WGM110 API Reference Manual

This document contains the full API reference for the Wizard Gecko WGM110 Wi-Fi Module, firmware version 1.2.0 .

The Wizard Gecko family of the Silicon Labs' Wi-Fi modules deliver a high-performance, low energy and easy-to-use Wi-Fi solution integrated into a small form factor package. Wizard Gecko Wi-Fi modules combine an integrated antenna, a high-performance Wi-Fi transceiver, an energy efficient 32-bit MCU and ready to use Wi-Fi software and SDK.

KEY POINTS

- Wizard Gecko SDK version 1.2.0 API Reference Manual for the WGM110 Wi-Fi Module

Table of Contents

1. Introduction to Wizard Gecko Software Architecture 4

1.1 Wizard Gecko Wi-Fi BGAPI™ Protocol 4

1.2 Wizard Gecko BGLib™ Library 6

1.3 Wizard Gecko BGScript™ Scripting Language 6

1.4 Wizard Gecko Wi-FI Stack and Endpoints 7

1.4.1 Using Endpoints 8

1.4.2 Predefined Endpoints 9

2. API Definition 10

2.1 BGAPI Protocol Definition 10

2.1.1 Packet Format 10

2.1.2 Message Types 10

2.1.3 Command Class IDs 11

2.1.4 Packet Exchange. 11

2.1.5 Introduction to BGAPI over SPI 11

2.2 BGLib Functions Definition 12

2.3 BGScript API Definition 12

2.4 Data Types 13

3. API Reference 15

3.1 Configuration Command Class 15

3.1.1 config commands 15

3.1.2 config events 18

3.2 Device Firmware Upgrade (DFU) Command Class 20

3.2.1 dfu commands 21

3.2.2 dfu events 25

3.3 Endpoint Command Class 26

3.3.1 endpoint commands. 27

3.3.2 endpoint events 39

3.3.3 endpoint enumerations 43

3.4 Persistent Store Command Class. 44

3.4.1 flash commands 44

3.4.2 flash events 51

3.4.3 flash defines 52

3.5 Hardware Command Class 54

3.5.1 hardware commands 55

3.5.2 hardware events 81

3.5.3 hardware enumerations 83

3.6 HTTP Server Command Class. 85

3.6.1 https commands 85

3.6.2 https events 91

3.6.3 https enumerations 97

3.7 I2C Command Class 98

3.7.1 i2c commands 98
3.8 SDCard Command Class 102
3.8.1 sdhc commands 103
3.8.2 sdhc events 116
3.9 Wi-Fi Command Class 120
3.9.1 sme commands 121
3.9.2 sme events 167
3.9.3 sme enumerations 195
3.10 System Class Commands 196
3.10.1 system commands 197
3.10.2 system events 201
3.10.3 system enumerations 203
3.11 TCP Stack Command Class 205
3.11.1 tcpip commands 206
3.11.2 tcpip events 245
3.12 Utilities for BGScript Command Class 259
3.12.1 util commands 259
3.13 X.509 Command Class 266
3.13.1 x509 commands 266
3.13.2 x509 events 275
3.13.3 x509 enumerations 278
3.14 Error codes 278
4. Document Revision History 281

1. Introduction to Wizard Gecko Software Architecture

The Silicon Labs Wizard Gecko WGM110 Wi-Fi Module contains a complete 802.11 MAC and IP networking stack, providing everything required for creating wireless devices that works together with existing Wi-Fi infrastructure and devices.

Wizard Gecko software supports three different modes of use:

- **Standalone mode:** all software including the application software runs on the Wizard Gecko Module.
- **Hosted mode:** an external host runs the application software which controls the Wizard Gecko Device using a well-defined binary based transport protocol called BGAPI™.
- **Mixed mode:** a part of the application runs on the Wizard Gecko Module while the rest of the application runs on an external host. In mixed mode commands are accepted from both host and BGScript, and when command is invoked in BGScript its response is received also on host interface (responses for command received from host are not received in BGScript). Care must be taken in designing mixed mode application so that BGScritp and host do not collide, so that correct and desired action is taken.

In all of these cases, the Wizard Gecko software provides a complete 802.11 MAC and IP networking stack, additional 802.11 or IP stack software is not required, which enables simple and fast application development.

The figure below shows both the standalone and mixed mode stack architectures.

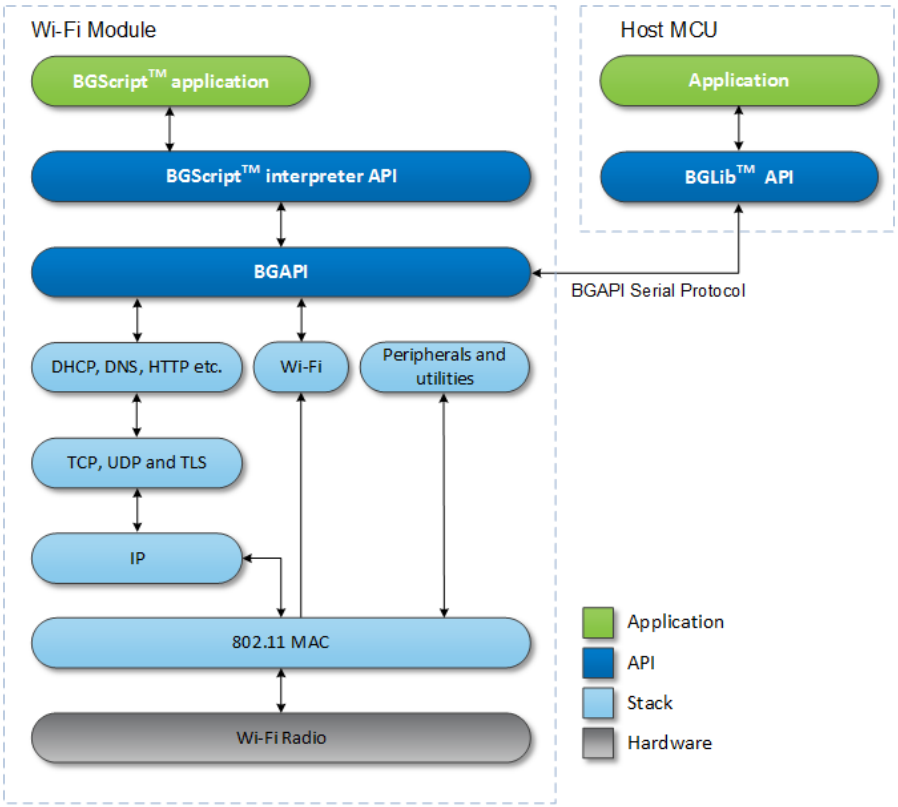

Figure 1.1. Wizard Gecko Software Architecture

1.1 Wizard Gecko Wi-Fi BGAPI™ Protocol

For applications, where a separate host (MCU) is used to implement the end user application, a transport protocol is needed between the host and the Wizard Gecko Wi-Fi stack. This transport protocol is used to communicate with the Wi-Fi stack as well to transmit and receive data packets. This protocol is called BGAPI and it is a binary communication protocol designed specifically to ease integration with external host devices having limited resources.

The BGAPI protocol provides access to layers (also referred to as command classes) listed in the following table.

Table 1.1. BGAPI Protocol Layers (Command Classes)

BGAPI Layer	Description
System	Various system functions, such as querying the hardware status or system reset

BGAPI Layer	Description
Configuration	Provides access to the device parameters such as the MAC address
TCP/IP	Gives access to the TCP/IP stack and various protocols like TCP and UDP
SME	Provides access to 802.11 MAC and procedures like Access Point discovery
Endpoint	Provides functions for controlling the data and peripheral endpoints
Hardware	An interface for accessing the various hardware modules such as timers and ADC
Persistent Store	Allows the user to read/write from/to non-volatile memory
Device Firmware Upgrade	Provides access to firmware update functions
I2C	An interface for accessing I2C functionality
HTTP Server	Provides functions for interfacing with the built-in HTTP Server
X.509	Allows the user to manage the X.509 certificate store
SD Card	Gives access to the built-in SD card functions
Util	Provides helper functions for BGScript development

Note: BGAPI protocol is designed to be used over UART, SPI or USB.

The BGAPI protocol is a **command - response** type protocol similar to AT commands. BGAPI uses binary format instead of the ASCII format. The basic principle of BGAPI messaging is shown in the the next figure. The user must wait for a response after sending each command and before issuing the next command. Some commands also invoke events. The BGAPI packet format uses a single bit to identify between commands/responses (0x08) and events (0x88) as shown in the figure below.

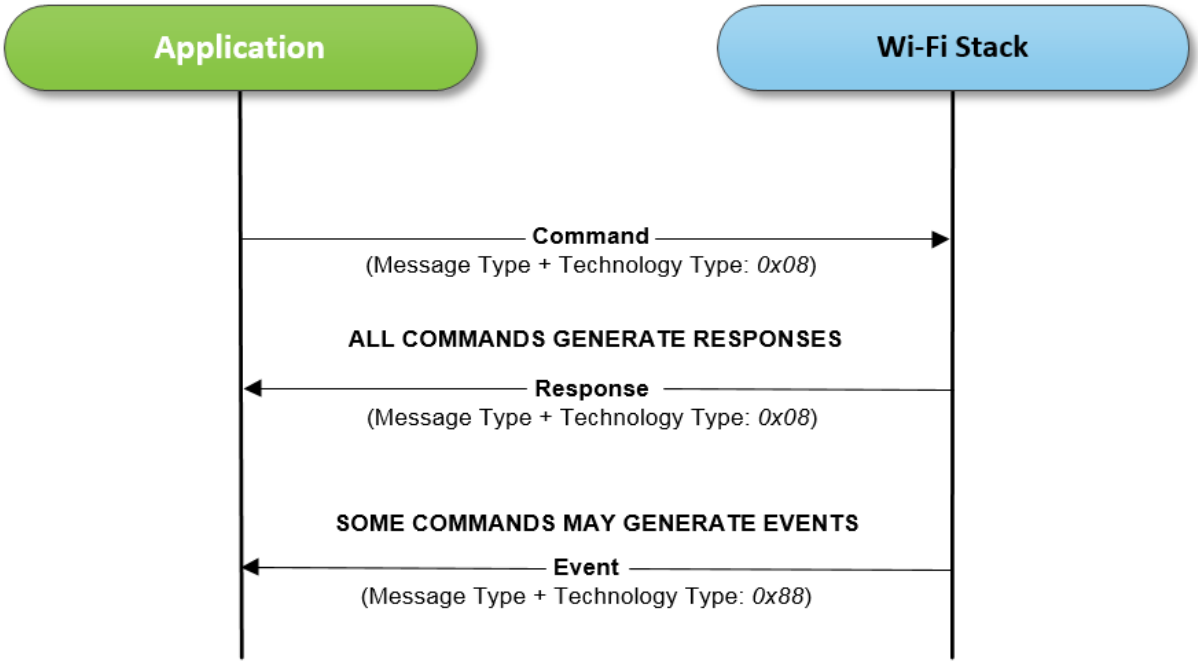

Figure 1.2. BGAPI Messaging is Based on the Command - Response - Event Principle

Note: When sending commands over BGAPI you always have to wait for the BGAPI response before issuing a new command to the Module.

1.2 Wizard Gecko BGLib™ Library

To enable easy development using the BGAPI protocol, a C host library is provided. This library is based on easily portable ANSI C code, which is delivered as part of the Wizard Gecko Software Development Kit (SDK). The purpose of the BGLib library is to simplify application development within different host environments.

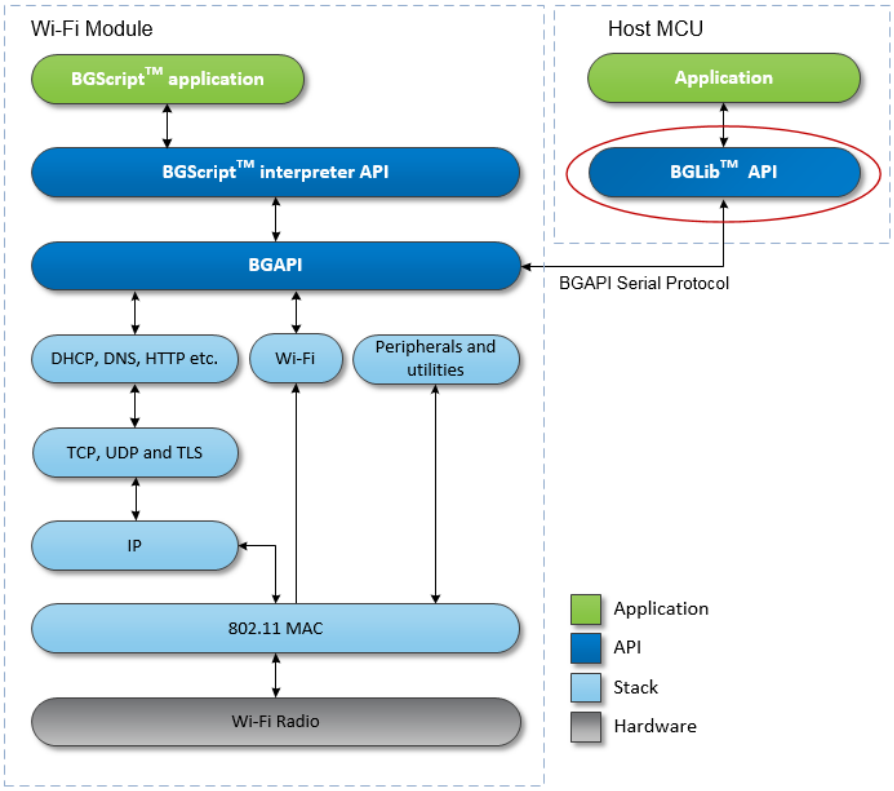

Figure 1.3. Purpose of BGLib in the Wizard Gecko Software Architecture

1.3 Wizard Gecko BGScript™ Scripting Language

Wizard Gecko software allows BGScript application developers to create standalone devices without the need for a separate host. The Wizard Gecko Module can run BGScript applications along the Module software stack and this is beneficial in cases where the end product size, cost and current consumption needs to be minimized. BGScript provides access to the same software and hardware interfaces as the BGAPI protocol. BGScript code can be developed and compiled with free tools provided as part of the Wizard Gecko Software Development Kit.

The purpose of the BGScript scripting language in the Wizard Gecko Software Architecture is to enable software applications to run inside the Module and without the need for an external host, as shown in the following figure.

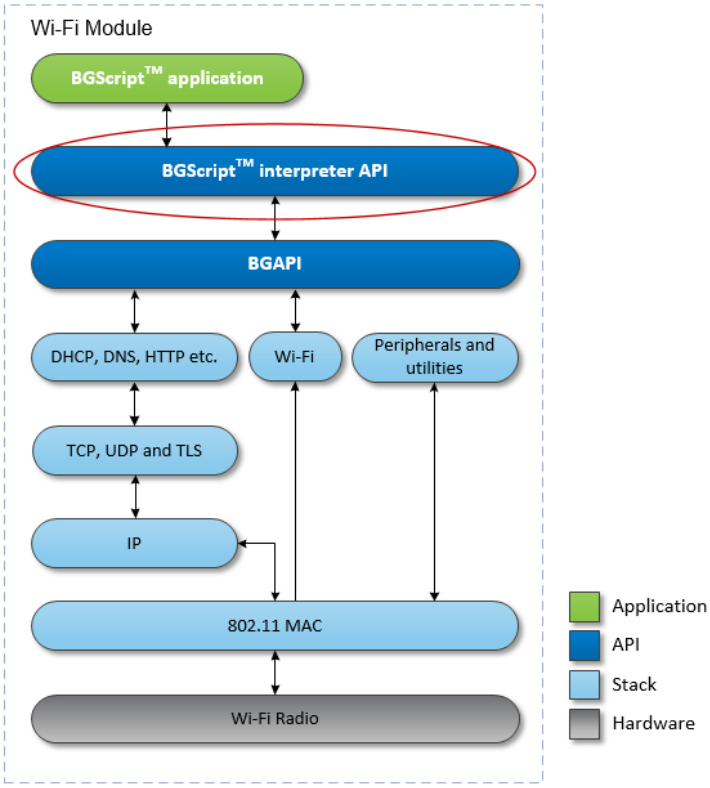

Figure 1.4. Purpose of BGScript in the Wizard Gecko Software Architecture

For more information on BGScript, its use and example applications please see the *UG170: Wizard Gecko BGScript™ User's Guide*. This guide also contains a short description on the benefits and limitations of BGScript as a guidance to designers contemplating the question whether to start application development based on a standalone or host architecture approach.

1.4 Wizard Gecko Wi-FI Stack and Endpoints

This section describes how endpoints are used in the Wizard Gecko Wi-Fi stack.

1.4.1 Using Endpoints

Endpoints is the mechanism to route data traffic between destinations, i.e. it is a flexible tool to manage from where the data originates and where it will be sent. Each endpoint has an *Endpoint type* (e.g. UART, TCP Server or BGScript) enumerated by the *Endpoint type number*, which defines the endpoint characteristics.

Each time a new endpoint is created it gets a unique *Endpoint index* assigned by the Wi-Fi module. Some Endpoint types have fixed Endpoint indexes, while others get the index number generated on a demand basis. Examples of endpoints with fixed index numbers are UART, USB and BGScript.

A good example of a dynamically allocated Endpoint index is when a new TCP Server is created. Upon an incoming TCP connection the Wi-Fi Module will look at what the next available Endpoint index is, and give the new connection the first available index. This TCP/IP connection is also called a socket. In the future if any data needs to be written to that socket, it is done by writing to the specific Endpoint index. Since the socket also receives incoming data, the endpoint for where that data is routed is configurable.

It is important to understand that with bi-directional peripherals, such as a serial port or a TCP socket where data can move both into and out of the Wi-Fi Module, the connection is configured by defining the source and destination for each endpoint link. For example, in order to route the data in both directions between a serial port and a TCP socket, the serial port is configured to have the socket as its endpoint, and the socket is configured to have the serial port as its endpoint.

A typical peripheral (SPI, UART or USB) endpoint will automatically route data to its configured endpoint using BGAPI framing. However, for UART endpoints, it is possible to make the streaming (raw data) endpoint by setting the UART parameter api to "false" in the project configuration file. During runtime streaming mode enabling is done using the BGAPI command `cmd_endpoint_set_streaming`.

Setting SPI,UART or USB endpoint in streaming mode will cause the module to send every received byte on this endpoint to be sent to its defined destination and this endpoint will transmit every byte from endpoints that have defined it as their destination (however it will not be possible to distinguish the source of the data). For example: Assuming that UART endpoint is set in streaming mode and its destination is set to TCP client endpoint. TCP endpoint destination is set to UART endpoint. All bytes that are received in UART will be sent to TCP endpoint and further via TCP protocol to TCP server and all data received on TCP client will be transmitted in UART without any additional framing.

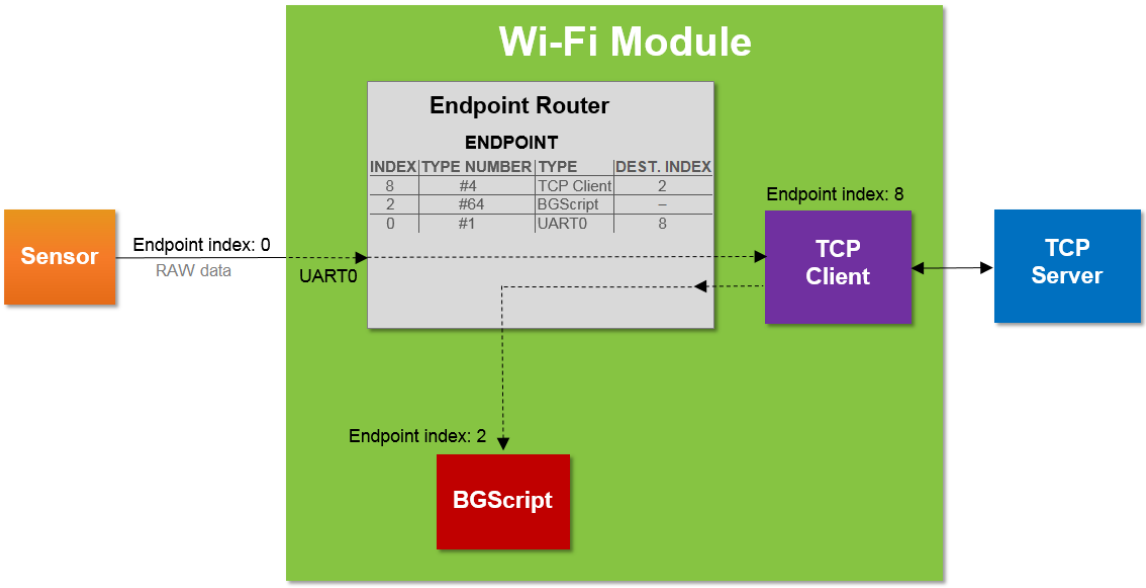

Figure 1.5. Bi-Directional Streaming in Standalone Mode

If an external microcontroller (MCU) is used, the setup could look quite similar. In the next figure, the data from the Server would get sent to the MCU, and the sensor would send its data straight to the Server.

