is unread

**enumerator** **LWGSM\_SMS\_STATUS\_SENT**

SMS status is sent

**enumerator** **LWGSM\_SMS\_STATUS\_UNSENT**

SMS status is unsent

**enumerator** **LWGSM\_SMS\_STATUS\_INBOX**

SMS status, used only for mass delete operation

## Functions

*lwgsmr\_t* **lwgsms\_enable**(**const** *lwgsms\_api\_cmd\_evt\_fn* *evt\_fn*, void \***const** *evt\_arg*, **const** uint32\_t *blocking*)

Enable SMS functionality.

**Return** *lwgsmsOK* on success, member of *lwgsmr\_t* otherwise

### Parameters

- [in] *evt\_fn*: Callback function called when command has finished. Set to NULL when not used
- [in] *evt\_arg*: Custom argument for event callback function
- [in] *blocking*: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsms\_disable**(**const** *lwgsms\_api\_cmd\_evt\_fn* *evt\_fn*, void \***const** *evt\_arg*, **const** uint32\_t *blocking*)

Disable SMS functionality.

**Return** *lwgsmsOK* on success, member of *lwgsmr\_t* otherwise

### Parameters

- [in] *evt\_fn*: Callback function called when command has finished. Set to NULL when not used
- [in] *evt\_arg*: Custom argument for event callback function
- [in] *blocking*: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsms\_send**(**const** char \**num*, **const** char \**text*, **const** *lwgsms\_api\_cmd\_evt\_fn* *evt\_fn*, void \***const** *evt\_arg*, **const** uint32\_t *blocking*)

Send SMS text to phone number.

**Return** *lwgsmsOK* on success, member of *lwgsmr\_t* otherwise

### Parameters

- [in] *num*: String number
- [in] *text*: Text to send. Maximal 160 characters
- [in] *evt\_fn*: Callback function called when command has finished. Set to NULL when not used
- [in] *evt\_arg*: Custom argument for event callback function
- [in] *blocking*: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsms\_read**(*lwgsms\_mem\_t* mem, size\_t pos, *lwgsms\_sms\_entry\_t* \*entry, uint8\_t update, **const** *lwgsms\_api\_cmd\_evt\_fn* evt\_fn, void \***const** evt\_arg, **const** uint32\_t blocking)

Read SMS entry at specific memory and position.

**Return** *lwgsmsOK* on success, member of *lwgsmsmr\_t* otherwise

#### Parameters

- [in] mem: Memory used to read message from
- [in] pos: Position number in memory to read
- [out] entry: Pointer to SMS entry structure to fill data to
- [in] update: Flag indicates update. Set to 1 to change UNREAD messages to READ or 0 to leave as is
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not used
- [in] evt\_arg: Custom argument for event callback function
- [in] blocking: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsms\_delete**(*lwgsms\_mem\_t* mem, size\_t pos, **const** *lwgsms\_api\_cmd\_evt\_fn* evt\_fn, void \***const** evt\_arg, **const** uint32\_t blocking)

Delete SMS entry at specific memory and position.

**Return** *lwgsmsOK* on success, member of *lwgsmsmr\_t* otherwise

#### Parameters

- [in] mem: Memory used to read message from
- [in] pos: Position number in memory to read
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not used
- [in] evt\_arg: Custom argument for event callback function
- [in] blocking: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsms\_delete\_all**(*lwgsms\_sms\_status\_t* status, **const** *lwgsms\_api\_cmd\_evt\_fn* evt\_fn, void \***const** evt\_arg, **const** uint32\_t blocking)

Delete all SMS entries with specific status.

**Return** *lwgsmsOK* on success, member of *lwgsmsmr\_t* otherwise

#### Parameters

- [in] status: SMS status. This parameter can be one of all possible types in *lwgsms\_sms\_status\_t* enumeration
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not used
- [in] evt\_arg: Custom argument for event callback function
- [in] blocking: Status whether command should be blocking or not

```
lwgsmr_t lwgsm_sms_list(lwgsmr_t mem, lwgsmr_sms_status_t stat, lwgsmr_sms_entry_t
                        *entries, size_t etr, size_t *er, uint8_t update, const
                        lwgsmr_api_cmd_evt_fn evt_fn, void *const evt_arg, const uint32_t
                        blocking)
```

List SMS from SMS memory.

**Return** *lwgsmrOK* on success, member of *lwgsmr\_t* otherwise

**Parameters**

- [in] mem: Memory to read entries from. Use *LWGSM\_MEM\_CURRENT* to read from current memory
- [in] stat: SMS status to read, either read, unread, sent, unsent or all
- [out] entries: Pointer to array to save SMS entries
- [in] etr: Number of entries to read
- [out] er: Pointer to output variable to save number of entries in array
- [in] update: Flag indicates update. Set to 1 to change UNREAD messages to READ or 0 to leave as is
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not used
- [in] evt\_arg: Custom argument for event callback function
- [in] blocking: Status whether command should be blocking or not

```
lwgsmr_t lwgsm_sms_set_preferred_storage(lwgsmr_mem_t mem1, lwgsmr_mem_t
                                         mem2, lwgsmr_mem_t mem3, const
                                         lwgsmr_api_cmd_evt_fn evt_fn, void *const
                                         evt_arg, const uint32_t blocking)
```

Set preferred storage for SMS.

**Return** *lwgsmrOK* on success, member of *lwgsmr\_t* otherwise

**Parameters**

- [in] mem1: Preferred memory for read/delete SMS operations. Use *LWGSM\_MEM\_CURRENT* to keep it as is
- [in] mem2: Preferred memory for sent/write SMS operations. Use *LWGSM\_MEM\_CURRENT* to keep it as is
- [in] mem3: Preferred memory for received SMS entries. Use *LWGSM\_MEM\_CURRENT* to keep it as is
- [in] evt\_fn: Callback function called when command has finished. Set to NULL when not used
- [in] evt\_arg: Custom argument for event callback function
- [in] blocking: Status whether command should be blocking or not

```
struct lwgsm_sms_mem_t
#include <lwgsmr_private.h> SMS memory information.
```

**Public Members**

`uint32_t mem_available`  
Bit field of available memories

`lwgsm_mem_t current`  
Current memory choice

`size_t total`  
Size of memory in units of entries

`size_t used`  
Number of used entries

**struct lwgsm\_sms\_t**  
*#include <lwgsm\_private.h>* SMS structure.

**Public Members**

`uint8_t ready`  
Flag indicating feature ready by device

`uint8_t enabled`  
Flag indicating feature enabled

`lwgsm_sms_mem_t mem[3]`  
3 memory info for operation, receive, sent storage

**struct lwgsm\_pb\_mem\_t**  
*#include <lwgsm\_private.h>* SMS memory information.

**Public Members**

`uint32_t mem_available`  
Bit field of available memories

`lwgsm_mem_t current`  
Current memory choice

`size_t total`  
Size of memory in units of entries

`size_t used`  
Number of used entries

**struct lwgsm\_sms\_entry\_t**  
*#include <lwgsm\_typedefs.h>* SMS entry structure.

### Public Members

*lwgsm\_mem\_t* **mem**  
Memory storage

size\_t **pos**  
Memory position

*lwgsm\_datetime\_t* **datetime**  
Date and time

*lwgsm\_sms\_status\_t* **status**  
Message status

char **number**[26]  
Phone number

char **name**[20]  
Name in phonebook if exists

char **data**[161]  
Data memory

size\_t **length**  
Length of SMS data

### Timeout manager

Timeout manager allows application to call specific function at desired time. It is used in middleware (and can be used by application too) to poll active connections.

---

**Note:** Callback function is called from *processing* thread. It is not allowed to call any blocking API function from it.

---

When application registers timeout, it needs to set timeout, callback function and optional user argument. When timeout elapses, GSM middleware will call timeout callback.

This feature can be considered as single-shot software timer.

group **LWGSM\_TIMEOUT**  
Timeout manager.

### Typedefs

**typedef** void (\***lwgsm\_timeout\_fn**) (void \*arg)  
Timeout callback function prototype.

#### Parameters

- [in] **arg**: Custom user argument

## Functions

*lwgsmr\_t* **lwgsm\_timeout\_add** (uint32\_t *time*, *lwgsm\_timeout\_fn* *fn*, void \**arg*)

Add new timeout to processing list.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

### Parameters

- [in] *time*: Time in units of milliseconds for timeout execution
- [in] *fn*: Callback function to call when timeout expires
- [in] *arg*: Pointer to user specific argument to call when timeout callback function is executed

*lwgsmr\_t* **lwgsm\_timeout\_remove** (*lwgsm\_timeout\_fn* *fn*)

Remove callback from timeout list.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

### Parameters

- [in] *fn*: Callback function to identify timeout to remove

**struct** *lwgsm\_timeout\_t*

*#include <lwgsm\_typedefs.h>* Timeout structure.

## Public Members

**struct** *lwgsm\_timeout* \***next**

Pointer to next timeout entry

uint32\_t **time**

Time difference from previous entry

void \***arg**

Argument to pass to callback function

*lwgsm\_timeout\_fn* **fn**

Callback function for timeout

## Structures and enumerations

*group* **LWGSM\_TYPEDEFS**

List of core structures and enumerations.

## Typedefs

**typedef** uint16\_t *lwgsm\_port\_t*

Port variable.

**typedef** void (\**lwgsm\_api\_cmd\_evt\_fn*) (*lwgsmr\_t* *res*, void \**arg*)

Function declaration for API function command event callback function.

### Parameters

- [in] *res*: Operation result, member of *lwgsmr\_t* enumeration

- [in] `arg`: Custom user argument

## Enums

**enum** `lwgsm_cmd_t`

List of possible messages.

*Values:*

**enumerator** `LWGSMD_CMD_IDLE`

IDLE mode

**enumerator** `LWGSMD_CMD_RESET`

Reset device

**enumerator** `LWGSMD_CMD_RESET_DEVICE_FIRST_CMD`

Reset device first driver specific command

**enumerator** `LWGSMD_CMD_ATE0`

Disable ECHO mode on AT commands

**enumerator** `LWGSMD_CMD_ATE1`

Enable ECHO mode on AT commands

**enumerator** `LWGSMD_CMD_GSLP`

Set GSM to sleep mode

**enumerator** `LWGSMD_CMD_RESTORE`

Restore GSM internal settings to default values

**enumerator** `LWGSMD_CMD_UART`

**enumerator** `LWGSMD_CMD_CGACT_SET_0`

**enumerator** `LWGSMD_CMD_CGACT_SET_1`

**enumerator** `LWGSMD_CMD_CGATT_SET_0`

**enumerator** `LWGSMD_CMD_CGATT_SET_1`

**enumerator** `LWGSMD_CMD_NETWORK_ATTACH`

Attach to a network

**enumerator** `LWGSMD_CMD_NETWORK_DETACH`

Detach from network

**enumerator** `LWGSMD_CMD_CIPMUX_SET`

**enumerator** `LWGSMD_CMD_CIPRXGET_SET`

**enumerator** `LWGSMD_CMD_CSTT_SET`

**enumerator** `LWGSMD_CMD_CALL_ENABLE`

**enumerator** `LWGSMD_CMD_A`

Re-issues the Last Command Given

**enumerator** `LWGSMD_CMD_ATA`

Answer an Incoming Call

**enumerator** `LWGSMD_CMD_ATD`

Mobile Originated Call to Dial A Number

**enumerator** `LWGSMD_CMD_ATD_N`

Originate Call to Phone Number in Current Memory: ATD<n>



**enumerator LWGSM\_CMD\_ATD\_STR**  
Originate Call to Phone Number in Memory Which Corresponds to Field “str”: ATD>str

**enumerator LWGSM\_CMD\_ATDL**  
Redial Last Telephone Number Used

**enumerator LWGSM\_CMD\_ATE**  
Set Command Echo Mode

**enumerator LWGSM\_CMD\_ATH**  
Disconnect Existing

**enumerator LWGSM\_CMD\_ATI**  
Display Product Identification Information

**enumerator LWGSM\_CMD\_ATL**  
Set Monitor speaker

**enumerator LWGSM\_CMD\_ATM**  
Set Monitor Speaker Mode

**enumerator LWGSM\_CMD\_PPP**  
Switch from Data Mode or PPP Online Mode to Command Mode, “+++” originally

**enumerator LWGSM\_CMD\_ATO**  
Switch from Command Mode to Data Mode

**enumerator LWGSM\_CMD\_ATP**  
Select Pulse Dialing

**enumerator LWGSM\_CMD\_ATQ**  
Set Result Code Presentation Mode

**enumerator LWGSM\_CMD\_ATS0**  
Set Number of Rings before Automatically Answering the Call

**enumerator LWGSM\_CMD\_ATS3**  
Set Command Line Termination Character

**enumerator LWGSM\_CMD\_ATS4**  
Set Response Formatting Character

**enumerator LWGSM\_CMD\_ATS5**  
Set Command Line Editing Character

**enumerator LWGSM\_CMD\_ATS6**  
Pause Before Blind

**enumerator LWGSM\_CMD\_ATS7**  
Set Number of Seconds to Wait for Connection Completion

**enumerator LWGSM\_CMD\_ATS8**  
Set Number of Seconds to Wait for Comma Dial Modifier Encountered in Dial String of D Command

**enumerator LWGSM\_CMD\_ATS10**  
Set Disconnect Delay after Indicating the Absence of Data Carrier

**enumerator LWGSM\_CMD\_ATT**  
Select Tone Dialing

**enumerator LWGSM\_CMD\_ATV**  
TA Response Format

**enumerator LWGSM\_CMD\_ATX**  
Set CONNECT Result Code Format and Monitor Call Progress

**enumerator LWGSM\_CMD\_ATZ**  
Reset Default Configuration

**enumerator LWGSM\_CMD\_AT\_C**  
Set DCD Function Mode, AT&C

**enumerator LWGSM\_CMD\_AT\_D**  
Set DTR Function, AT&D

**enumerator LWGSM\_CMD\_AT\_F**  
Factory Defined Configuration, AT&F

**enumerator LWGSM\_CMD\_AT\_V**  
Display Current Configuration, AT&V

**enumerator LWGSM\_CMD\_AT\_W**  
Store Active Profile, AT&W

**enumerator LWGSM\_CMD\_GCAP**  
Request Complete TA Capabilities List

**enumerator LWGSM\_CMD\_GMI**  
Request Manufacturer Identification

**enumerator LWGSM\_CMD\_GMM**  
Request TA Model Identification

**enumerator LWGSM\_CMD\_GMR**  
Request TA Revision Identification of Software Release

**enumerator LWGSM\_CMD\_GOI**  
Request Global Object Identification

**enumerator LWGSM\_CMD\_GSN**  
Request TA Serial Number Identification (IMEI)

**enumerator LWGSM\_CMD\_ICF**  
Set TE-TA Control Character Framing

**enumerator LWGSM\_CMD\_IFC**  
Set TE-TA Local Data Flow Control

**enumerator LWGSM\_CMD\_IPR**  
Set TE-TA Fixed Local Rate

**enumerator LWGSM\_CMD\_HVOIC**  
Disconnect Voice Call Only

**enumerator LWGSM\_CMD\_COPS\_SET**  
Set operator

**enumerator LWGSM\_CMD\_COPS\_GET**  
Get current operator

**enumerator LWGSM\_CMD\_COPS\_GET\_OPT**  
Get a list of available operators

**enumerator LWGSM\_CMD\_CPAS**  
Phone Activity Status

**enumerator LWGSM\_CMD\_CGMI\_GET**  
Request Manufacturer Identification

**enumerator LWGSM\_CMD\_CGMM\_GET**  
Request Model Identification

**enumerator LWGSM\_CMD\_CGMR\_GET**  
Request TA Revision Identification of Software Release

**enumerator LWGSM\_CMD\_CGSN\_GET**  
Request Product Serial Number Identification (Identical with +GSN)

**enumerator LWGSM\_CMD\_CLCC\_SET**  
List Current Calls of ME

**enumerator LWGSM\_CMD\_CLCK**  
Facility Lock

**enumerator LWGSM\_CMD\_CACM**  
Accumulated Call Meter (ACM) Reset or Query

**enumerator LWGSM\_CMD\_CAMM**  
Accumulated Call Meter Maximum (ACM max) Set or Query

**enumerator LWGSM\_CMD\_CAOC**  
Advice of Charge

**enumerator LWGSM\_CMD\_CBST**  
Select Bearer Service Type

**enumerator LWGSM\_CMD\_CCFC**  
Call Forwarding Number and Conditions Control

**enumerator LWGSM\_CMD\_CCWA**  
Call Waiting Control

**enumerator LWGSM\_CMD\_CEER**  
Extended Error Report

**enumerator LWGSM\_CMD\_CSCS**  
Select TE Character Set

**enumerator LWGSM\_CMD\_CSTA**  
Select Type of Address

**enumerator LWGSM\_CMD\_CHLD**  
Call Hold and Multiparty

**enumerator LWGSM\_CMD\_CIMI**  
Request International Mobile Subscriber Identity

**enumerator LWGSM\_CMD\_CLIP**  
Calling Line Identification Presentation

**enumerator LWGSM\_CMD\_CLIR**  
Calling Line Identification Restriction

**enumerator LWGSM\_CMD\_CMEE\_SET**  
Report Mobile Equipment Error

**enumerator LWGSM\_CMD\_COLP**  
Connected Line Identification Presentation

**enumerator LWGSM\_CMD\_PHONEBOOK\_ENABLE**

**enumerator LWGSM\_CMD\_CPBF**  
Find Phonebook Entries

**enumerator LWGSM\_CMD\_CPBR**  
Read Current Phonebook Entries

**enumerator LWGSM\_CMD\_CPBS\_SET**  
Select Phonebook Memory Storage

**enumerator LWGSM\_CMD\_CPBS\_GET**  
Get current Phonebook Memory Storage

**enumerator LWGSM\_CMD\_CPBS\_GET\_OPT**  
Get available Phonebook Memory Storages

**enumerator LWGSM\_CMD\_CPBW\_SET**  
Write Phonebook Entry

**enumerator LWGSM\_CMD\_CPBW\_GET\_OPT**  
Get options for write Phonebook Entry

**enumerator LWGSM\_CMD\_SIM\_PROCESS\_BASIC\_CMDS**  
Command setup, executed when SIM is in READY state

**enumerator LWGSM\_CMD\_CPIN\_SET**  
Enter PIN

**enumerator LWGSM\_CMD\_CPIN\_GET**  
Read current SIM status

**enumerator LWGSM\_CMD\_CPIN\_ADD**  
Add new PIN to SIM if pin is not set

**enumerator LWGSM\_CMD\_CPIN\_CHANGE**  
Change already active SIM

**enumerator LWGSM\_CMD\_CPIN\_REMOVE**  
Remove current PIN

**enumerator LWGSM\_CMD\_CPUK\_SET**  
Enter PUK and set new PIN

**enumerator LWGSM\_CMD\_CSQ\_GET**  
Signal Quality Report

**enumerator LWGSM\_CMD\_CFUN\_SET**  
Set Phone Functionality

**enumerator LWGSM\_CMD\_CFUN\_GET**  
Get Phone Functionality

**enumerator LWGSM\_CMD\_CREG\_SET**  
Network Registration set output

**enumerator LWGSM\_CMD\_CREG\_GET**  
Get current network registration status

**enumerator LWGSM\_CMD\_CBC**  
Battery Charge

**enumerator LWGSM\_CMD\_CNUM**  
Subscriber Number

**enumerator LWGSM\_CMD\_CPWD**  
Change Password

**enumerator LWGSM\_CMD\_CR**  
Service Reporting Control

**enumerator LWGSM\_CMD\_CRC**  
Set Cellular Result Codes for Incoming Call Indication

**enumerator LWGSM\_CMD\_CRLP**  
Select Radio Link Protocol Parameters

**enumerator LWGSM\_CMD\_CRSM**  
Restricted SIM Access

**enumerator LWGSM\_CMD\_VTD**  
Tone Duration

**enumerator LWGSM\_CMD\_VTS**  
DTMF and Tone Generation

**enumerator LWGSM\_CMD\_CMUX**  
Multiplexer Control

**enumerator LWGSM\_CMD\_CPOL**  
Preferred Operator List

**enumerator LWGSM\_CMD\_COPN**  
Read Operator Names

**enumerator LWGSM\_CMD\_CCLK**  
Clock

**enumerator LWGSM\_CMD\_CSIM**  
Generic SIM Access

**enumerator LWGSM\_CMD\_CALM**  
Alert Sound Mode

**enumerator LWGSM\_CMD\_CALS**  
Alert Sound Select

**enumerator LWGSM\_CMD\_CRSL**  
Ringer Sound Level

**enumerator LWGSM\_CMD\_CLVL**  
Loud Speaker Volume Level

**enumerator LWGSM\_CMD\_CMUT**  
Mute Control

**enumerator LWGSM\_CMD\_CPUC**  
Price Per Unit and Currency Table

**enumerator LWGSM\_CMD\_CCWE**  
Call Meter Maximum Event

**enumerator LWGSM\_CMD\_CUSD\_SET**  
Unstructured Supplementary Service Data, Set command

**enumerator LWGSM\_CMD\_CUSD\_GET**  
Unstructured Supplementary Service Data, Get command

**enumerator LWGSM\_CMD\_CUSD**  
Unstructured Supplementary Service Data, Execute command

**enumerator LWGSM\_CMD\_CSSN**  
Supplementary Services Notification

**enumerator LWGSM\_CMD\_CIPMUX**  
Start Up Multi-IP Connection

**enumerator LWGSM\_CMD\_CIPSTART**  
Start Up TCP or UDP Connection

**enumerator LWGSM\_CMD\_CIPSEND**  
Send Data Through TCP or UDP Connection

**enumerator LWGSM\_CMD\_CIPQSEND**  
Select Data Transmitting Mode

**enumerator LWGSM\_CMD\_CIPACK**  
Query Previous Connection Data Transmitting State

**enumerator LWGSM\_CMD\_CIPCLOSE**  
Close TCP or UDP Connection

**enumerator LWGSM\_CMD\_CIPSHUT**  
Deactivate GPRS PDP Context

**enumerator LWGSM\_CMD\_CLPORT**  
Set Local Port

**enumerator LWGSM\_CMD\_CSTT**  
Start Task and Set APN, username, password

**enumerator LWGSM\_CMD\_CIICR**  
Bring Up Wireless Connection with GPRS or CSD

**enumerator LWGSM\_CMD\_CIFSR**  
Get Local IP Address

**enumerator LWGSM\_CMD\_CIPSTATUS**  
Query Current Connection Status

**enumerator LWGSM\_CMD\_CDNSCFG**  
Configure Domain Name Server

**enumerator LWGSM\_CMD\_CDNSGIP**  
Query the IP Address of Given Domain Name

**enumerator LWGSM\_CMD\_CIPHEAD**  
Add an IP Head at the Beginning of a Package Received

**enumerator LWGSM\_CMD\_CIPATS**  
Set Auto Sending Timer

**enumerator LWGSM\_CMD\_CIPSPRT**  
Set Prompt of greater than sign When Module Sends Data

**enumerator LWGSM\_CMD\_CIPSERVER**  
Configure Module as Server

**enumerator LWGSM\_CMD\_CIPCSGP**  
Set CSD or GPRS for Connection Mode

**enumerator LWGSM\_CMD\_CIPSRIP**  
Show Remote IP Address and Port When Received Data

**enumerator LWGSM\_CMD\_CIPDPDP**  
Set Whether to Check State of GPRS Network Timing

**enumerator LWGSM\_CMD\_CIPMODE**  
Select TCPIP Application Mode

**enumerator LWGSM\_CMD\_CIPCCFG**  
Configure Transparent Transfer Mode

**enumerator LWGSM\_CMD\_CIPSHOWTP**  
Display Transfer Protocol in IP Head When Received Data

**enumerator LWGSM\_CMD\_CIPUDPMODE**  
UDP Extended Mode

**enumerator LWGSM\_CMD\_CIPRXGET**  
Get Data from Network Manually

**enumerator LWGSM\_CMD\_CIPSCONT**  
Save TCPIP Application Context

**enumerator LWGSM\_CMD\_CIPRDTIMER**  
Set Remote Delay Timer

**enumerator LWGSM\_CMD\_CIPSGTXT**  
Select GPRS PDP context

**enumerator LWGSM\_CMD\_CIPTKA**  
Set TCP Keepalive Parameters

**enumerator LWGSM\_CMD\_CIPSSL**  
Connection SSL function

**enumerator LWGSM\_CMD\_SMS\_ENABLE**

**enumerator LWGSM\_CMD\_CMGD**  
Delete SMS Message

**enumerator LWGSM\_CMD\_CMGF**  
Select SMS Message Format

**enumerator LWGSM\_CMD\_CMGL**  
List SMS Messages from Preferred Store

**enumerator LWGSM\_CMD\_CMGR**  
Read SMS Message

**enumerator LWGSM\_CMD\_CMGS**  
Send SMS Message

**enumerator LWGSM\_CMD\_CMGW**  
Write SMS Message to Memory

**enumerator LWGSM\_CMD\_CMSS**  
Send SMS Message from Storage

**enumerator LWGSM\_CMD\_CMGDA**  
MASS SMS delete

**enumerator LWGSM\_CMD\_CNMI**  
New SMS Message Indications

**enumerator LWGSM\_CMD\_CPMS\_SET**  
Set preferred SMS Message Storage

**enumerator LWGSM\_CMD\_CPMS\_GET**  
Get preferred SMS Message Storage

**enumerator LWGSM\_CMD\_CPMS\_GET\_OPT**  
Get optional SMS message storages

**enumerator LWGSM\_CMD\_CRES**  
Restore SMS Settings

**enumerator LWGSM\_CMD\_CSAS**  
Save SMS Settings

**enumerator LWGSM\_CMD\_CSCA**  
SMS Service Center Address

**enumerator LWGSM\_CMD\_CSCB**  
Select Cell Broadcast SMS Messages

**enumerator LWGSM\_CMD\_CSDH**  
Show SMS Text Mode Parameters

**enumerator LWGSM\_CMD\_CSMP**  
Set SMS Text Mode Parameters

**enumerator LWGSM\_CMD\_CSMS**  
Select Message Service

**enumerator LWGSM\_CMD\_END**  
Last CMD entry

**enum lwgsm\_conn\_connect\_res\_t**  
Connection result on connect command.

*Values:*

**enumerator LWGSM\_CONN\_CONNECT\_UNKNOWN**  
No valid result

**enumerator LWGSM\_CONN\_CONNECT\_OK**  
Connected OK

**enumerator LWGSM\_CONN\_CONNECT\_ERROR**  
Error on connection

**enumerator LWGSM\_CONN\_CONNECT\_ALREADY**  
Already connected

**enum lwgsmr\_t**  
Result enumeration used across application functions.

*Values:*

**enumerator lwgsmOK**  
Function returned OK

**enumerator lwgsmOKIGNOREMORE**  
Function succeeded, should continue as *lwgsmOK* but ignore sending more data. This result is possible on connection data receive callback

**enumerator lwgsmERR**  
Generic error



```

enumerator lwgsmPARERR
    Wrong parameters on function call

enumerator lwgsmERRMEM
    Memory error occurred

enumerator lwgsmTIMEOUT
    Timeout occurred on command

enumerator lwgsmCONT
    There is still some command to be processed in current command

enumerator lwgsmCLOSED
    Connection just closed

enumerator lwgsmINPROG
    Operation is in progress

enumerator lwgsmERRNOTENABLED
    Feature not enabled error

enumerator lwgsmERRNOIP
    Station does not have IP address

enumerator lwgsmERRNOFREECONN
    There is no free connection available to start

enumerator lwgsmERRCONNTIMEOUT
    Timeout received when connection to access point

enumerator lwgsmERRPASS
    Invalid password for access point

enumerator lwgsmERRNOAP
    No access point found with specific SSID and MAC address

enumerator lwgsmERRCONNFAIL
    Connection failed to access point

enumerator lwgsmERRWIFINOTCONNECTED
    Wifi not connected to access point

enumerator lwgsmERRNODEVICE
    Device is not present

enumerator lwgsmERRBLOCKING
    Blocking mode command is not allowed

enum lwgsm_device_model_t
    GSM device model type.

    Values:

enumerator LWGSM_DEVICE_MODEL_END
    End of device model

enumerator LWGSM_DEVICE_MODEL_UNKNOWN
    Unknown device model

enum lwgsm_mem_t
    Available device memories.

    Values:

```

```
enumerator LWGSM_MEM_END
    End of memory list

enumerator LWGSM_MEM_CURRENT
    Use current memory for read/delete operation

enumerator LWGSM_MEM_UNKNOWN
    Unknown memory

enum lwgsm_number_type_t
    GSM number type.

    Values:

    enumerator LWGSM_NUMBER_TYPE_NATIONAL
        Number is national

    enumerator LWGSM_NUMBER_TYPE_INTERNATIONAL
        Number is international

struct lwgsm_conn_t
    #include <lwgsm_private.h> Connection structure.
```

### Public Members

```
lwgsm_conn_type_t type
    Connection type

uint8_t num
    Connection number

lwgsm_ip_t remote_ip
    Remote IP address

lwgsm_port_t remote_port
    Remote port number

lwgsm_port_t local_port
    Local IP address

lwgsm_evt_fn evt_func
    Callback function for connection

void *arg
    User custom argument

uint8_t val_id
    Validation ID number. It is increased each time a new connection is established. It protects sending
    data to wrong connection in case we have data in send queue, and connection was closed and active
    again in between.

lwgsm_linbuff_t buff
    Linear buffer structure

size_t total_recved
    Total number of bytes received

uint8_t active
    Status whether connection is active

uint8_t client
    Status whether connection is in client mode
```

```

uint8_t data_received
    Status whether first data were received on connection

uint8_t in_closing
    Status if connection is in closing mode. When in closing mode, ignore any possible received data
    from function

uint8_t bearer
    Bearer used. Can be 1 or 0

struct lwgsm_conn_t::[anonymous]::[anonymous] f
    Connection flags

union lwgsm_conn_t::[anonymous] status
    Connection status union with flag bits

struct lwgsm_pbuf_t
    #include <lwgsm_private.h> Packet buffer structure.

```

### Public Members

```

struct lwgsm_pbuf *next
    Next pbuf in chain list

size_t tot_len
    Total length of pbuf chain

size_t len
    Length of payload

size_t ref
    Number of references to this structure

uint8_t *payload
    Pointer to payload memory

lwgsm_ip_t ip
    Remote address for received IPD data

lwgsm_port_t port
    Remote port for received IPD data

struct lwgsm_ipd_t
    #include <lwgsm_private.h> Incoming network data read structure.

```

### Public Members

```

uint8_t read
    Set to 1 when we should process input data as connection data

size_t tot_len
    Total length of packet

size_t rem_len
    Remaining bytes to read in current +IPD statement

lwgsm_conn_p conn
    Pointer to connection for network data

```

`size_t buff_ptr`

Buffer pointer to save data to. When set to NULL while `read = 1`, reading should ignore incoming data

`lwgsm_pbuf_p buff`

Pointer to data buffer used for receiving data

`struct lwgsm_msg_t`

*#include <lwgsm\_private.h>* Message queue structure to share between threads.

## Public Members

`lwgsm_cmd_t cmd_def`

Default message type received from queue

`lwgsm_cmd_t cmd`

Since some commands can have different subcommands, sub command is used here

`uint8_t i`

Variable to indicate order number of subcommands

`lwgsm_sys_sem_t sem`

Semaphore for the message

`uint8_t is_blocking`

Status if command is blocking

`uint32_t block_time`

Maximal blocking time in units of milliseconds. Use 0 to for non-blocking call

`lwgsmr_t res`

Result of message operation

`lwgsmr_t (*fn) (struct lwgsm_msg*)`

Processing callback function to process packet

`uint32_t delay`

Delay to use before sending first reset AT command

`struct lwgsm_msg_t::[anonymous]::[anonymous] reset`

Reset device

`uint32_t baudrate`

Baudrate for AT port

`struct lwgsm_msg_t::[anonymous]::[anonymous] uart`

UART configuration

`uint8_t mode`

Functionality mode

`struct lwgsm_msg_t::[anonymous]::[anonymous] cfun`

Set phone functionality

`const char *pin`

Pin code to write

New pin code

Current pin code

New PIN code

```

struct lwgsm_msg_t::[anonymous]::[anonymous] cpin_enter
    Enter pin code

struct lwgsm_msg_t::[anonymous]::[anonymous] cpin_add
    Add pin code if previously wasn't set

const char *current_pin
    Current pin code

const char *new_pin
    New pin code

struct lwgsm_msg_t::[anonymous]::[anonymous] cpin_change
    Change current pin code

struct lwgsm_msg_t::[anonymous]::[anonymous] cpin_remove
    Remove PIN code

const char *puk
    PUK code

struct lwgsm_msg_t::[anonymous]::[anonymous] cpuk_enter
    Enter PUK and new PIN

size_t cnum_tries
    Number of tries

struct lwgsm_msg_t::[anonymous]::[anonymous] sim_info
    Get information for SIM card

char *str
    Pointer to output string array

size_t len
    Length of output string array including trailing zero memory

struct lwgsm_msg_t::[anonymous]::[anonymous] device_info
    All kind of device info, serial number, model, manufacturer, revision

int16_t *rssi
    Pointer to RSSI variable

struct lwgsm_msg_t::[anonymous]::[anonymous] csq
    Signal strength

uint8_t read
    Flag indicating we can read the COPS actual data

    Read the data flag

lwgsm_operator_t *ops
    Pointer to operators array

size_t ops1
    Length of operators array

size_t opsi
    Current operator index array

size_t *opf
    Pointer to number of operators found

struct lwgsm_msg_t::[anonymous]::[anonymous] cops_scan
    Scan operators

```

*lwgsm\_operator\_curr\_t* **\*curr**  
Pointer to output current operator

**struct** *lwgsm\_msg\_t*::[anonymous]::[anonymous] **cops\_get**  
Get current operator info

*lwgsm\_operator\_mode\_t* **mode**  
COPS mode

*lwgsm\_operator\_format\_t* **format**  
Operator format to print

**const** char **\*name**  
Short or long name, according to format  
Entry name

uint32\_t **num**  
Number in case format is number

**struct** *lwgsm\_msg\_t*::[anonymous]::[anonymous] **cops\_set**  
Set operator settings

*lwgsm\_conn\_t* **\*\*conn**  
Pointer to pointer to save connection used

**const** char **\*host**  
Host to use for connection

*lwgsm\_port\_t* **port**  
Remote port used for connection

*lwgsm\_conn\_type\_t* **type**  
Connection type

void **\*arg**  
Connection custom argument

*lwgsm\_evt\_fn* **evt\_func**  
Callback function to use on connection

uint8\_t **num**  
Connection number used for start

*lwgsm\_conn\_connect\_res\_t* **conn\_res**  
Connection result status

**struct** *lwgsm\_msg\_t*::[anonymous]::[anonymous] **conn\_start**  
Structure for starting new connection

*lwgsm\_conn\_t* **\*conn**  
Pointer to connection to close  
Pointer to connection to send data

uint8\_t **val\_id**  
Connection current validation ID when command was sent to queue

**struct** *lwgsm\_msg\_t*::[anonymous]::[anonymous] **conn\_close**  
Close connection

size\_t **btw**  
Number of remaining bytes to write

```

size_t ptr
    Current write pointer for data

const uint8_t *data
    Data to send

size_t sent
    Number of bytes sent in last packet

size_t sent_all
    Number of bytes sent all together

uint8_t tries
    Number of tries used for last packet

uint8_t wait_send_ok_err
    Set to 1 when we wait for SEND OK or SEND ERROR

const lwgsm_ip_t *remote_ip
    Remote IP address for UDP connection

lwgsm_port_t remote_port
    Remote port address for UDP connection

uint8_t fau
    Free after use flag to free memory after data are sent (or not)

size_t *bw
    Number of bytes written so far

struct lwgsm_msg_t::[anonymous]::[anonymous] conn_send
    Structure to send data on connection

const char *num
    Phone number

    Entry number

const char *text
    SMS content to send

uint8_t format
    SMS format, 0 = PDU, 1 = text

size_t pos
    Set on +CMGS response if command is OK

    SMS position in memory

    Memory position. Set to 0 to use new one or SIZE_T MAX to delete entry

struct lwgsm_msg_t::[anonymous]::[anonymous] sms_send
    Send SMS

lwgsm_mem_t mem
    Memory to read from

    Memory to delete from

    Memory to use for read

    Array of memories

    Memory to use

```

*lwgsm\_sms\_entry\_t* \***entry**  
Pointer to entry to write info

uint8\_t **update**  
Update SMS status after read operation

**struct** *lwgsm\_msg\_t*::[anonymous]::[anonymous] **sms\_read**  
Read single SMS

**struct** *lwgsm\_msg\_t*::[anonymous]::[anonymous] **sms\_delete**  
Delete SMS message

*lwgsm\_sms\_status\_t* **status**  
SMS status to delete  
SMS entries status

**struct** *lwgsm\_msg\_t*::[anonymous]::[anonymous] **sms\_delete\_all**  
Mass delete SMS messages

*lwgsm\_sms\_entry\_t* \***entries**  
Pointer to entries

size\_t **etr**  
Entries to read (array length)  
Number of entries to read

size\_t **ei**  
Current entry index in array  
Current entry index

size\_t \***er**  
Final entries read pointer for user

**struct** *lwgsm\_msg\_t*::[anonymous]::[anonymous] **sms\_list**  
List SMS messages

**struct** *lwgsm\_msg\_t*::[anonymous]::[anonymous] **sms\_memory**  
Set preferred memories

const char \***number**  
Phone number to dial

**struct** *lwgsm\_msg\_t*::[anonymous]::[anonymous] **call\_start**  
Start a new call

*lwgsm\_number\_type\_t* **type**  
Entry phone number type

uint8\_t **del**  
Flag indicates delete

**struct** *lwgsm\_msg\_t*::[anonymous]::[anonymous] **pb\_write**  
Write/Edit/Delete entry

size\_t **start\_index**  
Start index in phonebook to read

*lwgsm\_pb\_entry\_t* \***entries**  
Pointer to entries array

**struct** *lwgsm\_msg\_t*::[anonymous]::[anonymous] **pb\_list**  
List phonebook entries



```

const char *search
    Search string

struct lwgsm_msg_t::[anonymous]::[anonymous] pb_search
    Search phonebook entries

const char *code
    Code to send

char *resp
    Response array

size_t resp_len
    Length of response array

size_t resp_write_ptr
    Write pointer for response

uint8_t quote_det
    Information if quote has been detected

struct lwgsm_msg_t::[anonymous]::[anonymous] ussd
    Execute USSD command

const char *apn
    APN address

const char *user
    APN username

const char *pass
    APN password

struct lwgsm_msg_t::[anonymous]::[anonymous] network_attach
    Settings for network attach

union lwgsm_msg_t::[anonymous] msg
    Group of different possible message contents

struct lwgsm_ip_mac_t
    #include <lwgsm_private.h> IP and MAC structure with netmask and gateway addresses.

```

### Public Members

```

lwgsm_ip_t ip
    IP address

lwgsm_ip_t gw
    Gateway address

lwgsm_ip_t nm
    Netmask address

lwgsm_mac_t mac
    MAC address

struct lwgsm_link_conn_t
    #include <lwgsm_private.h> Link connection active info.

```

### Public Members

**uint8\_t failed**  
Status if connection successful

**uint8\_t num**  
Connection number

**uint8\_t is\_server**  
Status if connection is client or server

*lwgsm\_conn\_type\_t* **type**  
Connection type

*lwgsm\_ip\_t* **remote\_ip**  
Remote IP address

*lwgsm\_port\_t* **remote\_port**  
Remote port

*lwgsm\_port\_t* **local\_port**  
Local port number

**struct lwgsm\_evt\_func\_t**  
*#include <lwgsm\_private.h>* Callback function linked list prototype.

### Public Members

**struct lwgsm\_evt\_func \*next**  
Next function in the list

*lwgsm\_evt\_fn* **fn**  
Function pointer itself

**struct lwgsm\_sms\_mem\_t**  
*#include <lwgsm\_private.h>* SMS memory information.

### Public Members

**uint32\_t mem\_available**  
Bit field of available memories

*lwgsm\_mem\_t* **current**  
Current memory choice

**size\_t total**  
Size of memory in units of entries

**size\_t used**  
Number of used entries

**struct lwgsm\_sms\_t**  
*#include <lwgsm\_private.h>* SMS structure.

**Public Members**

**uint8\_t ready**  
Flag indicating feature ready by device

**uint8\_t enabled**  
Flag indicating feature enabled

*lwgsm\_sms\_mem\_t* **mem**[3]  
3 memory info for operation, receive, sent storage

**struct lwgsm\_pb\_mem\_t**  
*#include <lwgsm\_private.h>* SMS memory information.

**Public Members**

**uint32\_t mem\_available**  
Bit field of available memories

*lwgsm\_mem\_t* **current**  
Current memory choice

**size\_t total**  
Size of memory in units of entries

**size\_t used**  
Number of used entries

**struct lwgsm\_pb\_t**  
*#include <lwgsm\_private.h>* Phonebook structure.

**Public Members**

**uint8\_t ready**  
Flag indicating feature ready by device

**uint8\_t enabled**  
Flag indicating feature enabled

*lwgsm\_pb\_mem\_t* **mem**  
Memory information

**struct lwgsm\_sim\_t**  
*#include <lwgsm\_private.h>* SIM structure.

**Public Members**

*lwgsm\_sim\_state\_t* **state**  
Current SIM status

**struct lwgsm\_network\_t**  
*#include <lwgsm\_private.h>* Network info.

### Public Members

*lwgsm\_network\_reg\_status\_t* **status**

Network registration status

*lwgsm\_operator\_curr\_t* **curr\_operator**

Current operator information

**uint8\_t is\_attached**

Flag indicating device is attached and PDP context is active

*lwgsm\_ip\_t* **ip\_addr**

Device IP address when network PDP context is enabled

**struct lwgsm\_modules\_t**

*#include <lwgsm\_private.h>* GSM modules structure.

### Public Members

char **model\_manufacturer**[20]

Device manufacturer

char **model\_number**[20]

Device model number

char **model\_serial\_number**[20]

Device serial number

char **model\_revision**[20]

Device revision

*lwgsm\_device\_model\_t* **model**

Device model

*lwgsm\_sim\_t* **sim**

SIM data

*lwgsm\_network\_t* **network**

Network status

**int16\_t rssi**

RSSI signal strength. 0 = invalid, -53 ~ -113 = valid

**uint8\_t active\_conns\_cur\_parse\_num**

Current connection number used for parsing

*lwgsm\_conn\_t* **conns**[LWGSM\_CFG\_MAX\_CONNS]

Array of all connection structures

*lwgsm\_ipd\_t* **ipd**

Connection incoming data structure

**uint8\_t conn\_val\_id**

Validation ID increased each time device connects to network

*lwgsm\_sms\_t* **sms**

SMS information

*lwgsm\_pb\_t* **pb**

Phonebook information

```

lwgsm_call_t call
    Call information

struct lwgsm_t
    #include <lwgsm_private.h> GSM global structure.

Public Members

size_t locked_cnt
    Counter how many times (recursive) stack is currently locked

lwgsm_sys_sem_t sem_sync
    Synchronization semaphore between threads

lwgsm_sys_mbox_t mbox_producer
    Producer message queue handle

lwgsm_sys_mbox_t mbox_process
    Consumer message queue handle

lwgsm_sys_thread_t thread_produce
    Producer thread handle

lwgsm_sys_thread_t thread_process
    Processing thread handle

lwgsm_buff_t buff
    Input processing buffer

lwgsm_ll_t ll
    Low level functions

lwgsm_msg_t *msg
    Pointer to current user message being executed

lwgsm_evt_t evt
    Callback processing structure

lwgsm_evt_func_t *evt_func
    Callback function linked list

lwgsm_modules_t m
    All modules. When resetting, reset structure

uint8_t initialized
    Flag indicating GSM library is initialized

uint8_t dev_present
    Flag indicating GSM device is present

struct lwgsm_t::[anonymous]::[anonymous] f
    Flags structure

union lwgsm_t::[anonymous] status
    Status structure

struct lwgsm_dev_mem_map_t
    #include <lwgsm_private.h> Memory mapping structure between string and value in app.

```

### Public Members

*lwgsm\_mem\_t* **mem**

Mem indication

**const** char \***mem\_str**

Memory string

**struct** **lwgsm\_dev\_model\_map\_t**

*#include <lwgsm\_private.h>* Device models map between model and other information.

### Public Members

*lwgsm\_device\_model\_t* **model**

Device model

**const** char \***id\_str**

Model string identification

uint8\_t **is\_2g**

Status if modem is 2G

uint8\_t **is\_lte**

Status if modem is LTE

**struct** **lwgsm\_unicode\_t**

*#include <lwgsm\_private.h>* Unicode support structure.

### Public Members

uint8\_t **ch**[4]

UTF-8 max characters

uint8\_t **t**

Total expected length in UTF-8 sequence

uint8\_t **r**

Remaining bytes in UTF-8 sequence

*lwgsmr\_t* **res**

Current result of processing

**struct** **lwgsm\_ip\_t**

*#include <lwgsm\_typedefs.h>* IP structure.

### Public Members

uint8\_t **ip**[4]

IPv4 address

**struct** **lwgsm\_mac\_t**

*#include <lwgsm\_typedefs.h>* MAC address.

### Public Members

`uint8_t mac[6]`  
MAC address

**struct lwgsm\_datetime\_t**  
*#include <lwgsm\_typedefs.h>* Date and time structure.

### Public Members

`uint8_t date`  
Day in a month, from 1 to up to 31

`uint8_t month`  
Month in a year, from 1 to 12

`uint16_t year`  
Year

`uint8_t day`  
Day in a week, from 1 to 7, 0 = invalid

`uint8_t hours`  
Hours in a day, from 0 to 23

`uint8_t minutes`  
Minutes in a hour, from 0 to 59

`uint8_t seconds`  
Seconds in a minute, from 0 to 59

**struct lwgsm\_linbuff\_t**  
*#include <lwgsm\_typedefs.h>* Linear buffer structure.

### Public Members

`uint8_t *buff`  
Pointer to buffer data array

`size_t len`  
Length of buffer array

`size_t ptr`  
Current buffer pointer

## Unicode

Unicode decoder block. It can decode sequence of *UTF-8* characters, between 1 and 4 bytes long.

---

**Note:** This is simple implementation and does not support string encoding.

---

*group* **LWGSM\_UNICODE**  
Unicode support manager.

## Functions

*lwgsmr\_t* **lwgsml\_unicode\_decode** (*lwgsm\_unicode\_t* \*uni, uint8\_t ch)

Decode single character for unicode (UTF-8 only) format.