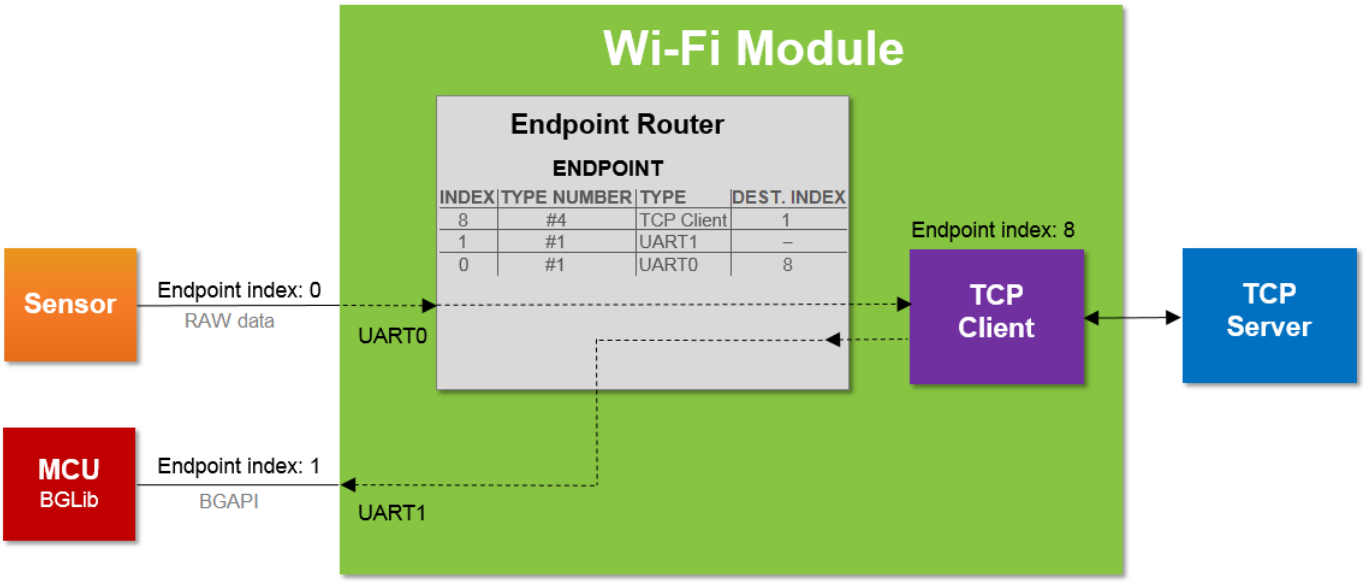

Figure 1.6. Bi-Directional Streaming in Mixed Mode

1.4.2 Predefined Endpoints

Wizard Gecko Wi-Fi stack contains several predefined endpoints which are automatically reserved to provide access to some of the Module's hardware interfaces.

For more information on endpoints see the related Command Class Description section *Endpoints* in this manual.

Table 1.2. Endpoint Mappings

Endpoint type	Endpoint type number	Endpoint index	Endpoint index details
Free	0	0 - 30	-
UART	1	USART0: 0 USART1: 1	Depends on the USART module used
USB	2	3	-
TCP Client	4	0 - 30	Dynamically assigned
TCP Server	8	0 - 30	Dynamically assigned
UDP Client	16	0 - 30	Dynamically assigned
UDP Server	32	0 - 30	Dynamically assigned
BGScript	64	2	-
Waiting for closing	128	0 - 30	-
SPI	256	USART0: 0 USART1: 1	Depends on the USART module used
Drop	1024	31	-
TLS Client	2048	0 - 30	Dynamically assigned

2. API Definition

This section contains the generic Wizard Gecko Wi-Fi software API definition. The definition consists of three parts:

- BGAPI protocol definition
- BGLib C library description
- BGScript scripting API description

This section of the document only provides the generic definition and description of the API. The actual commands, responses and events are described in the *API Reference* section of this manual.

2.1 BGAPI Protocol Definition

For applications, where a separate host is used to implement the application, a transport protocol is required between the host and the Wizard Gecko Module. The transport protocol is used for controlling the Module software as well as for transmitting and receiving data packets. This protocol is called BGAPI. It is a binary based communication protocol designed specifically for ease of implementation within host devices with limited resources.

The BGAPI protocol is available on these peripheral interfaces:

- **UART**
- **SPI** (Wizard Gecko Module as the slave and external host as the master)
- **USB** (Wizard Gecko Module as the device and external host as the host)

Note: The BGAPI transport protocol must explicitly be configured to be in use for the intended host interface, when using the Wi-Fi module in Network Co-Processor mode. For more details see the *Module Configuration User's Guide* of the Module.

2.1.1 Packet Format

Packets in either direction use the format indicated in the table below.

Table 2.1. BGAPI Packet Format

Octet	Octet bits	Length	Description
Octet 0	7	**Message Type (MT)**	0: Command/Response 1: Event
-	6:3	**Technology Type (TT)**	0001: Wi-Fi
-	2:0	**Length High (LH)**	Payload length (high bits)
Octet 1	7:0	**Length Low (LL)**	Payload length (low bits)
Octet 2	7:0	**Class ID (CID)**	Command class ID
Octet 3	7:0	**Command ID (CMD)**	Command ID
Octet 4 + N	-	**Payload (PL)**	Up to 2047 bytes of payload 0 ≤ N ≤ 2047 N = (LH << 8) | LL **Note:** N is an 11 bit value which is formed by left shifting LH and OR-ing with LL.

2.1.2 Message Types

BGAPI protocol includes the message types listed in the following table.

Note: Command and Response types have the same value by purpose.

Table 2.2. BGAPI Message Types

Message Type (MT) + Technology Type (TT)	Value	Description
Command	0x08	Command from the host to the Module
Response	0x08	Response from the Module to the host
Event	0x88	Event from the Module to the host

2.1.3 Command Class IDs

BGAPI contains the command classes listed in the following table. Each command class is described in more detail in the appropriate section later in this manual.

Table 2.3. BGAPI Command Classes

Class ID	Description	Explanation
0x00	DFU	Device Firmware Update
0x01	System	Common system control
0x02	Configuration	Module configuration
0x03	WiFi	Wi-Fi connection control
0x04	TCPIP	Module TCP/IP stack control
0x05	Endpoint	Endpoints handling
0x06	Hardware	MCU peripherals control
0x07	PS Store	Persistent Store handling
0x08	I2C	MCU I2C control
0x09	HTTP Server	Module internal HTTP Server
0x0B	X509	Certificate handling
0x0C	SD Card	microSD memory card handling
0x0D	Utilities for BGScript	BGScript utilities

2.1.4 Packet Exchange

Note: When sending commands over BGAPI you always have to wait for the BGAPI response before issuing a new command to the Module.

2.1.5 Introduction to BGAPI over SPI

When using SPI as a host interface to the Wi-Fi Module, the host controller is the master while the Wi-Fi module is the slave. SPI is a synchronous interface with the same clock driving both input and output. During BGAPI command sending, the host (master) should also read possible responses or events sent by the Module. The Module also informs the host that it has data to send using a specified notify GPIO pin and the host must generate the SPI clock by sending zeros (0's) to the Module.

Note: The Module will assert the configured notify pin to active state (from low to high) when it has data to send to the master. The notify pin will be set back to low when there are two bytes left to send to the host. The user should always read the header (4 bytes) and the payload, the size of which is declared in the header.

- **Command & Response**

1. Master sends Wi-Fi on command.

```
MASTER 08 00 03 00
```

```
SLAVE 00 00 00 00
```

2. Slave notifies master by using a notify bit (if configured).
3. Master reads response.

```
MASTER 00 00 00 00 00 00
```

```
SLAVE 08 02 03 00 00 00
```

- **Event**

1. Slave notifies master by using a notify bit (if configured).
2. Master reads event.

```
MASTER 00 00 00 00 00 00
```

```
SLAVE 88 02 03 00 00 00
```

2.2 BGLib Functions Definition

All the BGAPI commands are also available as ANSI C functions in a separate host library called BGLib. The library consists of ANSI C header files including macros for command sending and BGAPI message handling as well as all command, response and event message identifiers. BGLib function, data structure and identifier are documented in the API reference for each command as follows:

Command

```
/* Function */
void wifi_cmd_sme_wifi_on();

/* Response id */
wifi_rsp_sme_wifi_on_id

/* Response structure */
struct wifi_msg_sme_wifi_on_rsp_t
{
    uint16 result;
};
```

Event

```
/* Event id */
wifi_evt_sme_wifi_is_on

/* Event structure */
struct wifi_msg_sme_wifi_is_on_evt_t
{
    uint61 result;
};
```

2.3 BGScript API Definition

The BGScript functions are documented in the *API Reference* section. The format of the commands varies slightly from the C library functions. Instead of using callbacks the BGScript functions take the return values as parameters.

BGScript commands are documented as follows:

- **BGScript functions**

```
call system_hello()
```

The BGScript command parameters and return values are the same as used in the BGAPI binary protocol so they are not documented separately.

2.4 Data Types

The following table lists all available data types and their length in bytes and includes examples in both human readable and hexadecimal format.

Table 2.4. Data Types

Name	Length (bytes)	Description
hw_addr	6	MAC Address
		Example Human readable: 00:07:80:1A:2B:3C Packet data (hex): 00 07 80 1A 2B 3C
ipv4	4	IPv4 address in big endian format
		Example Human readable: 192.168.1.1 Packet data (hex): C0 A8 01 01
uint8array	1 - 256	Variable length byte array. The first byte defines the length *n* of the data that follows, n having the value 0 ≤ *n* ≤ 255.
		Example Human readable: “Hello” Packet data (hex): 05 48 65 6c 6c 6f
uint16array	2 - 2047	Variable length byte array. The first two bytes define the length *n* of the payload data that follows, n having the value 0 ≤ *n* ≤ 2045. Length field is in little endian format. **Note:** The payload data length is in bytes (not in multiples of 2)! **Note:** The upper limit 2045 is the theoretical maximum value for *n* because the actual maximum value depends on the other data in the message array (excluding the message header).
		Example Human readable: “World!” Packet data (hex): 06 00 57 6f 72 6c 64 21
uint8	1	8-bit unsigned integer
		Example Human readable: 167 Packet data (hex): a7

Name	Length (bytes)	Description	
int8	1	8-bit signed integer, 2's complement	
		Example	
		Human readable	-22
		Packet data (hex)	ea
uint16	2	16-bit unsigned integer	
		Example	
		Human readable	4567
		Packet data (hex)	d7 11
int16	2	16-bit signed integer, 2's complement	
		Example	
		Human readable	-4567
		Packet data (hex)	29 ee
uint32	4	32-bit unsigned integer	
		Example	
		Human readable	2864434397
		Packet data (hex)	dd cc bb aa
int32	4	32-bit signed integer, 2's complement	
		Example	
		Human readable	-45678
		Packet data (hex)	92 4d ff ff

3. API Reference

This section describes all commands, enumerations, responses, events and errors. Commands with related enumerations, responses and events are grouped according to command classes.

3.1 Configuration Command Class

The commands in this class are used to read and write the MAC address of the Module.

This class consists of the following items:

- *Commands*
  - `cmd_config_get_mac`
  - `cmd_config_set_mac`
- *Events*
  - `evt_config_mac_address`

Troubleshooting information

Problem

The command `cmd_config_get_mac` returns the address *"00:0b:57:ff:ff:ff"*.

Possible reason

The Module is totally erased after a flasher tool was used – contact the Silicon Labs Customer Support or set the MAC address always after a Module reset with the `cmd_config_set_mac` command. Avoid erasing the Module memory completely when using flasher tools.

3.1.1 config commands

3.1.1.1 cmd_config_get_mac

This command is used to read the MAC address of the device.

Table 3.1. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x02	class	Message class: Configuration
3	0x00	method	Message ID
4	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

Table 3.2. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x02	class	Message class: Configuration
3	0x00	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript command

```
call config_get_mac(hw_interface)(result, hw_interface)
```

BGLIB C API

```
/* Function */
void wifi_cmd_config_get_mac(uint8 hw_interface);

/* Response id */
wifi_rsp_config_get_mac_id

/* Response structure */
struct wifi_msg_config_get_mac_rsp_t
{
  uint16 result;,
  uint8 hw_interface;
};
```

Table 3.3. Events Generated

Event	Description
config_mac_address	Device MAC address

3.1.1.2 cmd_config_set_mac

This command is used to write the device MAC address.

The MAC address will be taken into use when the Wi-Fi radio is switched on using the sme_wifi_on command.

Table 3.4. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x07	lolen	Minimum payload length
2	0x02	class	Message class: Configuration
3	0x01	method	Message ID
4	uint8	hw_interface	Hardware interface • **0:** Wi-Fi
5-10	hw_addr	mac	The new MAC address to set

Table 3.5. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x02	class	Message class: Configuration
3	0x01	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript command

```
call config_set_mac(hw_interface, mac)(result, hw_interface)
```

BGLIB C API

```
/* Function */
void wifi_cmd_config_set_mac(uint8 hw_interface, hw_addr mac);

/* Response id */
wifi_rsp_config_set_mac_id

/* Response structure */
struct wifi_msg_config_set_mac_rsp_t
{
  uint16 result;,
  uint8 hw_interface;
};
```

3.1.2 config events

3.1.2.1 evt_config_mac_address

This event indicates the current MAC address of the device.

Table 3.6. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x07	lolen	Minimum payload length
2	0x02	class	Message class: Configuration
3	0x00	method	Message ID
4	uint8	hw_interface	Hardware interface • **0:** Wi-Fi
5-10	hw_addr	mac	The current MAC address

BGScript event

```
event config_mac_address(hw_interface, mac)
```

C Functions

```
/* Event id */
wifi_evt_config_mac_address_id

/* Event structure */
struct wifi_msg_config_mac_address_evt_t
{
  uint8 hw_interface;,
  hw_addr mac;
};
```

3.2 Device Firmware Upgrade (DFU) Command Class

The commands in this class are used to perform firmware updates over one of the host interfaces (e.g. UART, USB or SPI). DFU firmware updates are intended for field updates but not for development time updates.

This class consists of the following items:

- *Commands*
  - `cmd_dfu_flash_set_address`
  - `cmd_dfu_flash_upload`
  - `cmd_dfu_flash_upload_finish`
  - `cmd_dfu_reset`
- *Events*
  - `event_dfu_boot` indicates that the command `cmd_dfu_reset` was finished succesfully

Correct DFU upload sequence

1. Set the Module's power on. The configuration of the host serial communication is the same as the one configured during previous firmware load.
2. Send the `cmd_dfu_reset` command with the parameter *dfu mode (1)*.
3. Wait until the `event_dfu_boot` event is received.
4. Send the `cmd_dfu_flash_set_address` command with the address value *0* (this is the only possible value).
5. Upload the DFU image file by sending the `cmd_dfu_flash_upload` command.

   Note: Maximum payload size for a single `cmd_dfu_flash_upload` command is 128 bytes so typically multiple commands must be sent to upload the entire DFU image file.

   Note: Response time will vary according to several parameters related to the the flash memory write operation.

6. Repeat step 5 until the entire DFU image file has been loaded.
7. Send the `cmd_dfu_flash_upload_finish` command to finalize the loading of the flash.
8. Send *cmd_dfu_reset* command with the parameter *normal mode (0)* to leave the module upgrade.
9. Wait for the `system_event_boot` event.

   Note: Error response and module reset is performed, if delay in command sending is more than 3.5 seconds.

Troubleshooting information

Problem

No boot event received after issuing the `cmd_dfu_reset` command in DFU upload sequence (see step 3 of the DFU Upload sequence).

Possible reasons

- The Module is not powered up.
- The host serial communication (UART, SPI, USB) is not working, for example the UART bit rate has been configured erroneosly.
- The flash memory has been erased completely, use another tool, like SimplicityCommander from SDK, to load the DFU flash image file. For more details see document *QSG122: WGM110 Wi-Fi® Module Software Quick-Start Guide*.
- The Module has been damaged.

Problem

No boot event received after issuing the last `cmd_dfu_reset` command in DFU upload sequence (see step 9 of the DFU Upload sequence).

Possible reasons

- The host serial communication (UART, SPI, USB) is not working, for example the communication part has not been enabled or the UART bitrate was changed due to the settings defined in the DFU image file. Check UART, SPI or USB settings from the project configuration file (*project.xml)*.
- Verification of the DFU image has failed. Reset the device using the `cmd_dfu_reset` command and repeat the DFU Upload sequence.
- Incorrect DFU image. Rebuild the project and/or check that the correct image file is selected for DFU Upload.

Other comments

If the DFU Upload sequence is not completed, for example if the device is unpowered during the upload or if the `cmd_dfu_flash_upload_finish` command is not sent, there is a possibility that the configuration of the serial host communication interface is lost in which case the Module will enter into Recovery Mode. If the Module is in Recovery Mode only **UART0** in **Location 0** can be used for host communication.

Table 3.7. UART Default Settings in Recovery Mode

Parameter	Value
UART #	UART 0
Location	0
Bitrate	115200
Stop bits	1
Parity	None
Handshake	None

3.2.1 dfu commands

3.2.1.1 cmd_dfu_flash_set_address

This command is used to set the flash address offset for the DFU upgrade.

Table 3.8. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x01	method	Message ID
4-7	uint32	address	The flash address offset. Set to zero.

Table 3.9. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x01	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call dfu_flash_set_address(address)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_dfu_flash_set_address(uint32 address);

/* Response id */
wifi_rsp_dfu_flash_set_address_id

/* Response structure */
struct wifi_msg_dfu_flash_set_address_rsp_t
{
  uint16 result;
};
```

3.2.1.2 cmd_dfu_flash_upload

This command is used to upload a block of firmware data.

Table 3.10. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x02	method	Message ID
4	uint8array	data	Block of firmware data, up to 128 bytes

Table 3.11. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x02	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call dfu_flash_upload(data_len, data_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_dfu_flash_upload(uint8 data_len, const uint8 *data_data);

/* Response id */
wifi_rsp_dfu_flash_upload_id

/* Response structure */
struct wifi_msg_dfu_flash_upload_rsp_t
{
  uint16 result;
};
```

3.2.1.3 cmd_dfu_flash_upload_finish

This command is used to finish the DFU upgrade.

The command must be called once all firmware data has been uploaded.

Table 3.12. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x03	method	Message ID

Table 3.13. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x03	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call dfu_flash_upload_finish()(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_dfu_flash_upload_finish();

/* Response id */
wifi_rsp_dfu_flash_upload_finish_id

/* Response structure */
struct wifi_msg_dfu_flash_upload_finish_rsp_t
{
  uint16 result;
};
```

3.2.1.4 cmd_dfu_reset

This command is used to reset the device.

The command does not have a response, but it triggers a boot event.

Table 3.14. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x00	method	Message ID
4	uint8	dfu	Boot mode • **0:** Boot to normal mode • **1:** Boot to DFU mode

BGScript command

```
call dfu_reset(dfu)
```

BGLIB C API

```
/* Function */
void wifi_cmd_dfu_reset(uint8 dfu);

/* Command does not have a response */
```

Table 3.15. Events Generated

Event	Description
system_boot	Sent after the device has booted to normal mode
dfu_boot	Sent after the device has booted to DFU mode

3.2.2 dfu events

3.2.2.1 evt_dfu_boot

This event indicates that the device booted into DFU mode, and that it is ready to receive commands related to DFU firmware upgrade.

Table 3.16. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x04	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x00	method	Message ID
4-7	uint32	version	Version of the bootloader

BGScript event

```
event dfu_boot(version)
```

C Functions

```
/* Event id */
wifi_evt_dfu_boot_id

/* Event structure */
struct wifi_msg_dfu_boot_evt_t
{
  uint32 version;
};
```

3.3 Endpoint Command Class

The commands in this class are used to create and delete endpoints and to route data from and to an endpoint. Defined endpoint types are listed in the adjoining enumeration.

This class consists of the following items:

- *Commands*
  - `cmd_endpoint_close`
  - `cmd_endpoint_disable`
  - `cmd_endpoint_send`
  - `cmd_endpoint_set_active`
  - `cmd_endpoint_set_streaming`
  - `cmd_endpoint_set_streaming_destination`
  - `cmd_endpoint_set_transmit_size`
- *Events*
  - `evt_endpoint_closing`
  - `evt_endpoint_data`
  - `evt_endpoint_error`
  - `evt_endpoint_status`
  - `evt_endpoint_syntax_error`
- *Enumerations*
  - `enum_endpoint_type`

All host interfaces which have been configured for BGAPI use in the project configuration file (using parameter; api=”true”) will send BGAPI events generated by the Module and also can receive BGAPI commands. If more than one host interface has been configured into BGAPI mode, care must be taken to prevent problems related to issuance of parallel commands.

3.3.1 endpoint commands

3.3.1.1 cmd_endpoint_close

This command is used to close a protocol endpoint.

The command must only be used for protocol endpoints such as TCP, UDP and TLS endpoints.

TCP server endpoint close is denied when one or more clients are connected to the server

Table 3.17. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x04	method	Message ID
4	uint8	endpoint	Index of the endpoint to close **Range:** 0 - 30

Table 3.18. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x04	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	endpoint	Index of the endpoint closed **Range:** 0 - 30

BGScript command

```
call endpoint_close(endpoint)(result, endpoint)
```

BGLIB C API

```
/* Function */
void wifi_cmd_endpoint_close(uint8 endpoint);