### Parameters

- [inout] s: Pointer to unicode decode control structure
- [in] c: UTF-8 character sequence to test for device

### Return Value

- lwgsmOK: Function succeeded, there is a valid UTF-8 sequence
- lwgsmINPROG: Function continues well but expects some more data to finish sequence
- lwgsmERR: Error in UTF-8 sequence

**struct lwgsm\_unicode\_t**

#include <lwgsm\_private.h> Unicode support structure.

### Public Members

uint8\_t **ch**[4]

UTF-8 max characters

uint8\_t **t**

Total expected length in UTF-8 sequence

uint8\_t **r**

Remaining bytes in UTF-8 sequence

*lwgsmr\_t* **res**

Current result of processing

## Unstructured Supplementary Service Data

group **LWGSM\_USSD**

Unstructured Supplementary Service Data.

## Functions

*lwgsmr\_t* **lwgsm\_ussd\_run** (const char \*code, char \*resp, size\_t resp\_len, const *lwgsm\_api\_cmd\_evt\_fn* evt\_fn, void \*const evt\_arg, const uint32\_t blocking)

Run USSD command, such as \*123# to get balance on SIM card.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

### Parameters

- [in] code: Code to run, such as \*123#
- [out] resp: Pointer to array to save response
- [in] resp\_len: Length of array, including string NULL termination



- [in] `evt_fn`: Callback function called when command has finished. Set to NULL when not used
- [in] `evt_arg`: Custom argument for event callback function
- [in] `blocking`: Status whether command should be blocking or not

## Utilities

Utility functions for various cases. These function are used across entire middleware and can also be used by application.

group **LWGSM\_UTILS**

Utilities.

## Defines

**LWGSM\_ASSERT** (*msg, c*)

Assert an input parameter if in valid range.

**Note** Since this is a macro, it may only be used on a functions where return status is of type *lwgsmr\_t* enumeration

### Parameters

- [in] `msg`: message to print to debug if test fails
- [in] `c`: Condition to test

**LWGSM\_MEM\_ALIGN** (*x*)

Align `x` value to specific number of bytes, provided by *LWGSM\_CFG\_MEM\_ALIGNMENT* configuration.

**Return** Input value aligned to specific number of bytes

### Parameters

- [in] `x`: Input value to align

**LWGSM\_MIN** (*x, y*)

Get minimal value between `x` and `y` inputs.

**Return** Minimal value between `x` and `y` parameters

### Parameters

- [in] `x`: First input to test
- [in] `y`: Second input to test

**LWGSM\_MAX** (*x, y*)

Get maximal value between `x` and `y` inputs.

**Return** Maximal value between `x` and `y` parameters

### Parameters

- [in] `x`: First input to test
- [in] `y`: Second input to test

**LWGSMA\_ARRAYSIZE** (*x*)

Get size of statically declared array.

**Return** Number of array elements

**Parameters**

- [*in*] *x*: Input array

**LWGSMA\_UNUSED** (*x*)

Unused argument in a function call.

**Note** Use this on all parameters in a function which are not used to prevent compiler warnings complaining about “unused variables”

**Parameters**

- [*in*] *x*: Variable which is not used

**LWGSMA\_U32** (*x*)

Get input value casted to unsigned 32-bit value.

**Parameters**

- [*in*] *x*: Input value

**LWGSMA\_U16** (*x*)

Get input value casted to unsigned 16-bit value.

**Parameters**

- [*in*] *x*: Input value

**LWGSMA\_U8** (*x*)

Get input value casted to unsigned 8-bit value.

**Parameters**

- [*in*] *x*: Input value

**LWGSMA\_I32** (*x*)

Get input value casted to signed 32-bit value.

**Parameters**

- [*in*] *x*: Input value

**LWGSMA\_I16** (*x*)

Get input value casted to signed 16-bit value.

**Parameters**

- [*in*] *x*: Input value

**LWGSMA\_I8** (*x*)

Get input value casted to signed 8-bit value.

**Parameters**

- [in] x: Input value

**LwGSM\_SZ** (*x*)

Get input value casted to `size_t` value.

**Parameters**

- [in] x: Input value

**lwgsm\_u32\_to\_str** (*num, out*)

Convert unsigned 32-bit number to string.

**Return** Pointer to output variable

**Parameters**

- [in] num: Number to convert
- [out] out: Output variable to save string

**lwgsm\_u32\_to\_hex\_str** (*num, out, w*)

Convert unsigned 32-bit number to HEX string.

**Return** Pointer to output variable

**Parameters**

- [in] num: Number to convert
- [out] out: Output variable to save string
- [in] w: Width of output string. When number is shorter than width, leading 0 characters will apply.

**lwgsm\_i32\_to\_str** (*num, out*)

Convert signed 32-bit number to string.

**Return** Pointer to output variable

**Parameters**

- [in] num: Number to convert
- [out] out: Output variable to save string

**lwgsm\_u16\_to\_str** (*num, out*)

Convert unsigned 16-bit number to string.

**Return** Pointer to output variable

**Parameters**

- [in] num: Number to convert
- [out] out: Output variable to save string

**lwgsm\_u16\_to\_hex\_str** (*num, out, w*)

Convert unsigned 16-bit number to HEX string.

**Return** Pointer to output variable

**Parameters**

- [in] num: Number to convert
- [out] out: Output variable to save string
- [in] w: Width of output string. When number is shorter than width, leading 0 characters will apply.

**lwgsm\_i16\_to\_str**(*num, out*)

Convert signed 16-bit number to string.

**Return** Pointer to output variable

**Parameters**

- [in] num: Number to convert
- [out] out: Output variable to save string

**lwgsm\_u8\_to\_str**(*num, out*)

Convert unsigned 8-bit number to string.

**Return** Pointer to output variable

**Parameters**

- [in] num: Number to convert
- [out] out: Output variable to save string

**lwgsm\_u8\_to\_hex\_str**(*num, out, w*)

Convert unsigned 16-bit number to HEX string.

**Return** Pointer to output variable

**Parameters**

- [in] num: Number to convert
- [out] out: Output variable to save string
- [in] w: Width of output string. When number is shorter than width, leading 0 characters will apply.

**lwgsm\_i8\_to\_str**(*num, out*)

Convert signed 8-bit number to string.

**Return** Pointer to output variable

**Parameters**

- [in] num: Number to convert
- [out] out: Output variable to save string

## Functions

char **\*lwgsm\_u32\_to\_gen\_str** (uint32\_t *num*, char *\*out*, uint8\_t *is\_hex*, uint8\_t *padding*)  
Convert unsigned 32-bit number to string.

**Return** Pointer to output variable

### Parameters

- [in] *num*: Number to convert
- [out] *out*: Output variable to save string
- [in] *is\_hex*: Set to 1 to output hex, 0 otherwise
- [in] *width*: Width of output string. When number is shorter than width, leading 0 characters will apply. This parameter is valid only when formatting hex numbers

char **\*lwgsm\_i32\_to\_gen\_str** (int32\_t *num*, char *\*out*)  
Convert signed 32-bit number to string.

**Return** Pointer to output variable

### Parameters

- [in] *num*: Number to convert
- [out] *out*: Output variable to save string

group **LWGSM**

Lightweight GSM-AT parser library.

## Functions

*lwgsmr\_t* **lwgsm\_init** (*lwgsm\_evt\_fn* *evt\_func*, **const** uint32\_t *blocking*)  
Init and prepare GSM stack for device operation.

**Note** Function must be called from operating system thread context. It creates necessary threads and waits them to start, thus running operating system is important.

- When *LWGSM\_CFG\_RESET\_ON\_INIT* is enabled, reset sequence will be sent to device otherwise manual call to *lwgsm\_reset* is required to setup device

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

### Parameters

- [in] *evt\_func*: Global event callback function for all major events
- [in] *blocking*: Status whether command should be blocking or not. Used when *LWGSM\_CFG\_RESET\_ON\_INIT* is enabled.

*lwgsmr\_t* **lwgsm\_reset** (**const** *lwgsm\_api\_cmd\_evt\_fn* *evt\_fn*, void *\*const* *evt\_arg*, **const** uint32\_t *blocking*)  
Execute reset and send default commands.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

### Parameters

- [in] `evt_fn`: Callback function called when command is finished. Set to `NULL` when not used
- [in] `evt_arg`: Custom argument for event callback function
- [in] `blocking`: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsm\_reset\_with\_delay** (uint32\_t *delay*, const *lwgsm\_api\_cmd\_evt\_fn* *evt\_fn*, void \*const *evt\_arg*, const uint32\_t *blocking*)

Execute reset and send default commands with delay.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] `delay`: Number of milliseconds to wait before initiating first command to device
- [in] `evt_fn`: Callback function called when command is finished. Set to `NULL` when not used
- [in] `evt_arg`: Custom argument for event callback function
- [in] `blocking`: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsm\_set\_func\_mode** (uint8\_t *mode*, const *lwgsm\_api\_cmd\_evt\_fn* *evt\_fn*, void \*const *evt\_arg*, const uint32\_t *blocking*)

Set modem function mode.

**Note** Use this function to set modem to normal or low-power mode

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] `mode`: Mode status. Set to 1 for full functionality or 0 for low-power mode (no functionality)
- [in] `evt_fn`: Callback function called when command is finished. Set to `NULL` when not used
- [in] `evt_arg`: Custom argument for event callback function
- [in] `blocking`: Status whether command should be blocking or not

*lwgsmr\_t* **lwgsm\_core\_lock** (void)

Lock stack from multi-thread access, enable atomic access to core.

If lock was 0 prior function call, lock is enabled and increased

**Note** Function may be called multiple times to increase locks. Application must take care to call *lwgsm\_core\_unlock* the same amount of time to make sure lock gets back to 0

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

*lwgsmr\_t* **lwgsm\_core\_unlock** (void)

Unlock stack for multi-thread access.

Used in conjunction with *lwgsm\_core\_lock* function

If lock was non-zero before function call, lock is decreased. When `lock == 0`, protection is disabled and other threads may access to core

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

*lwgsmr\_t* **lwgsmd\_device\_set\_present** (uint8\_t *present*, const *lwgsmd\_api\_cmd\_evt\_fn* *evt\_fn*, void \*const *evt\_arg*, const uint32\_t *blocking*)

Notify stack if device is present or not.

Use this function to notify stack that device is not physically connected and not ready to communicate with host device

**Return** *lwgsmdOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] *present*: Flag indicating device is present
- [in] *evt\_fn*: Callback function called when command is finished. Set to NULL when not used
- [in] *evt\_arg*: Custom argument for event callback function
- [in] *blocking*: Status whether command should be blocking or not

uint8\_t **lwgsmd\_device\_is\_present** (void)

Check if device is present.

**Return** 1 on success, 0 otherwise

uint8\_t **lwgsmd\_delay** (uint32\_t *ms*)

Delay for amount of milliseconds.

Delay is based on operating system semaphores. It locks semaphore and waits for timeout in *ms* time. Based on operating system, thread may be put to *blocked* list during delay and may improve execution speed

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] *ms*: Milliseconds to delay

## 5.3.2 Configuration

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

---

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

---

group **LWGSMD\_OPT**  
GSM-AT options.

## Defines

### **LWGSMS\_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

### **LWGSMS\_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 *lwgsms\_mem\_malloc*, *lwgsms\_mem\_calloc*, *lwgsms\_mem\_realloc* and *lwgsms\_mem\_free*

**Note** Function declaration follows standard C functions `malloc`, `calloc`, `realloc`, `free`. Declaration is available in `lwgsms/lwgsms_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

### **LWGSMS\_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

### **LWGSMS\_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`.

### **LWGSMS\_CFG\_MAX\_CONNS**

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

### **LWGSMS\_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

### **LWGSMS\_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.

### **LWGSMS\_CFG\_IPD\_MAX\_BUFF\_SIZE**

Maximum single buffer size for network receive data (TCP/UDP connections)

**Note** When GSM sends buffer bigger than maximal, multiple buffers are created



**LWGSM\_CFG\_AT\_PORT\_BAUDRATE**

Default baudrate used for AT port.

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

**LWGSM\_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 *LWGSM\_CFG\_INPUT\_USE\_PROCESS* is enabled

**LWGSM\_CFG\_RESET\_ON\_INIT**

Enables 1 or disables 0 reset sequence after *lwgsm\_init* call.

**Note** When this functionality is disabled, user must manually call *lwgsm\_reset* to send reset sequence to GSM device.

**LWGSM\_CFG\_RESET\_ON\_DEVICE\_PRESENT**

Enables 1 or disables 0 reset sequence after *lwgsm\_device\_set\_present* call.

**Note** When this functionality is disabled, user must manually call *lwgsm\_reset* to send reset sequence to GSM device.

**LWGSM\_CFG\_RESET\_DELAY\_DEFAULT**

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

**LWGSM\_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

*group* **LWGSM\_OPT\_DBG**

Debugging configurations.

**Defines****LWGSM\_CFG\_DBG**

Set global debug support.

Possible values are *LWGSM\_DBG\_ON* or *LWGSM\_DBG\_OFF*

**Note** Set to *LWGSM\_DBG\_OFF* to globally disable all debugs

**LWGSM\_CFG\_DBG\_OUT** (*fmt, ...*)

Debugging output function.

Called with format and optional parameters for printf style debug

**LWGSM\_CFG\_DBG\_LVL\_MIN**

Minimal debug level.

Check LWGSM\_DBG\_LVL for possible values

**LWGSM\_CFG\_DBG\_TYPES\_ON**

Enabled debug types.

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

Check LWGSM\_DBG\_TYPE for possible options. Separate values with bitwise OR operator

**LWGSM\_CFG\_DBG\_INIT**

Set debug level for init function.

Possible values are *LWGSM\_DBG\_ON* or *LWGSM\_DBG\_OFF*

**LWGSM\_CFG\_DBG\_MEM**

Set debug level for memory manager.

Possible values are *LWGSM\_DBG\_ON* or *LWGSM\_DBG\_OFF*

**LWGSM\_CFG\_DBG\_INPUT**

Set debug level for input module.

Possible values are *LWGSM\_DBG\_ON* or *LWGSM\_DBG\_OFF*

**LWGSM\_CFG\_DBG\_THREAD**

Set debug level for GSM threads.

Possible values are *LWGSM\_DBG\_ON* or *LWGSM\_DBG\_OFF*

**LWGSM\_CFG\_DBG\_ASSERT**

Set debug level for asserting of input variables.

Possible values are *LWGSM\_DBG\_ON* or *LWGSM\_DBG\_OFF*

**LWGSM\_CFG\_DBG\_IPD**

Set debug level for incoming data received from device.

Possible values are *LWGSM\_DBG\_ON* or *LWGSM\_DBG\_OFF*

**LWGSM\_CFG\_DBG\_PBUF**

Set debug level for packet buffer manager.

Possible values are *LWGSM\_DBG\_ON* or *LWGSM\_DBG\_OFF*

**LWGSM\_CFG\_DBG\_CONN**

Set debug level for connections.

Possible values are *LWGSM\_DBG\_ON* or *LWGSM\_DBG\_OFF*

**LWGSM\_CFG\_DBG\_VAR**

Set debug level for dynamic variable allocations.

Possible values are *LWGSM\_DBG\_ON* or *LWGSM\_DBG\_OFF*

**LWGSM\_CFG\_DBG\_NETCONN**

Set debug level for netconn sequential API.

Possible values are *LWGSM\_DBG\_ON* or *LWGSM\_DBG\_OFF*

**LWGSM\_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

*group* **LWGSM\_OPT\_OS**

Operating system dependant configuration.

## Defines

### **LWGSM\_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

### **LWGSM\_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

### **LWGSM\_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 *LWGSM\_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