/* Response id */
wifi_rsp_endpoint_close_id

/* Response structure */
struct wifi_msg_endpoint_close_rsp_t
{
  uint16 result;,
  uint8 endpoint;
};
```

Table 3.19. Events Generated

Event	Description
endpoint_status	Sent when the endpoint status changes

3.3.1.2 cmd_endpoint_disable

This command is used to disable an UART endpoint.

The command effectively switches off an UART interface until the device is reset or power-cycled. When an UART interface is disabled its pins go to high-impedance state.

Table 3.20. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x06	method	Message ID
4	uint8	endpoint	Index of the endpoint to disable **Range:** 0 - 30

Table 3.21. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x06	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	endpoint	Index of the endpoint disabled **Range:** 0 - 30

BGScript command

```
call endpoint_disable(endpoint)(result, endpoint)
```

BGLIB C API

```
/* Function */
void wifi_cmd_endpoint_disable(uint8 endpoint);

/* Response id */
wifi_rsp_endpoint_disable_id

/* Response structure */
struct wifi_msg_endpoint_disable_rsp_t
{
  uint16 result;,
  uint8 endpoint;
};
```

3.3.1.3 cmd_endpoint_send

This command is used to send data to a given endpoint.

Table 3.22. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x00	method	Message ID
4	uint8	endpoint	Index of the endpoint to which the data will be sent **Range:** 0 - 31
5	uint8array	data	Data to be sent

Table 3.23. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x00	method	Message ID
4-5	uint16	result	This code can be set as wifi_err_buffers_full. This means that there is not enough memory to handle this request. It needs to be resent some time later to be processed.
6	uint8	endpoint	Index of the endpoint where the data was sent **Range:** 0 - 31

BGScript command

```
call endpoint_send(endpoint, data_len, data_data)(result, endpoint)
```

BGLIB C API

```
/* Function */
void wifi_cmd_endpoint_send(uint8 endpoint, uint8 data_len, const uint8 *data_data);

/* Response id */
wifi_rsp_endpoint_send_id

/* Response structure */
struct wifi_msg_endpoint_send_rsp_t
{
  uint16 result;,
  uint8 endpoint;
};
```

3.3.1.4 cmd_endpoint_set_active

This command is used to activate or deactivate endpoints.

Configured endpoints are active by default, i.e., you can send data to them, and data can be received from them. This command allows you to temporarily halt the outgoing data from an endpoint by deactivating it. For example, deactivating a BGAPI UART endpoint will prevent BGAPI events and responses from being sent out of the UART interface. Similarly, deactivating the BGScript endpoint will prevent events from being passed to the script, thus preventing the calls from being executed.

Table 3.24. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x02	method	Message ID
4	uint8	endpoint	Index of the endpoint to configure **Range:** 0 - 30
5	uint8	active	Endpoint status • **0:** inactive • **1:** active

Table 3.25. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x02	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	endpoint	Index of the endpoint configured **Range:** 0 - 30

BGScript command

```
call endpoint_set_active(endpoint, active)(result, endpoint)
```

BGLIB C API

```
/* Function */
void wifi_cmd_endpoint_set_active(uint8 endpoint, uint8 active);

/* Response id */
wifi_rsp_endpoint_set_active_id

/* Response structure */
struct wifi_msg_endpoint_set_active_rsp_t
```

```
{
  uint16 result;,
  uint8 endpoint;
};
```

Table 3.26. Events Generated

Event	Description
endpoint_status	Sent when the endpoint status changes

3.3.1.5 cmd_endpoint_set_streaming

This command is used to switch an endpoint between streaming or BGAPI modes.

When an endpoint is in streaming mode, received data is passed unmodified to another endpoint such as TCP and sent data written as-is to a peripheral interface. In BGAPI mode, received data is handled as BGAPI protocol messages and sent data is encapsulated in evt_endpoint_data events.

This command can currently be used only with UART endpoints. If the switch from streaming to BGAPI mode is made while there is incoming or outgoing UART data, data loss may occur.

Table 3.27. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x01	method	Message ID
4	uint8	endpoint	Index of the endpoint to configure **Range:** 0 - 30
5	uint8	streaming	Endpoint mode • **0:** BGAPI mode • **1:** streaming mode

Table 3.28. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x01	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	endpoint	Index of the endpoint configured **Range:** 0 - 30

BGScript command

```
call endpoint_set_streaming(endpoint, streaming)(result, endpoint)
```

BGLIB C API

```
/* Function */
void wifi_cmd_endpoint_set_streaming(uint8 endpoint, uint8 streaming);

/* Response id */
wifi_rsp_endpoint_set_streaming_id
```

```
/* Response structure */
struct wifi_msg_endpoint_set_streaming_rsp_t
{
  uint16 result;,
  uint8 endpoint;
};
```

Table 3.29. Events Generated

Event	Description
endpoint_status	Sent when the endpoint status changes

3.3.1.6 cmd_endpoint_set_streaming_destination

This command is used to set the destination endpoint to which the received data from an endpoint will be routed to.

The command is only applicable for endpoints that can be configured in streaming mode.

Table 3.30. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x03	method	Message ID
4	uint8	endpoint	Index of the endpoint in streaming mode to configure **Range:** 0 - 30
5	int8	destination_endpoint	Index of the destination endpoint • **-1:** routed to all BGAPI/BGScript endpoints • **0 - 30:** routed to a specific endpoint • **31:** received data is discarded

Table 3.31. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x03	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	endpoint	Index of the endpoint configured **Range:** 0 - 30

BGScript command

```
call endpoint_set_streaming_destination(endpoint, destination_endpoint)(result, endpoint)
```

BGLIB C API

```
/* Function */
void wifi_cmd_endpoint_set_streaming_destination(uint8 endpoint, int8 destination_endpoint);

/* Response id */
wifi_rsp_endpoint_set_streaming_destination_id

/* Response structure */
struct wifi_msg_endpoint_set_streaming_destination_rsp_t
{
  uint16 result;,
```

```
  uint8 endpoint;
};
```

Table 3.32. Events Generated

Event	Description
endpoint_status	Sent when the endpoint status changes

3.3.1.7 cmd_endpoint_set_transmit_size

This command is used to set the desired transmit size of a protocol endpoint.

If defined, the endpoint will buffer outgoing data until transmit size is reached and send the data to remote end. This is meant for UDP endpoints and must not be used with any other type of endpoint, including TCP. Transmit size is not set by default.

Table 3.33. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x05	method	Message ID
4	uint8	endpoint	Index of the endpoint to configure **Range:** 0 - 30
5-6	uint16	size	Transmit size • **0:** no defined size • **1 - 1472:** defined size

Table 3.34. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x05	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	endpoint	Index of the endpoint configured **Range:** 0 - 30

BGScript command

```
call endpoint_set_transmit_size(endpoint, size)(result, endpoint)
```

BGLIB C API

```
/* Function */
void wifi_cmd_endpoint_set_transmit_size(uint8 endpoint, uint16 size);

/* Response id */
wifi_rsp_endpoint_set_transmit_size_id

/* Response structure */
struct wifi_msg_endpoint_set_transmit_size_rsp_t
{
  uint16 result;,
```

```
  uint8 endpoint;
};
```

3.3.2 endpoint events

3.3.2.1 evt_endpoint_closing

This event indicates that a protocol endpoint has been closed.

The event must be acknowledged by calling the endpoint_close command or otherwise the device will not re-use the endpoint index.

This event is only applicable to dynamically assigned endpoints such as TCP, UDP or TLS.

Table 3.35. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x03	method	Message ID
4-5	uint16	reason	Reason of endpoint close, unsigned 16-bit integer in **little endian** format • **0:** no error • **non-zero:** an error occurred For other values refer to the error codes.
6	uint8	endpoint	Index of the endpoint **Range:** 0 - 30

BGScript event

```
event endpoint_closing(reason, endpoint)
```

C Functions

```
/* Event id */
wifi_evt_endpoint_closing_id

/* Event structure */
struct wifi_msg_endpoint_closing_evt_t
{
  uint16 reason;,
  uint8 endpoint;
};
```

3.3.2.2 evt_endpoint_data

This event contains received data from an endpoint.

Table 3.36. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x02	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x01	method	Message ID
4	uint8	endpoint	Index of the endpoint which received the data **Range:** 0 - 30
5	uint8array	data	The received data

BGScript event

```
event endpoint_data(endpoint, data_len, data_data)
```

C Functions

```
/* Event id */
wifi_evt_endpoint_data_id

/* Event structure */
struct wifi_msg_endpoint_data_evt_t
{
  uint8 endpoint;,
  uint8array data;
};
```

3.3.2.3 evt_endpoint_error

This event indicates an error in an endpoint.

Table 3.37. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x04	method	Message ID
4-5	uint16	reason	Error reason For values refer to the error codes.
6	uint8	endpoint	Index of the endpoint that generated the event **Range:** 0 - 30

BGScript event

```
event endpoint_error(reason, endpoint)
```

C Functions

```
/* Event id */
wifi_evt_endpoint_error_id

/* Event structure */
struct wifi_msg_endpoint_error_evt_t
{
  uint16 reason;,
  uint8 endpoint;
};
```

3.3.2.4 evt_endpoint_status

This event indicates the status of an endpoint.

Additionally, this event will be sent after connecting and disconnecting USB cable.

Table 3.38. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x09	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x02	method	Message ID
4	uint8	endpoint	Index of the endpoint **Range:** 0 - 30
5-8	uint32	type	Type of the endpoint
9	uint8	streaming	Endpoint mode • **0:** BGAPI mode • **1:** streaming mode
10	int8	destination	Index of the endpoint in to which the received data is routed to • **-1:** routed to all BGAPI/BGScript endpoints • **0 - 30:** routed to a specific endpoint • **31:** received data is discarded
11	uint8	active	Endpoint status • **0:** inactive • **1:** active
12	uint8	flags	Flags indicating internal state of the endpoint

BGScript event

```
event endpoint_status(endpoint, type, streaming, destination, active, flags)
```

C Functions

```
/* Event id */
wifi_evt_endpoint_status_id

/* Event structure */
struct wifi_msg_endpoint_status_evt_t
{
  uint8 endpoint;,
  uint32 type;,
  uint8 streaming;,
  int8 destination;,
  uint8 active;,
  uint8 flags;
};
```

3.3.2.5 evt_endpoint_syntax_error

This event indicates that an endpoint detected a BGAPI protocol error in the received data.

The event is triggered if a BGAPI command from the external host contains syntax error(s) or if a command is only partially received. The BGAPI parser has a 1 second command timeout, and if a valid command is not received within this timeout, an error is raised and the partial or wrong command is ignored.

Table 3.39. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x00	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	endpoint	Index of the endpoint that generated the event **Range:** 0 - 30

BGScript event

```
event endpoint_syntax_error(result, endpoint)
```

C Functions

```
/* Event id */
wifi_evt_endpoint_syntax_error_id

/* Event structure */
struct wifi_msg_endpoint_syntax_error_evt_t
{
  uint16 result;,
  uint8 endpoint;
};
```

3.3.3 endpoint enumerations

3.3.3.1 enum_endpoint_type

This enumeration defines the endpoint types.

Table 3.40. Enumerations

Value	Name	Description
0	endpoint_type_free	Endpoint is not in use
1	endpoint_type_uart	UART
2	endpoint_type_usb	USB
4	endpoint_type_tcp	TCP client
8	endpoint_type_tcp_server	TCP server
16	endpoint_type_udp	UDP client
32	endpoint_type_udp_server	UDP server
64	endpoint_type_script	Scripting
128	endpoint_type_wait_close	Waiting for closing
256	endpoint_type_spi	SPI
1024	endpoint_type_drop	All received data is discarded
2048	endpoint_type_tls	TLS client

3.4 Persistent Store Command Class

The commands in this class are related to reading and saving data of the Module's internal non-volatile 4 KB memory (NVM). This memory is duplicated to avoid lost data. The structure of the storage is based on "key – value" pairs, where "key" is a unique identifier of the related data. Key id's above address "0x8000" are reserved for user data. An enumeration defines the system PS keys.

The Module has a minimum internal flash erase cycle value of 20000 before errors. Thus the total amount of data that can be saved in to the NVM totals 8 MB. This must be taken into consideration when erasing data from the NVM frequently. The erase cycle life time depends on the amount and variation of saved data, on the saving frequency and on the total memory usage of the NVM.

This class consists of the following items:

- *Commands*
  - `cmd_flash_ps_defrag`
  - `cmd_flash_ps_dump`
  - `cmd_flash_ps_erase`
  - `cmd_flash_ps_erase_all`
  - `cmd_flash_ps_load`
  - `cmd_flash_ps_save`
- *Events*
  - `evt_flash_ps_key` carries the value of the saved PS Key.
  - `evt_flash_ps_key_changed` is sent whenever a saved PS Key value is changed.

Recommendations

The NVM of the Module is defragmented automatically during normal operation so the command `cmd_flash_ps_defrag` is rarely needed. However, to keep the response time of the `cmd_flash_ps_save` command deterministic, it is recommended that the defragment command `cmd_flash_ps_defrag` is called every now and then.

3.4.1 flash commands

3.4.1.1 cmd_flash_ps_defrag

This command is used to manually initiate the defragmentation of the Persistent Store.

The Persistent Store is automatically defragmented if there is not enough space when adding a PS key.

Table 3.41. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x00	method	Message ID

Table 3.42. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x00	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call flash_ps_defrag()(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_flash_ps_defrag();

/* Response id */
wifi_rsp_flash_ps_defrag_id

/* Response structure */
struct wifi_msg_flash_ps_defrag_rsp_t
{
  uint16 result;
};
```

3.4.1.2 cmd_flash_ps_dump

This command is used to dump all the PS keys from the Persistent Store.

The command will generate a series of PS key events. The last PS key event is identified by the key index value 65535, indicating that the dump has finished listing all PS keys.

Table 3.43. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x01	method	Message ID

Table 3.44. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x01	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call flash_ps_dump()(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_flash_ps_dump();

/* Response id */
wifi_rsp_flash_ps_dump_id

/* Response structure */
struct wifi_msg_flash_ps_dump_rsp_t
{
  uint16 result;
};
```

Table 3.45. Events Generated

Event	Description
flash_ps_key	Sent for each PS key in the Persistent Store

3.4.1.3 cmd_flash_ps_erase

This command is used to erase a single PS key and its value from the Persistent Store.

Table 3.46. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x05	method	Message ID
4-5	uint16	key	Key index • **0 - 32767:** system keys • **32768 - 65534:** user keys

Table 3.47. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x05	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call flash_ps_erase(key)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_flash_ps_erase(uint16 key);

/* Response id */
wifi_rsp_flash_ps_erase_id

/* Response structure */
struct wifi_msg_flash_ps_erase_rsp_t
{
  uint16 result;
};
```

Table 3.48. Events Generated

Event	Description
flash_ps_key_changed	Sent when a PS key changes in the Persistent Store

3.4.1.4 cmd_flash_ps_erase_all

This command is used to erase all PS keys from the Persistent Store.

The command removes all system and user keys except the MAC address.

Table 3.49. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x02	method	Message ID

Table 3.50. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x02	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call flash_ps_erase_all()(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_flash_ps_erase_all();

/* Response id */
wifi_rsp_flash_ps_erase_all_id

/* Response structure */
struct wifi_msg_flash_ps_erase_all_rsp_t
{
  uint16 result;
};
```

3.4.1.5 cmd_flash_ps_load

This command is used to retrieve the value of a PS key from the Persistent Store.

Table 3.51. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x04	method	Message ID
4-5	uint16	key	Key index • **0 - 32767:** system keys • **32768 - 65534:** user keys

Table 3.52. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x04	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8array	value	Key value **Length:** 0 - 255 bytes

BGScript command

```
call flash_ps_load(key)(result, value_len, value_data)
```

BGLIB C API

```
/* Function */
void wifi_cmd_flash_ps_load(uint16 key);

/* Response id */
wifi_rsp_flash_ps_load_id

/* Response structure */
struct wifi_msg_flash_ps_load_rsp_t
{
  uint16 result;,
  uint8array value;
};
```

3.4.1.6 cmd_flash_ps_save

This command is used to store a value in the Persistent Store.

Table 3.53. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x03	method	Message ID
4-5	uint16	key	Key index • **0 - 32767:** system keys • **32768 - 65534:** user keys
6	uint8array	value	Key value **Length:** 0 - 255 bytes

Table 3.54. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x03	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call flash_ps_save(key, value_len, value_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_flash_ps_save(uint16 key, uint8 value_len, const uint8 *value_data);

/* Response id */
wifi_rsp_flash_ps_save_id

/* Response structure */
struct wifi_msg_flash_ps_save_rsp_t
{
  uint16 result;
};
```

Table 3.55. Events Generated

Event	Description
flash_ps_key_changed	Sent when a PS key changes in the Persistent Store

3.4.2 flash events

3.4.2.1 evt_flash_ps_key

This event indicates the value of a PS key in the Persistent Store.

Table 3.56. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x00	method	Message ID
4-5	uint16	key	Key index • **0 - 32767:** system keys • **32768 - 65534:** user keys
6	uint8array	value	Key value **Length:** 0 - 255 bytes

BGScript event

```
event flash_ps_key(key, value_len, value_data)
```

C Functions

```
/* Event id */
wifi_evt_flash_ps_key_id

/* Event structure */
struct wifi_msg_flash_ps_key_evt_t
{
  uint16 key;,
  uint8array value;
};
```

3.4.2.2 evt_flash_ps_key_changed

This event indicates a PS key has changed in the Persistent Store.

Event is generated by BGAPI command as well as module internal Persist Store operation.

Table 3.57. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x01	method	Message ID
4-5	uint16	key	Key index • **0 - 32767:** system keys • **32768 - 65534:** user keys

BGScript event

```
event flash_ps_key_changed(key)
```

C Functions

```
/* Event id */
wifi_evt_flash_ps_key_changed_id

/* Event structure */
struct wifi_msg_flash_ps_key_changed_evt_t
{
  uint16 key;
};
```

3.4.3 flash defines

3.4.3.1 define_flash_ps_keys

This enumeration defines the system PS keys.

Table 3.58. Defines

Value	Name	Description
1	FLASH_ps_key_mac	Device MAC address
2	FLASH_ps_key_ipv4_settings	Device IP configuration
3	FLASH_ps_key_dns0_settings	Device primary DNS server
4	FLASH_ps_key_dns1_settings	Device secondary DNS server
5	FLASH_ps_key_module_service	Device operating mode
20	FLASH_ps_key_ap_ssid	Access Point mode SSID
21	FLASH_ps_key_ap_channel	Access Point mode channel
22	FLASH_ps_key_ap_pw	Access point mode password
23	FLASH_ps_key_ap_wifi_n	Access Point mode 802.11n support enabled or disabled
24	FLASH_ps_key_ap_security	Access Point mode security mode • **0:** open security • **1:** WPA security • **2:** WPA2 security • **3:** WEP security
25	FLASH_ps_key_client_ssid	Client mode SSID
26	FLASH_ps_key_client_pw	Client mode password
30	FLASH_ps_key_dhcps_enable	DHCP server enabled or disabled
31	FLASH_ps_key_dhcps_space	First client IPv4 address assigned by the DHCP server
32	FLASH_ps_key_dhcps_disable_routing	DHCP server gateway and DNS router options enabled or disabled
33	FLASH_ps_key_dhcps_mask	DHCP server IPv4 address mask
34	FLASH_ps_key_dhcps_leasetime	DHCP server IPv4 address lease timeout
35	FLASH_ps_key_dnss_enable	DNS server enabled or disabled
36	FLASH_ps_key_dnss_url	DNS server URL
37	FLASH_ps_key_dnss_any_url	DNS server reply to any URL enabled or disabled
38	FLASH_ps_key_eap_buffer_size	Buffer size used for eap handshake. Value should be within 0x400 and 0x1800 range.
60	FLASH_ps_key_tcp_client_port_range	TCP Client random local port range
61	FLASH_ps_key_disable_arp_check	Disable ARP checking mechanism
65535	FLASH_ps_key_eof	Last PS key index

3.5 Hardware Command Class

The commands in this class are used to configure and control peripherals and timers of the Module MCU. There are three enumerations which are used for defining the GPIO (General Purpose Input Output) pin mode (input, output, pull-up, pull-down, filter etc.), the GPIO pin trigger mode, and the ADC input channel.

This class consists of the following items:

GPIO related

- *Commands*
  - `cmd_hardware_configure_gpio`
  - `cmd_hardware_configure_gpio_interrupt`
  - `cmd_hardware_write_gpio`
  - `cmd_hardware_read_gpio`
- *Events*
  - `evt_hardware_interrupt`

ADC related

- *Commands*
  - `cmd_hardware_adc_read`

Timer related

- *Commands*
  - `cmd_hardware_set_soft_timer`
  - `cmd_hardware_timer_init`
  - `cmd_hardware_timer_initcc`
  - `cmd_hardware_rtc_init`
  - `cmd_hardware_rtc_set_time`
  - `cmd_hardware_rtc_get_time`
  - `cmd_hardware_rtc_set_alarm`
- *Events*
  - `evt_hardware_rtc_alarm`
  - `evt_hardware_soft_timer`

SPI related

- *Commands*
  - `cmd_hardware_spi_transfer`

UART related

- *Commands*
  - `cmd_hardware_uart_conf_set`
  - `cmd_hardware_uart_conf_get`
- *Events*
  - `evt_hardware_uart_conf`

Examples

1. Reading a value of an ADC input channel.
   a. Enable the ADC in project configuration by selecting a suitable reference voltage source.
   b. Compile the project and load the firmware into the Module.
   c. Read the ADC value using the command `cmd_hardware_adc_read` with the ADC input channel as a parameter.
   d. The value of the analog voltage at the input of the selected ADC channel will be returned in the response.
2. Using an RTC alarm.
   a. Initialize the RTC using the `cmd_hardware_rtc_init` command.
   b. Set the date and time of the RTC with `cmd_hardware_rtc_set_time` command.
   c. Wait for the `evt_hardware_rtc_alarm` event.