### **LWGSM\_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.

### **LWGSM\_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.

*group* **LWGSM\_OPT\_STD\_LIB**

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

### **LWGSM\_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);
```

**Return** Destination memory start address

**Parameters**

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

**LWGSM\_MEMSET** (*dst, b, len*)

Memory set function declaration.

Function prototype must be similar to:

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

**Return** Destination memory start address

**Parameters**

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

*group* **LWGSM\_OPT\_MODULES**

Configuration of specific modules.

**Defines****LWGSM\_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.

**LWGSM\_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

**LWGSM\_CFG\_CONN**

Enables 1 or disables 0 connection API.

**Note** *LWGSM\_CFG\_NETWORK* must be enabled to use connection feature

**LWGSM\_CFG\_SMS**

Enables 1 or disables 0 SMS API.

**LWGSM\_CFG\_CALL**

Enables 1 or disables 0 call API.

**LWGSM\_CFG\_PHONEBOOK**

Enables 1 or disables 0 phonebook API.

**LWGSM\_CFG\_HTTP**

Enables 1 or disables 0 HTTP API.

**Note** *LWGSM\_CFG\_NETWORK* must be enabled to use connection feature

**LWGSM\_CFG\_FTP**

Enables 1 or disables 0 FTP API.

**Note** *LWGSM\_CFG\_NETWORK* must be enabled to use connection feature

**LWGSM\_CFG\_PING**

Enables 1 or disables 0 PING API.

**Note** *LWGSM\_CFG\_NETWORK* must be enabled to use connection feature

**LWGSM\_CFG\_USSD**

Enables 1 or disables 0 USSD API.

*group* **LWGSM\_OPT\_MODULES\_NETCONN**

Configuration of netconn API module.

**Defines****LWGSM\_CFG\_NETCONN**

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

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

**See** *LWGSM\_CFG\_OS*

**LWGSM\_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.

**LWGSM\_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

**LWGSM\_CFG\_NETCONN\_RECEIVE\_QUEUE\_LEN**

Receive queue length for pbuf entries.

Defines maximal number of pbuf data packet references for receive

*group* **LWGSM\_OPT\_MODULES\_MQTT**

Configuration of MQTT and MQTT API client modules.

## Defines

### **LWGSM\_CFG\_MQTT\_MAX\_REQUESTS**

Maximal number of open MQTT requests at a time.

### **LWGSM\_CFG\_DBG\_MQTT**

Set debug level for MQTT client module.

Possible values are *LWGSM\_DBG\_ON* or *LWGSM\_DBG\_OFF*

### **LWGSM\_CFG\_DBG\_MQTT\_API**

Set debug level for MQTT API client module.

Possible values are *LWGSM\_DBG\_ON* or *LWGSM\_DBG\_OFF*

## 5.3.3 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* **LWGSM\_LL**

Low-level communication functions.

## Typedefs

**typedef** size\_t (\*lwgsm\_ll\_send\_fn) (const void \*data, size\_t len)

Function prototype for AT output data.

**Return** Number of bytes sent

#### **Parameters**

- [in] data: Pointer to data to send. This parameter can be set to NULL
- [in] len: 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

**typedef** uint8\_t (\*lwgsm\_ll\_reset\_fn) (uint8\_t state)

Function prototype for hardware reset of GSM device.

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

#### **Parameters**

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

## Functions

*lwgsmr\_t* **lwgsml1\_init** (*lwgsml1\_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 *LWGSML\_CFG\_INPUT\_USE\_PROCESS* is set to 1, this function may be called from user UART thread.

**Return** lwgsmlOK on success, member of *lwgsmr\_t* enumeration otherwise

### Parameters

- [inout] ll: Pointer to *lwgsml1\_t* structure to fill data for communication functions

*lwgsmr\_t* **lwgsml1\_deinit** (*lwgsml1\_t* \*ll)

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

**Return** *lwgsmlOK* on success, member of *lwgsmr\_t* enumeration otherwise

### Parameters

- [inout] ll: Pointer to *lwgsml1\_t* structure to fill data for communication functions

**struct lwgsml1\_t**

#include <lwgsml\_typedefs.h> Low level user specific functions.

## Public Members

*lwgsml1\_send\_fn* **send\_fn**

Callback function to transmit data

*lwgsml1\_reset\_fn* **reset\_fn**

Reset callback function

uint32\_t **baudrate**

UART baudrate value

**struct lwgsml1\_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 **LWGSM\_SYS**

System based function for OS management, timings, etc.

#### Main

`uint8_t lwgsm_sys_init (void)`

Init system dependant parameters.

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

**Return** 1 on success, 0 otherwise

`uint32_t lwgsm_sys_now (void)`

Get current time in units of milliseconds.

**Return** Current time in units of milliseconds

`uint8_t lwgsm_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.

**Return** 1 on success, 0 otherwise

`uint8_t lwgsm_sys_unprotect (void)`

Unprotect middleware core.

This function must follow number of calls of *lwgsm\_sys\_protect* and unlock access only when counter reached back zero.

**Note** Most operating systems support recursive mutexes.

**Return** 1 on success, 0 otherwise

#### Mutex

`uint8_t lwgsm_sys_mutex_create (lwgsm_sys_mutex_t *p)`

Create new recursive mutex.

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

**Return** 1 on success, 0 otherwise

##### Parameters

- [out] p: Pointer to mutex structure to allocate

`uint8_t lwgsm_sys_mutex_delete (lwgsm_sys_mutex_t *p)`

Delete recursive mutex from system.



**Return** 1 on success, 0 otherwise

**Parameters**

- [in] p: Pointer to mutex structure

uint8\_t **lwgsm\_sys\_mutex\_lock** (*lwgsm\_sys\_mutex\_t \*p*)

Lock recursive mutex, wait forever to lock.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] p: Pointer to mutex structure

uint8\_t **lwgsm\_sys\_mutex\_unlock** (*lwgsm\_sys\_mutex\_t \*p*)

Unlock recursive mutex.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] p: Pointer to mutex structure

uint8\_t **lwgsm\_sys\_mutex\_isvalid** (*lwgsm\_sys\_mutex\_t \*p*)

Check if mutex structure is valid system.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] p: Pointer to mutex structure

uint8\_t **lwgsm\_sys\_mutex\_invalid** (*lwgsm\_sys\_mutex\_t \*p*)

Set recursive mutex structure as invalid.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] p: Pointer to mutex structure

## Semaphores

uint8\_t **lwgsm\_sys\_sem\_create** (*lwgsm\_sys\_sem\_t \*p*, uint8\_t *cnt*)

Create a new binary semaphore and set initial state.

**Note** Semaphore may only have 1 token available

**Return** 1 on success, 0 otherwise

**Parameters**

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

uint8\_t **lwgsm\_sys\_sem\_delete** (*lwgsm\_sys\_sem\_t \*p*)

Delete binary semaphore.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] p: Pointer to semaphore structure

uint32\_t **lwgsm\_sys\_sem\_wait** (*lwgsm\_sys\_sem\_t* \*p, uint32\_t timeout)

Wait for semaphore to be available.

**Return** Number of milliseconds waited for semaphore to become available or *LWGSM\_SYS\_TIMEOUT* if not available within given time

**Parameters**

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

uint8\_t **lwgsm\_sys\_sem\_release** (*lwgsm\_sys\_sem\_t* \*p)

Release semaphore.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] p: Pointer to semaphore structure

uint8\_t **lwgsm\_sys\_sem\_isvalid** (*lwgsm\_sys\_sem\_t* \*p)

Check if semaphore is valid.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] p: Pointer to semaphore structure

uint8\_t **lwgsm\_sys\_sem\_invalid** (*lwgsm\_sys\_sem\_t* \*p)

Invalid semaphore.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] p: Pointer to semaphore structure

## Message queues

uint8\_t **lwgsm\_sys\_mbox\_create** (*lwgsm\_sys\_mbox\_t* \*b, size\_t size)

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

**Return** 1 on success, 0 otherwise

**Parameters**

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

uint8\_t **lwgsm\_sys\_mbox\_delete** (*lwgsm\_sys\_mbox\_t* \*b)

Delete message queue.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] b: Pointer to message queue structure

uint32\_t **lwgsm\_sys\_mbox\_put** (*lwgsm\_sys\_mbox\_t* \*b, void \*m)

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

**Return** Time in units of milliseconds needed to put a message to queue

**Parameters**

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

uint32\_t **lwgsm\_sys\_mbox\_get** (*lwgsm\_sys\_mbox\_t* \*b, void \*\*m, uint32\_t timeout)

Get a new entry from message queue with timeout.

**Return** Time in units of milliseconds needed to put a message to queue or *LWGSM\_SYS\_TIMEOUT* if it was not successful

**Parameters**

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

uint8\_t **lwgsm\_sys\_mbox\_putnow** (*lwgsm\_sys\_mbox\_t* \*b, void \*m)

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

**Return** 1 on success, 0 otherwise

**Parameters**

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

uint8\_t **lwgsm\_sys\_mbox\_getnow** (*lwgsm\_sys\_mbox\_t* \*b, void \*\*m)

Get an entry from message queue immediately.

**Return** 1 on success, 0 otherwise

**Parameters**

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

uint8\_t **lwgsm\_sys\_mbox\_isvalid** (*lwgsm\_sys\_mbox\_t* \*b)

Check if message queue is valid.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] b: Pointer to message queue structure

`uint8_t lwgsm_sys_mbox_invalid(lwgsm_sys_mbox_t *b)`  
Invalid message queue.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] b: Pointer to message queue structure

## Threads

`uint8_t lwgsm_sys_thread_create(lwgsm_sys_thread_t *t, const char *name, lwgsm_sys_thread_fn thread_func, void *const arg, size_t stack_size, lwgsm_sys_thread_prio_t prio)`

Create a new thread.

**Return** 1 on success, 0 otherwise

**Parameters**

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

`uint8_t lwgsm_sys_thread_terminate(lwgsm_sys_thread_t *t)`  
Terminate thread (shut it down and remove)

**Return** 1 on success, 0 otherwise

**Parameters**

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

`uint8_t lwgsm_sys_thread_yield(void)`  
Yield current thread.

**Return** 1 on success, 0 otherwise

## Defines

**LWGSMSYS\_MUTEX\_NULL**

Mutex invalid value.

Value assigned to `lwgsmsys_mutex_t` type when it is not valid.

**LWGSMSYS\_SEM\_NULL**

Semaphore invalid value.

Value assigned to `lwgsmsys_sem_t` type when it is not valid.

**LWGSM\_SYS\_MBOX\_NULL**

Message box invalid value.

Value assigned to *lwgsm\_sys\_mbox\_t* type when it is not valid.

**LWGSM\_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

**LWGSM\_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 prior operation.

**LWGSM\_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 (\**lwgsm\_sys\_thread\_fn*) (void\*)

Thread function prototype.

**typedef** osMutexId\_t *lwgsm\_sys\_mutex\_t*

System mutex type.

It is used by middleware as base type of mutex.

**typedef** osSemaphoreId\_t *lwgsm\_sys\_sem\_t*

System semaphore type.

It is used by middleware as base type of mutex.

**typedef** osMessageQueueId\_t *lwgsm\_sys\_mbox\_t*

System message queue type.

It is used by middleware as base type of mutex.

**typedef** osThreadId\_t *lwgsm\_sys\_thread\_t*

System thread ID type.

**typedef** osPriority *lwgsm\_sys\_thread\_prio\_t*

System thread priority type.

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

**5.3.4 Applications****MQTT Client**

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

*group* **LWGSM\_APP\_MQTT\_CLIENT**

MQTT client.

## Typedefs

**typedef struct** lwgsm\_mqtt\_client \*lwgsm\_mqtt\_client\_p

Pointer to lwgsm\_mqtt\_client\_t structure.

**typedef void** (\*lwgsm\_mqtt\_evt\_fn) (lwgsm\_mqtt\_client\_p client, lwgsm\_mqtt\_evt\_t \*evt)

MQTT event callback function.

### Parameters

- [in] client: MQTT client
- [in] evt: MQTT event with type and related data

## Enums

**enum** lwgsm\_mqtt\_qos\_t

Quality of service enumeration.

*Values:*

**enumerator** LWGSM\_MQTT\_QOS\_AT\_MOST\_ONCE

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

**enumerator** LWGSM\_MQTT\_QOS\_AT\_LEAST\_ONCE

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

**enumerator** LWGSM\_MQTT\_QOS\_EXACTLY\_ONCE

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

**enum** lwgsm\_mqtt\_state\_t

State of MQTT client.

*Values:*

**enumerator** LWGSM\_MQTT\_CONN\_DISCONNECTED

Connection with server is not established

**enumerator** LWGSM\_MQTT\_CONN\_CONNECTING

Client is connecting to server

**enumerator** LWGSM\_MQTT\_CONN\_DISCONNECTING

Client connection is disconnecting from server

**enumerator** LWGSM\_MQTT\_CONNECTING

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

**enumerator** LWGSM\_MQTT\_CONNECTED

MQTT is fully connected and ready to send data on topics

**enum** lwgsm\_mqtt\_evt\_type\_t

MQTT event types.

*Values:*

**enumerator** LWGSM\_MQTT\_EVT\_CONNECT

MQTT client connect event

**enumerator** LWGSM\_MQTT\_EVT\_SUBSCRIBE

MQTT client subscribed to specific topic

**enumerator LWGSM\_MQTT\_EVT\_UNSUBSCRIBE**

MQTT client unsubscribed from specific topic

**enumerator LWGSM\_MQTT\_EVT\_PUBLISH**

MQTT client publish message to server event.

**Note** When publishing packet with quality of service *LWGSM\_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 = LWGSM_MQTT_QOS_AT_MOST_ONCE`

**enumerator LWGSM\_MQTT\_EVT\_PUBLISH\_RECV**

MQTT client received a publish message from server

**enumerator LWGSM\_MQTT\_EVT\_DISCONNECT**

MQTT client disconnected from MQTT server

**enumerator LWGSM\_MQTT\_EVT\_KEEP\_ALIVE**

MQTT keep-alive sent to server and reply received

**enum lwgsm\_mqtt\_conn\_status\_t**

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

*Values:*

**enumerator LWGSM\_MQTT\_CONN\_STATUS\_ACCEPTED**

Connection accepted and ready to use

**enumerator LWGSM\_MQTT\_CONN\_STATUS\_REFUSED\_PROTOCOL\_VERSION**

Connection Refused, unacceptable protocol version

**enumerator LWGSM\_MQTT\_CONN\_STATUS\_REFUSED\_ID**

Connection refused, identifier rejected

**enumerator LWGSM\_MQTT\_CONN\_STATUS\_REFUSED\_SERVER**

Connection refused, server unavailable

**enumerator LWGSM\_MQTT\_CONN\_STATUS\_REFUSED\_USER\_PASS**

Connection refused, bad user name or password

**enumerator LWGSM\_MQTT\_CONN\_STATUS\_REFUSED\_NOT\_AUTHORIZED**

Connection refused, not authorized

**enumerator LWGSM\_MQTT\_CONN\_STATUS\_TCP\_FAILED**

TCP connection to server was not successful

## Functions

*lwgsm\_mqtt\_client\_p* **lwgsm\_mqtt\_client\_new** (size\_t tx\_buff\_len, size\_t rx\_buff\_len)

Allocate a new MQTT client structure.

**Return** Pointer to new allocated MQTT client structure or NULL on failure

### Parameters

- [in] tx\_buff\_len: Length of raw data output buffer
- [in] rx\_buff\_len: Length of raw data input buffer

void **lwgsm\_mqtt\_client\_delete** (*lwgsm\_mqtt\_client\_p* client)

Delete MQTT client structure.

**Note** MQTT client must be disconnected first

**Parameters**

- [in] client: MQTT client

*lwgsmr\_t* **lwgsmr\_mqtt\_client\_connect** (*lwgsmr\_mqtt\_client\_p* client, **const** char \*host, *lwgsmr\_port\_t* port, *lwgsmr\_mqtt\_evt\_fn* evt\_fn, **const** *lwgsmr\_mqtt\_client\_info\_t* \*info)

Connect to MQTT server.

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

**Return** *lwgsmrOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] client: MQTT client
- [in] host: Host address for server
- [in] port: Host port number
- [in] evt\_fn: Callback function for all events on this MQTT client
- [in] info: Information structure for connection

*lwgsmr\_t* **lwgsmr\_mqtt\_client\_disconnect** (*lwgsmr\_mqtt\_client\_p* client)

Disconnect from MQTT server.

**Return** *lwgsmrOK* if request sent to queue or member of *lwgsmr\_t* otherwise

**Parameters**

- [in] client: MQTT client

*uint8\_t* **lwgsmr\_mqtt\_client\_is\_connected** (*lwgsmr\_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

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] client: MQTT client

*lwgsmr\_t* **lwgsmr\_mqtt\_client\_subscribe** (*lwgsmr\_mqtt\_client\_p* client, **const** char \*topic, *lwgsmr\_mqtt\_qos\_t* qos, void \*arg)

Subscribe to MQTT topic.

**Return** *lwgsmrOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] client: MQTT client
- [in] topic: Topic name to subscribe to
- [in] qos: Quality of service. This parameter can be a value of *lwgsmr\_mqtt\_qos\_t*
- [in] arg: User custom argument used in callback



*lwgsmr\_t* **lwgsm\_mqtt\_client\_unsubscribe** (*lwgsm\_mqtt\_client\_p* client, const char \*topic, void \*arg)

Unsubscribe from MQTT topic.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

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

*lwgsmr\_t* **lwgsm\_mqtt\_client\_publish** (*lwgsm\_mqtt\_client\_p* client, const char \*topic, const void \*payload, uint16\_t len, *lwgsm\_mqtt\_qos\_t* qos, uint8\_t retain, void \*arg)

Publish a new message on specific topic.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] client: MQTT client
- [in] topic: Topic to send message to
- [in] payload: Message data
- [in] payload\_len: Length of payload data
- [in] qos: Quality of service. This parameter can be a value of *lwgsm\_mqtt\_qos\_t* enumeration
- [in] retain: Retian parameter value
- [in] arg: User custom argument used in callback

void **lwgsm\_mqtt\_client\_get\_arg** (*lwgsm\_mqtt\_client\_p* client)

Get user argument on client.

**Return** User argument

**Parameters**

- [in] client: MQTT client handle

void **lwgsm\_mqtt\_client\_set\_arg** (*lwgsm\_mqtt\_client\_p* client, void \*arg)

Set user argument on client.

**Parameters**

- [in] client: MQTT client handle
- [in] arg: User argument

**struct lwgsm\_mqtt\_client\_info\_t**

#include <lwgsm\_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

*lwgsm\_mqtt\_qos\_t* **will\_qos**  
Will topic quality of service

**struct lwgsm\_mqtt\_request\_t**  
*#include <lwgsm\_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 lwgsm\_mqtt\_evt\_t**  
*#include <lwgsm\_mqtt\_client.h>* MQTT event structure for callback function.

### Public Members

*lwgsm\_mqtt\_evt\_type\_t* **type**  
Event type

*lwgsm\_mqtt\_conn\_status\_t* **status**  
Connection status with MQTT

**struct lwgsm\_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 lwgsm_mqtt_evt_t::[anonymous]::[anonymous] disconnect
    Event for disconnecting from server

void *arg
    User argument for callback function

lwgsmr_t res
    Response status

struct lwgsm_mqtt_evt_t::[anonymous]::[anonymous] sub_unsub_scribed
    Event for (un)subscribe to/from topics

struct lwgsm_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

lwgsm_mqtt_qos_t qos
    Received packet quality of service

struct lwgsm_mqtt_evt_t::[anonymous]::[anonymous] publish_recv
    Publish received event

union lwgsm_mqtt_evt_t::[anonymous] evt
    Event data parameters

group LWGSM_APP_MQTT_CLIENT_EVT
    Event helper functions.

```

### Connect event

**Note** Use these functions on *LWGSM\_MQTT\_EVT\_CONNECT* event

**lwgsm\_mqtt\_client\_evt\_connect\_get\_status** (*client*, *evt*)  
Get connection status.

**Return** Connection status. Member of *lwgsm\_mqtt\_conn\_status\_t*

#### Parameters

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

### Disconnect event

**Note** Use these functions on *LWGSM\_MQTT\_EVT\_DISCONNECT* event

**lwgsm\_mqtt\_client\_evt\_disconnect\_is\_accepted** (*client*, *evt*)

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

**Return** 1 on success, 0 otherwise

**Parameters**

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

### Subscribe/unsubscribe event

**Note** Use these functions on *LWGSM\_MQTT\_EVT\_SUBSCRIBE* or *LWGSM\_MQTT\_EVT\_UNSUBSCRIBE* events

**lwgsm\_mqtt\_client\_evt\_subscribe\_get\_argument** (*client*, *evt*)

Get user argument used on *lwgsm\_mqtt\_client\_subscribe*.

**Return** User argument

**Parameters**

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

**lwgsm\_mqtt\_client\_evt\_subscribe\_get\_result** (*client*, *evt*)

Get result of subscribe event.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* otherwise

**Parameters**

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

**lwgsm\_mqtt\_client\_evt\_unsubscribe\_get\_argument** (*client*, *evt*)

Get user argument used on *lwgsm\_mqtt\_client\_unsubscribe*.

**Return** User argument

**Parameters**

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

**lwgsm\_mqtt\_client\_evt\_unsubscribe\_get\_result** (*client*, *evt*)

Get result of unsubscribe event.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* otherwise

**Parameters**

- [in] `client`: MQTT client
- [in] `evt`: Event handle

### Publish receive event

**Note** Use these functions on *LWGSM\_MQTT\_EVT\_PUBLISH\_RECV* event

**lwgsm\_mqtt\_client\_evt\_publish\_recv\_get\_topic** (*client*, *evt*)  
Get topic from received publish packet.

**Return** Topic name

**Parameters**

- [in] `client`: MQTT client
- [in] `evt`: Event handle

**lwgsm\_mqtt\_client\_evt\_publish\_recv\_get\_topic\_len** (*client*, *evt*)  
Get topic length from received publish packet.

**Return** Topic length

**Parameters**

- [in] `client`: MQTT client
- [in] `evt`: Event handle

**lwgsm\_mqtt\_client\_evt\_publish\_recv\_get\_payload** (*client*, *evt*)  
Get payload from received publish packet.

**Return** Packet payload

**Parameters**

- [in] `client`: MQTT client
- [in] `evt`: Event handle

**lwgsm\_mqtt\_client\_evt\_publish\_recv\_get\_payload\_len** (*client*, *evt*)  
Get payload length from received publish packet.

**Return** Payload length

**Parameters**

- [in] `client`: MQTT client
- [in] `evt`: Event handle

**lwgsm\_mqtt\_client\_evt\_publish\_recv\_is\_duplicate** (*client*, *evt*)  
Check if packet is duplicated.

**Return** 1 if duplicated, 0 otherwise

**Parameters**

- [in] `client`: MQTT client

- [in] evt: Event handle

**lwgsm\_mqtt\_client\_evt\_publish\_rcv\_get\_qos** (*client*, *evt*)

Get received quality of service.

**Return** Member of *lwgsm\_mqtt\_qos\_t* enumeration

**Parameters**

- [in] client: MQTT client
- [in] evt: Event handle

## Publish event

**Note** Use these functions on *LWGSM\_MQTT\_EVT\_PUBLISH* event

**lwgsm\_mqtt\_client\_evt\_publish\_get\_argument** (*client*, *evt*)

Get user argument used on *lwgsm\_mqtt\_client\_publish*.

**Return** User argument

**Parameters**

- [in] client: MQTT client
- [in] evt: Event handle

**lwgsm\_mqtt\_client\_evt\_publish\_get\_result** (*client*, *evt*)

Get result of publish event.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* otherwise

**Parameters**

- [in] client: MQTT client
- [in] evt: Event handle

## Defines

**lwgsm\_mqtt\_client\_evt\_get\_type** (*client*, *evt*)

Get MQTT event type.

**Return** MQTT Event type, value of *lwgsm\_mqtt\_evt\_type\_t* enumeration

**Parameters**

- [in] client: MQTT client
- [in] evt: Event handle

## MQTT Client API

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

Listing 20: 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 "lwgsm_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 "lwgsm/apps/lwgsm_mqtt_client_api.h"
14  #include "mqtt_client_api.h"
15  #include "lwgsm/lwgsm_mem.h"
16  #include "lwgsm/lwgsm_network_api.h"
17
18  /**
19   * \brief      Connection information for MQTT CONNECT packet
20   */
21  static const lwgsm_mqtt_client_info_t
22  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
37  mqtt_topic_str[256];
38
39  /**
40   * \brief      Generate random number and write it to string
41   * \param[out] str: Output string with new number
42   */
43  void
44  generate_random(char* str) {
45      static uint32_t random_beg = 0x8916;
46      random_beg = random_beg * 0x00123455 + 0x85654321;
47      sprintf(str, "%u", (unsigned)((random_beg >> 8) & 0xFFFF));
48  }
49
50  /**
51   * \brief      MQTT client API thread

```

(continues on next page)

(continued from previous page)

```

52  */
53  void
54  mqtt_client_api_thread(void const* arg) {
55      lwgsm_mqtt_client_api_p client;
56      lwgsm_mqtt_conn_status_t conn_status;
57      lwgsm_mqtt_client_api_buf_p buf;
58      lwgsmr_t res;
59      char random_str[10];
60
61      /* Request network attach */
62      while (lwgsm_network_request_attach() != lwgsmOK) {
63          lwgsm_delay(1000);
64      }
65
66      /* Create new MQTT API */
67      client = lwgsm_mqtt_client_api_new(256, 128);
68      if (client == 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 = lwgsm_mqtt_client_api_connect(client, "mqtt.mydevices.com", ↵
↵1883, &mqtt_client_info);
78          if (conn_status == LWGSM_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              lwgsm_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 (lwgsm_mqtt_client_api_subscribe(client, mqtt_topic_str, LWGSM_MQTT_QOS_AT_
↵LEAST_ONCE) == lwgsmOK) {
90              printf("Subscribed to topic\r\n");
91          } else {
92              printf("Problem subscribing to topic!\r\n");
93          }
94
95          while (1) {
96              /* Receive MQTT packet with 1000ms timeout */
97              res = lwgsm_mqtt_client_api_receive(client, &buf, 5000);
98              if (res == lwgsmOK) {
99                  if (buf != NULL) {
100                      printf("Publish received!\r\n");
101                      printf("Topic: %s, payload: %s\r\n", buf->topic, buf->payload);
102                      lwgsm_mqtt_client_api_buf_free(buf);
103                      buf = NULL;
104                  }
105              } else if (res == lwgsmCLOSED) {

```

(continues on next page)



(continued from previous page)

```

106         printf("MQTT connection closed!\r\n");
107         break;
108     } else if (res == lwgsmTIMEOUT) {
109         printf("Timeout on MQTT receive function. Manually publishing.\r\n");
110
111         /* Publish data on channel 1 */
112         generate_random(random_str);
113         sprintf(mqtt_topic_str, "v1/%s/things/%s/data/1", mqtt_client_info.
↪user, mqtt_client_info.id);
114         lwgsm_mqtt_client_api_publish(client, mqtt_topic_str, random_str, ↪
↪strlen(random_str), LWGSM_MQTT_QOS_AT_LEAST_ONCE, 0);
115     }
116 }
117 goto terminate;
118 }
119
120 terminate:
121     lwgsm_mqtt_client_api_delete(client);
122     lwgsm_network_request_detach();
123     printf("MQTT client thread terminate\r\n");
124     lwgsm_sys_thread_terminate(NULL);
125 }

```

group **LWGSM\_APP\_MQTT\_CLIENT\_API**

Sequential, single thread MQTT client API.

## Typedefs

**typedef struct** lwgsm\_mqtt\_client\_api\_buf \*lwgsm\_mqtt\_client\_api\_buf\_p  
 Pointer to *lwgsm\_mqtt\_client\_api\_buf\_t* structure.

## Functions

lwgsm\_mqtt\_client\_api\_p **lwgsm\_mqtt\_client\_api\_new** (size\_t tx\_buff\_len, size\_t rx\_buff\_len)  
 Create new MQTT client API.

**Return** Client handle on success, NULL otherwise

### Parameters

- [in] tx\_buff\_len: Maximal TX buffer for maximal packet length
- [in] rx\_buff\_len: Maximal RX buffer

void **lwgsm\_mqtt\_client\_api\_delete** (lwgsm\_mqtt\_client\_api\_p client)  
 Delete client from memory.

### Parameters

- [in] client: MQTT API client handle

*lwgsm\_mqtt\_conn\_status\_t* **lwgsm\_mqtt\_client\_api\_connect** (*lwgsm\_mqtt\_client\_api\_p* *client*, **const** *char* *\*host*, *lwgsm\_port\_t* *port*, **const** *lwgsm\_mqtt\_client\_info\_t* *\*info*)

Connect to MQTT broker.

**Return** *LWGSM\_MQTT\_CONN\_STATUS\_ACCEPTED* on success, member of *lwgsm\_mqtt\_conn\_status\_t* otherwise

**Parameters**

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

*lwgsmr\_t* **lwgsm\_mqtt\_client\_api\_close** (*lwgsm\_mqtt\_client\_api\_p* *client*)  
Close MQTT connection.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* otherwise

**Parameters**

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

*lwgsmr\_t* **lwgsm\_mqtt\_client\_api\_subscribe** (*lwgsm\_mqtt\_client\_api\_p* *client*, **const** *char* *\*topic*, *lwgsm\_mqtt\_qos\_t* *qos*)  
Subscribe to topic.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* otherwise

**Parameters**

- [in] *client*: MQTT API client handle
- [in] *topic*: Topic to subscribe on
- [in] *qos*: Quality of service. This parameter can be a value of *lwgsm\_mqtt\_qos\_t*

*lwgsmr\_t* **lwgsm\_mqtt\_client\_api\_unsubscribe** (*lwgsm\_mqtt\_client\_api\_p* *client*, **const** *char* *\*topic*)  
Unsubscribe from topic.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* otherwise

**Parameters**

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

*lwgsmr\_t* **lwgsm\_mqtt\_client\_api\_publish** (*lwgsm\_mqtt\_client\_api\_p* *client*, **const** *char* *\*topic*, **const** *void* *\*data*, *size\_t* *btw*, *lwgsm\_mqtt\_qos\_t* *qos*, *uint8\_t* *retain*)  
Publish new packet to MQTT network.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* otherwise

**Parameters**

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

`uint8_t lwgsm_mqtt_client_api_is_connected` (`lwgsm_mqtt_client_api_p client`)

Check if client MQTT connection is active.

**Return** 1 on success, 0 otherwise

**Parameters**

- [in] `client`: MQTT API client handle

*lwgsmr\_t* `lwgsm_mqtt_client_api_receive` (`lwgsm_mqtt_client_api_p client`,  
*lwgsm\_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

**Return** *lwgsmOK* on success, *lwgsmCLOSED* if MQTT is closed, *lwgsmTIMEOUT* on timeout

**Parameters**

- [in] `client`: MQTT API client handle
- [in] `p`: Pointer to output buffer
- [in] `timeout`: Maximal time to wait before function returns timeout

`void lwgsm_mqtt_client_api_buf_free` (`lwgsm_mqtt_client_api_buf_p p`)

Free buffer memory after usage.

**Parameters**

- [in] `p`: Buffer to free

`struct lwgsm_mqtt_client_api_buf_t`

*#include <lwgsm\_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

`lwgsm_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. 8: 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 21: Netconn client example

```
1 #include "netconn_client.h"
2 #include "lwgsm/lwgsm.h"
3 #include "lwgsm/lwgsm_network_api.h"
4
5 #if LWGSM_CFG_NETCONN
6
7 /**
```

(continues on next page)

(continued from previous page)

```

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
17 request_header[] = ""
18     "GET / HTTP/1.1\r\n"
19     "Host: " NETCONN_HOST "\r\n"
20     "Connection: close\r\n"
21     "\r\n";
22
23 /**
24  * \brief      Netconn client thread implementation
25  * \param[in]  arg: User argument
26  */
27 void
28 netconn_client_thread(void const* arg) {
29     lwgsmr_t res;
30     lwgsm_pbuf_p pbuf;
31     lwgsm_netconn_p client;
32     lwgsm_sys_sem_t* sem = (void*)arg;
33
34     /* Request attach to network */
35     while (lwgsm_network_request_attach() != lwgsmOK) {
36         lwgsm_delay(1000);
37     }
38
39     /*
40      * First create a new instance of netconn
41      * connection and initialize system message boxes
42      * to accept received packet buffers
43      */
44     client = lwgsm_netconn_new(LWGSM_NETCONN_TYPE_TCP);
45     if (client != NULL) {
46         /*
47          * Connect to external server as client
48          * with custom NETCONN_HOST and NETCONN_PORT values
49          *
50          * Function will block thread until we are successfully connected (or not) to
51          ↪ server
52          */
53         res = lwgsm_netconn_connect(client, NETCONN_HOST, NETCONN_PORT);
54         if (res == lwgsmOK) { /* Are we successfully connected? */
55             printf("Connected to " NETCONN_HOST "\r\n");
56             res = lwgsm_netconn_write(client, request_header, sizeof(request_header) -
57 ↪ 1); /* Send data to server */
58             if (res == lwgsmOK) {
59                 res = lwgsm_netconn_flush(client); /* Flush data to output */
60             }
61             if (res == lwgsmOK) { /* Were data sent? */
62                 printf("Data were successfully sent to server\r\n");
63             }
64         }
65     }
66 }

```

(continues on next page)

(continued from previous page)

```

63         * Since we sent HTTP request,
64         * we are expecting some data from server
65         * or at least forced connection close from remote side
66         */
67     do {
68         /*
69         * Receive single packet of data
70         *
71         * Function will block thread until new packet
72         * is ready to be read from remote side
73         *
74         * After function returns, don't forgot the check value.
75         * Returned status will give you info in case connection
76         * was closed too early from remote side
77         */
78         res = lwgsm_netconn_receive(client, &pbuf);
79         if (res == lwgsmCLOSED) { /* Was the connection closed? This
↳can be checked by return status of receive function */
80             printf("Connection closed by remote side...\r\n");
81             break;
82         } else if (res == lwgsmTIMEOUT) {
83             printf("Netconn timeout while receiving data. You may try
↳multiple readings before deciding to close manually\r\n");
84         }
85
86         if (res == lwgsmOK && 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)lwgsm_
↳pbuf_length(pbuf, 1));
95             lwgsm_pbuf_free(pbuf); /* Free the memory after usage */
96             pbuf = NULL;
97         }
98     } while (1);
99     } else {
100         printf("Error writing data to remote host!\r\n");
101     }
102
103     /*
104     * Check if connection was closed by remote server
105     * and in case it wasn't, close it manually
106     */
107     if (res != lwgsmCLOSED) {
108         lwgsm_netconn_close(client);
109     }
110     } else {
111         printf("Cannot connect to remote host %s:%d!\r\n", NETCONN_HOST, NETCONN_
↳PORT);
112     }
113     lwgsm_netconn_delete(client); /* Delete netconn structure */
114 }

```

(continues on next page)

(continued from previous page)

```

115     lwgsm_network_request_detach();           /* Detach from network */
116
117     if (lwgsm_sys_sem_isvalid(sem)) {
118         lwgsm_sys_sem_release(sem);
119     }
120     lwgsm_sys_thread_terminate(NULL);         /* Terminate current thread */
121 }
122
123 #endif /* LWGSM_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, `lwgsm_netconn_receive()` will block for maximal timeout set with `lwgsm_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:** `LWGSM_CFG_NETCONN_RECEIVE_TIMEOUT` must be set to 1 to use this feature.

---

*group* **LWGSM\_NETCONN**  
Network connection.

## Typedefs

**typedef struct** lwgsm\_netconn \*lwgsm\_netconn\_p  
Netconn object structure.

## Enums

**enum** lwgsm\_netconn\_type\_t  
Netconn connection type.

*Values:*

**enumerator** LWGSM\_NETCONN\_TYPE\_TCP  
TCP connection

**enumerator** LWGSM\_NETCONN\_TYPE\_UDP  
UDP connection

**enumerator** LWGSM\_NETCONN\_TYPE\_SSL  
TCP connection over SSL

## Functions

*lwgsm\_netconn\_p* **lwgsm\_netconn\_new** (*lwgsm\_netconn\_type\_t* type)

Create new netconn connection.

**Return** New netconn connection on success, NULL otherwise

**Parameters**

- [in] type: Netconn connection type

*lwgsmr\_t* **lwgsm\_netconn\_delete** (*lwgsm\_netconn\_p* nc)

Delete netconn connection.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] nc: Netconn handle

*lwgsmr\_t* **lwgsm\_netconn\_connect** (*lwgsm\_netconn\_p* nc, const char \*host, *lwgsm\_port\_t* port)

Connect to server as client.

**Return** *lwgsmOK* if successfully connected, member of *lwgsmr\_t* otherwise

**Parameters**

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

*lwgsmr\_t* **lwgsm\_netconn\_receive** (*lwgsm\_netconn\_p* nc, *lwgsm\_pbuf\_p* \*pbuf)

Receive data from connection.

**Return** *lwgsmOK* when new data ready,

**Return** *lwgsmCLOSED* when connection closed by remote side,

**Return** *lwgsmTIMEOUT* when receive timeout occurs

**Return** Any other member of *lwgsmr\_t* otherwise

**Parameters**

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

*lwgsmr\_t* **lwgsm\_netconn\_close** (*lwgsm\_netconn\_p* nc)

Close a netconn connection.

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] nc: Netconn handle to close

int8\_t **lwgsm\_netconn\_getconnnum** (*lwgsm\_netconn\_p* nc)

Get connection number used for netconn.



**Return** -1 on failure, connection number between 0 and *LWGSM\_CFG\_MAX\_CONNS* otherwise

**Parameters**

- [in] *nc*: Netconn handle

void **lwgsm\_netconn\_set\_receive\_timeout** (*lwgsm\_netconn\_p nc*, uint32\_t *timeout*)

Set timeout value for receiving data.

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

**Parameters**

- [in] *nc*: Netconn handle
- [in] *timeout*: Timeout in units of milliseconds. Set to 0 to disable timeout for *lwgsm\_netconn\_receive* function

uint32\_t **lwgsm\_netconn\_get\_receive\_timeout** (*lwgsm\_netconn\_p nc*)

Get netconn receive timeout value.

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

**Parameters**

- [in] *nc*: Netconn handle

*lwgsmr\_t* **lwgsm\_netconn\_write** (*lwgsm\_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

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

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

*lwgsmr\_t* **lwgsm\_netconn\_flush** (*lwgsm\_netconn\_p nc*)

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

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

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] *nc*: Netconn handle to flush data

*lwgsmr\_t* **lwgsm\_netconn\_send** (*lwgsm\_netconn\_p nc*, const void \**data*, size\_t *btw*)

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

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

- [in] *nc*: Netconn handle used to send

- [in] data: Pointer to data to write
- [in] btw: Number of bytes to write

*lwgsmr\_t* **lwgsm\_netconn\_sendto**(*lwgsm\_netconn\_p* nc, **const** *lwgsm\_ip\_t* \*ip, *lwgsm\_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

**Return** *lwgsmOK* on success, member of *lwgsmr\_t* enumeration otherwise

**Parameters**

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

## 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 [Visual Studio Community](#) projects
- ARM Cortex-M examples for STM32, prepared as [STM32CubeIDE](#) GCC projects

<b>Warning:</b> Library is platform independent and can be used on any platform.
--

### 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 prepared as [Visual Studio Community](#) projects. You can directly open project in the IDE, compile & debug.

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 `/lwgsm/src/system/lwgsm_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 3: 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.

## B

BUF\_PREF (*C macro*), 60

## L

lwgsm\_api\_cmd\_evt\_fn (*C++ type*), 115

LWGSMS\_ARRAYSIZE (*C macro*), 141

LWGSMS\_ASSERT (*C macro*), 141

lwgsm\_buff\_advance (*C++ function*), 62

lwgsm\_buff\_free (*C++ function*), 60

lwgsm\_buff\_get\_free (*C++ function*), 61

lwgsm\_buff\_get\_full (*C++ function*), 61

lwgsm\_buff\_get\_linear\_block\_read\_address  
(*C++ function*), 61

lwgsm\_buff\_get\_linear\_block\_read\_length  
(*C++ function*), 61

lwgsm\_buff\_get\_linear\_block\_write\_address  
(*C++ function*), 62

lwgsm\_buff\_get\_linear\_block\_write\_length  
(*C++ function*), 62

lwgsm\_buff\_init (*C++ function*), 60

lwgsm\_buff\_peek (*C++ function*), 61

lwgsm\_buff\_read (*C++ function*), 60

lwgsm\_buff\_reset (*C++ function*), 60

lwgsm\_buff\_skip (*C++ function*), 61

lwgsm\_buff\_t (*C++ struct*), 62

lwgsm\_buff\_t::buff (*C++ member*), 62

lwgsm\_buff\_t::r (*C++ member*), 62

lwgsm\_buff\_t::size (*C++ member*), 62

lwgsm\_buff\_t::w (*C++ member*), 62

lwgsm\_buff\_write (*C++ function*), 60

LWGSMS\_CFG\_AT\_ECHO (*C macro*), 150

LWGSMS\_CFG\_AT\_PORT\_BAUDRATE (*C macro*), 148

LWGSMS\_CFG\_CALL (*C macro*), 152

LWGSMS\_CFG\_CONN (*C macro*), 152

LWGSMS\_CFG\_CONN\_MAX\_DATA\_LEN (*C macro*), 148

LWGSMS\_CFG\_CONN\_POLL\_INTERVAL (*C macro*), 149

LWGSMS\_CFG\_DBG (*C macro*), 149

LWGSMS\_CFG\_DBG\_ASSERT (*C macro*), 150

LWGSMS\_CFG\_DBG\_CONN (*C macro*), 150

LWGSMS\_CFG\_DBG\_INIT (*C macro*), 150

LWGSMS\_CFG\_DBG\_INPUT (*C macro*), 150

LWGSMS\_CFG\_DBG\_IPD (*C macro*), 150

LWGSMS\_CFG\_DBG\_LVL\_MIN (*C macro*), 149

LWGSMS\_CFG\_DBG\_MEM (*C macro*), 150

LWGSMS\_CFG\_DBG\_MQTT (*C macro*), 154

LWGSMS\_CFG\_DBG\_MQTT\_API (*C macro*), 154

LWGSMS\_CFG\_DBG\_NETCONN (*C macro*), 150

LWGSMS\_CFG\_DBG\_OUT (*C macro*), 149

LWGSMS\_CFG\_DBG\_PBUF (*C macro*), 150

LWGSMS\_CFG\_DBG\_THREAD (*C macro*), 150

LWGSMS\_CFG\_DBG\_TYPES\_ON (*C macro*), 150

LWGSMS\_CFG\_DBG\_VAR (*C macro*), 150

LWGSMS\_CFG\_FTP (*C macro*), 152

LWGSMS\_CFG\_HTTP (*C macro*), 152

LWGSMS\_CFG\_INPUT\_USE\_PROCESS (*C macro*), 151

LWGSMS\_CFG\_IPD\_MAX\_BUFF\_SIZE (*C macro*), 148

LWGSMS\_CFG\_MAX\_CONNS (*C macro*), 148

LWGSMS\_CFG\_MAX\_SEND\_RETRIES (*C macro*), 148

LWGSMS\_CFG\_MEM\_ALIGNMENT (*C macro*), 148

LWGSMS\_CFG\_MEM\_CUSTOM (*C macro*), 148

LWGSMS\_CFG\_MQTT\_MAX\_REQUESTS (*C macro*), 154

LWGSMS\_CFG\_NETCONN (*C macro*), 153

LWGSMS\_CFG\_NETCONN\_ACCEPT\_QUEUE\_LEN (*C macro*), 153

LWGSMS\_CFG\_NETCONN\_RECEIVE\_QUEUE\_LEN (*C macro*), 153

LWGSMS\_CFG\_NETCONN\_RECEIVE\_TIMEOUT (*C macro*), 153

LWGSMS\_CFG\_NETWORK (*C macro*), 152

LWGSMS\_CFG\_NETWORK\_IGNORE\_CGACT\_RESULT  
(*C macro*), 152

LWGSMS\_CFG\_OS (*C macro*), 148

LWGSMS\_CFG\_PHONEBOOK (*C macro*), 152

LWGSMS\_CFG\_PING (*C macro*), 153

LWGSMS\_CFG\_RCV\_BUFF\_SIZE (*C macro*), 149

LWGSMS\_CFG\_RESET\_DELAY\_DEFAULT (*C macro*), 149

LWGSMS\_CFG\_RESET\_ON\_DEVICE\_PRESENT (*C macro*), 149

LWGSMS\_CFG\_RESET\_ON\_INIT (*C macro*), 149

LWGSMS\_CFG\_SMS (*C macro*), 152

LWGSMS\_CFG\_THREAD\_PROCESS\_MBOX\_SIZE (*C macro*), 151

LWGSMS\_CFG\_THREAD\_PRODUCER\_MBOX\_SIZE (*C macro*), 151

*macro*), 151

LWGSM\_CFG\_USE\_API\_FUNC\_EVT (*C macro*), 148

LWGSM\_CFG\_USSD (*C macro*), 153

lwgsm\_cmd\_t (*C++ enum*), 116

lwgsm\_cmd\_t::LWGSM\_CMD\_A (*C++ enumerator*), 116

lwgsm\_cmd\_t::LWGSM\_CMD\_AT\_C (*C++ enumerator*), 118

lwgsm\_cmd\_t::LWGSM\_CMD\_AT\_D (*C++ enumerator*), 118

lwgsm\_cmd\_t::LWGSM\_CMD\_AT\_F (*C++ enumerator*), 118

lwgsm\_cmd\_t::LWGSM\_CMD\_AT\_V (*C++ enumerator*), 118

lwgsm\_cmd\_t::LWGSM\_CMD\_AT\_W (*C++ enumerator*), 118

lwgsm\_cmd\_t::LWGSM\_CMD\_ATA (*C++ enumerator*), 116

lwgsm\_cmd\_t::LWGSM\_CMD\_ATD (*C++ enumerator*), 116

lwgsm\_cmd\_t::LWGSM\_CMD\_ATD\_N (*C++ enumerator*), 116

lwgsm\_cmd\_t::LWGSM\_CMD\_ATD\_STR (*C++ enumerator*), 116

lwgsm\_cmd\_t::LWGSM\_CMD\_ATDL (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATE (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATE0 (*C++ enumerator*), 116

lwgsm\_cmd\_t::LWGSM\_CMD\_ATE1 (*C++ enumerator*), 116

lwgsm\_cmd\_t::LWGSM\_CMD\_ATH (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATI (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATL (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATM (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATO (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATP (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATQ (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATS0 (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATS10 (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATS3 (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATS4 (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATS5 (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATS6 (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATS7 (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATS8 (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATT (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATV (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATX (*C++ enumerator*), 117

lwgsm\_cmd\_t::LWGSM\_CMD\_ATZ (*C++ enumerator*), 118

lwgsm\_cmd\_t::LWGSM\_CMD\_CACM (*C++ enumerator*), 119

lwgsm\_cmd\_t::LWGSM\_CMD\_CALL\_ENABLE (*C++ enumerator*), 116

lwgsm\_cmd\_t::LWGSM\_CMD\_CALM (*C++ enumerator*), 121

lwgsm\_cmd\_t::LWGSM\_CMD\_CALS (*C++ enumerator*), 121

lwgsm\_cmd\_t::LWGSM\_CMD\_CAMM (*C++ enumerator*), 119

lwgsm\_cmd\_t::LWGSM\_CMD\_CAOC (*C++ enumerator*), 119

lwgsm\_cmd\_t::LWGSM\_CMD\_CBC (*C++ enumerator*), 120

lwgsm\_cmd\_t::LWGSM\_CMD\_CBST (*C++ enumerator*), 119

lwgsm\_cmd\_t::LWGSM\_CMD\_CCFC (*C++ enumerator*), 119

lwgsm\_cmd\_t::LWGSM\_CMD\_CCLK (*C++ enumerator*), 121

lwgsm\_cmd\_t::LWGSM\_CMD\_CCWA (*C++ enumerator*), 119

lwgsm\_cmd\_t::LWGSM\_CMD\_CCWE (*C++ enumerator*), 121

lwgsm\_cmd\_t::LWGSM\_CMD\_CDNSCFG (*C++ enumerator*), 122

lwgsm\_cmd\_t::LWGSM\_CMD\_CDNSGIP (*C++ enumerator*), 122

lwgsm\_cmd\_t::LWGSM\_CMD\_CEER (*C++ enumerator*), 119

lwgsm\_cmd\_t::LWGSM\_CMD\_CFUN\_GET (*C++ enumerator*), 120

lwgsm\_cmd\_t::LWGSM\_CMD\_CFUN\_SET (*C++ enumerator*), 120

lwgsm\_cmd\_t::LWGSM\_CMD\_CGACT\_SET\_0 (*C++ enumerator*), 116

lwgsm\_cmd\_t::LWGSM\_CMD\_CGACT\_SET\_1 (*C++ enumerator*), 116

<code>lwgsm_cmd_t::LWGSM_CMD_CGATT_SET_0</code> (C++ <i>enumerator</i> ), 116	<code>lwgsm_cmd_t::LWGSM_CMD_CIPSGTXT</code> (C++ <i>enumerator</i> ), 123
<code>lwgsm_cmd_t::LWGSM_CMD_CGATT_SET_1</code> (C++ <i>enumerator</i> ), 116	<code>lwgsm_cmd_t::LWGSM_CMD_CIPSHOWTP</code> (C++ <i>enumerator</i> ), 123
<code>lwgsm_cmd_t::LWGSM_CMD_CGMI_GET</code> (C++ <i>enumerator</i> ), 118	<code>lwgsm_cmd_t::LWGSM_CMD_CIPSHUT</code> (C++ <i>enumerator</i> ), 122
<code>lwgsm_cmd_t::LWGSM_CMD_CGMM_GET</code> (C++ <i>enumerator</i> ), 119	<code>lwgsm_cmd_t::LWGSM_CMD_CIPSPRT</code> (C++ <i>enumerator</i> ), 122
<code>lwgsm_cmd_t::LWGSM_CMD_CGMR_GET</code> (C++ <i>enumerator</i> ), 119	<code>lwgsm_cmd_t::LWGSM_CMD_CIPSRIP</code> (C++ <i>enumerator</i> ), 122
<code>lwgsm_cmd_t::LWGSM_CMD_CGSN_GET</code> (C++ <i>enumerator</i> ), 119	<code>lwgsm_cmd_t::LWGSM_CMD_CIPSSL</code> (C++ <i>enumerator</i> ), 123
<code>lwgsm_cmd_t::LWGSM_CMD_CHLD</code> (C++ <i>enumerator</i> ), 119	<code>lwgsm_cmd_t::LWGSM_CMD_CIPSTART</code> (C++ <i>enumerator</i> ), 122
<code>lwgsm_cmd_t::LWGSM_CMD_CIFSR</code> (C++ <i>enumerator</i> ), 122	<code>lwgsm_cmd_t::LWGSM_CMD_CIPSTATUS</code> (C++ <i>enumerator</i> ), 122
<code>lwgsm_cmd_t::LWGSM_CMD_CIIICR</code> (C++ <i>enumerator</i> ), 122	<code>lwgsm_cmd_t::LWGSM_CMD_CIPTKA</code> (C++ <i>enumerator</i> ), 123
<code>lwgsm_cmd_t::LWGSM_CMD_CIMI</code> (C++ <i>enumerator</i> ), 119	<code>lwgsm_cmd_t::LWGSM_CMD_CIPUDPMODE</code> (C++ <i>enumerator</i> ), 123
<code>lwgsm_cmd_t::LWGSM_CMD_CIPACK</code> (C++ <i>enumerator</i> ), 122	<code>lwgsm_cmd_t::LWGSM_CMD_CLCC_SET</code> (C++ <i>enumerator</i> ), 119
<code>lwgsm_cmd_t::LWGSM_CMD_CIPATS</code> (C++ <i>enumerator</i> ), 122	<code>lwgsm_cmd_t::LWGSM_CMD_CLCK</code> (C++ <i>enumerator</i> ), 119
<code>lwgsm_cmd_t::LWGSM_CMD_CIPCCFG</code> (C++ <i>enumerator</i> ), 123	<code>lwgsm_cmd_t::LWGSM_CMD_CLIP</code> (C++ <i>enumerator</i> ), 119
<code>lwgsm_cmd_t::LWGSM_CMD_CIPCLOSE</code> (C++ <i>enumerator</i> ), 122	<code>lwgsm_cmd_t::LWGSM_CMD_CLIR</code> (C++ <i>enumerator</i> ), 119
<code>lwgsm_cmd_t::LWGSM_CMD_CIPCSGP</code> (C++ <i>enumerator</i> ), 122	<code>lwgsm_cmd_t::LWGSM_CMD_CLPORT</code> (C++ <i>enumerator</i> ), 122
<code>lwgsm_cmd_t::LWGSM_CMD_CIPDPDP</code> (C++ <i>enumerator</i> ), 123	<code>lwgsm_cmd_t::LWGSM_CMD_CLVL</code> (C++ <i>enumerator</i> ), 121
<code>lwgsm_cmd_t::LWGSM_CMD_CIPHEAD</code> (C++ <i>enumerator</i> ), 122	<code>lwgsm_cmd_t::LWGSM_CMD_CMEE_SET</code> (C++ <i>enumerator</i> ), 119
<code>lwgsm_cmd_t::LWGSM_CMD_CIPMODE</code> (C++ <i>enumerator</i> ), 123	<code>lwgsm_cmd_t::LWGSM_CMD_CMGD</code> (C++ <i>enumerator</i> ), 123
<code>lwgsm_cmd_t::LWGSM_CMD_CIPMUX</code> (C++ <i>enumerator</i> ), 122	<code>lwgsm_cmd_t::LWGSM_CMD_CMGDA</code> (C++ <i>enumerator</i> ), 123
<code>lwgsm_cmd_t::LWGSM_CMD_CIPMUX_SET</code> (C++ <i>enumerator</i> ), 116	<code>lwgsm_cmd_t::LWGSM_CMD_CMGF</code> (C++ <i>enumerator</i> ), 123
<code>lwgsm_cmd_t::LWGSM_CMD_CIPQSEND</code> (C++ <i>enumerator</i> ), 122	<code>lwgsm_cmd_t::LWGSM_CMD_CMGL</code> (C++ <i>enumerator</i> ), 123
<code>lwgsm_cmd_t::LWGSM_CMD_CIPRDTIMER</code> (C++ <i>enumerator</i> ), 123	<code>lwgsm_cmd_t::LWGSM_CMD_CMGR</code> (C++ <i>enumerator</i> ), 123
<code>lwgsm_cmd_t::LWGSM_CMD_CIPRXGET</code> (C++ <i>enumerator</i> ), 123	<code>lwgsm_cmd_t::LWGSM_CMD_CMGS</code> (C++ <i>enumerator</i> ), 123
<code>lwgsm_cmd_t::LWGSM_CMD_CIPRXGET_SET</code> (C++ <i>enumerator</i> ), 116	<code>lwgsm_cmd_t::LWGSM_CMD_CMGW</code> (C++ <i>enumerator</i> ), 123
<code>lwgsm_cmd_t::LWGSM_CMD_CIPSCONT</code> (C++ <i>enumerator</i> ), 123	<code>lwgsm_cmd_t::LWGSM_CMD_CMSS</code> (C++ <i>enumerator</i> ), 123
<code>lwgsm_cmd_t::LWGSM_CMD_CIPSEND</code> (C++ <i>enumerator</i> ), 122	<code>lwgsm_cmd_t::LWGSM_CMD_CMUT</code> (C++ <i>enumerator</i> ), 121
<code>lwgsm_cmd_t::LWGSM_CMD_CIPSERVER</code> (C++ <i>enumerator</i> ), 122	<code>lwgsm_cmd_t::LWGSM_CMD_CMUX</code> (C++ <i>enumerator</i> ), 121