   Note: RTC date and time are lost on the module power down.

3. Using the PWM (Pulse Width Modulation) output.
    a. Initialize the hardware timer with the `cmd_hardware_timer_init` command and set the PWM frequency.
    b. Set the PWM pulse ratio with the `cmd_hardware_timer_initcc` command.
4. Using GPIO as an external interrupt.
    a. Configure the GPIO pin into *input* mode using the `cmd_hardware_configure_gpio` command.
    b. Configure the same GPIO pin as an *interrupt* using the `cmd_hardware_configure_gpio_interrupt`.
    c. You can detect an external interrupt occurrance by catching the `evt_hardware_interrupt` event.
5. Using a GPIO pin as an input.
    a. Configure the GPIO pin into *input* mode using the `cmd_hardware_configure_gpio` command.
    b. Read the status of the GPIO pin defined in step 1 above using the `cmd_hardware_read_gpio` command.

        Note: This command requires that port and pin are defined as masks.

    c. Pin status is returned in the response of *cmd_hardware_read_gpio* command.
6. Using a GPIO pin as an output.
    a. Configure the GPIO pin into *output* mode using the `cmd_hardware_configure_gpio.` command.
    b. Set or clear the pin with the `cmd_hardware_write_gpio` command.

        Note: This command requires that port and pin are defined as masks.

3.5.1 hardware commands

3.5.1.1 cmd_hardware_adc_read

This command is used to trigger a single ADC sample conversion.

Conversion parameters are 12-bit resolution and 32 clock cycle conversion time.

Table 3.59. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x07	method	Message ID
4	uint8	input	ADC input

Table 3.60. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x05	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x07	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	input	ADC input
7-8	uint16	value	ADC sample value

BGScript command

```
call hardware_adc_read(input)(result, input, value)
```

BGLIB C API

```
/* Function */
void wifi_cmd_hardware_adc_read(uint8 input);

/* Response id */
wifi_rsp_hardware_adc_read_id

/* Response structure */
struct wifi_msg_hardware_adc_read_rsp_t
{
  uint16 result;,
  uint8 input;,
  uint16 value;
};
```

3.5.1.2 cmd_hardware_configure_gpio

This command is used to configure the mode of a specified I/O pin.

Table 3.61. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x01	method	Message ID
4	uint8	port	Port index • **0:** PA • **1:** PB • **2:** PC • **3:** PD • **4:** PE • **5:** PF
5	uint8	pin	Pin within the defined port **Range:** 0 - 15
6	uint8	mode	Pin mode
7	uint8	output	Pin default function Values in mode 0: • **0:** tri-stated • **1:** tri-stated with pull-up enabled Values in mode 1: • **0:** glitch filter disabled • **1:** glitch filter enabled Values in mode 2: • **0:** pull-down enabled • **1:** pull-up enabled Values in mode 3: • **0:** pull-down enabled • **1:** pull-up enabled Values in mode 4: • **0:** output low • **1:** output high

Table 3.62. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x01	method	Message ID

Byte	Type	Name	Description
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call hardware_configure_gpio(port, pin, mode, output)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_hardware_configure_gpio(uint8 port, uint8 pin, uint8 mode, uint8 output);

/* Response id */
wifi_rsp_hardware_configure_gpio_id

/* Response structure */
struct wifi_msg_hardware_configure_gpio_rsp_t
{
  uint16 result;
};
```

3.5.1.3 cmd_hardware_configure_gpio_interrupt

This command is used to configure an interrupt on a specified I/O pin.

Table 3.63. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x02	method	Message ID
4	uint8	port	Port index • **0:** PA • **1:** PB • **2:** PC • **3:** PD • **4:** PE • **5:** PF
5	uint8	pin	Pin within the defined port **Range:** 1 - 15
6	uint8	trigger	Trigger for the interrupt

Table 3.64. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x02	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call hardware_configure_gpio_interrupt(port, pin, trigger)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_hardware_configure_gpio_interrupt(uint8 port, uint8 pin, uint8 trigger);

/* Response id */
wifi_rsp_hardware_configure_gpio_interrupt_id

/* Response structure */
struct wifi_msg_hardware_configure_gpio_interrupt_rsp_t
{
```

```
  uint16 result;
};
```

3.5.1.4 cmd_hardware_dbg_interface_set_active

This command is used to disable or enable debug interface. Disabling debug interface allows user to use pins PF0 and PF1 as GPIO. By default debug interface is enabled.

Caution when using as this might lead to module being unable to be accessed via debug interface.

Table 3.65. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0f	method	Message ID
4	uint8	active	Debug interface status • **0:** inactive • **1:** active (default)

Table 3.66. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0f	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call hardware_dbg_interface_set_active(active)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_hardware_dbg_interface_set_active(uint8 active);

/* Response id */
wifi_rsp_hardware_dbg_interface_set_active_id

/* Response structure */
struct wifi_msg_hardware_dbg_interface_set_active_rsp_t
{
  uint16 result;
};
```

3.5.1.5 cmd_hardware_read_gpio

This command is used to read the logic state of pins of a specified I/O-port.

Table 3.67. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x04	method	Message ID
4	uint8	port	Port index • **0:** PA • **1:** PB • **2:** PC • **3:** PD • **4:** PE • **5:** PF
5-6	uint16	mask	Bitmask of the pins within in the defined port to read. If a bit is zero, the corresponding pin is ignored. If a bit is one, the corresponding pin is read.

Table 3.68. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x04	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x04	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6-7	uint16	data	Bitmask of the pin state for the pins in the requeset mask. If a bit is zero, the corresponding pin is low. If a bit is one, the corresponding pin is high. The state of the pins that are not part of the request mask is zero.

BGScript command

```
call hardware_read_gpio(port, mask)(result, data)
```

BGLIB C API

```
/* Function */
void wifi_cmd_hardware_read_gpio(uint8 port, uint16 mask);

/* Response id */
```

```
wifi_rsp_hardware_read_gpio_id

/* Response structure */
struct wifi_msg_hardware_read_gpio_rsp_t
{
  uint16 result;,
  uint16 data;
};
```

3.5.1.6 cmd_hardware_rtc_get_time

This command is used to read the current Real Time Clock (RTC) time.

Table 3.69. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0a	method	Message ID

Table 3.70. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x0a	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0a	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6-7	int16	year	Current year **Range:** 2000 - 2099
8	int8	month	Current month **Range:** 1 - 12
9	int8	day	Current day **Range:** 1 - 31
10	int8	weekday	Current weekday • **0:** Sunday • **1:** Monday • **2:** Tuesday • **3:** Wednesday • **4:** Thursday • **5:** Friday • **6:** Saturday
11	int8	hour	Current hour **Range:** 0 - 23
12	int8	minute	Current minute **Range:** 0 - 59
13	int8	second	Current second **Range:** 0 - 59

BGScript command

```
call hardware_rtc_get_time()(result, year, month, day, weekday, hour, minute, second)
```

BGLIB C API

```
/* Function */
void wifi_cmd_hardware_rtc_get_time();

/* Response id */
wifi_rsp_hardware_rtc_get_time_id

/* Response structure */
struct wifi_msg_hardware_rtc_get_time_rsp_t
{
  uint16 result;,
  int16 year;,
  int8 month;,
  int8 day;,
  int8 weekday;,
  int8 hour;,
  int8 minute;,
  int8 second;
};
```

3.5.1.7 cmd_hardware_rtc_init

This command is used to enable or disable the Real Time Clock.

The clock must be initialized using the hardware_rtc_set_time command before it can be used.

Table 3.71. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x08	method	Message ID
4	uint8	enable	RTC state • **0:** disabled • **1:** enabled

Table 3.72. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x08	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call hardware_rtc_init(enable)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_hardware_rtc_init(uint8 enable);

/* Response id */
wifi_rsp_hardware_rtc_init_id

/* Response structure */
struct wifi_msg_hardware_rtc_init_rsp_t
{
  uint16 result;
};
```

3.5.1.8 cmd_hardware_rtc_set_alarm

This command is used to set an alarm for the Real Time Clock.

Only one alarm can be active at one time. The alarm can be disabled by disabling the Real Time Clock using the hardware_rtc_init command.

Table 3.73. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x07	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0b	method	Message ID
4-5	int16	year	Current year **Range:** 2000 - 2099
6	int8	month	Current month **Range:** 1 - 12
7	int8	day	Current day **Range:** 1 - 31
8	int8	hour	Current hour **Range:** 0 - 23
9	int8	minute	Current minute **Range:** 0 - 59
10	int8	second	Current second **Range:** 0 - 59

Table 3.74. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0b	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call hardware_rtc_set_alarm(year, month, day, hour, minute, second)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_hardware_rtc_set_alarm(int16 year, int8 month, int8 day, int8 hour, int8 minute, int8 second);

/* Response id */
wifi_rsp_hardware_rtc_set_alarm_id

/* Response structure */
struct wifi_msg_hardware_rtc_set_alarm_rsp_t
{
  uint16 result;
};
```

Table 3.75. Events Generated

Event	Description
hardware_rtc_alarm	Sent when the RTC alarm is triggered

3.5.1.9 cmd_hardware_rtc_set_time

This command is used to set the current Real Time Clock time.

Table 3.76. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x07	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x09	method	Message ID
4-5	int16	year	Current year **Range:** 2000 - 2099
6	int8	month	Current month **Range:** 1 - 12
7	int8	day	Current day **Range:** 1 - 31
8	int8	hour	Current hour **Range:** 0 - 23
9	int8	minute	Current minute **Range:** 0 - 59
10	int8	second	Current second **Range:** 0 - 59

Table 3.77. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x09	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call hardware_rtc_set_time(year, month, day, hour, minute, second)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_hardware_rtc_set_time(int16 year, int8 month, int8 day, int8 hour, int8 minute, int8 second);
```

```
/* Response id */
wifi_rsp_hardware_rtc_set_time_id

/* Response structure */
struct wifi_msg_hardware_rtc_set_time_rsp_t
{
  uint16 result;
};
```

3.5.1.10 cmd_hardware_set_soft_timer

This command is used to enable a software timer.

Multiple concurrent timers can be running simultaneously as long as an unique handle is used for each.

Table 3.78. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x06	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x00	method	Message ID
4-7	uint32	time	Interval between events in milliseconds. If 0, the timer is disabled.
8	uint8	handle	Timer handle **Range:** 0 - 255
9	uint8	single_shot	Timer repeat • **0:** a repeating timer • **1:** a single-shot timer

Table 3.79. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x00	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call hardware_set_soft_timer(time, handle, single_shot)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_hardware_set_soft_timer(uint32 time, uint8 handle, uint8 single_shot);

/* Response id */
wifi_rsp_hardware_set_soft_timer_id

/* Response structure */
struct wifi_msg_hardware_set_soft_timer_rsp_t
{
  uint16 result;
};
```

Table 3.80. Events Generated

Event	Description
hardware_soft_timer	Sent after the specified interval has elapsed

3.5.1.11 cmd_hardware_spi_transfer

This command is used to transfer data using an SPI interface.

The command can only be used when the interface is in SPI master mode. Due to synchronous full duplex nature of SPI, the command response will contain the same amount of read bytes as was written.

Table 3.81. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0e	method	Message ID
4	uint8	channel	SPI channel to use
5	uint8array	data	Data to write

Table 3.82. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0e	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8array	data	Data read

BGScript command

```
call hardware_spi_transfer(channel, data_len, data_data)(result, data_len, data_data)
```

BGLIB C API

```
/* Function */
void wifi_cmd_hardware_spi_transfer(uint8 channel, uint8 data_len, const uint8 *data_data);

/* Response id */
wifi_rsp_hardware_spi_transfer_id

/* Response structure */
struct wifi_msg_hardware_spi_transfer_rsp_t
{
  uint16 result;,
  uint8array data;
};
```

3.5.1.12 cmd_hardware_timer_init

This command is used to configure a hardware timer.

The timer clock signal is derived from the peripheral clock signal using the prescale factor. The timer counter is increased by one at every clock cycle. When the timer reaches the configured maximum value, it wraps around to zero and continues counting.

Table 3.83. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x06	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x05	method	Message ID
4	uint8	index	Index of the timer **Range:** 0 - 1
5	uint8	location	Location of the timer pins **Range:** 0 - 6
6-7	uint16	prescale	Prescale factor • **1:** corresponds to 48 MHz • **2:** corresponds to 24 MHz • **4:** corresponds to 12 MHz • **8:** corresponds to 6 MHz • **16:** corresponds to 3 MHz • **32:** corresponds to 1500 kHz • **64:** corresponds to 750 kHz • **128:** corresponds to 375 kHz • **256:** corresponds to 187.5 kHz • **512:** corresponds to 93.75 kHz • **1024:** corresponds to 46.875 kHz
8-9	uint16	top_value	Top value of the timer

Table 3.84. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x05	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call hardware_timer_init(index, location, prescale, top_value)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_hardware_timer_init(uint8 index, uint8 location, uint16 prescale, uint16 top_value);

/* Response id */
wifi_rsp_hardware_timer_init_id

/* Response structure */
struct wifi_msg_hardware_timer_init_rsp_t
{
  uint16 result;
};
```

3.5.1.13 cmd_hardware_timer_initcc

This command is used to configure a hardware timer compare channel.

In PWM mode, the compare channel pin will be held high until the timer reaches the configured compare value, at which point the pin will be set low until the timer reaches the configured maximum value. After reaching the maximum value, the timer wraps around to zero and the cycle restarts.

Table 3.85. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x06	method	Message ID
4	uint8	index	Index of the timer **Range:** 0 - 1
5	uint8	channel	Compare channel **Range:** 0 - 2
6	uint8	mode	Compare mode • **0:** disabled • **1:** PWM
7-8	uint16	compare_value	Value to compare the timer against

Table 3.86. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x06	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call hardware_timer_initcc(index, channel, mode, compare_value)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_hardware_timer_initcc(uint8 index, uint8 channel, uint8 mode, uint16 compare_value);

/* Response id */
wifi_rsp_hardware_timer_initcc_id

/* Response structure */
struct wifi_msg_hardware_timer_initcc_rsp_t
```

```
{
  uint16 result;
};
```

3.5.1.14 cmd_hardware_uart_conf_get

This command is used to read the current configuration of a UART interface.

Table 3.87. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0d	method	Message ID
4	uint8	id	Uart id

Table 3.88. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0d	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call hardware_uart_conf_get(id)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_hardware_uart_conf_get(uint8 id);

/* Response id */
wifi_rsp_hardware_uart_conf_get_id

/* Response structure */
struct wifi_msg_hardware_uart_conf_get_rsp_t
{
  uint16 result;
};
```

Table 3.89. Events Generated

Event	Description
hardware_uart_conf	UART configuration

3.5.1.15 cmd_hardware_uart_conf_set

This command is used to configure a UART interface.

Table 3.90. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x09	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0c	method	Message ID
4	uint8	id	Uart id UART must be enabled in hw configuration
5-8	uint32	rate	Data rate in bps **Range:**9600 - 6000000
9	uint8	data_bits	Data bits 8 or 9
10	uint8	stop_bits	Stop bits 1 or 2
11	uint8	parity	Parity 0=none, 1=odd, 2=even
12	uint8	flow_ctrl	Flow control 0=none, 1=rts/cts, 2=rts

Table 3.91. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0c	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call hardware_uart_conf_set(id, rate, data_bits, stop_bits, parity, flow_ctrl)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_hardware_uart_conf_set(uint8 id, uint32 rate, uint8 data_bits, uint8 stop_bits, uint8 parity,
uint8 flow_ctrl);

/* Response id */
wifi_rsp_hardware_uart_conf_set_id

/* Response structure */
struct wifi_msg_hardware_uart_conf_set_rsp_t
{
```

```
  uint16 result;
};
```

3.5.1.16 cmd_hardware_write_gpio

This command is used to set the logic state of pins on a specified I/O-port.

Table 3.92. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x03	method	Message ID
4	uint8	port	Port index • **0:** PA • **1:** PB • **2:** PC • **3:** PD • **4:** PE • **5:** PF
5-6	uint16	mask	Bitmask of the pins within the defined port to write. If a bit is zero, the corresponding pin is ignored. If a bit is one, the corresponding pin is written.
7-8	uint16	data	Bitmask of the pin state for the pins in the request mask. If a bit is zero, the corresponding pin is set low. If a bit is one, the corresponding pin is set high. The state of the pins that are not part of the request mask is not changed.

Table 3.93. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x03	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call hardware_write_gpio(port, mask, data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_hardware_write_gpio(uint8 port, uint16 mask, uint16 data);
```

```
/* Response id */
wifi_rsp_hardware_write_gpio_id

/* Response structure */
struct wifi_msg_hardware_write_gpio_rsp_t
{
  uint16 result;
};
```

3.5.2 hardware events

3.5.2.1 evt_hardware_interrupt

This event indicates that one or more I/O interrupts have been triggered.

Table 3.94. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x08	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x01	method	Message ID
4-7	uint32	interrupts	Bitmask of the triggered interrupts. If a bit is one, the corresponding interrupt was triggered.
8-11	uint32	timestamp	Timestamp when the event was generated. The timestamp represents the time in milliseconds since the last reset.

BGScript event

```
event hardware_interrupt(interrupts, timestamp)
```

C Functions

```
/* Event id */
wifi_evt_hardware_interrupt_id

/* Event structure */
struct wifi_msg_hardware_interrupt_evt_t
{
  uint32 interrupts;,
  uint32 timestamp;
};
```

3.5.2.2 evt_hardware_rtc_alarm

This event indicates an RTC alarm has occurred.

Table 3.95. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x00	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x02	method	Message ID

BGScript event

```
event hardware_rtc_alarm()
```

C Functions

```
/* Event id */
wifi_evt_hardware_rtc_alarm_id

/* Event structure */
struct wifi_msg_hardware_rtc_alarm_evt_t
{
};
```

3.5.2.3 evt_hardware_soft_timer

This event indicates that a software timer has elapsed.

Table 3.96. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x00	method	Message ID
4	uint8	handle	Handle of the elapsed timer

BGScript event

```
event hardware_soft_timer(handle)
```

C Functions

```
/* Event id */
wifi_evt_hardware_soft_timer_id

/* Event structure */
struct wifi_msg_hardware_soft_timer_evt_t
{
  uint8 handle;
};
```

3.5.2.4 evt_hardware_uart_conf

This event indicates the current configuration of a UART interface.

Table 3.97. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x09	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x03	method	Message ID
4	uint8	id	Uart id
5-8	uint32	rate	Actual data rate in bps
9	uint8	data_bits	Data bits
10	uint8	stop_bits	Stop bits
11	uint8	parity	Parity: 0=none, 1=even, 2=odd
12	uint8	flow_ctrl	Flow control: 0=none, 1=rts/cts, 2=rts

BGScript event

```
event hardware_uart_conf(id, rate, data_bits, stop_bits, parity, flow_ctrl)
```

C Functions

```
/* Event id */
wifi_evt_hardware_uart_conf_id

/* Event structure */
struct wifi_msg_hardware_uart_conf_evt_t
{
  uint8 id;,
  uint32 rate;,
  uint8 data_bits;,
  uint8 stop_bits;,
  uint8 parity;,
  uint8 flow_ctrl;
};
```

3.5.3 hardware enumerations

3.5.3.1 enum_hardware_adc_input

This enumeration defines the ADC inputs.

Table 3.98. Enumerations

Value	Name	Description
0	hardware_adc_input_ch0	ADC CH0
1	hardware_adc_input_ch1	ADC CH1
2	hardware_adc_input_ch2	ADC CH2
3	hardware_adc_input_ch3	ADC CH3
4	hardware_adc_input_ch4	ADC CH4
5	hardware_adc_input_ch5	ADC CH5
6	hardware_adc_input_ch6	ADC CH6
7	hardware_adc_input_ch7	ADC CH7
9	hardware_adc_input_vdddiv3	ADC VDD/3

3.5.3.2 enum_hardware_gpio_mode

This enumeration defines the I/O pin modes.

Table 3.99. Enumerations

Value	Name	Description
0	hardware_gpio_mode_disabled	Tri-stated with optional pull-up
1	hardware_gpio_mode_input	Input with optional glitch filter
2	hardware_gpio_mode_input_pull	Input with pull-down or pull-up
3	hardware_gpio_mode_input_pull_filter	Input with glitch filter and pull-down or pull-up
4	hardware_gpio_mode_push_pull	Push-pull output

3.5.3.3 enum_hardware_gpio_port

I/O port definition

Table 3.100. Enumerations

Value	Name	Description
0	hardware_gpio_porta	Port PA
1	hardware_gpio_portb	Port PB
2	hardware_gpio_portc	Port PC
3	hardware_gpio_portd	Port PD
4	hardware_gpio_porte	Port PE
5	hardware_gpio_portf	Port PF

3.5.3.4 enum_hardware_gpio_trigger

This enumeration defines the triggers for an I/O pin interrupt.

Table 3.101. Enumerations

Value	Name	Description
0	hardware_gpio_trigger_disabled	Interrupt disabled
1	hardware_gpio_trigger_rising	Interrupt triggered by a rising edge
2	hardware_gpio_trigger_falling	Interrupt triggered by a falling edge
3	hardware_gpio_trigger_both	Interrupt triggered by both edges

3.5.3.5 enum_hardware_uart_config

This enumeration defines the UART configuration flags.