<code>lwgsm_cmd_t::LWGSM_CMD_CNMI</code> ( <i>C++ enumerator</i> ), 123	<code>lwgsm_cmd_t::LWGSM_CMD_CR</code> ( <i>C++ enumerator</i> ), 121
<code>lwgsm_cmd_t::LWGSM_CMD_CNUM</code> ( <i>C++ enumerator</i> ), 120	<code>lwgsm_cmd_t::LWGSM_CMD_CRC</code> ( <i>C++ enumerator</i> ), 121
<code>lwgsm_cmd_t::LWGSM_CMD_COLP</code> ( <i>C++ enumerator</i> ), 119	<code>lwgsm_cmd_t::LWGSM_CMD_CREG_GET</code> ( <i>C++ enumerator</i> ), 120
<code>lwgsm_cmd_t::LWGSM_CMD_COPN</code> ( <i>C++ enumerator</i> ), 121	<code>lwgsm_cmd_t::LWGSM_CMD_CREG_SET</code> ( <i>C++ enumerator</i> ), 120
<code>lwgsm_cmd_t::LWGSM_CMD_COPS_GET</code> ( <i>C++ enumerator</i> ), 118	<code>lwgsm_cmd_t::LWGSM_CMD_CRES</code> ( <i>C++ enumerator</i> ), 124
<code>lwgsm_cmd_t::LWGSM_CMD_COPS_GET_OPT</code> ( <i>C++ enumerator</i> ), 118	<code>lwgsm_cmd_t::LWGSM_CMD_CRLP</code> ( <i>C++ enumerator</i> ), 121
<code>lwgsm_cmd_t::LWGSM_CMD_COPS_SET</code> ( <i>C++ enumerator</i> ), 118	<code>lwgsm_cmd_t::LWGSM_CMD_CRSL</code> ( <i>C++ enumerator</i> ), 121
<code>lwgsm_cmd_t::LWGSM_CMD_CPAS</code> ( <i>C++ enumerator</i> ), 118	<code>lwgsm_cmd_t::LWGSM_CMD_CRSM</code> ( <i>C++ enumerator</i> ), 121
<code>lwgsm_cmd_t::LWGSM_CMD_CPBF</code> ( <i>C++ enumerator</i> ), 119	<code>lwgsm_cmd_t::LWGSM_CMD_CSAS</code> ( <i>C++ enumerator</i> ), 124
<code>lwgsm_cmd_t::LWGSM_CMD_CPBR</code> ( <i>C++ enumerator</i> ), 120	<code>lwgsm_cmd_t::LWGSM_CMD_CSCA</code> ( <i>C++ enumerator</i> ), 124
<code>lwgsm_cmd_t::LWGSM_CMD_CPBS_GET</code> ( <i>C++ enumerator</i> ), 120	<code>lwgsm_cmd_t::LWGSM_CMD_CSCB</code> ( <i>C++ enumerator</i> ), 124
<code>lwgsm_cmd_t::LWGSM_CMD_CPBS_GET_OPT</code> ( <i>C++ enumerator</i> ), 120	<code>lwgsm_cmd_t::LWGSM_CMD_CSCS</code> ( <i>C++ enumerator</i> ), 119
<code>lwgsm_cmd_t::LWGSM_CMD_CPBS_SET</code> ( <i>C++ enumerator</i> ), 120	<code>lwgsm_cmd_t::LWGSM_CMD_CSDH</code> ( <i>C++ enumerator</i> ), 124
<code>lwgsm_cmd_t::LWGSM_CMD_CPBW_GET_OPT</code> ( <i>C++ enumerator</i> ), 120	<code>lwgsm_cmd_t::LWGSM_CMD_CSIM</code> ( <i>C++ enumerator</i> ), 121
<code>lwgsm_cmd_t::LWGSM_CMD_CPBW_SET</code> ( <i>C++ enumerator</i> ), 120	<code>lwgsm_cmd_t::LWGSM_CMD_CSMP</code> ( <i>C++ enumerator</i> ), 124
<code>lwgsm_cmd_t::LWGSM_CMD_CPIN_ADD</code> ( <i>C++ enumerator</i> ), 120	<code>lwgsm_cmd_t::LWGSM_CMD_CSMS</code> ( <i>C++ enumerator</i> ), 124
<code>lwgsm_cmd_t::LWGSM_CMD_CPIN_CHANGE</code> ( <i>C++ enumerator</i> ), 120	<code>lwgsm_cmd_t::LWGSM_CMD_CSQ_GET</code> ( <i>C++ enumerator</i> ), 120
<code>lwgsm_cmd_t::LWGSM_CMD_CPIN_GET</code> ( <i>C++ enumerator</i> ), 120	<code>lwgsm_cmd_t::LWGSM_CMD_CSSN</code> ( <i>C++ enumerator</i> ), 122
<code>lwgsm_cmd_t::LWGSM_CMD_CPIN_REMOVE</code> ( <i>C++ enumerator</i> ), 120	<code>lwgsm_cmd_t::LWGSM_CMD_CSTA</code> ( <i>C++ enumerator</i> ), 119
<code>lwgsm_cmd_t::LWGSM_CMD_CPIN_SET</code> ( <i>C++ enumerator</i> ), 120	<code>lwgsm_cmd_t::LWGSM_CMD_CSTT</code> ( <i>C++ enumerator</i> ), 122
<code>lwgsm_cmd_t::LWGSM_CMD_CPMS_GET</code> ( <i>C++ enumerator</i> ), 124	<code>lwgsm_cmd_t::LWGSM_CMD_CSTT_SET</code> ( <i>C++ enumerator</i> ), 116
<code>lwgsm_cmd_t::LWGSM_CMD_CPMS_GET_OPT</code> ( <i>C++ enumerator</i> ), 124	<code>lwgsm_cmd_t::LWGSM_CMD_CUSD</code> ( <i>C++ enumerator</i> ), 121
<code>lwgsm_cmd_t::LWGSM_CMD_CPMS_SET</code> ( <i>C++ enumerator</i> ), 123	<code>lwgsm_cmd_t::LWGSM_CMD_CUSD_GET</code> ( <i>C++ enumerator</i> ), 121
<code>lwgsm_cmd_t::LWGSM_CMD_CPOL</code> ( <i>C++ enumerator</i> ), 121	<code>lwgsm_cmd_t::LWGSM_CMD_CUSD_SET</code> ( <i>C++ enumerator</i> ), 121
<code>lwgsm_cmd_t::LWGSM_CMD_CPUC</code> ( <i>C++ enumerator</i> ), 121	<code>lwgsm_cmd_t::LWGSM_CMD_END</code> ( <i>C++ enumerator</i> ), 124
<code>lwgsm_cmd_t::LWGSM_CMD_CPUK_SET</code> ( <i>C++ enumerator</i> ), 120	<code>lwgsm_cmd_t::LWGSM_CMD_GCAP</code> ( <i>C++ enumerator</i> ), 118
<code>lwgsm_cmd_t::LWGSM_CMD_CPWD</code> ( <i>C++ enumerator</i> ), 120	<code>lwgsm_cmd_t::LWGSM_CMD_GMI</code> ( <i>C++ enumerator</i> ), 118



- lwgsm\_cmd\_t::LWGSM\_CMD\_GMM (C++ *enumerator*), 118
- lwgsm\_cmd\_t::LWGSM\_CMD\_GMR (C++ *enumerator*), 118
- lwgsm\_cmd\_t::LWGSM\_CMD\_GOI (C++ *enumerator*), 118
- lwgsm\_cmd\_t::LWGSM\_CMD\_GSLP (C++ *enumerator*), 116
- lwgsm\_cmd\_t::LWGSM\_CMD\_GSN (C++ *enumerator*), 118
- lwgsm\_cmd\_t::LWGSM\_CMD\_HVOIC (C++ *enumerator*), 118
- lwgsm\_cmd\_t::LWGSM\_CMD\_ICF (C++ *enumerator*), 118
- lwgsm\_cmd\_t::LWGSM\_CMD\_IDLE (C++ *enumerator*), 116
- lwgsm\_cmd\_t::LWGSM\_CMD\_IFC (C++ *enumerator*), 118
- lwgsm\_cmd\_t::LWGSM\_CMD\_IPR (C++ *enumerator*), 118
- lwgsm\_cmd\_t::LWGSM\_CMD\_NETWORK\_ATTACH (C++ *enumerator*), 116
- lwgsm\_cmd\_t::LWGSM\_CMD\_NETWORK\_DETACH (C++ *enumerator*), 116
- lwgsm\_cmd\_t::LWGSM\_CMD\_PHONEBOOK\_ENABLE (C++ *enumerator*), 119
- lwgsm\_cmd\_t::LWGSM\_CMD\_PPP (C++ *enumerator*), 117
- lwgsm\_cmd\_t::LWGSM\_CMD\_RESET (C++ *enumerator*), 116
- lwgsm\_cmd\_t::LWGSM\_CMD\_RESET\_DEVICE\_FIRST (C++ *enumerator*), 116
- lwgsm\_cmd\_t::LWGSM\_CMD\_RESTORE (C++ *enumerator*), 116
- lwgsm\_cmd\_t::LWGSM\_CMD\_SIM\_PROCESS\_BASIC (C++ *enumerator*), 120
- lwgsm\_cmd\_t::LWGSM\_CMD\_SMS\_ENABLE (C++ *enumerator*), 123
- lwgsm\_cmd\_t::LWGSM\_CMD\_UART (C++ *enumerator*), 116
- lwgsm\_cmd\_t::LWGSM\_CMD\_VTD (C++ *enumerator*), 121
- lwgsm\_cmd\_t::LWGSM\_CMD\_VTS (C++ *enumerator*), 121
- lwgsm\_conn\_close (C++ *function*), 66
- lwgsm\_conn\_connect\_res\_t (C++ *enum*), 124
- lwgsm\_conn\_connect\_res\_t::LWGSM\_CONN\_CONNECT\_RES\_ALREADY (C++ *enumerator*), 124
- lwgsm\_conn\_connect\_res\_t::LWGSM\_CONN\_CONNECT\_RES\_ERROR (C++ *enumerator*), 124
- lwgsm\_conn\_connect\_res\_t::LWGSM\_CONN\_CONNECT\_RES\_OK (C++ *enumerator*), 124
- lwgsm\_conn\_connect\_res\_t::LWGSM\_CONN\_CONNECT\_RES\_UNKNOWN (C++ *enumerator*), 124
- lwgsm\_conn\_get\_arg (C++ *function*), 68
- lwgsm\_conn\_get\_from\_evt (C++ *function*), 68
- lwgsm\_conn\_get\_local\_port (C++ *function*), 70
- lwgsm\_conn\_get\_remote\_ip (C++ *function*), 69
- lwgsm\_conn\_get\_remote\_port (C++ *function*), 70
- lwgsm\_conn\_get\_total\_recved\_count (C++ *function*), 69
- lwgsm\_conn\_getnum (C++ *function*), 68
- lwgsm\_conn\_is\_active (C++ *function*), 68
- lwgsm\_conn\_is\_client (C++ *function*), 68
- lwgsm\_conn\_is\_closed (C++ *function*), 68
- lwgsm\_conn\_p (C++ *type*), 66
- lwgsm\_conn\_recved (C++ *function*), 69
- lwgsm\_conn\_send (C++ *function*), 67
- lwgsm\_conn\_sendto (C++ *function*), 67
- lwgsm\_conn\_set\_arg (C++ *function*), 67
- lwgsm\_conn\_start (C++ *function*), 66
- lwgsm\_conn\_t (C++ *struct*), 126
- lwgsm\_conn\_t::active (C++ *member*), 126
- lwgsm\_conn\_t::arg (C++ *member*), 126
- lwgsm\_conn\_t::bearer (C++ *member*), 127
- lwgsm\_conn\_t::buff (C++ *member*), 126
- lwgsm\_conn\_t::client (C++ *member*), 126
- lwgsm\_conn\_t::data\_received (C++ *member*), 126
- lwgsm\_conn\_t::evt\_func (C++ *member*), 126
- lwgsm\_conn\_t::f (C++ *member*), 127
- lwgsm\_conn\_t::in\_closing (C++ *member*), 127
- lwgsm\_conn\_t::local\_port (C++ *member*), 126
- lwgsm\_conn\_t::num (C++ *member*), 126
- lwgsm\_conn\_t::remote\_ip (C++ *member*), 126
- lwgsm\_conn\_t::remote\_port (C++ *member*), 126
- lwgsm\_conn\_t::status (C++ *member*), 127
- lwgsm\_conn\_t::total\_recved (C++ *member*), 126
- lwgsm\_conn\_t::type (C++ *member*), 126
- lwgsm\_conn\_t::val\_id (C++ *member*), 126
- lwgsm\_conn\_type\_t (C++ *enum*), 66
- lwgsm\_conn\_type\_t::LWGSM\_CONN\_TYPE\_SSL (C++ *enumerator*), 66
- lwgsm\_conn\_type\_t::LWGSM\_CONN\_TYPE\_TCP (C++ *enumerator*), 66
- lwgsm\_conn\_type\_t::LWGSM\_CONN\_TYPE\_UDP (C++ *enumerator*), 66
- lwgsm\_core\_already\_write (C++ *function*), 69
- lwgsm\_core\_lock (C++ *function*), 146
- lwgsm\_core\_unlock (C++ *function*), 146
- lwgsm\_datetime\_t (C++ *struct*), 139
- lwgsm\_datetime\_t::date (C++ *member*), 139
- lwgsm\_datetime\_t::day (C++ *member*), 139
- lwgsm\_datetime\_t::hours (C++ *member*), 139

lwgsm\_datetime\_t::minutes (C++ member), 139  
 lwgsm\_datetime\_t::month (C++ member), 139  
 lwgsm\_datetime\_t::seconds (C++ member), 139  
 lwgsm\_datetime\_t::year (C++ member), 139  
 LWGSM\_DBG\_LVL\_ALL (C macro), 71  
 LWGSM\_DBG\_LVL\_DANGER (C macro), 71  
 LWGSM\_DBG\_LVL\_MASK (C macro), 71  
 LWGSM\_DBG\_LVL\_SEVERE (C macro), 71  
 LWGSM\_DBG\_LVL\_WARNING (C macro), 71  
 LWGSM\_DBG\_OFF (C macro), 72  
 LWGSM\_DBG\_ON (C macro), 72  
 LWGSM\_DBG\_TYPE\_ALL (C macro), 71  
 LWGSM\_DBG\_TYPE\_STATE (C macro), 71  
 LWGSM\_DBG\_TYPE\_TRACE (C macro), 71  
 LWGSM\_DEBUGF (C macro), 72  
 LWGSM\_DEBUGW (C macro), 72  
 lwgsm\_delay (C++ function), 147  
 lwgsm\_dev\_mem\_map\_t (C++ struct), 137  
 lwgsm\_dev\_mem\_map\_t::mem (C++ member), 138  
 lwgsm\_dev\_mem\_map\_t::mem\_str (C++ member), 138  
 lwgsm\_dev\_model\_map\_t (C++ struct), 138  
 lwgsm\_dev\_model\_map\_t::id\_str (C++ member), 138  
 lwgsm\_dev\_model\_map\_t::is\_2g (C++ member), 138  
 lwgsm\_dev\_model\_map\_t::is\_lte (C++ member), 138  
 lwgsm\_dev\_model\_map\_t::model (C++ member), 138  
 lwgsm\_device\_get\_manufacturer (C++ function), 72  
 lwgsm\_device\_get\_model (C++ function), 72  
 lwgsm\_device\_get\_revision (C++ function), 73  
 lwgsm\_device\_get\_serial\_number (C++ function), 73  
 lwgsm\_device\_is\_present (C++ function), 147  
 lwgsm\_device\_model\_t (C++ enum), 125  
 lwgsm\_device\_model\_t::LWGSM\_DEVICE\_MODEL\_2G (C++ enumerator), 125  
 lwgsm\_device\_model\_t::LWGSM\_DEVICE\_MODEL\_UNKNOWN (C++ enumerator), 125  
 lwgsm\_device\_set\_present (C++ function), 147  
 lwgsm\_evt\_call\_changed\_get\_call (C++ function), 80  
 lwgsm\_evt\_conn\_active\_get\_conn (C++ function), 75  
 lwgsm\_evt\_conn\_active\_is\_client (C++ function), 75  
 lwgsm\_evt\_conn\_close\_get\_conn (C++ function), 76  
 lwgsm\_evt\_conn\_close\_get\_result (C++ function), 76  
 lwgsm\_evt\_conn\_close\_is\_client (C++ function), 76  
 lwgsm\_evt\_conn\_close\_is\_forced (C++ function), 76  
 lwgsm\_evt\_conn\_error\_get\_arg (C++ function), 77  
 lwgsm\_evt\_conn\_error\_get\_error (C++ function), 77  
 lwgsm\_evt\_conn\_error\_get\_host (C++ function), 77  
 lwgsm\_evt\_conn\_error\_get\_port (C++ function), 77  
 lwgsm\_evt\_conn\_error\_get\_type (C++ function), 77  
 lwgsm\_evt\_conn\_poll\_get\_conn (C++ function), 76  
 lwgsm\_evt\_conn\_recv\_get\_buff (C++ function), 74  
 lwgsm\_evt\_conn\_recv\_get\_conn (C++ function), 74  
 lwgsm\_evt\_conn\_send\_get\_conn (C++ function), 75  
 lwgsm\_evt\_conn\_send\_get\_length (C++ function), 75  
 lwgsm\_evt\_conn\_send\_get\_result (C++ function), 75  
 lwgsm\_evt\_fn (C++ type), 80  
 lwgsm\_evt\_func\_t (C++ struct), 134  
 lwgsm\_evt\_func\_t::fn (C++ member), 134  
 lwgsm\_evt\_func\_t::next (C++ member), 134  
 lwgsm\_evt\_get\_type (C++ function), 83  
 lwgsm\_evt\_network\_operator\_get\_current (C++ function), 74  
 lwgsm\_evt\_operator\_scan\_get\_entries (C++ function), 80  
 lwgsm\_evt\_operator\_scan\_get\_length (C++ function), 80  
 lwgsm\_evt\_operator\_scan\_get\_result (C++ function), 80  
 lwgsm\_evt\_register (C++ function), 82  
 lwgsm\_evt\_reset\_get\_result (C++ function), 74  
 lwgsm\_evt\_restore\_get\_result (C++ function), 74  
 lwgsm\_evt\_signal\_strength\_get\_rssi (C++ function), 78  
 lwgsm\_evt\_sms\_delete\_get\_mem (C++ function), 79  
 lwgsm\_evt\_sms\_delete\_get\_pos (C++ function), 79  
 lwgsm\_evt\_sms\_delete\_get\_result (C++ function), 79  
 lwgsm\_evt\_sms\_read\_get\_entry (C++ function), 79

*tion*), 78  
 lwgsm\_evt\_sms\_read\_get\_result (C++ *function*), 78  
 lwgsm\_evt\_sms\_rcv\_get\_mem (C++ *function*), 78  
 lwgsm\_evt\_sms\_rcv\_get\_pos (C++ *function*), 78  
 lwgsm\_evt\_sms\_send\_get\_pos (C++ *function*), 79  
 lwgsm\_evt\_sms\_send\_get\_result (C++ *function*), 79  
 lwgsm\_evt\_t (C++ *struct*), 83  
 lwgsm\_evt\_t::arg (C++ *member*), 84  
 lwgsm\_evt\_t::buff (C++ *member*), 84  
 lwgsm\_evt\_t::call (C++ *member*), 85  
 lwgsm\_evt\_t::call\_changed (C++ *member*), 85  
 lwgsm\_evt\_t::call\_enable (C++ *member*), 85  
 lwgsm\_evt\_t::client (C++ *member*), 84  
 lwgsm\_evt\_t::conn (C++ *member*), 84  
 lwgsm\_evt\_t::conn\_active\_close (C++ *member*), 84  
 lwgsm\_evt\_t::conn\_data\_rcv (C++ *member*), 84  
 lwgsm\_evt\_t::conn\_data\_send (C++ *member*), 84  
 lwgsm\_evt\_t::conn\_error (C++ *member*), 84  
 lwgsm\_evt\_t::conn\_poll (C++ *member*), 84  
 lwgsm\_evt\_t::cpin (C++ *member*), 83  
 lwgsm\_evt\_t::entries (C++ *member*), 85  
 lwgsm\_evt\_t::entry (C++ *member*), 85  
 lwgsm\_evt\_t::err (C++ *member*), 84  
 lwgsm\_evt\_t::evt (C++ *member*), 86  
 lwgsm\_evt\_t::forced (C++ *member*), 84  
 lwgsm\_evt\_t::host (C++ *member*), 84  
 lwgsm\_evt\_t::mem (C++ *member*), 85  
 lwgsm\_evt\_t::operator\_current (C++ *member*), 83  
 lwgsm\_evt\_t::operator\_scan (C++ *member*), 84  
 lwgsm\_evt\_t::opf (C++ *member*), 84  
 lwgsm\_evt\_t::ops (C++ *member*), 83  
 lwgsm\_evt\_t::pb\_enable (C++ *member*), 85  
 lwgsm\_evt\_t::pb\_list (C++ *member*), 85  
 lwgsm\_evt\_t::pb\_search (C++ *member*), 86  
 lwgsm\_evt\_t::port (C++ *member*), 84  
 lwgsm\_evt\_t::pos (C++ *member*), 85  
 lwgsm\_evt\_t::res (C++ *member*), 83  
 lwgsm\_evt\_t::reset (C++ *member*), 83  
 lwgsm\_evt\_t::restore (C++ *member*), 83  
 lwgsm\_evt\_t::rssi (C++ *member*), 84  
 lwgsm\_evt\_t::search (C++ *member*), 85  
 lwgsm\_evt\_t::sent (C++ *member*), 84  
 lwgsm\_evt\_t::size (C++ *member*), 85  
 lwgsm\_evt\_t::sms\_delete (C++ *member*), 85  
 lwgsm\_evt\_t::sms\_enable (C++ *member*), 85  
 lwgsm\_evt\_t::sms\_list (C++ *member*), 85  
 lwgsm\_evt\_t::sms\_read (C++ *member*), 85  
 lwgsm\_evt\_t::sms\_rcv (C++ *member*), 85  
 lwgsm\_evt\_t::sms\_send (C++ *member*), 85  
 lwgsm\_evt\_t::state (C++ *member*), 83  
 lwgsm\_evt\_t::status (C++ *member*), 85  
 lwgsm\_evt\_t::type (C++ *member*), 83, 84  
 lwgsm\_evt\_type\_t (C++ *enum*), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_CALL\_BUSY (C++ *enumerator*), 82  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_CALL\_CHANGED (C++ *enumerator*), 82  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_CALL\_ENABLE (C++ *enumerator*), 82  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_CALL\_NO\_CARRIER (C++ *enumerator*), 82  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_CALL\_READY (C++ *enumerator*), 82  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_CALL\_RING (C++ *enumerator*), 82  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_CMD\_TIMEOUT (C++ *enumerator*), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_CONN\_ACTIVE (C++ *enumerator*), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_CONN\_CLOSE (C++ *enumerator*), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_CONN\_ERROR (C++ *enumerator*), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_CONN\_POLL (C++ *enumerator*), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_CONN\_RECV (C++ *enumerator*), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_CONN\_SEND (C++ *enumerator*), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_DEVICE\_IDENTIFIED (C++ *enumerator*), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_DEVICE\_PRESENT (C++ *enumerator*), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_INIT\_FINISH (C++ *enumerator*), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_NETWORK\_ATTACHED (C++ *enumerator*), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_NETWORK\_DETACHED (C++ *enumerator*), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_NETWORK\_OPERATOR\_CURRENT (C++ *enumerator*), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_NETWORK\_REG\_CHANGED (C++ *enumerator*), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_OPERATOR\_SCAN (C++ *enumerator*), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_PB\_ENABLE (C++ *enumerator*), 82  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_PB\_LIST

(C++ enumerator), 82  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_PB\_SEARCH (C++ enumerator), 82  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_RESET (C++ enumerator), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_RESTORE (C++ enumerator), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_SIGNAL\_STRENGTH (C++ enumerator), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_SIM\_STATE\_CHANGED (C++ enumerator), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_SMS\_DELETE (C++ enumerator), 82  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_SMS\_ENABLE (C++ enumerator), 81  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_SMS\_LIST (C++ enumerator), 82  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_SMS\_READ (C++ enumerator), 82  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_SMS\_READY (C++ enumerator), 82  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_SMS\_RECV (C++ enumerator), 82  
 lwgsm\_evt\_type\_t::LWGSM\_EVT\_SMS\_SEND (C++ enumerator), 82  
 lwgsm\_evt\_unregister (C++ function), 82  
 lwgsm\_get\_conns\_status (C++ function), 68  
 LWGSM\_I16 (C macro), 142  
 lwgsm\_i16\_to\_str (C macro), 144  
 LWGSM\_I32 (C macro), 142  
 lwgsm\_i32\_to\_gen\_str (C++ function), 145  
 lwgsm\_i32\_to\_str (C macro), 143  
 LWGSM\_I8 (C macro), 142  
 lwgsm\_i8\_to\_str (C macro), 144  
 lwgsm\_init (C++ function), 145  
 lwgsm\_input (C++ function), 87  
 lwgsm\_input\_process (C++ function), 87  
 lwgsm\_ip\_mac\_t (C++ struct), 133  
 lwgsm\_ip\_mac\_t::gw (C++ member), 133  
 lwgsm\_ip\_mac\_t::ip (C++ member), 133  
 lwgsm\_ip\_mac\_t::mac (C++ member), 133  
 lwgsm\_ip\_mac\_t::nm (C++ member), 133  
 lwgsm\_ip\_t (C++ struct), 138  
 lwgsm\_ip\_t::ip (C++ member), 138  
 lwgsm\_ipd\_t (C++ struct), 127  
 lwgsm\_ipd\_t::buff (C++ member), 128  
 lwgsm\_ipd\_t::buff\_ptr (C++ member), 127  
 lwgsm\_ipd\_t::conn (C++ member), 127  
 lwgsm\_ipd\_t::read (C++ member), 127  
 lwgsm\_ipd\_t::rem\_len (C++ member), 127  
 lwgsm\_ipd\_t::tot\_len (C++ member), 127  
 lwgsm\_linbuff\_t (C++ struct), 139  
 lwgsm\_linbuff\_t::buff (C++ member), 139  
 lwgsm\_linbuff\_t::len (C++ member), 139  
 lwgsm\_linbuff\_t::ptr (C++ member), 139  
 lwgsm\_link\_conn\_t (C++ struct), 133  
 lwgsm\_link\_conn\_t::failed (C++ member), 134  
 lwgsm\_link\_conn\_t::is\_server (C++ member), 134  
 lwgsm\_link\_conn\_t::local\_port (C++ member), 134  
 lwgsm\_link\_conn\_t::num (C++ member), 134  
 lwgsm\_link\_conn\_t::remote\_ip (C++ member), 134  
 lwgsm\_link\_conn\_t::remote\_port (C++ member), 134  
 lwgsm\_link\_conn\_t::type (C++ member), 134  
 lwgsm\_ll\_deinit (C++ function), 155  
 lwgsm\_ll\_init (C++ function), 155  
 lwgsm\_ll\_reset\_fn (C++ type), 154  
 lwgsm\_ll\_send\_fn (C++ type), 154  
 lwgsm\_ll\_t (C++ struct), 155  
 lwgsm\_ll\_t::baudrate (C++ member), 155  
 lwgsm\_ll\_t::reset\_fn (C++ member), 155  
 lwgsm\_ll\_t::send\_fn (C++ member), 155  
 lwgsm\_ll\_t::uart (C++ member), 155  
 lwgsm\_mac\_t (C++ struct), 138  
 lwgsm\_mac\_t::mac (C++ member), 139  
 LWGSM\_MAX (C macro), 141  
 LWGSM\_MEM\_ALIGN (C macro), 141  
 lwgsm\_mem\_assignmemory (C++ function), 88  
 lwgsm\_mem\_calloc (C++ function), 88  
 lwgsm\_mem\_free (C++ function), 89  
 lwgsm\_mem\_free\_s (C++ function), 89  
 lwgsm\_mem\_malloc (C++ function), 88  
 lwgsm\_mem\_realloc (C++ function), 88  
 lwgsm\_mem\_region\_t (C++ struct), 89  
 lwgsm\_mem\_region\_t::size (C++ member), 89  
 lwgsm\_mem\_region\_t::start\_addr (C++ member), 89  
 lwgsm\_mem\_t (C++ enum), 125  
 lwgsm\_mem\_t::LWGSM\_MEM\_CURRENT (C++ enumerator), 126  
 lwgsm\_mem\_t::LWGSM\_MEM\_END (C++ enumerator), 125  
 lwgsm\_mem\_t::LWGSM\_MEM\_UNKNOWN (C++ enumerator), 126  
 LWGSM\_MEMCPY (C macro), 151  
 LWGSM\_MEMSET (C macro), 152  
 LWGSM\_MIN (C macro), 141  
 lwgsm\_modules\_t (C++ struct), 136  
 lwgsm\_modules\_t::active\_conns\_cur\_parse\_num (C++ member), 136  
 lwgsm\_modules\_t::call (C++ member), 136  
 lwgsm\_modules\_t::conn\_val\_id (C++ member), 136  
 lwgsm\_modules\_t::conns (C++ member), 136

lwgsm\_modules\_t::ipd (C++ *member*), 136  
 lwgsm\_modules\_t::model (C++ *member*), 136  
 lwgsm\_modules\_t::model\_manufacturer (C++ *member*), 136  
 lwgsm\_modules\_t::model\_number (C++ *member*), 136  
 lwgsm\_modules\_t::model\_revision (C++ *member*), 136  
 lwgsm\_modules\_t::model\_serial\_number (C++ *member*), 136  
 lwgsm\_modules\_t::network (C++ *member*), 136  
 lwgsm\_modules\_t::pb (C++ *member*), 136  
 lwgsm\_modules\_t::rssi (C++ *member*), 136  
 lwgsm\_modules\_t::sim (C++ *member*), 136  
 lwgsm\_modules\_t::sms (C++ *member*), 136  
 lwgsm\_mqtt\_client\_api\_buf\_free (C++ *function*), 175  
 lwgsm\_mqtt\_client\_api\_buf\_p (C++ *type*), 173  
 lwgsm\_mqtt\_client\_api\_buf\_t (C++ *struct*), 175  
 lwgsm\_mqtt\_client\_api\_buf\_t::payload (C++ *member*), 176  
 lwgsm\_mqtt\_client\_api\_buf\_t::payload\_len (C++ *member*), 176  
 lwgsm\_mqtt\_client\_api\_buf\_t::qos (C++ *member*), 176  
 lwgsm\_mqtt\_client\_api\_buf\_t::topic (C++ *member*), 176  
 lwgsm\_mqtt\_client\_api\_buf\_t::topic\_len (C++ *member*), 176  
 lwgsm\_mqtt\_client\_api\_close (C++ *function*), 174  
 lwgsm\_mqtt\_client\_api\_connect (C++ *function*), 173  
 lwgsm\_mqtt\_client\_api\_delete (C++ *function*), 173  
 lwgsm\_mqtt\_client\_api\_is\_connected (C++ *function*), 175  
 lwgsm\_mqtt\_client\_api\_new (C++ *function*), 173  
 lwgsm\_mqtt\_client\_api\_publish (C++ *function*), 174  
 lwgsm\_mqtt\_client\_api\_receive (C++ *function*), 175  
 lwgsm\_mqtt\_client\_api\_subscribe (C++ *function*), 174  
 lwgsm\_mqtt\_client\_api\_unsubscribe (C++ *function*), 174  
 lwgsm\_mqtt\_client\_connect (C++ *function*), 164  
 lwgsm\_mqtt\_client\_delete (C++ *function*), 163  
 lwgsm\_mqtt\_client\_disconnect (C++ *function*), 164  
 lwgsm\_mqtt\_client\_evt\_connect\_get\_status (C *macro*), 167  
 lwgsm\_mqtt\_client\_evt\_disconnect\_is\_accepted (C *macro*), 168  
 lwgsm\_mqtt\_client\_evt\_get\_type (C *macro*), 170  
 lwgsm\_mqtt\_client\_evt\_publish\_get\_argument (C *macro*), 170  
 lwgsm\_mqtt\_client\_evt\_publish\_get\_result (C *macro*), 170  
 lwgsm\_mqtt\_client\_evt\_publish\_recv\_get\_payload (C *macro*), 169  
 lwgsm\_mqtt\_client\_evt\_publish\_recv\_get\_payload\_len (C *macro*), 169  
 lwgsm\_mqtt\_client\_evt\_publish\_recv\_get\_qos (C *macro*), 170  
 lwgsm\_mqtt\_client\_evt\_publish\_recv\_get\_topic (C *macro*), 169  
 lwgsm\_mqtt\_client\_evt\_publish\_recv\_get\_topic\_len (C *macro*), 169  
 lwgsm\_mqtt\_client\_evt\_publish\_recv\_is\_duplicate (C *macro*), 169  
 lwgsm\_mqtt\_client\_evt\_subscribe\_get\_argument (C *macro*), 168  
 lwgsm\_mqtt\_client\_evt\_subscribe\_get\_result (C *macro*), 168  
 lwgsm\_mqtt\_client\_evt\_unsubscribe\_get\_argument (C *macro*), 168  
 lwgsm\_mqtt\_client\_evt\_unsubscribe\_get\_result (C *macro*), 168  
 lwgsm\_mqtt\_client\_get\_arg (C++ *function*), 165  
 lwgsm\_mqtt\_client\_info\_t (C++ *struct*), 165  
 lwgsm\_mqtt\_client\_info\_t::id (C++ *member*), 166  
 lwgsm\_mqtt\_client\_info\_t::keep\_alive (C++ *member*), 166  
 lwgsm\_mqtt\_client\_info\_t::pass (C++ *member*), 166  
 lwgsm\_mqtt\_client\_info\_t::user (C++ *member*), 166  
 lwgsm\_mqtt\_client\_info\_t::will\_message (C++ *member*), 166  
 lwgsm\_mqtt\_client\_info\_t::will\_qos (C++ *member*), 166  
 lwgsm\_mqtt\_client\_info\_t::will\_topic (C++ *member*), 166  
 lwgsm\_mqtt\_client\_is\_connected (C++ *function*), 164  
 lwgsm\_mqtt\_client\_new (C++ *function*), 163  
 lwgsm\_mqtt\_client\_p (C++ *type*), 162  
 lwgsm\_mqtt\_client\_publish (C++ *function*), 165  
 lwgsm\_mqtt\_client\_set\_arg (C++ *function*), 165



lwgsm\_mqtt\_client\_subscribe (C++ *function*), 164  
 lwgsm\_mqtt\_client\_unsubscribe (C++ *function*), 164  
 lwgsm\_mqtt\_conn\_status\_t (C++ *enum*), 163  
 lwgsm\_mqtt\_conn\_status\_t::LWGSMMQTT\_CONN\_STATUS\_ACCEPTED (C++ *enumerator*), 163  
 lwgsm\_mqtt\_conn\_status\_t::LWGSMMQTT\_CONN\_STATUS\_CONNECTED (C++ *enumerator*), 163  
 lwgsm\_mqtt\_conn\_status\_t::LWGSMMQTT\_CONN\_STATUS\_DISCONNECTED (C++ *enumerator*), 163  
 lwgsm\_mqtt\_conn\_status\_t::LWGSMMQTT\_CONN\_STATUS\_FAILED (C++ *enumerator*), 163  
 lwgsm\_mqtt\_conn\_status\_t::LWGSMMQTT\_CONN\_STATUS\_REFUSED (C++ *enumerator*), 163  
 lwgsm\_mqtt\_conn\_status\_t::LWGSMMQTT\_CONN\_STATUS\_REFUSED\_PROTOCOL\_VERSION\_NOT\_SUPPORTED (C++ *enumerator*), 163  
 lwgsm\_mqtt\_conn\_status\_t::LWGSMMQTT\_CONN\_STATUS\_REFUSED\_SERVER (C++ *enumerator*), 163  
 lwgsm\_mqtt\_conn\_status\_t::LWGSMMQTT\_CONN\_STATUS\_REFUSED\_USER (C++ *enumerator*), 163  
 lwgsm\_mqtt\_conn\_status\_t::LWGSMMQTT\_CONN\_STATUS\_TCP\_FAILED (C++ *enumerator*), 163  
 lwgsm\_mqtt\_evt\_fn (C++ *type*), 162  
 lwgsm\_mqtt\_evt\_t (C++ *struct*), 166  
 lwgsm\_mqtt\_evt\_t::arg (C++ *member*), 167  
 lwgsm\_mqtt\_evt\_t::connect (C++ *member*), 166  
 lwgsm\_mqtt\_evt\_t::disconnect (C++ *member*), 166  
 lwgsm\_mqtt\_evt\_t::dup (C++ *member*), 167  
 lwgsm\_mqtt\_evt\_t::evt (C++ *member*), 167  
 lwgsm\_mqtt\_evt\_t::is\_accepted (C++ *member*), 166  
 lwgsm\_mqtt\_evt\_t::payload (C++ *member*), 167  
 lwgsm\_mqtt\_evt\_t::payload\_len (C++ *member*), 167  
 lwgsm\_mqtt\_evt\_t::publish (C++ *member*), 167  
 lwgsm\_mqtt\_evt\_t::publish\_recv (C++ *member*), 167  
 lwgsm\_mqtt\_evt\_t::qos (C++ *member*), 167  
 lwgsm\_mqtt\_evt\_t::res (C++ *member*), 167  
 lwgsm\_mqtt\_evt\_t::status (C++ *member*), 166  
 lwgsm\_mqtt\_evt\_t::sub\_unsubscribed (C++ *member*), 167  
 lwgsm\_mqtt\_evt\_t::topic (C++ *member*), 167  
 lwgsm\_mqtt\_evt\_t::topic\_len (C++ *member*), 167  
 lwgsm\_mqtt\_evt\_t::type (C++ *member*), 166  
 lwgsm\_mqtt\_evt\_type\_t (C++ *enum*), 162  
 lwgsm\_mqtt\_evt\_type\_t::LWGSMMQTT\_EVT\_CONNECTED (C++ *enumerator*), 162  
 lwgsm\_mqtt\_evt\_type\_t::LWGSMMQTT\_EVT\_DISCONNECTED (C++ *enumerator*), 163  
 lwgsm\_mqtt\_evt\_type\_t::LWGSMMQTT\_EVT\_KEEP\_ALIVE (C++ *enumerator*), 163  
 lwgsm\_mqtt\_evt\_type\_t::LWGSMMQTT\_EVT\_PUBLISH (C++ *enumerator*), 163  
 lwgsm\_mqtt\_evt\_type\_t::LWGSMMQTT\_EVT\_PUBLISH\_RECV (C++ *enumerator*), 163  
 lwgsm\_mqtt\_evt\_type\_t::LWGSMMQTT\_EVT\_SUBSCRIBE (C++ *enumerator*), 162  
 lwgsm\_mqtt\_evt\_type\_t::LWGSMMQTT\_EVT\_UNSUBSCRIBE (C++ *enumerator*), 162  
 lwgsm\_mqtt\_qos\_t (C++ *enum*), 162  
 lwgsm\_mqtt\_qos\_t::LWGSMMQTT\_QOS\_AT\_LEAST\_ONCE (C++ *enumerator*), 162  
 lwgsm\_mqtt\_qos\_t::LWGSMMQTT\_QOS\_AT\_MOST\_ONCE (C++ *enumerator*), 162  
 lwgsm\_mqtt\_qos\_t::LWGSMMQTT\_QOS\_EXACTLY\_ONCE (C++ *enumerator*), 162  
 lwgsm\_mqtt\_request\_t (C++ *struct*), 166  
 lwgsm\_mqtt\_request\_t::arg (C++ *member*), 166  
 lwgsm\_mqtt\_request\_t::expected\_sent\_len (C++ *member*), 166  
 lwgsm\_mqtt\_request\_t::packet\_id (C++ *member*), 166  
 lwgsm\_mqtt\_request\_t::status (C++ *member*), 166  
 lwgsm\_mqtt\_request\_t::timeout\_start\_time (C++ *member*), 166  
 lwgsm\_mqtt\_state\_t (C++ *enum*), 162  
 lwgsm\_mqtt\_state\_t::LWGSMMQTT\_CONN\_CONNECTING (C++ *enumerator*), 162  
 lwgsm\_mqtt\_state\_t::LWGSMMQTT\_CONN\_DISCONNECTED (C++ *enumerator*), 162  
 lwgsm\_mqtt\_state\_t::LWGSMMQTT\_CONN\_DISCONNECTING (C++ *enumerator*), 162  
 lwgsm\_mqtt\_state\_t::LWGSMMQTT\_CONNECTED (C++ *enumerator*), 162  
 lwgsm\_mqtt\_state\_t::LWGSMMQTT\_CONNECTING (C++ *enumerator*), 162  
 lwgsm\_msg\_t (C++ *struct*), 128  
 lwgsm\_msg\_t::apn (C++ *member*), 133  
 lwgsm\_msg\_t::arg (C++ *member*), 130  
 lwgsm\_msg\_t::baudrate (C++ *member*), 128  
 lwgsm\_msg\_t::block\_time (C++ *member*), 128  
 lwgsm\_msg\_t::btw (C++ *member*), 130  
 lwgsm\_msg\_t::bw (C++ *member*), 131  
 lwgsm\_msg\_t::call\_start (C++ *member*), 132  
 lwgsm\_msg\_t::cfun (C++ *member*), 128  
 lwgsm\_msg\_t::cmd (C++ *member*), 128  
 lwgsm\_msg\_t::cmd\_def (C++ *member*), 128  
 lwgsm\_msg\_t::cnum\_tries (C++ *member*), 129  
 lwgsm\_msg\_t::code (C++ *member*), 133  
 lwgsm\_msg\_t::conn (C++ *member*), 130  
 lwgsm\_msg\_t::conn\_close (C++ *member*), 130  
 lwgsm\_msg\_t::conn\_res (C++ *member*), 130  
 lwgsm\_msg\_t::conn\_send (C++ *member*), 131