Table 3.102. Enumerations

Value	Name	Description
0	hardware_uart_conf_parity_none	No parity
1	hardware_uart_conf_parity_odd	Odd parity
2	hardware_uart_conf_parity_even	Even parity
3	hardware_uart_conf_flow_ctrl_none	No flow control
1	hardware_uart_conf_flow_ctrl_rtscts	RTS/CTS flow control
2	hardware_uart_conf_flow_ctrl_rts	RTS flow control

3.6 HTTP Server Command Class

The commands in this class are used to control the internal HTTP, DHCP and DNS Server operations of the Module.

This class consists of the following items:

- *Commands*
  - `cmd_https_add_path`
  - `cmd_https_api_response`
  - `cmd_https_api_response_finish`
  - `cmd_https_enable`
- *Events*
  - `evt_https_api_request`
  - `evt_api_request_data`
  - `evt_https_api_request_finished`
  - `evt_https_api_request_header`
  - `evt_https_error`

3.6.1 https commands

3.6.1.1 cmd_https_add_path

This command is used to add a mapping between an HTTP server path and a resource.

Table 3.103. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x01	method	Message ID
4	uint8	resource	Resource • **0:** flash • **1:** BGAPI/BGScript • **2:** SD card
5	uint8array	path	Server path **Length:** 1 - 127 bytes

Table 3.104. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x01	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call https_add_path(resource, path_len, path_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_https_add_path(uint8 resource, uint8 path_len, const uint8 *path_data);

/* Response id */
wifi_rsp_https_add_path_id

/* Response structure */
struct wifi_msg_https_add_path_rsp_t
{
  uint16 result;
};
```

3.6.1.2 cmd_https_api_response

This command is used to send HTTP response data to a pending HTTP request.

Table 3.105. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x02	method	Message ID
4-7	uint32	request	Request number
8	uint8array	data	Response data

Table 3.106. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x02	method	Message ID
4-5	uint16	result	If there is not enough memory for processing this request, this code will be set as wifi_err_buffers_full. If this happens, command should be resent after some time to respond to API request.

BGScript command

```
call https_api_response(request, data_len, data_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_https_api_response(uint32 request, uint8 data_len, const uint8 *data_data);

/* Response id */
wifi_rsp_https_api_response_id

/* Response structure */
struct wifi_msg_https_api_response_rsp_t
{
  uint16 result;
};
```

3.6.1.3 cmd_https_api_response_finish

This command is used to signal that all HTTP response data has been sent and that the pending HTTP request can be closed.

Table 3.107. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x03	method	Message ID
4-7	uint32	request	Request number

Table 3.108. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x03	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call https_api_response_finish(request)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_https_api_response_finish(uint32 request);

/* Response id */
wifi_rsp_https_api_response_finish_id

/* Response structure */
struct wifi_msg_https_api_response_finish_rsp_t
{
  uint16 result;
};
```

3.6.1.4 cmd_https_enable

This command is used to enable or disable built-in HTTP, DHCP or DNS servers.

Table 3.109. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x00	method	Message ID
4	uint8	https	HTTP server configuration as a bitmask. • **bit 0:** if bit is set, the http server is enabled • **bit 1:** if bit is set, handling of GET requests for SD card is disabled • **bit 2:** if bit is set, handling of PUT requests for SD card is disabled • **bit 3:** if bit is set, handling of POST requests for SD card is disabled • **bit 4:** if bit is set, handling of DELETE requests for SD card is disabled • **bit 5:** if bit is set, possibility to list files in directory on SD card is disabled • **bit 6-7:** unused
5	uint8	dhcps	DHCP server • **0:** disabled • **1:** enabled
6	uint8	dnss	DNS server • **0:** disabled • **1:** enabled

Table 3.110. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x00	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call https_enable(https, dhcps, dnss)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_https_enable(uint8 https, uint8 dhcps, uint8 dnss);

/* Response id */
wifi_rsp_https_enable_id

/* Response structure */
struct wifi_msg_https_enable_rsp_t
{
  uint16 result;
};
```

3.6.1.5 cmd_https_enable_detailed_error

This command is used to enable https_error_detailed instead of https_error.

Table 3.111. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x04	method	Message ID
4	uint8	enable	Use 0 to disable, any other value enables detailed event.

Table 3.112. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x04	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call https_enable_detailed_error(enable)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_https_enable_detailed_error(uint8 enable);

/* Response id */
wifi_rsp_https_enable_detailed_error_id

/* Response structure */
struct wifi_msg_https_enable_detailed_error_rsp_t
{
  uint16 result;
};
```

3.6.2 https events

3.6.2.1 evt_https_api_request

This event indicates that an HTTP request has been received.

Table 3.113. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x06	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x00	method	Message ID
4-7	uint32	request	Request number
8	uint8	method	Request method • **0:** GET • **1:** PUT • **2:** POST • **3:** DELETE
9	uint8array	resource	Request resource

BGScript event

```
event https_api_request(request, method, resource_len, resource_data)
```

C Functions

```
/* Event id */
wifi_evt_https_api_request_id

/* Event structure */
struct wifi_msg_https_api_request_evt_t
{
  uint32 request;,
  uint8 method;,
  uint8array resource;
};
```

3.6.2.2 evt_https_api_request_data

This event contains HTTP payload data of a particular HTTP request.

Multiple events may be received, or none if the HTTP request does not have any payload data.

Table 3.114. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x05	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x02	method	Message ID
4-7	uint32	request	Request number
8	uint8array	data	Request data

BGScript event

```
event https_api_request_data(request, data_len, data_data)
```

C Functions

```
/* Event id */
wifi_evt_https_api_request_data_id

/* Event structure */
struct wifi_msg_https_api_request_data_evt_t
{
  uint32 request;,
  uint8array data;
};
```

3.6.2.3 evt_https_api_request_finished

This event indicates that all HTTP header data, and payload data, if any, have been fully received for a particular HTTP request.

Table 3.115. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x04	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x03	method	Message ID
4-7	uint32	request	Request number

BGScript event

```
event https_api_request_finished(request)
```

C Functions

```
/* Event id */
wifi_evt_https_api_request_finished_id

/* Event structure */
struct wifi_msg_https_api_request_finished_evt_t
{
  uint32 request;
};
```

3.6.2.4 evt_https_api_request_header

This event contains HTTP header data of a particular HTTP request.

At least one event per an HTTP request is received.

Table 3.116. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x05	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x01	method	Message ID
4-7	uint32	request	Request number
8	uint8array	data	Request header data

BGScript event

```
event https_api_request_header(request, data_len, data_data)
```

C Functions

```
/* Event id */
wifi_evt_https_api_request_header_id

/* Event structure */
struct wifi_msg_https_api_request_header_evt_t
{
  uint32 request;,
  uint8array data;
};
```

3.6.2.5 evt_https_error

This event indicates error in HTTP server

Table 3.117. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x04	method	Message ID
4	uint8	reason	Error reason • **1:** Timeout - means that request/response was not fully received/sent and 2 seconds have passed since last action on it • **254:** Unknown

BGScript event

```
event https_error(reason)
```

C Functions

```
/* Event id */
wifi_evt_https_error_id

/* Event structure */
struct wifi_msg_https_error_evt_t
{
  uint8 reason;
};
```

3.6.2.6 evt_https_error_detailed

This event indicates error in HTTP server

Table 3.118. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x05	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x05	method	Message ID
4	uint8	reason	Error reason • **2:** Request timeout - means that request was not fully received and 2 seconds have passed since last action on it • **3:** Response timeout - means that response was not fully sent and 2 seconds have passed since last action on it • **254:** Unknown
5-8	uint32	request	Request number. If all bits are set, it means that this property is invalid.

BGScript event

```
event https_error_detailed(reason, request)
```

C Functions

```
/* Event id */
wifi_evt_https_error_detailed_id

/* Event structure */
struct wifi_msg_https_error_detailed_evt_t
{
  uint8 reason;,
  uint32 request;
};
```

3.6.3 https enumerations

3.6.3.1 enum_https_errors

This enumeration defines HTTP Server Errors

Table 3.119. Enumerations

Value	Name	Description
1	https_timeout	General timeout
2	https_request_timeout	Request timeout
3	https_response_timeout	Response timeout
254	https_unknown	Unknown

3.7 I2C Command Class

The commands in this class are used to control I2C serial communication.

This class consists of the following commands and events:

- *Commands*
  - `cmd_i2c_start_read`
  - `cmd_i2c_start_write`
  - `cmd_i2c_stop`

Examples

1. Using I2C for reading and writing data.
   a. Enable I2C in the project configuration file by defining the I2C Channels and the Location.
   b. Build and load the firmware image file into the Module.
   c. Send the `cmd_i2c_start_write` command using parameters suitable for the connected I2C device.
   d. Complete the write function by sending the `cmd_i2c_stop` command.
   e. Start the read by sending the `cmd_i2c_read` command.
   f. Complete the read function by sending the *cmd_i2c_stop* command
   g. The value read from the sensor is contained in the response.

Troubleshooting information

Problem

No data in response of `cmd_i2c_start read` command .

Possible reasons

- Wrong configuration (e.g. I2C Channel or Location). Check the project configuration file.
- Wrong I2C Channel or Slave address in the command parameters. Check the command call.
- Hardware error. Check the schematics and the PCB layout for any errors.

3.7.1 i2c commands

3.7.1.1 cmd_i2c_start_read

This command is used to start an I2C transaction for reading data.

The transaction must be terminated with the i2c_stop command.

Table 3.120. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x08	class	Message class: I2C
3	0x00	method	Message ID
4	uint8	channel	I2C module to use **Range:** 0 - 1
5	uint8	slave_address	right-aligned 7-bit slave address to use
6	uint8	length	Amount to read **Length:** 1 - 255 bytes

Table 3.121. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x08	class	Message class: I2C
3	0x00	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8array	data	Data read if command was successful

BGScript command

```
call i2c_start_read(channel, slave_address, length)(result, data_len, data_data)
```

BGLIB C API

```
/* Function */
void wifi_cmd_i2c_start_read(uint8 channel, uint8 slave_address, uint8 length);

/* Response id */
wifi_rsp_i2c_start_read_id

/* Response structure */
struct wifi_msg_i2c_start_read_rsp_t
{
  uint16 result;,
  uint8array data;
};
```

3.7.1.2 cmd_i2c_start_write

This command is used to start an I2C transaction for writing data.

The transaction must be terminated with the i2c_stop command.

Table 3.122. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x08	class	Message class: I2C
3	0x01	method	Message ID
4	uint8	channel	I2C module to use **Range:** 0 - 1
5	uint8	slave_address	right-aligned 7-bit slave address to use
6	uint8array	data	Data to write

Table 3.123. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x08	class	Message class: I2C
3	0x01	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call i2c_start_write(channel, slave_address, data_len, data_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_i2c_start_write(uint8 channel, uint8 slave_address, uint8 data_len, const uint8 *data_data);

/* Response id */
wifi_rsp_i2c_start_write_id

/* Response structure */
struct wifi_msg_i2c_start_write_rsp_t
{
  uint16 result;
};
```

3.7.1.3 cmd_i2c_stop

This command is used to stop a read or write I2C transaction.

Table 3.124. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x08	class	Message class: I2C
3	0x02	method	Message ID
4	uint8	channel	I2C module to use **Range:** 0 - 1

Table 3.125. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x08	class	Message class: I2C
3	0x02	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call i2c_stop(channel)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_i2c_stop(uint8 channel);

/* Response id */
wifi_rsp_i2c_stop_id

/* Response structure */
struct wifi_msg_i2c_stop_rsp_t
{
  uint16 result;
};
```

3.8 SDCard Command Class

The commands in this class are intended for file and directory handling of the microSD memory card. The use of these commands requires that the SD card is enabled in the device configuration file and that the microSD card is physically inserted into the microSD card slot before the Module is powered up. Supported microSD card types are SD and SDHC with up to 32 GB memory size. Supported file systems are FAT16 and FAT32. LFN (long) and 8.3 (short) filenames are supported. Maximum supported path length is 254 characters with slash (/) and backslash (\) characters allowed in the path name string.

For more information on configuration of the microSD card slot see *UG161: WGM110 Wi-Fi® Module Configuration User's Guide*.

This class consists of the following commands and events:

- *Commands*
  - `cmd_sdhc_fchdir`
  - `cmd_sdhc_fchmode`
  - `cmd_sdhc_fclose`
  - `cmd_sdhc_fdelete`
  - `cmd_sdhc_fdir`
  - `cmd_sdhc_fmkdir`
  - `cmd_sdhc_fopen`
  - `cmd_sdhc_fread`
  - `cmd_sdhc_frename`
  - `cmd_sdhc_fwrite`
- *Events*
  - `evt_sdhc_fdata`
  - `evt_sdhc_ffile`
  - `evt_sdhc_fpwd`
  - `evt_sdhc_ready`

Before using the `cmd_sdhc_fread`, `cmd_sdhc_fwrite` or `cmd_sdhc_frename` command, the file must first be opened by using the `cmd_sdhc_fopen` command, using a suitable *mode* parameter. The file will be closed automatically after renaming with `cmd_sdhc_frename` command.

EXAMPLES

1. **File write sequence**
   a. Open the file with the `cmd_sdhc_fopen` command.
      i. In case of an existing file use parameter *mode 0x02*.
      ii. In case of a new file use parameter *mode 0x0a*.

   Note: In case the file already exists new data will be appended to the end of the file data.

   Note: In case an existing file is opened using parameter *mode new file (0x0a)*, all data in the file will be deleted.

   b. Send the `cmd_sdhc_fwrite` command and include data to write.

   Note: The maximum data payload that can be included in the command is 512 bytes.

   c. Wait for the `evt_sdhc_fready` event.
   d. Repeat steps 2 and 3 until all data is written into the file.
   e. Close the file with the `cmd_sdhc_fclose` command.
2. **File read sequence**
   a. Open the file with the `cmd_sdhc_fopen` including the parameter *mode bit 0x01*.
   b. Send the `cmd_sdhc_fread` command including the parameter *fsize* which defines the required amount of data in bytes.

   Note: *fsize* value *"0"* indicates that the entire file is read.

   c. Wait for and handle `evt_sdhc_fdata` event(s).

   Note: A single event can contain 512 bytes of data, larger amounts of data wll be divided into several events.

   d. Wait for the `evt_sdhc_fready` event which indicates that all data has been read.
   e. Close the file with the `cmd_sdhc_fclose` command.

3. **Reading the current active directory**
    a. Send the `cmd_sdhc_fchdir` using parameter " '.' " to get the current active directory.

Note: File read with the `cmd_sdhc_fread` command continues from the previous position until the file is closed.

Note: File read using parameter *fsize 0* (reading of the entire file) means that file read is always started at the beginning of the file. The file pointer will be left pointing to the end of the file after the file has been read. Partial reading of the file is not possible before the file has first been closed and reopened to reset the file pointer to point to the beginning of the file.

Troubleshooting information

Problem

The microSD card was removed from the card clot during an operation.

Possible actions

- The only way to continue using the microSD card is to reset the Module.
- If the microSD card was removed from the slot during a write operation, the file system of the memory card might be damaged. Format the microSD card to the proper file system (FAT16 or FAT32) in a PC and retry.

Problem

The command `cmd_sdhc_open` returns an error code.

Possible actions

- File permissions do not allow opening of the file in the selected mode. An example case would be trying to write into a Read Only file. Change file attributes.
- The microSD card was removed and inserted back into the slot after Module initialization. Reset the Module.
- The microSD card is full. Remove unnecessary files.
- Incorrect file path. Check the path or make the directory before trying to open the file.

Problem

The event **evt_sdhc_ready** with an error result code is received at Module initialization.

Possible actions

- The microSD card is not in the slot. Insert the microSD card into the slot and reset the Module.
- The use of microSD cards has not been enabled in the project configuration file or the USART or the chip select pin have been configured erroneosly. Check the project configuration file.
- The microSD card has a not supported file system. Format the microSD card into FAT16 or FAT32 using a PC.
- The microSD card type is not supported (e.g. SDXC). Use only SD or SDHC type microSD cards.

3.8.1 sdhc commands

3.8.1.1 cmd_sdhc_fchdir

This command is used to change the active directory.

Table 3.126. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x07	method	Message ID
4	uint8array	dir_name	Relative or absolute path of the directory

Table 3.127. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x07	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sdhc_fchdir(dir_name_len, dir_name_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sdhc_fchdir(uint8 dir_name_len, const uint8 *dir_name_data);

/* Response id */
wifi_rsp_sdhc_fchdir_id

/* Response structure */
struct wifi_msg_sdhc_fchdir_rsp_t
{
  uint16 result;
};
```

Table 3.128. Events Generated

Event	Description
sdhc_fpwd	Sent when the active directory changed

3.8.1.2 cmd_sdhc_fchmode

This command is used to change the attributes of a file in current active directory.

Table 3.129. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x09	method	Message ID
4	uint8	value	File attributes as a bitmask • **0x01:** read only • **0x02:** hidden • **0x04:** system
5	uint8array	fname	File name

Table 3.130. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x09	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sdhc_fchmode(value, fname_len, fname_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sdhc_fchmode(uint8 value, uint8 fname_len, const uint8 *fname_data);

/* Response id */
wifi_rsp_sdhc_fchmode_id

/* Response structure */
struct wifi_msg_sdhc_fchmode_rsp_t
{
  uint16 result;
};
```

3.8.1.3 cmd_sdhc_fclose

This command is used to close an open file.

Table 3.131. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x01	method	Message ID
4	uint8	fhandle	Handle of the opened file

Table 3.132. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x01	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sdhc_fclose(fhandle)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sdhc_fclose(uint8 fhandle);

/* Response id */
wifi_rsp_sdhc_fclose_id

/* Response structure */
struct wifi_msg_sdhc_fclose_rsp_t
{
  uint16 result;
};
```

3.8.1.4 cmd_sdhc_fdelete

This command is used to delete a file or an empty subdirectory in current active directory.

Table 3.133. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x05	method	Message ID
4	uint8array	fname	File or directory name

Table 3.134. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x05	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sdhc_fdelete(fname_len, fname_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sdhc_fdelete(uint8 fname_len, const uint8 *fname_data);

/* Response id */
wifi_rsp_sdhc_fdelete_id

/* Response structure */
struct wifi_msg_sdhc_fdelete_rsp_t
{
  uint16 result;
};
```

3.8.1.5 cmd_sdhc_fdir

This command is used to list files of a directory.

The command also lists files and folders with the hidden attribute set.

Table 3.135. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x02	method	Message ID
4	uint8	mode	File list mode • **0:** single directory • **1:** recursive
5	uint8array	path	Path of the directory to list **Length:** 1 - 254 bytes

Table 3.136. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x02	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sdhc_fdir(mode, path_len, path_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sdhc_fdir(uint8 mode, uint8 path_len, const uint8 *path_data);

/* Response id */
wifi_rsp_sdhc_fdir_id

/* Response structure */
struct wifi_msg_sdhc_fdir_rsp_t
{
  uint16 result;
};
```

Table 3.137. Events Generated

Event	Description
sdhc_ffile	Sent for each listed file
sdhc_ready	Sent after all the files have been listed with operation value 3.

3.8.1.6 cmd_sdhc_fmkdir

This command is used to create a directory.

Table 3.138. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x06	method	Message ID
4	uint8array	dir_name	Relative or absolute path of the directory

Table 3.139. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x06	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sdhc_fmkdir(dir_name_len, dir_name_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sdhc_fmkdir(uint8 dir_name_len, const uint8 *dir_name_data);

/* Response id */
wifi_rsp_sdhc_fmkdir_id

/* Response structure */
struct wifi_msg_sdhc_fmkdir_rsp_t
{
  uint16 result;
};
```

3.8.1.7 cmd_sdhc_fopen

This command is used to open a file.

The maximum amount of open files is 10.

Table 3.140. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x00	method	Message ID
4	uint8	mode	File mode as a bitmask • **0x01:** read • **0x02:** write • **0x08:** create new
5	uint8array	fname	File name **Length:** 1 - 254 bytes

Table 3.141. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x00	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sdhc_fopen(mode, fname_len, fname_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sdhc_fopen(uint8 mode, uint8 fname_len, const uint8 *fname_data);

/* Response id */
wifi_rsp_sdhc_fopen_id

/* Response structure */
struct wifi_msg_sdhc_fopen_rsp_t
{
  uint16 result;
};
```

Table 3.142. Events Generated

Event	Description
sdhc_ffile	Sent when the file has been successfully opened.

3.8.1.8 cmd_sdhc_fread

This command is used to read data from a file.

The read data is sent in multiple evt_sdhc_fdata events, each containing up to 512 bytes of data.

Table 3.143. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x03	method	Message ID
4	uint8	fhandle	Handle of the opened file
5-8	uint32	fsize	Amount to read • **0:** whole file • **non-zero:** amount of bytes

Table 3.144. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x03	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sdhc_fread(fhandle, fsize)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sdhc_fread(uint8 fhandle, uint32 fsize);

/* Response id */
wifi_rsp_sdhc_fread_id

/* Response structure */
struct wifi_msg_sdhc_fread_rsp_t
{
  uint16 result;
};
```

Table 3.145. Events Generated

Event	Description
sdhc_fdata	Sent for each block of data
sdhc_ready	Sent after the read operation has been completed with operation value 1.

3.8.1.9 cmd_sdhc_frename

This command is used to rename a file in the current active directory.

Table 3.146. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x08	method	Message ID
4	uint8	fhandle	Handle of the opened file
5	uint8array	new_name	New file name

Table 3.147. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x08	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sdhc_frename(fhandle, new_name_len, new_name_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sdhc_frename(uint8 fhandle, uint8 new_name_len, const uint8 *new_name_data);

/* Response id */
wifi_rsp_sdhc_frename_id

/* Response structure */
struct wifi_msg_sdhc_frename_rsp_t
{
  uint16 result;
};
```

3.8.1.10 cmd_sdhc_fwrite

This command is used to a write a block of data to a file.

The data will be appended to the end of the file.

Table 3.148. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x04	method	Message ID
4	uint8	fhandle	Handle of the opened file
5-6	uint16array	fdata	Data to write **Length:** 1 - 512 bytes

Table 3.149. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x04	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sdhc_fwrite(fhandle, fdata_len, fdata_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sdhc_fwrite(uint8 fhandle, uint8 fdata_len, const uint8 *fdata_data);

/* Response id */
wifi_rsp_sdhc_fwrite_id

/* Response structure */
struct wifi_msg_sdhc_fwrite_rsp_t
{
  uint16 result;
};
```

Table 3.150. Events Generated

Event	Description
sdhc_ready	Sent after the write operation has been completed with operation value 2.

3.8.2 sdhc events

3.8.2.1 evt_sdhc_fdata

This event contains a block of file data.

Table 3.151. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x01	method	Message ID
4	uint8	fhandle	Handle of the opened file
5-6	uint16array	data	File data

BGScript event

```
event sdhc_fdata(fhandle, data_len, data_data)
```

C Functions

```
/* Event id */
wifi_evt_sdhc_fdata_id

/* Event structure */
struct wifi_msg_sdhc_fdata_evt_t
{
  uint8 fhandle;,
  uint16array data;
};
```

3.8.2.2 evt_sdhc_ffile

This event contains file information.

Table 3.152. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x07	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x02	method	Message ID
4	uint8	fhandle	Handle of the opened file **255:** file not opened
5-8	uint32	fsize	File size in bytes
9	uint8	fattrib	File attributes as a bitmask • **0x01:** read only • **0x02:** hidden • **0x04:** system • **0x10:** directory
10	uint8array	fname	File name

BGScript event

```
event sdhc_ffile(fhandle, fsize, fattrib, fname_len, fname_data)
```

C Functions

```
/* Event id */
wifi_evt_sdhc_ffile_id

/* Event structure */
struct wifi_msg_sdhc_ffile_evt_t
{
  uint8 fhandle;,
  uint32 fsize;,
  uint8 fattrib;,
  uint8array fname;
};
```

3.8.2.3 evt_sdhc_fpwd

This event indicates the current active directory.

Table 3.153. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x03	method	Message ID
4	uint8array	fdir	Directory name with path

BGScript event

```
event sdhc_fpwd(fdir_len, fdir_data)
```

C Functions

```
/* Event id */
wifi_evt_sdhc_fpwd_id

/* Event structure */
struct wifi_msg_sdhc_fpwd_evt_t
{
  uint8array fdir;
};
```

3.8.2.4 evt_sdhc_ready

This event indicates that an SD card operation has finished.

Table 3.154. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x04	lolen	Minimum payload length
2	0x0c	class	Message class: SD card
3	0x00	method	Message ID
4	uint8	fhandle	Handle of the opened file
5	uint8	operation	Operation which caused the event • **0:** card detection state • **1:** file read • **2:** file write • **3:** directory listing
6-7	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript event

```
event sdhc_ready(fhandle, operation, result)
```

C Functions

```
/* Event id */
wifi_evt_sdhc_ready_id

/* Event structure */
struct wifi_msg_sdhc_ready_evt_t
{
  uint8 fhandle;,
  uint8 operation;,
  uint16 result;
};
```

3.9 Wi-Fi Command Class

The commands in this class are used to control Wi-Fi Station Management Entity (SME) functionality, more specifically to control the Wi-Fi chip functionality of the Module, network scanning and connections of the Wi-Fi client and access point.

This class consists of the following commands and events:

Wi-Fi chip control related

- *Commands*
  - `cmd_sme_wifi_on`
  - `cmd_sme_wifi_off`
  - `cmd_sme_set_operating_mode`
  - `cmd_sme_get_signal_quality`
- *Events*
  - `evt_sme_wifi_is_on`
  - `evt_sme_wifi_is_off`
  - `evt_sme_signal_quality`

Wi-Fi network scan related

- *Commands*
  - `cmd_sme_set_scan_channels`
  - `cmd_sme_start_scan`
  - `cmd_sme_stop_scan`
  - `cmd_sme_scan_results_sort_rssi`
- *Events*
  - `evt_sme_scan_result`
  - `evt_sme_scan_result_drop`
  - `evt_sme_scanned`
  - `evt_sme_scan_sort_result`
  - `evt_sme_scan_sort_finished`

Client mode connection related

- *Commands*
  - `cmd_sme_set_password`
  - `cmd_sme_connect_bssid`
  - `cmd_sme_connect_ssid`
  - `cmd_sme_disconnect`
- *Events*
  - `evt_sme_connected`
  - `evt_sme_connect_failed`
  - `evt_sme_connect_retry`
  - `evt_sme_interface_status`
  - `evt_sme_disconnected`

Client WPS (Wi-Fi Protected Setup) related

- *Commands*
  - `cmd_sme_start_wps`
  - `cmd_sme_stop_wps`
- *Events*
  - `evt_sme_wps_completed`
  - `evt_sme_wps_failed`
  - `evt_sme_wps_credential_ssid`
  - `evt_sme_wps_credential_password`
  - `evt_sme_wps_stopped`

Access Point Mode related

- *Commands*
  - `cmd_sme_start_ap_mode`
  - `cmd_sme_stop_ap_mode`
  - `cmd_sme_set_ap_password`
  - `cmd_sme_set_ap_max_clients`
  - `cmd_sme_ap_client_config`
- *Events*
  - `evt_sme_ap_mode_started`
  - `evt_sme_ap_mode_failed`
  - `evt_sme_ap_mode_stopped`
  - `evt_sme_ap_client_joined`
  - `evt_sme_ap_client_left`

Wi-Fi Direct related

- *Commands*
  - `cmd_sme_start_p2p_group`
  - `cmd_sme_stop_p2p_group`
  - `cmd_sme_p2p_accept_client`
- *Events*
  - `evt_sme_p2p_group_started`
  - `evt_sme_p2p_group_stopped`
  - `evt_sme_p2p_group_failed`
  - `evt_sme_p2p_client_wants_to_join`

Examples

1. **Connecting to a Wi-Fi network using SSID and DHCP client**
   a. Select the operation mode of the client using the `cmd_sme_set_operation_mode` command.
   b. Send the `cmd_sme_wifi_on` command to power up the Wi-Fi chip.
   c. Wait for the `evt_sme_wifi_is_on` event.

      Note: Wi-Fi chip power-up may take upto 10 seconds.

   d. Set the network password using the `cmd_sme_set_password` command.
   e. Connect to the network using the `cmd_sme_connect_ssid` command.
   f. Wait for the `evt_sme_connected` event.
   g. The Module will send the events `evt_tcpip_configuration` and `evt_tcpip_dns_configuration` while resolving the network parameters.
   h. The event `evt_sme_interface_status` indicates that a connection has been established.

3.9.1 sme commands

3.9.1.1 cmd_sme_add_scan_result_filter

Add a scan result filter. If AP with matched IE is found, wifi_evt_sme_scan_result_filter will be sent.

Number of applied filters is limited to 3. If limit is exceeded, function returns wifi_err_out_of_memory.

Table 3.155. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x23	method	Message ID
4	uint8	ie_id	Information Element ID to be matched
5	uint8array	ie_data	Information Element data to be matched. Maximum length of data is 7 bytes. If provided argument exceeds length limit, function returns wifi_err_invalid_param.

Table 3.156. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x23	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	int8	index	Index of the filter

BGScript command

```
call sme_add_scan_result_filter(ie_id, ie_data_len, ie_data_data)(result, index)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_add_scan_result_filter(uint8 ie_id, uint8 ie_data_len, const uint8 *ie_data_data);

/* Response id */
wifi_rsp_sme_add_scan_result_filter_id

/* Response structure */
struct wifi_msg_sme_add_scan_result_filter_rsp_t
{
  uint16 result;,
  int8 index;
};
```

3.9.1.2 cmd_sme_ap_client_config

This command is used to configure clients connected in access point mode

Table 3.157. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x20	method	Message ID
4	uint8	max_clients	The maximum amount of simultaneously connected clients. The command does not affect currently connected clients. **Range:** 0 - 10. Zero keeps previous value **Default:** 5
5-6	uint16	cleanup_period	Client activity check period in seconds. **Range:** 0-65535. Zero keeps previous value **Default:** 45
7-8	uint16	keepalive_interval	Keepalive frame sending interval in seconds **Range:** 0-65535. Zero keeps previous value **Default:** 14

Table 3.158. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x20	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_ap_client_config(max_clients, cleanup_period, keepalive_interval)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_ap_client_config(uint8 max_clients, uint16 cleanup_period, uint16 keepalive_interval);

/* Response id */
wifi_rsp_sme_ap_client_config_id

/* Response structure */
```

```
struct wifi_msg_sme_ap_client_config_rsp_t
{
  uint16 result;
};
```

3.9.1.3 cmd_sme_ap_client_disconnect

This command is used to disconnect a client from the Access Point.

Table 3.159. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x06	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0e	method	Message ID
4-9	hw_addr	mac_address	MAC address of the client

Table 3.160. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0e	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript command

```
call sme_ap_client_disconnect(mac_address)(result, hw_interface)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_ap_client_disconnect(hw_addr mac_address);

/* Response id */
wifi_rsp_sme_ap_client_disconnect_id

/* Response structure */
struct wifi_msg_sme_ap_client_disconnect_rsp_t
{
  uint16 result;,
  uint8 hw_interface;
};
```

Table 3.161. Events Generated

Event	Description
sme_ap_client_left	Sent after the client has been disconnected

3.9.1.4 cmd_sme_connect_bssid

This command is used to try to connect to a specific Access Point using its unique BSSID.

The command requires a preceding sme_start_scan or sme_start_ssid_scan command and that the desired Access Point was found during that scan. When connecting to an Access Point on channel 12 or 13, at least one of the Access Points discovered in the scan must advertise the use of channels up to 13.

An ongoing connection attempt can be canceled using the sme_disconnect command.

Table 3.162. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x06	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x06	method	Message ID
4-9	hw_addr	bssid	The BSSID of the Access Point to connect to

Table 3.163. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x09	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x06	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi
7-12	hw_addr	bssid	The BSSID of the Access Point that the device will attempt to connect to

BGScript command

```
call sme_connect_bssid(bssid)(result, hw_interface, bssid)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_connect_bssid(hw_addr bssid);

/* Response id */
wifi_rsp_sme_connect_bssid_id

/* Response structure */
struct wifi_msg_sme_connect_bssid_rsp_t
{
  uint16 result;,
  uint8 hw_interface;,
```

```
  hw_addr bssid;
};
```

Table 3.164. Events Generated

Event	Description
sme_connect_retry	Sent when the connection attempt fails and it is automatically retried
sme_connected	Sent when the connection attempt succeeds
sme_connect_failed	Sent when the connection attempt fails
tcpip_configuration	Sent when the IP address of the Wi-Fi interface has been activated
tcpip_dns_configuration	Sent when the DNS address of the Wi-Fi interface has been activated
sme_interface_status	Sent after the Wi-Fi interface has been activated

3.9.1.5 cmd_sme_connect_ssid

This command is used to start a connection establishment procedure with an Access Point with the given SSID.

Executing this command launches an internal scan procedure in order to discover the Access Points in range. The results of the scan are NOT exposed via BGAPI nor stored in the internal scan list. The channels used in the scan procedure can be defined with the sme_set_scan_channels command. If the command has not been executed, all channels will be scanned. When connecting to an Access Point on channel 12 or 13, at least one of the Access Points discovered in the scan must advertise the use of channels up to 13.

An ongoing connection attempt can be canceled using the sme_disconnect command.

Table 3.165. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x07	method	Message ID
4	uint8array	ssid	The SSID of the network to connect to

Table 3.166. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x09	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x07	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi
7-12	hw_addr	bssid	The BSSID of the Access Point that the device will attempt to connect to. Since the command attempts to connect based on the SSID, this parameter is not valid and is always zero.

BGScript command

```
call sme_connect_ssid(ssid_len, ssid_data)(result, hw_interface, bssid)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_connect_ssid(uint8 ssid_len, const uint8 *ssid_data);

/* Response id */
wifi_rsp_sme_connect_ssid_id

/* Response structure */
struct wifi_msg_sme_connect_ssid_rsp_t
```

```
{
  uint16 result;,
  uint8 hw_interface;,
  hw_addr bssid;
};
```

Table 3.167. Events Generated

Event	Description
sme_connect_retry	Sent when the connection attempt fails and it is automatically retried
sme_connected	Sent when the connection attempt succeeds
sme_connect_failed	Sent when the connection attempt fails
tcpip_configuration	Sent when the IP address of the Wi-Fi interface has been activated
tcpip_dns_configuration	Sent when the DNS address of the Wi-Fi interface has been activated
sme_interface_status	Sent after the Wi-Fi interface has been activated

3.9.1.6 cmd_sme_disconnect

This command is used to cancel an ongoing connection attempt or disconnect from the currently connected Access Point.

Table 3.168. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x08	method	Message ID

Table 3.169. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x08	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript command

```
call sme_disconnect()(result, hw_interface)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_disconnect();

/* Response id */
wifi_rsp_sme_disconnect_id

/* Response structure */
struct wifi_msg_sme_disconnect_rsp_t
{
  uint16 result;,
  uint8 hw_interface;
};
```

Table 3.170. Events Generated

Event	Description
sme_disconnected	Sent after disconnected
tcpip_configuration	Sent when the IP address of the Wi-Fi interface has been deactivated

Event	Description
tcpip_dns_configuration	Sent when the DNS address of the Wi-Fi interface has been deactivated
sme_interface_status	Sent after the Wi-Fi interface has been deactivated

3.9.1.7 cmd_sme_get_signal_quality

This command is used to get a value indicating the signal quality of the connection.

The command is applicable only in client mode

Table 3.171. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x13	method	Message ID

Table 3.172. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x13	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript command

```
call sme_get_signal_quality()(result, hw_interface)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_get_signal_quality();

/* Response id */
wifi_rsp_sme_get_signal_quality_id

/* Response structure */
struct wifi_msg_sme_get_signal_quality_rsp_t
{
  uint16 result;,
  uint8 hw_interface;
};
```

Table 3.173. Events Generated

Event	Description
sme_signal_quality	Connection signal quality

3.9.1.8 cmd_sme_p2p_accept_client

This command is used to accept the pending connection attempt.

Due to encryption used in communication, a small delay (up to 3 seconds) can appear before receiving response.

Table 3.174. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x06	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x1f	method	Message ID
4-9	hw_addr	mac_address	MAC address of the client to be accepted

Table 3.175. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x1f	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_p2p_accept_client(mac_address)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_p2p_accept_client(hw_addr mac_address);

/* Response id */
wifi_rsp_sme_p2p_accept_client_id

/* Response structure */
struct wifi_msg_sme_p2p_accept_client_rsp_t
{
  uint16 result;
};
```

3.9.1.9 cmd_sme_remove_scan_result_filter

Remove a scan result filter.

Table 3.176. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x24	method	Message ID
4	int8	index	Index of the filter

Table 3.177. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x24	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_remove_scan_result_filter(index)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_remove_scan_result_filter(int8 index);

/* Response id */
wifi_rsp_sme_remove_scan_result_filter_id

/* Response structure */
struct wifi_msg_sme_remove_scan_result_filter_rsp_t
{
  uint16 result;
};
```

3.9.1.10 cmd_sme_scan_results_sort_rssi

This command is used to retrieve results from the internal scan list, sorted according to RSSI value.

The command can be run only after the sme_start_scan command or the sme_start_ssid_scan command has been issued at least once during the current session.

Table 3.178. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0d	method	Message ID
4	uint8	amount	Maximum amount of entries to retrieve **Range:** 1 - 40

Table 3.179. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0d	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_scan_results_sort_rssi(amount)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_scan_results_sort_rssi(uint8 amount);

/* Response id */
wifi_rsp_sme_scan_results_sort_rssi_id

/* Response structure */
struct wifi_msg_sme_scan_results_sort_rssi_rsp_t
{
  uint16 result;
};
```

Table 3.180. Events Generated

Event	Description
sme_scan_sort_result	Sent for each Access Point
sme_scan_sort_finished	Sent after the operation completes

3.9.1.11 cmd_sme_set_11n_mode

This command is used to select whether 802.11n mode is enabled or disabled.

The mode is enabled by default.

Table 3.181. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x16	method	Message ID
4	uint8	mode	802.11n mode • **0:** disabled • **1:** enabled

Table 3.182. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x16	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript command

```
call sme_set_11n_mode(mode)(result, hw_interface)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_set_11n_mode(uint8 mode);

/* Response id */
wifi_rsp_sme_set_11n_mode_id

/* Response structure */
struct wifi_msg_sme_set_11n_mode_rsp_t
{
  uint16 result;,
  uint8 hw_interface;
};
```

3.9.1.12 cmd_sme_set_ap_client_isolation

Set whether Access Points clients are isolated from each other.

Table 3.183. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x21	method	Message ID
4	uint8	isolation	0 = disabled, 1 = enabled

Table 3.184. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x21	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	

BGScript command

```
call sme_set_ap_client_isolation(isolation)(result, hw_interface)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_set_ap_client_isolation(uint8 isolation);

/* Response id */
wifi_rsp_sme_set_ap_client_isolation_id

/* Response structure */
struct wifi_msg_sme_set_ap_client_isolation_rsp_t
{
  uint16 result;,
  uint8 hw_interface;
};
```

3.9.1.13 cmd_sme_set_ap_custom_ie

Set custom vendor-specific IEs in beacons and probe responses.

This function can only be called before starting Access Point. After reset all custom IEs are deleted.

Table 3.185. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x22	method	Message ID
4	uint8array	ie	List of vendor specific IEs. This array can contain multiple IEs and should be made of raw IEs data. If this parameter is empty, then all custom IEs will be deleted.

Table 3.186. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x22	method	Message ID
4-5	uint16	result	Wrong state is returned when Access Point has been started or device is in station mode. Invalid parameter is returned when size of IEs doesn't match the length of raw IEs data.

BGScript command

```
call sme_set_ap_custom_ie(ie_len, ie_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_set_ap_custom_ie(uint8 ie_len, const uint8 *ie_data);

/* Response id */
wifi_rsp_sme_set_ap_custom_ie_id

/* Response structure */
struct wifi_msg_sme_set_ap_custom_ie_rsp_t
{
  uint16 result;
};
```

3.9.1.14 cmd_sme_set_ap_hidden

This command is used to set whether the Access Point is hidden or visible.

The Access Point is set visible by default.

Table 3.187. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x15	method	Message ID
4	uint8	hidden	Visibility • **0:** visible • **1:** hidden

Table 3.188. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x15	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript command

```
call sme_set_ap_hidden(hidden)(result, hw_interface)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_set_ap_hidden(uint8 hidden);

/* Response id */
wifi_rsp_sme_set_ap_hidden_id

/* Response structure */
struct wifi_msg_sme_set_ap_hidden_rsp_t
{
  uint16 result;,
  uint8 hw_interface;
};
```

3.9.1.15 cmd_sme_set_ap_max_clients

This command is used to set the maximum amount of clients that can be associated to the Access Point simultaneously.

The command does not affect currently connected clients.

Table 3.189. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x10	method	Message ID
4	uint8	max_clients	The maximum amount of clients **Range:** 1 - 10 **Default:** 5

Table 3.190. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x10	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript command

```
call sme_set_ap_max_clients(max_clients)(result, hw_interface)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_set_ap_max_clients(uint8 max_clients);

/* Response id */
wifi_rsp_sme_set_ap_max_clients_id

/* Response structure */
struct wifi_msg_sme_set_ap_max_clients_rsp_t
{
  uint16 result;,
  uint8 hw_interface;
};
```

3.9.1.16 cmd_sme_set_ap_password

This command is used to set the Wi-Fi password for the Access Point mode.

Table 3.191. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0f	method	Message ID
4	uint8array	password	The password to be used for the Access Point • WPA/WPA2-PSK is either a hash of 64 hexadecimal characters, or a pass-phrase of 8 to 63 ASCII-encoded characters • 64-bit WEP key is either 5 ASCII-encoded characters or 10 HEX characters • 128-bit WEP key is either 13 ASCII-encoded characters or 26 HEX characters

Table 3.192. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0f	method	Message ID
4	uint8	status	Result code • **0:** success • **128:** invalid password

BGScript command

```
call sme_set_ap_password(password_len, password_data)(status)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_set_ap_password(uint8 password_len, const uint8 *password_data);

/* Response id */
wifi_rsp_sme_set_ap_password_id

/* Response structure */
struct wifi_msg_sme_set_ap_password_rsp_t
{
  uint8 status;
};
```

3.9.1.17 cmd_sme_set_eap_configuration

This command is used to set the EAP configuration, which is used when authenticating with a secure Access Point.