lwgsm\_msg\_t::conn\_start (C++ member), 130  
 lwgsm\_msg\_t::cops\_get (C++ member), 130  
 lwgsm\_msg\_t::cops\_scan (C++ member), 129  
 lwgsm\_msg\_t::cops\_set (C++ member), 130  
 lwgsm\_msg\_t::cpin\_add (C++ member), 129  
 lwgsm\_msg\_t::cpin\_change (C++ member), 129  
 lwgsm\_msg\_t::cpin\_enter (C++ member), 128  
 lwgsm\_msg\_t::cpin\_remove (C++ member), 129  
 lwgsm\_msg\_t::cpuk\_enter (C++ member), 129  
 lwgsm\_msg\_t::csq (C++ member), 129  
 lwgsm\_msg\_t::curr (C++ member), 129  
 lwgsm\_msg\_t::current\_pin (C++ member), 129  
 lwgsm\_msg\_t::data (C++ member), 131  
 lwgsm\_msg\_t::del (C++ member), 132  
 lwgsm\_msg\_t::delay (C++ member), 128  
 lwgsm\_msg\_t::device\_info (C++ member), 129  
 lwgsm\_msg\_t::ei (C++ member), 132  
 lwgsm\_msg\_t::entries (C++ member), 132  
 lwgsm\_msg\_t::entry (C++ member), 131  
 lwgsm\_msg\_t::er (C++ member), 132  
 lwgsm\_msg\_t::etr (C++ member), 132  
 lwgsm\_msg\_t::evt\_func (C++ member), 130  
 lwgsm\_msg\_t::fau (C++ member), 131  
 lwgsm\_msg\_t::fn (C++ member), 128  
 lwgsm\_msg\_t::format (C++ member), 130, 131  
 lwgsm\_msg\_t::host (C++ member), 130  
 lwgsm\_msg\_t::i (C++ member), 128  
 lwgsm\_msg\_t::is\_blocking (C++ member), 128  
 lwgsm\_msg\_t::len (C++ member), 129  
 lwgsm\_msg\_t::mem (C++ member), 131  
 lwgsm\_msg\_t::mode (C++ member), 128, 130  
 lwgsm\_msg\_t::msg (C++ member), 133  
 lwgsm\_msg\_t::name (C++ member), 130  
 lwgsm\_msg\_t::network\_attach (C++ member), 133  
 lwgsm\_msg\_t::new\_pin (C++ member), 129  
 lwgsm\_msg\_t::num (C++ member), 130, 131  
 lwgsm\_msg\_t::number (C++ member), 132  
 lwgsm\_msg\_t::opf (C++ member), 129  
 lwgsm\_msg\_t::ops (C++ member), 129  
 lwgsm\_msg\_t::opsi (C++ member), 129  
 lwgsm\_msg\_t::opsl (C++ member), 129  
 lwgsm\_msg\_t::pass (C++ member), 133  
 lwgsm\_msg\_t::pb\_list (C++ member), 132  
 lwgsm\_msg\_t::pb\_search (C++ member), 133  
 lwgsm\_msg\_t::pb\_write (C++ member), 132  
 lwgsm\_msg\_t::pin (C++ member), 128  
 lwgsm\_msg\_t::port (C++ member), 130  
 lwgsm\_msg\_t::pos (C++ member), 131  
 lwgsm\_msg\_t::ptr (C++ member), 130  
 lwgsm\_msg\_t::puk (C++ member), 129  
 lwgsm\_msg\_t::quote\_det (C++ member), 133  
 lwgsm\_msg\_t::read (C++ member), 129  
 lwgsm\_msg\_t::remote\_ip (C++ member), 131  
 lwgsm\_msg\_t::remote\_port (C++ member), 131  
 lwgsm\_msg\_t::res (C++ member), 128  
 lwgsm\_msg\_t::reset (C++ member), 128  
 lwgsm\_msg\_t::resp (C++ member), 133  
 lwgsm\_msg\_t::resp\_len (C++ member), 133  
 lwgsm\_msg\_t::resp\_write\_ptr (C++ member), 133  
 lwgsm\_msg\_t::rssi (C++ member), 129  
 lwgsm\_msg\_t::search (C++ member), 132  
 lwgsm\_msg\_t::sem (C++ member), 128  
 lwgsm\_msg\_t::sent (C++ member), 131  
 lwgsm\_msg\_t::sent\_all (C++ member), 131  
 lwgsm\_msg\_t::sim\_info (C++ member), 129  
 lwgsm\_msg\_t::sms\_delete (C++ member), 132  
 lwgsm\_msg\_t::sms\_delete\_all (C++ member), 132  
 lwgsm\_msg\_t::sms\_list (C++ member), 132  
 lwgsm\_msg\_t::sms\_memory (C++ member), 132  
 lwgsm\_msg\_t::sms\_read (C++ member), 132  
 lwgsm\_msg\_t::sms\_send (C++ member), 131  
 lwgsm\_msg\_t::start\_index (C++ member), 132  
 lwgsm\_msg\_t::status (C++ member), 132  
 lwgsm\_msg\_t::str (C++ member), 129  
 lwgsm\_msg\_t::text (C++ member), 131  
 lwgsm\_msg\_t::tries (C++ member), 131  
 lwgsm\_msg\_t::type (C++ member), 130, 132  
 lwgsm\_msg\_t::uart (C++ member), 128  
 lwgsm\_msg\_t::update (C++ member), 132  
 lwgsm\_msg\_t::user (C++ member), 133  
 lwgsm\_msg\_t::ussd (C++ member), 133  
 lwgsm\_msg\_t::val\_id (C++ member), 130  
 lwgsm\_msg\_t::wait\_send\_ok\_err (C++ member), 131  
 lwgsm\_netconn\_close (C++ function), 180  
 lwgsm\_netconn\_connect (C++ function), 180  
 lwgsm\_netconn\_delete (C++ function), 180  
 lwgsm\_netconn\_flush (C++ function), 181  
 lwgsm\_netconn\_get\_receive\_timeout (C++ function), 181  
 lwgsm\_netconn\_getconnnum (C++ function), 180  
 lwgsm\_netconn\_new (C++ function), 180  
 lwgsm\_netconn\_p (C++ type), 179  
 lwgsm\_netconn\_receive (C++ function), 180  
 lwgsm\_netconn\_send (C++ function), 181  
 lwgsm\_netconn\_sendto (C++ function), 182  
 lwgsm\_netconn\_set\_receive\_timeout (C++ function), 181  
 lwgsm\_netconn\_type\_t (C++ enum), 179  
 lwgsm\_netconn\_type\_t::LWGSM\_NETCONN\_TYPE\_SSL (C++ enumerator), 179  
 lwgsm\_netconn\_type\_t::LWGSM\_NETCONN\_TYPE\_TCP (C++ enumerator), 179  
 lwgsm\_netconn\_type\_t::LWGSM\_NETCONN\_TYPE\_UDP (C++ enumerator), 179

- lwgsm\_netconn\_write (C++ function), 181  
 lwgsm\_network\_attach (C++ function), 90  
 lwgsm\_network\_check\_status (C++ function), 91  
 lwgsm\_network\_copy\_ip (C++ function), 91  
 lwgsm\_network\_detach (C++ function), 90  
 lwgsm\_network\_get\_reg\_status (C++ function), 90  
 lwgsm\_network\_is\_attached (C++ function), 91  
 lwgsm\_network\_reg\_status\_t (C++ enum), 89  
 lwgsm\_network\_reg\_status\_t::LWGSM\_NETWORK\_REG\_STATUS\_CONNECTED (C++ enumerator), 89  
 lwgsm\_network\_reg\_status\_t::LWGSM\_NETWORK\_REG\_STATUS\_DISCONNECTED (C++ enumerator), 90  
 lwgsm\_network\_reg\_status\_t::LWGSM\_NETWORK\_REG\_STATUS\_ROAMING (C++ enumerator), 90  
 lwgsm\_network\_reg\_status\_t::LWGSM\_NETWORK\_REG\_STATUS\_SMS\_ONLY (C++ enumerator), 90  
 lwgsm\_network\_reg\_status\_t::LWGSM\_NETWORK\_REG\_STATUS\_UNKNOWN (C++ enumerator), 89  
 lwgsm\_network\_reg\_status\_t::LWGSM\_NETWORK\_REG\_STATUS\_SIM\_ONLY (C++ enumerator), 90  
 lwgsm\_network\_reg\_status\_t::LWGSM\_NETWORK\_REG\_STATUS\_SIM\_ONLY (C++ enumerator), 93  
 lwgsm\_network\_reg\_status\_t::LWGSM\_NETWORK\_REG\_STATUS\_UNKNOWN (C++ enumerator), 89  
 lwgsm\_network\_reg\_status\_t::LWGSM\_NETWORK\_REG\_STATUS\_UNKNOWN (C++ enumerator), 93  
 lwgsm\_network\_request\_attach (C++ function), 92  
 lwgsm\_network\_request\_detach (C++ function), 92  
 lwgsm\_network\_rssi (C++ function), 90  
 lwgsm\_network\_set\_credentials (C++ function), 92  
 lwgsm\_network\_t (C++ struct), 135  
 lwgsm\_network\_t::curr\_operator (C++ member), 136  
 lwgsm\_network\_t::ip\_addr (C++ member), 136  
 lwgsm\_network\_t::is\_attached (C++ member), 136  
 lwgsm\_network\_t::status (C++ member), 136  
 lwgsm\_number\_type\_t (C++ enum), 126  
 lwgsm\_number\_type\_t::LWGSM\_NUMBER\_TYPE\_INTERNATIONAL (C++ enumerator), 126  
 lwgsm\_number\_type\_t::LWGSM\_NUMBER\_TYPE\_NATIONAL (C++ enumerator), 126  
 lwgsm\_operator\_curr\_t (C++ struct), 94  
 lwgsm\_operator\_curr\_t::data (C++ member), 95  
 lwgsm\_operator\_curr\_t::format (C++ member), 95  
 lwgsm\_operator\_curr\_t::long\_name (C++ member), 95  
 lwgsm\_operator\_curr\_t::mode (C++ member), 95  
 lwgsm\_operator\_curr\_t::num (C++ member), 95  
 lwgsm\_operator\_curr\_t::short\_name (C++ member), 95  
 lwgsm\_operator\_format\_t (C++ enum), 93  
 lwgsm\_operator\_format\_t::LWGSM\_OPERATOR\_FORMAT\_INVALID (C++ enumerator), 93  
 lwgsm\_operator\_format\_t::LWGSM\_OPERATOR\_FORMAT\_LONG (C++ enumerator), 93  
 lwgsm\_operator\_format\_t::LWGSM\_OPERATOR\_FORMAT\_NUM (C++ enumerator), 93  
 lwgsm\_operator\_format\_t::LWGSM\_OPERATOR\_FORMAT\_SHORT (C++ enumerator), 93  
 lwgsm\_operator\_get (C++ function), 93  
 lwgsm\_operator\_mode\_t::LWGSM\_OPERATOR\_MODE\_AUTO (C++ enumerator), 93  
 lwgsm\_operator\_mode\_t::LWGSM\_OPERATOR\_MODE\_DEREGISTER (C++ enumerator), 93  
 lwgsm\_operator\_mode\_t::LWGSM\_OPERATOR\_MODE\_MANUAL (C++ enumerator), 93  
 lwgsm\_operator\_mode\_t::LWGSM\_OPERATOR\_MODE\_MANUAL\_A (C++ enumerator), 93  
 lwgsm\_operator\_scan (C++ function), 94  
 lwgsm\_operator\_status\_t (C++ enum), 92  
 lwgsm\_operator\_status\_t::LWGSM\_OPERATOR\_STATUS\_AVAILABLE (C++ enumerator), 92  
 lwgsm\_operator\_status\_t::LWGSM\_OPERATOR\_STATUS\_CURRENT (C++ enumerator), 92  
 lwgsm\_operator\_status\_t::LWGSM\_OPERATOR\_STATUS\_FORWARD (C++ enumerator), 92  
 lwgsm\_operator\_status\_t::LWGSM\_OPERATOR\_STATUS\_UNKNOWN (C++ enumerator), 92  
 lwgsm\_operator\_t (C++ struct), 94  
 lwgsm\_operator\_t::long\_name (C++ member), 94  
 lwgsm\_operator\_t::num (C++ member), 94  
 lwgsm\_operator\_t::short\_name (C++ member), 94  
 lwgsm\_operator\_t::stat (C++ member), 94  
 lwgsm\_operator\_t::stat (C++ member), 94  
 lwgsm\_pb\_delete (C++ function), 105  
 lwgsm\_pb\_disable (C++ function), 104  
 lwgsm\_pb\_edit (C++ function), 105  
 lwgsm\_pb\_enable (C++ function), 104  
 lwgsm\_pb\_list (C++ function), 106  
 lwgsm\_pb\_mem\_t (C++ struct), 113, 135  
 lwgsm\_pb\_mem\_t::current (C++ member), 113, 135  
 lwgsm\_pb\_mem\_t::mem\_available (C++ member), 113, 135  
 lwgsm\_pb\_mem\_t::total (C++ member), 113, 135  
 lwgsm\_pb\_mem\_t::used (C++ member), 113, 135  
 lwgsm\_pb\_read (C++ function), 106  
 lwgsm\_pb\_search (C++ function), 106



---

lwgsm\_pb\_t (C++ struct), 135  
 lwgsm\_pb\_t::enabled (C++ member), 135  
 lwgsm\_pb\_t::mem (C++ member), 135  
 lwgsm\_pb\_t::ready (C++ member), 135  
 lwgsm\_pbuf\_advance (C++ function), 103  
 lwgsm\_pbuf\_cat (C++ function), 101  
 lwgsm\_pbuf\_chain (C++ function), 101  
 lwgsm\_pbuf\_copy (C++ function), 100  
 lwgsm\_pbuf\_data (C++ function), 100  
 lwgsm\_pbuf\_free (C++ function), 100  
 lwgsm\_pbuf\_get\_at (C++ function), 101  
 lwgsm\_pbuf\_get\_linear\_addr (C++ function), 103  
 lwgsm\_pbuf\_length (C++ function), 100  
 lwgsm\_pbuf\_memcmp (C++ function), 102  
 lwgsm\_pbuf\_memfind (C++ function), 102  
 lwgsm\_pbuf\_new (C++ function), 100  
 lwgsm\_pbuf\_p (C++ type), 100  
 lwgsm\_pbuf\_ref (C++ function), 101  
 lwgsm\_pbuf\_set\_ip (C++ function), 103  
 lwgsm\_pbuf\_skip (C++ function), 103  
 lwgsm\_pbuf\_strcmp (C++ function), 102  
 lwgsm\_pbuf\_strfind (C++ function), 102  
 lwgsm\_pbuf\_t (C++ struct), 104, 127  
 lwgsm\_pbuf\_t::ip (C++ member), 104, 127  
 lwgsm\_pbuf\_t::len (C++ member), 104, 127  
 lwgsm\_pbuf\_t::next (C++ member), 104, 127  
 lwgsm\_pbuf\_t::payload (C++ member), 104, 127  
 lwgsm\_pbuf\_t::port (C++ member), 104, 127  
 lwgsm\_pbuf\_t::ref (C++ member), 104, 127  
 lwgsm\_pbuf\_t::tot\_len (C++ member), 104, 127  
 lwgsm\_pbuf\_take (C++ function), 100  
 lwgsm\_port\_t (C++ type), 115  
 lwgsm\_reset (C++ function), 145  
 lwgsm\_reset\_with\_delay (C++ function), 146  
 lwgsm\_set\_func\_mode (C++ function), 146  
 lwgsm\_sim\_get\_current\_state (C++ function), 108  
 lwgsm\_sim\_pin\_add (C++ function), 108  
 lwgsm\_sim\_pin\_change (C++ function), 109  
 lwgsm\_sim\_pin\_enter (C++ function), 108  
 lwgsm\_sim\_pin\_remove (C++ function), 108  
 lwgsm\_sim\_puk\_enter (C++ function), 109  
 lwgsm\_sim\_state\_t (C++ enum), 107  
 lwgsm\_sim\_state\_t::LWGSM\_SIM\_STATE\_NOT\_INSERTED (C++ enumerator), 107  
 lwgsm\_sim\_state\_t::LWGSM\_SIM\_STATE\_NOT\_READY (C++ enumerator), 107  
 lwgsm\_sim\_state\_t::LWGSM\_SIM\_STATE\_PH\_PIN (C++ enumerator), 107  
 lwgsm\_sim\_state\_t::LWGSM\_SIM\_STATE\_PH\_PUK (C++ enumerator), 108  
 lwgsm\_sim\_state\_t::LWGSM\_SIM\_STATE\_PIN (C++ enumerator), 107  
 lwgsm\_sim\_state\_t::LWGSM\_SIM\_STATE\_PUK (C++ enumerator), 107  
 lwgsm\_sim\_state\_t::LWGSM\_SIM\_STATE\_READY (C++ enumerator), 107  
 lwgsm\_sim\_t (C++ struct), 135  
 lwgsm\_sim\_t::state (C++ member), 135  
 lwgsm\_sms\_delete (C++ function), 111  
 lwgsm\_sms\_delete\_all (C++ function), 111  
 lwgsm\_sms\_disable (C++ function), 110  
 lwgsm\_sms\_enable (C++ function), 110  
 lwgsm\_sms\_entry\_t (C++ struct), 113  
 lwgsm\_sms\_entry\_t::data (C++ member), 114  
 lwgsm\_sms\_entry\_t::datetime (C++ member), 114  
 lwgsm\_sms\_entry\_t::length (C++ member), 114  
 lwgsm\_sms\_entry\_t::mem (C++ member), 114  
 lwgsm\_sms\_entry\_t::name (C++ member), 114  
 lwgsm\_sms\_entry\_t::number (C++ member), 114  
 lwgsm\_sms\_entry\_t::pos (C++ member), 114  
 lwgsm\_sms\_entry\_t::status (C++ member), 114  
 lwgsm\_sms\_list (C++ function), 111  
 lwgsm\_sms\_mem\_t (C++ struct), 112, 134  
 lwgsm\_sms\_mem\_t::current (C++ member), 113, 134  
 lwgsm\_sms\_mem\_t::mem\_available (C++ member), 113, 134  
 lwgsm\_sms\_mem\_t::total (C++ member), 113, 134  
 lwgsm\_sms\_mem\_t::used (C++ member), 113, 134  
 lwgsm\_sms\_read (C++ function), 110  
 lwgsm\_sms\_send (C++ function), 110  
 lwgsm\_sms\_set\_preferred\_storage (C++ function), 112  
 lwgsm\_sms\_status\_t (C++ enum), 109  
 lwgsm\_sms\_status\_t::LWGSM\_SMS\_STATUS\_ALL (C++ enumerator), 109  
 lwgsm\_sms\_status\_t::LWGSM\_SMS\_STATUS\_INBOX (C++ enumerator), 110  
 lwgsm\_sms\_status\_t::LWGSM\_SMS\_STATUS\_READ (C++ enumerator), 109  
 lwgsm\_sms\_status\_t::LWGSM\_SMS\_STATUS\_SENT (C++ enumerator), 109  
 lwgsm\_sms\_status\_t::LWGSM\_SMS\_STATUS\_UNREAD (C++ enumerator), 109  
 lwgsm\_sms\_status\_t::LWGSM\_SMS\_STATUS\_UNSENT (C++ enumerator), 110  
 lwgsm\_sms\_t (C++ struct), 113, 134  
 lwgsm\_sms\_t::enabled (C++ member), 113, 135  
 lwgsm\_sms\_t::mem (C++ member), 113, 135  
 lwgsm\_sms\_t::ready (C++ member), 113, 135  
 lwgsm\_sys\_init (C++ function), 156

- lwgsm\_sys\_mbox\_create (C++ function), 158
- lwgsm\_sys\_mbox\_delete (C++ function), 158
- lwgsm\_sys\_mbox\_get (C++ function), 159
- lwgsm\_sys\_mbox\_getnow (C++ function), 159
- lwgsm\_sys\_mbox\_invalid (C++ function), 159
- lwgsm\_sys\_mbox\_isvalid (C++ function), 159
- LWGSM\_SYS\_MBOX\_NULL (C macro), 160
- lwgsm\_sys\_mbox\_put (C++ function), 159
- lwgsm\_sys\_mbox\_putnow (C++ function), 159
- lwgsm\_sys\_mbox\_t (C++ type), 161
- lwgsm\_sys\_mutex\_create (C++ function), 156
- lwgsm\_sys\_mutex\_delete (C++ function), 156
- lwgsm\_sys\_mutex\_invalid (C++ function), 157
- lwgsm\_sys\_mutex\_isvalid (C++ function), 157
- lwgsm\_sys\_mutex\_lock (C++ function), 157
- LWGSM\_SYS\_MUTEX\_NULL (C macro), 160
- lwgsm\_sys\_mutex\_t (C++ type), 161
- lwgsm\_sys\_mutex\_unlock (C++ function), 157
- lwgsm\_sys\_now (C++ function), 156
- lwgsm\_sys\_protect (C++ function), 156
- lwgsm\_sys\_sem\_create (C++ function), 157
- lwgsm\_sys\_sem\_delete (C++ function), 157
- lwgsm\_sys\_sem\_invalid (C++ function), 158
- lwgsm\_sys\_sem\_isvalid (C++ function), 158
- LWGSM\_SYS\_SEM\_NULL (C macro), 160
- lwgsm\_sys\_sem\_release (C++ function), 158
- lwgsm\_sys\_sem\_t (C++ type), 161
- lwgsm\_sys\_sem\_wait (C++ function), 158
- lwgsm\_sys\_thread\_create (C++ function), 160
- lwgsm\_sys\_thread\_fn (C++ type), 161
- LWGSM\_SYS\_THREAD\_PRIO (C macro), 161
- lwgsm\_sys\_thread\_prio\_t (C++ type), 161
- LWGSM\_SYS\_THREAD\_SS (C macro), 161
- lwgsm\_sys\_thread\_t (C++ type), 161
- lwgsm\_sys\_thread\_terminate (C++ function), 160
- lwgsm\_sys\_thread\_yield (C++ function), 160
- LWGSM\_SYS\_TIMEOUT (C macro), 161
- lwgsm\_sys\_unprotect (C++ function), 156
- LWGSM\_SZ (C macro), 143
- lwgsm\_t (C++ struct), 137
- lwgsm\_t::buff (C++ member), 137
- lwgsm\_t::dev\_present (C++ member), 137
- lwgsm\_t::evt (C++ member), 137
- lwgsm\_t::evt\_func (C++ member), 137
- lwgsm\_t::f (C++ member), 137
- lwgsm\_t::initialized (C++ member), 137
- lwgsm\_t::ll (C++ member), 137
- lwgsm\_t::locked\_cnt (C++ member), 137
- lwgsm\_t::m (C++ member), 137
- lwgsm\_t::mbox\_process (C++ member), 137
- lwgsm\_t::mbox\_producer (C++ member), 137
- lwgsm\_t::msg (C++ member), 137
- lwgsm\_t::sem\_sync (C++ member), 137
- lwgsm\_t::status (C++ member), 137
- lwgsm\_t::thread\_process (C++ member), 137
- lwgsm\_t::thread\_produce (C++ member), 137
- LWGSM\_THREAD\_PROCESS\_HOOK (C macro), 151
- LWGSM\_THREAD\_PRODUCER\_HOOK (C macro), 151
- lwgsm\_timeout\_add (C++ function), 115
- lwgsm\_timeout\_fn (C++ type), 114
- lwgsm\_timeout\_remove (C++ function), 115
- lwgsm\_timeout\_t (C++ struct), 115
- lwgsm\_timeout\_t::arg (C++ member), 115
- lwgsm\_timeout\_t::fn (C++ member), 115
- lwgsm\_timeout\_t::next (C++ member), 115
- lwgsm\_timeout\_t::time (C++ member), 115
- LWGSM\_U16 (C macro), 142
- lwgsm\_u16\_to\_hex\_str (C macro), 143
- lwgsm\_u16\_to\_str (C macro), 143
- LWGSM\_U32 (C macro), 142
- lwgsm\_u32\_to\_gen\_str (C++ function), 145
- lwgsm\_u32\_to\_hex\_str (C macro), 143
- lwgsm\_u32\_to\_str (C macro), 143
- LWGSM\_U8 (C macro), 142
- lwgsm\_u8\_to\_hex\_str (C macro), 144
- lwgsm\_u8\_to\_str (C macro), 144
- lwgsm\_unicode\_t (C++ struct), 138, 140
- lwgsm\_unicode\_t::ch (C++ member), 138, 140
- lwgsm\_unicode\_t::r (C++ member), 138, 140
- lwgsm\_unicode\_t::res (C++ member), 138, 140
- lwgsm\_unicode\_t::t (C++ member), 138, 140
- LWGSM\_UNUSED (C macro), 142
- lwgsm\_ussd\_run (C++ function), 140
- lwgsml\_unicode\_decode (C++ function), 140
- lwgsmr\_t (C++ enum), 124
- lwgsmr\_t::lwgsmlCLOSED (C++ enumerator), 125
- lwgsmr\_t::lwgsmlCONT (C++ enumerator), 125
- lwgsmr\_t::lwgsmlERR (C++ enumerator), 124
- lwgsmr\_t::lwgsmlERRBLOCKING (C++ enumerator), 125
- lwgsmr\_t::lwgsmlERRCONNFAIL (C++ enumerator), 125
- lwgsmr\_t::lwgsmlERRCONNTIMEOUT (C++ enumerator), 125
- lwgsmr\_t::lwgsmlERRMEM (C++ enumerator), 125
- lwgsmr\_t::lwgsmlERRNOAP (C++ enumerator), 125
- lwgsmr\_t::lwgsmlERRNODEVICE (C++ enumerator), 125
- lwgsmr\_t::lwgsmlERRNOFREECONN (C++ enumerator), 125
- lwgsmr\_t::lwgsmlERRNOIP (C++ enumerator), 125
- lwgsmr\_t::lwgsmlERRNOTENABLED (C++ enumerator), 125
- lwgsmr\_t::lwgsmlERRPASS (C++ enumerator), 125
- lwgsmr\_t::lwgsmlERRWIFINOTCONNECTED (C++ enumerator), 125
- lwgsmr\_t::lwgsmlINPROG (C++ enumerator), 125

`lwgsmr_t::lwgsmsOK` (*C++ enumerator*), [124](#)  
`lwgsmr_t::lwgsmsOKIGNOREMORE` (*C++ enumerator*), [124](#)  
`lwgsmr_t::lwgsmsPARERR` (*C++ enumerator*), [124](#)  
`lwgsmr_t::lwgsmsTIMEOUT` (*C++ enumerator*), [125](#)