Table 3.193. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x17	method	Message ID
4	uint8	outer_type	Outer (tunneling) EAP type
5	uint8	inner_type	Inner EAP type
6	uint8array	identity	Identity **Length:** 0 - 255 bytes

Table 3.194. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x17	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_set_eap_configuration(outer_type, inner_type, identity_len, identity_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_set_eap_configuration(uint8 outer_type, uint8 inner_type, uint8 identity_len, const uint8
*identity_data);

/* Response id */
wifi_rsp_sme_set_eap_configuration_id

/* Response structure */
struct wifi_msg_sme_set_eap_configuration_rsp_t
{
  uint16 result;
};
```

3.9.1.18 cmd_sme_set_eap_type_ca_certificate

This command is used to set the required CA certificate of an EAP type.

Connection attempt will fail if the certificate chain given by the authentication server cannot be verified using the provided CA certificate. If no CA certificate has been set, the internal certificate store is checked for a suitable certificate.

Table 3.195. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x1a	method	Message ID
4	uint8	type	EAP type
5	uint8array	fingerprint	Fingerprint of the CA certificate **Length:** 16 bytes

Table 3.196. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x1a	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_set_eap_type_ca_certificate(type, fingerprint_len, fingerprint_data)(result)
```

BGLIB C API

```
/* Function */
void    wifi_cmd_sme_set_eap_type_ca_certificate(uint8    type,    uint8    fingerprint_len,    const    uint8
*fingerprint_data);

/* Response id */
wifi_rsp_sme_set_eap_type_ca_certificate_id

/* Response structure */
struct wifi_msg_sme_set_eap_type_ca_certificate_rsp_t
{
  uint16 result;
};
```

3.9.1.19 cmd_sme_set_eap_type_password

This command is used to set the password of an EAP type.

Table 3.197. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x19	method	Message ID
4	uint8	type	EAP type
5	uint8array	password	Password **Length:** 0 - 255 bytes

Table 3.198. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x19	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_set_eap_type_password(type, password_len, password_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_set_eap_type_password(uint8 type, uint8 password_len, const uint8 *password_data);

/* Response id */
wifi_rsp_sme_set_eap_type_password_id

/* Response structure */
struct wifi_msg_sme_set_eap_type_password_rsp_t
{
  uint16 result;
};
```

3.9.1.20 cmd_sme_set_eap_type_server_common_name

This command is used to set the required Common Name of an EAP type.

Connection attempt will fail if the Common Name in the server certificate given by the authentication server does not match the set value. If not set, Common Name value is ignored.

Table 3.199. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x1b	method	Message ID
4	uint8	type	EAP type
5	uint8array	common_name	Server common name **Length:** 0 - 255 bytes

Table 3.200. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x1b	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_set_eap_type_server_common_name(type, common_name_len, common_name_data)(result)
```

BGLIB C API

```
/* Function */
void   wifi_cmd_sme_set_eap_type_server_common_name(uint8   type,   uint8   common_name_len,   const   uint8
*common_name_data);

/* Response id */
wifi_rsp_sme_set_eap_type_server_common_name_id

/* Response structure */
struct wifi_msg_sme_set_eap_type_server_common_name_rsp_t
{
  uint16 result;
};
```

3.9.1.21 cmd_sme_set_eap_type_user_certificate

This command is used to set the client certificate of an EAP type.

If set, the certificate will be sent to the authentication server if requested by the server during authentication.

Table 3.201. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x1c	method	Message ID
4	uint8	type	EAP type
5	uint8array	fingerprint	Fingerprint of the client certificate **Length:** 16 bytes

Table 3.202. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x1c	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_set_eap_type_user_certificate(type, fingerprint_len, fingerprint_data)(result)
```

BGLIB C API

```
/* Function */
void   wifi_cmd_sme_set_eap_type_user_certificate(uint8   type,   uint8   fingerprint_len,   const   uint8
*fingerprint_data);

/* Response id */
wifi_rsp_sme_set_eap_type_user_certificate_id

/* Response structure */
struct wifi_msg_sme_set_eap_type_user_certificate_rsp_t
{
  uint16 result;
};
```

3.9.1.22 cmd_sme_set_eap_type_username

This command is used to set the user name of an EAP type.

Table 3.203. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x18	method	Message ID
4	uint8	type	EAP type
5	uint8array	username	User name **Length:** 0 - 255 bytes

Table 3.204. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x18	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_set_eap_type_username(type, username_len, username_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_set_eap_type_username(uint8 type, uint8 username_len, const uint8 *username_data);

/* Response id */
wifi_rsp_sme_set_eap_type_username_id

/* Response structure */
struct wifi_msg_sme_set_eap_type_username_rsp_t
{
  uint16 result;
};
```

3.9.1.23 cmd_sme_set_operating_mode

This command is used to set the Wi-Fi operating mode, either to Wi-Fi client or Wi-Fi Access Point.

The selected operating mode will become effective the next time the Wi-Fi radio is switched on using the sme_wifi_on command.

Table 3.205. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0a	method	Message ID
4	uint8	mode	Operating mode • **1:** client • **2:** Access Point • **4:** Wi-Fi direct

Table 3.206. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0a	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_set_operating_mode(mode)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_set_operating_mode(uint8 mode);

/* Response id */
wifi_rsp_sme_set_operating_mode_id

/* Response structure */
struct wifi_msg_sme_set_operating_mode_rsp_t
{
  uint16 result;
};
```

3.9.1.24 cmd_sme_set_password

This command is used to set the password used when authenticating with a secure Access Point.

There is no password set by default.

Table 3.207. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x05	method	Message ID
4	uint8array	password	WEP/WPA/WPA2 pre-shared password to be used when connecting to an Access Point • WPA/WPA2-PSK is either a hash of 64 hexadecimal characters, or a pass-phrase of 8 to 63 ASCII-encoded characters • 64-bit WEP key is either 5 ASCII-encoded characters or 10 HEX characters • 128-bit WEP key is either 13 ASCII-encoded characters or 26 HEX characters

Table 3.208. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x05	method	Message ID
4	uint8	status	Result code • **0:** success • **128:** invalid password

BGScript command

```
call sme_set_password(password_len, password_data)(status)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_set_password(uint8 password_len, const uint8 *password_data);

/* Response id */
wifi_rsp_sme_set_password_id

/* Response structure */
struct wifi_msg_sme_set_password_rsp_t
{
  uint8 status;
};
```

3.9.1.25 cmd_sme_set_scan_channels

This command is used to set the default scan channel list.

Table 3.209. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x09	method	Message ID
4	uint8	hw_interface	Hardware interface • **0:** Wi-Fi
5	uint8array	chList	List of channel numbers to scan. All channels are scanned by default if this command is never used.

Table 3.210. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x09	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_set_scan_channels(hw_interface, chList_len, chList_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_set_scan_channels(uint8 hw_interface, uint8 chList_len, const uint8 *chList_data);

/* Response id */
wifi_rsp_sme_set_scan_channels_id

/* Response structure */
struct wifi_msg_sme_set_scan_channels_rsp_t
{
  uint16 result;
};
```

3.9.1.26 cmd_sme_start_ap_mode

This command is used to start the Access Point mode.

In order to start the Access Point on channel 12 or 13, at least one Access Point discovered in a scan must advertise the use of channels up to 13.

Table 3.211. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0b	method	Message ID
4	uint8	channel	Channel to use **Range:** 1 - 13
5	uint8	security	Security mode • **0:** open security • **1:** WPA security • **2:** WPA2 security • **3:** WEP security
6	uint8array	ssid	SSID to use **Length:** 1 - 32 bytes UTF-8 US-ASCII characters allowed

Table 3.212. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0b	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript command

```
call sme_start_ap_mode(channel, security, ssid_len, ssid_data)(result, hw_interface)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_start_ap_mode(uint8 channel, uint8 security, uint8 ssid_len, const uint8 *ssid_data);
```

```
/* Response id */
wifi_rsp_sme_start_ap_mode_id

/* Response structure */
struct wifi_msg_sme_start_ap_mode_rsp_t
{
  uint16 result;,
  uint8 hw_interface;
};
```

Table 3.213. Events Generated

Event	Description
sme_ap_mode_started	Sent after Access Point mode successfully started
sme_ap_mode_failed	Sent after Access Point mode fails
tcpip_configuration	Sent when the IP address of the Wi-Fi interface has been activated
tcpip_dns_configuration	Sent when the DNS address of the Wi-Fi interface has been activated
sme_interface_status	Sent after Wi-Fi interface has been activated

3.9.1.27 cmd_sme_start_p2p_group

This command is used to start Wi-Fi Direct Group. Before using the command, set the operating mode of the module to 4 by using sme_set_operating_mode.

Table 3.214. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x1d	method	Message ID
4	uint8	channel	Access point channel.
5	uint8array	ssid	SSID to use.

Table 3.215. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x1d	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	

BGScript command

```
call sme_start_p2p_group(channel, ssid_len, ssid_data)(result, hw_interface)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_start_p2p_group(uint8 channel, uint8 ssid_len, const uint8 *ssid_data);

/* Response id */
wifi_rsp_sme_start_p2p_group_id

/* Response structure */
struct wifi_msg_sme_start_p2p_group_rsp_t
{
  uint16 result;,
  uint8 hw_interface;
};
```

3.9.1.28 cmd_sme_start_scan

This command initiates a scan for Access Points.

Scanning is not possible once connected or once Access Point mode has been started.

The internal scan list is cleared before the scan is initiated.

Table 3.216. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x03	method	Message ID
4	uint8	hw_interface	Hardware interface • **0:** Wi-Fi
5	uint8array	chList	The list of channel numbers which will be scanned for Access Points. If empty, the scan will be performed using the channels set with the sme_set_scan_channels command.

Table 3.217. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x03	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_start_scan(hw_interface, chList_len, chList_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_start_scan(uint8 hw_interface, uint8 chList_len, const uint8 *chList_data);

/* Response id */
wifi_rsp_sme_start_scan_id

/* Response structure */
struct wifi_msg_sme_start_scan_rsp_t
{
  uint16 result;
};
```

Table 3.218. Events Generated

Event	Description
sme_scan_result	Sent for each Access Point discovered
sme_scan_result_drop	Sent for each Access Point dropped from the internal scan list
sme_scanned	Sent after the scan completes

3.9.1.29 cmd_sme_start_ssid_scan

This command is used to initiate an active scan for Access Points.

Scanning is not possible once connected or once Access Point mode has been started.

The internal scan list is cleared before the scan is initiated.

Table 3.219. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x14	method	Message ID
4	uint8array	ssid	The SSID to scan for **Length:** 0 - 32 bytes If 0, all SSIDs are scanned.

Table 3.220. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x14	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_start_ssid_scan(ssid_len, ssid_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_start_ssid_scan(uint8 ssid_len, const uint8 *ssid_data);

/* Response id */
wifi_rsp_sme_start_ssid_scan_id

/* Response structure */
struct wifi_msg_sme_start_ssid_scan_rsp_t
{
  uint16 result;
};
```

Table 3.221. Events Generated

Event	Description
sme_scan_result	Sent for each Access Point discovered
sme_scan_result_drop	Sent for each Access Point dropped from the internal scan list
sme_scanned	Sent after the scan completes

3.9.1.30 cmd_sme_start_wps

This command is used to start the Wi-Fi Protected Setup (WPS) session.

Only WPS push-button mode (PBC) is supported. The session will timeout in 120 seconds if no Access Point in WPS mode is discovered.

This command can only be used in client mode.

Due to encryption used in communication, a small delay (up to 3 seconds) in API responses can appear before receiving events.

Table 3.222. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x11	method	Message ID

Table 3.223. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x11	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript command

```
call sme_start_wps()(result, hw_interface)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_start_wps();

/* Response id */
wifi_rsp_sme_start_wps_id

/* Response structure */
struct wifi_msg_sme_start_wps_rsp_t
{
  uint16 result;,
  uint8 hw_interface;
};
```

Table 3.224. Events Generated

Event	Description
sme_wps_completed	Sent when WPS session completes successfully
sme_wps_failed	Sent when WPS session fails or timeouts
sme_wps_credential_ssid	Received network name credential
sme_wps_credential_password	Received network password credential

3.9.1.31 cmd_sme_stop_ap_mode

This command is used to stop the Access Point mode.

Table 3.225. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0c	method	Message ID

Table 3.226. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0c	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript command

```
call sme_stop_ap_mode()(result, hw_interface)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_stop_ap_mode();

/* Response id */
wifi_rsp_sme_stop_ap_mode_id

/* Response structure */
struct wifi_msg_sme_stop_ap_mode_rsp_t
{
  uint16 result;,
  uint8 hw_interface;
};
```

Table 3.227. Events Generated

Event	Description
sme_ap_mode_stopped	Sent after Access Point mode was stopped
tcpip_configuration	Sent when the IP address of the Wi-Fi interface has been deactivated

Event	Description
tcpip_dns_configuration	Sent when the DNS address of the Wi-Fi interface has been deactivated
sme_interface_status	Sent after Wi-Fi interface has been deactivated

3.9.1.32 cmd_sme_stop_p2p_group

This command is used to stop Wi-Fi Direct Group

Table 3.228. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x1e	method	Message ID

Table 3.229. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x1e	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	

BGScript command

```
call sme_stop_p2p_group()(result, hw_interface)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_stop_p2p_group();

/* Response id */
wifi_rsp_sme_stop_p2p_group_id

/* Response structure */
struct wifi_msg_sme_stop_p2p_group_rsp_t
{
  uint16 result;,
  uint8 hw_interface;
};
```

Table 3.230. Events Generated

Event	Description
sme_ap_mode_stopped	Sent after Wi-Fi Direct Group mode was stopped
tcpip_configuration	Sent when the IP address of the Wi-Fi interface has been deactivated
tcpip_dns_configuration	Sent when the DNS address of the Wi-Fi interface has been deactivated

Event	Description
sme_interface_status	Sent after Wi-Fi interface has been deactivated

3.9.1.33 cmd_sme_stop_scan

This command is used to terminate the active scanning procedure.

Table 3.231. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x04	method	Message ID

Table 3.232. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x04	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_stop_scan()(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_stop_scan();

/* Response id */
wifi_rsp_sme_stop_scan_id

/* Response structure */
struct wifi_msg_sme_stop_scan_rsp_t
{
  uint16 result;
};
```

Table 3.233. Events Generated

Event	Description
sme_scanned	Sent after scan stopped

3.9.1.34 cmd_sme_stop_wps

This command is used to stop the Wi-Fi Protected Setup (WPS) session.

Table 3.234. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x12	method	Message ID

Table 3.235. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x12	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript command

```
call sme_stop_wps()(result, hw_interface)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_stop_wps();

/* Response id */
wifi_rsp_sme_stop_wps_id

/* Response structure */
struct wifi_msg_sme_stop_wps_rsp_t
{
  uint16 result;,
  uint8 hw_interface;
};
```

Table 3.236. Events Generated

Event	Description
sme_wps_stopped	Sent after WPS session was stopped

3.9.1.35 cmd_sme_wifi_off

This command is used to turn off the Wi-Fi radio.

Table 3.237. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x01	method	Message ID

Table 3.238. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x01	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_wifi_off()(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_wifi_off();

/* Response id */
wifi_rsp_sme_wifi_off_id

/* Response structure */
struct wifi_msg_sme_wifi_off_rsp_t
{
  uint16 result;
};
```

Table 3.239. Events Generated

Event	Description
sme_wifi_is_off	Sent after Wi-Fi radio has been switched off

3.9.1.36 cmd_sme_wifi_on

This command is used to switch on the Wi-Fi radio.

Table 3.240. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x00	method	Message ID

Table 3.241. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x00	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call sme_wifi_on()(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_sme_wifi_on();

/* Response id */
wifi_rsp_sme_wifi_on_id

/* Response structure */
struct wifi_msg_sme_wifi_on_rsp_t
{
  uint16 result;
};
```

Table 3.242. Events Generated

Event	Description
sme_wifi_is_on	Sent after Wi-Fi radio has been switched on

3.9.2 sme events

3.9.2.1 evt_sme_ap_client_joined

This event indicates that a Wi-Fi client has joined the Access Point.

Table 3.243. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x07	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0d	method	Message ID
4-9	hw_addr	mac_address	MAC address of the client
10	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript event

```
event sme_ap_client_joined(mac_address, hw_interface)
```

C Functions

```
/* Event id */
wifi_evt_sme_ap_client_joined_id

/* Event structure */
struct wifi_msg_sme_ap_client_joined_evt_t
{
  hw_addr mac_address;,
  uint8 hw_interface;
};
```

3.9.2.2 evt_sme_ap_client_left

This event indicates that a Wi-Fi client has left the Access Point.

In case a client moves out of range or is abruptly powered off, it might take from 180 to 270 seconds for this event to be issued.

Table 3.244. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x07	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0e	method	Message ID
4-9	hw_addr	mac_address	MAC address of the client
10	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript event

```
event sme_ap_client_left(mac_address, hw_interface)
```

C Functions

```
/* Event id */
wifi_evt_sme_ap_client_left_id

/* Event structure */
struct wifi_msg_sme_ap_client_left_evt_t
{
  hw_addr mac_address;,
  uint8 hw_interface;
};
```

3.9.2.3 evt_sme_ap_mode_failed

This event indicates that the Access Point mode has failed.

Table 3.245. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0c	method	Message ID
4-5	uint16	reason	Reason of fail For values refer to the error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript event

```
event sme_ap_mode_failed(reason, hw_interface)
```

C Functions

```
/* Event id */
wifi_evt_sme_ap_mode_failed_id

/* Event structure */
struct wifi_msg_sme_ap_mode_failed_evt_t
{
  uint16 reason;,
  uint8 hw_interface;
};
```

3.9.2.4 evt_sme_ap_mode_started

This event indicates that the Access Point mode has been successfully started.

Table 3.246. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0a	method	Message ID
4	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript event

```
event sme_ap_mode_started(hw_interface)
```

C Functions

```
/* Event id */
wifi_evt_sme_ap_mode_started_id

/* Event structure */
struct wifi_msg_sme_ap_mode_started_evt_t
{
  uint8 hw_interface;
};
```

3.9.2.5 evt_sme_ap_mode_stopped

This event indicates that the Access Point mode has been stopped.

Table 3.247. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0b	method	Message ID
4	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript event

```
event sme_ap_mode_stopped(hw_interface)
```

C Functions

```
/* Event id */
wifi_evt_sme_ap_mode_stopped_id

/* Event structure */
struct wifi_msg_sme_ap_mode_stopped_evt_t
{
  uint8 hw_interface;
};
```

3.9.2.6 evt_sme_connect_failed

This event indicates a failed connection to an Access Point.

The event may occur if the device is unable to establish a connection to an Access Point or if the Access Point disconnects the device during the connection.

Table 3.248. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x08	method	Message ID
4-5	uint16	reason	Reason of connect fail For values refer to the error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript event

```
event sme_connect_failed(reason, hw_interface)
```

C Functions

```
/* Event id */
wifi_evt_sme_connect_failed_id

/* Event structure */
struct wifi_msg_sme_connect_failed_evt_t
{
  uint16 reason;,
  uint8 hw_interface;
};
```

3.9.2.7 evt_sme_connect_retry

This event indicates that a connection attempt failed.

The connection is automatically retried until the retry limit of 10 attempts is exceeded. This event typically appears when the device is commanded to connect to a wireless network but the given password is wrong. Automatic retries can be stopped with the cmd_sme_disconnect command.

Table 3.249. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x09	method	Message ID
4	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript event

```
event sme_connect_retry(hw_interface)
```

C Functions

```
/* Event id */
wifi_evt_sme_connect_retry_id

/* Event structure */
struct wifi_msg_sme_connect_retry_evt_t
{
  uint8 hw_interface;
};
```

3.9.2.8 evt_sme_connected

This event indicates a connection status to an Access Point.

Table 3.250. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x08	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x05	method	Message ID
4	int8	status	Connection status • **0:** connected • **1:** not connected
5	uint8	hw_interface	Hardware interface • **0:** Wi-Fi
6-11	hw_addr	bssid	The BSSID of the device that the device connected to

BGScript event

```
event sme_connected(status, hw_interface, bssid)
```

C Functions

```
/* Event id */
wifi_evt_sme_connected_id

/* Event structure */
struct wifi_msg_sme_connected_evt_t
{
  int8 status;,
  uint8 hw_interface;,
  hw_addr bssid;
};
```

3.9.2.9 evt_sme_disconnected

This event indicates a disconnection from an Access Point.

The event occurs either because the sme_disconnect command was issued or connection to an Access Point was lost. Connection loss could occur because the Access Point has been switched off or the user has moved out of range. The timeout for detecting a lost connection is 50 beacons which under typical network configuration translates to roughly 5 seconds.

Table 3.251. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x06	method	Message ID
4-5	uint16	reason	Disconnect reason For values refer to the error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript event

```
event sme_disconnected(reason, hw_interface)
```

C Functions

```
/* Event id */
wifi_evt_sme_disconnected_id

/* Event structure */
struct wifi_msg_sme_disconnected_evt_t
{
  uint16 reason;,
  uint8 hw_interface;
};
```

3.9.2.10 evt_sme_interface_status

This event indicates the current Wi-Fi interface status.

Table 3.252. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x07	method	Message ID
4	uint8	hw_interface	Hardware interface • **0:** Wi-Fi
5	uint8	status	Interface status • **0:** interface down • **1:** interface up

BGScript event

```
event sme_interface_status(hw_interface, status)
```

C Functions

```
/* Event id */
wifi_evt_sme_interface_status_id

/* Event structure */
struct wifi_msg_sme_interface_status_evt_t
{
  uint8 hw_interface;,
  uint8 status;
};
```

3.9.2.11 evt_sme_p2p_client_wants_to_join

This event indicates that a Wi-Fi direct client wants to join our group. Use command p2p_accept_client to accept the connection within 5 seconds from receiving this event.

Table 3.253. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x06	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x1a	method	Message ID
4-9	hw_addr	mac_address	MAC address of the client

BGScript event

```
event sme_p2p_client_wants_to_join(mac_address)
```

C Functions

```
/* Event id */
wifi_evt_sme_p2p_client_wants_to_join_id

/* Event structure */
struct wifi_msg_sme_p2p_client_wants_to_join_evt_t
{
  hw_addr mac_address;
};
```

3.9.2.12 evt_sme_p2p_group_failed

This event indicates that Wi-Fi Direct Group is failed

Table 3.254. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x19	method	Message ID
4-5	uint16	reason	Failure reason
6	uint8	hw_interface	

BGScript event

```
event sme_p2p_group_failed(reason, hw_interface)
```

C Functions

```
/* Event id */
wifi_evt_sme_p2p_group_failed_id

/* Event structure */
struct wifi_msg_sme_p2p_group_failed_evt_t
{
  uint16 reason;,
  uint8 hw_interface;
};
```

3.9.2.13 evt_sme_p2p_group_started

This event indicates that Wi-Fi Direct Group is successfully started

Table 3.255. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x17	method	Message ID
4	uint8	hw_interface	

BGScript event

```
event sme_p2p_group_started(hw_interface)
```

C Functions

```
/* Event id */
wifi_evt_sme_p2p_group_started_id

/* Event structure */
struct wifi_msg_sme_p2p_group_started_evt_t
{
  uint8 hw_interface;
};
```

3.9.2.14 evt_sme_p2p_group_stopped

This event indicates that Wi-Fi Direct Group is stopped

Table 3.256. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x18	method	Message ID
4	uint8	hw_interface	

BGScript event

```
event sme_p2p_group_stopped(hw_interface)
```

C Functions

```
/* Event id */
wifi_evt_sme_p2p_group_stopped_id

/* Event structure */
struct wifi_msg_sme_p2p_group_stopped_evt_t
{
  uint8 hw_interface;
};
```

3.9.2.15 evt_sme_scan_result

This event indicates that an Access Point has been discovered.

After a scan has been started this event is sent every time an Access Point is discovered. Access Point is also added to internal scan list and it is possible to connect to it.

Table 3.257. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x0c	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x02	method	Message ID
4-9	hw_addr	bssid	The BSSID of the Access Point which was discovered
10	int8	channel	The channel on which the Access Point was detected
11-12	int16	rssi	The received signal strength indication of the Access Point
13	int8	snr	The signal to noise ratio of the Access Point
14	uint8	secure	Access Point security status as a bitmask • **bit 0:** defines whether the Access Point supports secure connections • **bit 1:** defines whether the Access Point supports WPS • **bit 2:** defines whether the Access Point supports WPA Enterprise • **bit 3:** defines whether SSID is hidden
15	uint8array	ssid	The SSID of the network the Access Point belongs to

BGScript event

```
event sme_scan_result(bssid, channel, rssi, snr, secure, ssid_len, ssid_data)
```

C Functions

```
/* Event id */
wifi_evt_sme_scan_result_id

/* Event structure */
struct wifi_msg_sme_scan_result_evt_t
{
  hw_addr bssid;,
  int8 channel;,
  int16 rssi;,
  int8 snr;,
  uint8 secure;,
  uint8array ssid;
};
```

3.9.2.16 evt_sme_scan_result_drop

This event indicates that the Access Point was dropped from the internal scan list.

The internal scan list has a limit of 40 Access Points. If the limit is exceeded, Access Point with the weakest signal strength is removed from the list to make room for new Access Points. It is no longer possible to connect to it anymore using the sme_connect_bssid command.

Table 3.258. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x06	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x03	method	Message ID
4-9	hw_addr	bssid	The BSSID of the Access Point that was dropped

BGScript event

```
event sme_scan_result_drop(bssid)
```

C Functions

```
/* Event id */
wifi_evt_sme_scan_result_drop_id

/* Event structure */
struct wifi_msg_sme_scan_result_drop_evt_t
{
  hw_addr bssid;
};
```

3.9.2.17 evt_sme_scan_result_filter

Scan result filter match

Table 3.259. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x08	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x1b	method	Message ID
4-9	hw_addr	bssid	The BSSID of the Access Point which was found
10	uint8	ie_id	Matching Information Element ID
11	uint8array	ie_data	Matching Information Element data

BGScript event

```
event sme_scan_result_filter(bssid, ie_id, ie_data_len, ie_data_data)
```

C Functions

```
/* Event id */
wifi_evt_sme_scan_result_filter_id

/* Event structure */
struct wifi_msg_sme_scan_result_filter_evt_t
{
  hw_addr bssid;,
  uint8 ie_id;,
  uint8array ie_data;
};
```

3.9.2.18 evt_sme_scan_sort_finished

This event indicates that the scan result sort is finished.

Table 3.260. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x00	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x10	method	Message ID

BGScript event

```
event sme_scan_sort_finished()
```

C Functions

```
/* Event id */
wifi_evt_sme_scan_sort_finished_id

/* Event structure */
struct wifi_msg_sme_scan_sort_finished_evt_t
{
};
```

3.9.2.19 evt_sme_scan_sort_result

This event indicates a scan result from the internal scan list.

Table 3.261. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x0c	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0f	method	Message ID
4-9	hw_addr	bssid	The BSSID of the Access Point discovered
10	int8	channel	The channel on which the Access Point was seen
11-12	int16	rssi	The received signal strength indication of the found Access Point
13	int8	snr	The signal to noise ratio of the Access Point
14	uint8	secure	Access Point security status as a bitmask • **bit 0:** Access Point supports secure connections • **bit 1:** Access Point supports WPS
15	uint8array	ssid	The SSID of the network the Access Point belongs to

BGScript event

```
event sme_scan_sort_result(bssid, channel, rssi, snr, secure, ssid_len, ssid_data)
```

C Functions

```
/* Event id */
wifi_evt_sme_scan_sort_result_id

/* Event structure */
struct wifi_msg_sme_scan_sort_result_evt_t
{
  hw_addr bssid;,
  int8 channel;,
  int16 rssi;,
  int8 snr;,
  uint8 secure;,
  uint8array ssid;
};
```

3.9.2.20 evt_sme_scanned

This event indicates that the scan for Access Points is finished.

Table 3.262. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x04	method	Message ID
4	int8	status	Scan status • **0:** scan finished successfully • **non-zero:** scan failed with an error

BGScript event

```
event sme_scanned(status)
```

C Functions

```
/* Event id */
wifi_evt_sme_scanned_id

/* Event structure */
struct wifi_msg_sme_scanned_evt_t
{
  int8 status;
};
```

3.9.2.21 evt_sme_signal_quality

This event indicates the signal quality (RSSI value) of the connection in dBm units.

Table 3.263. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x16	method	Message ID
4	int8	rssi	The received signal strength indication (RSSI) in dBm units
5	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript event

```
event sme_signal_quality(rssi, hw_interface)
```

C Functions

```
/* Event id */
wifi_evt_sme_signal_quality_id

/* Event structure */
struct wifi_msg_sme_signal_quality_evt_t
{
  int8 rssi;,
  uint8 hw_interface;
};
```

3.9.2.22 evt_sme_wifi_is_off

This event indicates that the Wi-Fi radio has been powered off.

The event can also indicate that the Wi-Fi radio had an internal error and was shut down, in which case the user application must re-initialize the radio by turning it on.

Table 3.264. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x01	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript event

```
event sme_wifi_is_off(result)
```

C Functions

```
/* Event id */
wifi_evt_sme_wifi_is_off_id

/* Event structure */
struct wifi_msg_sme_wifi_is_off_evt_t
{
  uint16 result;
};
```

3.9.2.23 evt_sme_wifi_is_on

This event indicates that the Wi-Fi radio is powered up and ready to receive commands.

Table 3.265. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x00	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript event

```
event sme_wifi_is_on(result)
```

C Functions

```
/* Event id */
wifi_evt_sme_wifi_is_on_id

/* Event structure */
struct wifi_msg_sme_wifi_is_on_evt_t
{
  uint16 result;
};
```

3.9.2.24 evt_sme_wps_completed

This event indicates that the Wi-Fi Protected Setup (WPS) session was completed successfully.

Table 3.266. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x12	method	Message ID
4	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript event

```
event sme_wps_completed(hw_interface)
```

C Functions

```
/* Event id */
wifi_evt_sme_wps_completed_id

/* Event structure */
struct wifi_msg_sme_wps_completed_evt_t
{
  uint8 hw_interface;
};
```

3.9.2.25 evt_sme_wps_credential_password

This event indicates the password of the network in relation to Wi-Fi Protected Setup (WPS) session.

Table 3.267. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x15	method	Message ID
4	uint8	hw_interface	Hardware interface • **0:** Wi-Fi
5	uint8array	password	Password of the network • WPA/WPA2-PSK is either a hash of 64 hexadecimal characters, or a pass-phrase of 8 to 63 ASCII-encoded characters • 64-bit WEP key is either 5 ASCII-encoded characters or 10 HEX characters • 128-bit WEP key is either 13 ASCII-encoded characters or 26 HEX characters

BGScript event

```
event sme_wps_credential_password(hw_interface, password_len, password_data)
```

C Functions

```
/* Event id */
wifi_evt_sme_wps_credential_password_id

/* Event structure */
struct wifi_msg_sme_wps_credential_password_evt_t
{
  uint8 hw_interface;,
  uint8array password;
};
```

3.9.2.26 evt_sme_wps_credential_ssid

This event indicates the SSID of the network in relation to Wi-Fi Protected Setup (WPS) session.

Table 3.268. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x14	method	Message ID
4	uint8	hw_interface	Hardware interface • **0:** Wi-Fi
5	uint8array	ssid	SSID of the network **Length:** 1 - 32 bytes

BGScript event

```
event sme_wps_credential_ssid(hw_interface, ssid_len, ssid_data)
```

C Functions

```
/* Event id */
wifi_evt_sme_wps_credential_ssid_id

/* Event structure */
struct wifi_msg_sme_wps_credential_ssid_evt_t
{
  uint8 hw_interface;,
  uint8array ssid;
};
```

3.9.2.27 evt_sme_wps_failed

This event indicates that the Wi-Fi Protected Setup (WPS) session failed.

Table 3.269. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x13	method	Message ID
4-5	uint16	reason	Reason of fail For values refer to the error codes.
6	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript event

```
event sme_wps_failed(reason, hw_interface)
```

C Functions

```
/* Event id */
wifi_evt_sme_wps_failed_id

/* Event structure */
struct wifi_msg_sme_wps_failed_evt_t
{
  uint16 reason;,
  uint8 hw_interface;
};
```

3.9.2.28 evt_sme_wps_stopped

This event indicates that the Wi-Fi Protected Setup (WPS) session was stopped.

Table 3.270. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x11	method	Message ID
4	uint8	hw_interface	Hardware interface • **0:** Wi-Fi

BGScript event

```
event sme_wps_stopped(hw_interface)
```

C Functions

```
/* Event id */
wifi_evt_sme_wps_stopped_id

/* Event structure */
struct wifi_msg_sme_wps_stopped_evt_t
{
  uint8 hw_interface;
};
```

3.9.3 sme enumerations

3.9.3.1 enum_sme_eap_type

This enumeration defines the EAP types.

Table 3.271. Enumerations

Value	Name	Description
0	sme_eap_type_none	None
1	sme_eap_type_tls	TLS
2	sme_eap_type_peap	PEAP
3	sme_eap_type_mschapv2	MSCHAPv2

3.10 System Class Commands

The commands in this class are related to general control of the Module.

This class consists of the following items:

- *Commands*
  - `cmd_system_hello`
  - `cmd_system_reset`
  - `cmd_system_set_max_power_saving_state`
  - `cmd_system_sync`
- *Events*
  - `evt_system_boot` indicates that the Module has started and is ready to receive commands.
  - `evt_system_power_saving_state` indicates the power saving state the Module has entered into.
  - `evt_system_sw_exception` indicates occurrence of SW exception with Module resetting automatically to normal mode.
- *Enumerations*
  - `enum_system_power_saving_state` defines the four possible power saving states available for the Module.

The command `cmd_system_sync` can generate the following events from other classes

- `evt_sme_wifi_on`
- `evt_sme_wifi_off`
- `evt_sme_scan_result`
- `evt_sme_connected`
- `evt_tcpip_endpoint_status`
- `evt_endpoint_status`
- `evt_config_mac_address`
- `evt_tcpip_configuration`
- `evt_tcpip_dns_configuration`

A typical use for the `cmd_system_sync` command is to get the Module information and status for a GUI program startup.

Note: Events are emitted one after another and can cause unwanted impacts to BGScript execution.

The command `cmd_system_hello` can be used to check that the Module is up and running.

The command `cmd_system_set_max_power_saving_state` is used to set the deepest power saving state into which the Module is allowed to go into. Note that in power save states 0, 1 and 2 the host communication interface is available but in 3 and higher host communication interface is set down. In the case where the Module is in deep sleep (power saving state 3 or 4) an external interrupt is required to wake up the Module.

Note: After an interrupt signal wakes the Module up, host communication must be initiated within 2 seconds, because otherwise the Module will return to deep sleep mode.

EXAMPLES

1. **Using power savings modes**
   a. Configure one GPIO pin of the Module as an external interrupt in the project configuration file.
   b. Build and load the firmware image file into the Module.
   c. With the Module in state where power saving is possible, like connected to a Wi-Fi Access Point, send the `cmd_system_set_max_power_saving_state` using parameter "4" which defines that the Module is allowed to go into deepest power saving state. The Module will wake up autonomously to handle internal operations, like TCP keep alive.
   d. Wake up the Module using the external interrupt pin configured in Step 1.
   e. Set the Module maximum power saving state to *"0"* using the `cmd_system_set_max_power_saving_state` command.
   f. Wait for the `evt_system_power_saving_state` event.
   g. Execute the required communication.
   h. Wait until communication is finished.
   i. Enable the power saving like in Step 3.

Note: In power saving modes **0**, **1** and **2** timers time varion is less than *1ms*, in power saving mode **3** variation is less than *100ms* and in power saving mode **4** time turns in unspecified value and makes timers unreliable.

Troubleshooting information

Problem

The `evt_system_sw_execption` event is received .

Possible reasons and solutions

- Error in BGScript or stack usage of BGScript is too large. Debug the BGScript or try with another script, for example some script from SDK's example.
- Noise in power supply line. Improve power supply design.
- Firmware bug. Contact Silicon Labs Customer Support.

3.10.1 system commands

3.10.1.1 cmd_system_hello

This command is used to verify that communication works between the external host and the device.

The device sends a response without doing anything else.

Table 3.272. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x02	method	Message ID

Table 3.273. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x00	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x02	method	Message ID

BGScript command

```
call system_hello()()
```

BGLIB C API

```
/* Function */
void wifi_cmd_system_hello();

/* Response id */
wifi_rsp_system_hello_id

/* Response structure */
struct wifi_msg_system_hello_rsp_t
{
};
```

3.10.1.2 cmd_system_reset

This command is used to reset the device.

The command does not have a response, but it triggers a boot event.

Table 3.274. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x01	method	Message ID
4	uint8	dfu	Boot mode • **0:** Boot to normal mode • **1:** Boot to DFU mode See the DFU class for further details.

BGScript command

```
call system_reset(dfu)
```

BGLIB C API

```
/* Function */
void wifi_cmd_system_reset(uint8 dfu);

/* Command does not have a response */
```

Table 3.275. Events Generated

Event	Description
system_boot	Sent after the device has booted to normal mode
dfu_boot	Sent after the device has booted to DFU mode

3.10.1.3 cmd_system_set_max_power_saving_state

This command is used to set the maximum power saving state allowed for the device.

Initial state is power_saving_state_0

Table 3.276. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x03	method	Message ID
4	uint8	state	Power saving state **Range:** 0 - 4

Table 3.277. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x03	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call system_set_max_power_saving_state(state)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_system_set_max_power_saving_state(uint8 state);

/* Response id */
wifi_rsp_system_set_max_power_saving_state_id

/* Response structure */
struct wifi_msg_system_set_max_power_saving_state_rsp_t
{
  uint16 result;
};
```

3.10.1.4 cmd_system_sync

This command is used to retrieve the device status.

When the sync command is sent, multiple events are output representing the device status.

Table 3.278. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x00	method	Message ID

Table 3.279. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x00	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x00	method	Message ID

BGScript command

```
call system_sync()()
```

BGLIB C API

```
/* Function */
void wifi_cmd_system_sync();

/* Response id */
wifi_rsp_system_sync_id

/* Response structure */
struct wifi_msg_system_sync_rsp_t
{
};
```

Table 3.280. Events Generated

Event	Description
sme_wifi_is_on	Sent if Wi-Fi has been switched on
sme_wifi_is_off	Sent if Wi-Fi has been switched off
sme_scan_result	Sent for each cached scan result
sme_connected	Device connection status
tcpip_endpoint_status	Sent for each TCP/IP endpoint
endpoint_status	Sent for each endpoint
config_mac_address	Device MAC address

Event	Description
tcpip_configuration	Device TCP/IP configuration
tcpip_dns_configuration	Device DNS configuration

3.10.2 system events

3.10.2.1 evt_system_boot

This event indicates that the device has started and is ready to receive commands.

Table 3.281. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x0b	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x00	method	Message ID
4-5	uint16	hw	Hardware version
6-7	uint16	bootloader_version	Bootloader version
8-9	uint16	major	Software major version
10-11	uint16	minor	Software minor version
12-13	uint16	build	Software build number
14	uint8array	revision	Software revision string

BGScript event

```
event system_boot(hw, bootloader_version, major, minor, build, revision_len, revision_data)
```

C Functions

```
/* Event id */
wifi_evt_system_boot_id

/* Event structure */
struct wifi_msg_system_boot_evt_t
{
  uint16 hw;,
  uint16 bootloader_version;,
  uint16 major;,
  uint16 minor;,
  uint16 build;,
  uint8array revision;
};
```

3.10.2.2 evt_system_power_saving_state

This event indicates the power saving state into which the device has entered.

Table 3.282. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x03	method	Message ID
4	uint8	state	Power saving state

BGScript event

```
event system_power_saving_state(state)
```

C Functions

```
/* Event id */
wifi_evt_system_power_saving_state_id

/* Event structure */
struct wifi_msg_system_power_saving_state_evt_t
{
  uint8 state;
};
```

3.10.2.3 evt_system_sw_exception

This event indicates that a software exception has occurred.

The device resets automatically to normal mode when an exception occurs.

Table 3.283. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x05	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x02	method	Message ID
4-7	uint32	address	Address where exception occurred
8	uint8	type	Type of exception

BGScript event

```
event system_sw_exception(address, type)
```

C Functions

```
/* Event id */
wifi_evt_system_sw_exception_id

/* Event structure */
struct wifi_msg_system_sw_exception_evt_t
{
  uint32 address;,
  uint8 type;
};
```

3.10.3 system enumerations

3.10.3.1 enum_system_main_state

This enumeration defines the system states of the device.

Table 3.284. Enumerations

Value	Name	Description
1	system_idle	Idle
2	system_powered	Wi-Fi powered and initialized
4	system_connecting	Connecting to an Access Point
8	system_connected	Connected to an Access Point
16	system_wps	Wi-Fi Protected Setup mode
32	system_ap	AP mode
64	system_p2p	Wi-Fi Direct discovery
128	system_p2p_go	Wi-Fi Direct GO

3.10.3.2 enum_system_power_saving_state

This enumeration defines the power saving state of the device.

Table 3.285. Enumerations

Value	Name	Description
0	system_power_saving_state_0	No powersave. MCU in emode EM0, Wi-Fi chip no power save. Only this state can be used in access point mode
1	system_power_saving_state_1	MCU in emode EM1, Wi-Fi chip in middle power save mode.
2	system_power_saving_state_2	MCU in emode EM1, Wi-Fi chip in lowest power consumption mode.
3	system_power_saving_state_3	MCU in emode EM2, Wi-Fi chip on lowest power consumption mode. MCU wake in every 2 second for 50 milliseconds. Keeps IP stack timers and soft timers in time with lower resolution. Wakeup with interrupt pin for 2 seconds
4	system_power_saving_state_4	MCU in emode EM2, Wi-Fi chip on lowest power save mode. IP stack timers and software timers time is lost. Wakeup with interrupt pin for 2 seconds

3.11 TCP Stack Command Class

The commands in this class are used to control the TCP/IP stack of the Module.

This class consists of the following items:

IP stack related

- *Commands*
  - `cmd_tcpip_configure`
  - `cmd_tcpip_multicast_join`
  - `cmd_tcpip_multicast_leave`
- *Events*
  - `evt_tcpip_configuration`

TCP and UDP related

- *Commands*
  - `cmd_tcpip_start_tcp_server`
  - `cmd_tcpip_tcp_connect`
  - `cmd_tcpip_start_udp_server`
  - `cmd_tcpip_udp_connect`
  - `cmd_tcpip_udp_bind`
- *Events*
  - `evt_tcpip_endpoint_status`
  - `evt_tcpip_udp_data`

DNS, MSDN and DNS-SD related

- *Commands*
  - `cmd_tcpip_dns_configure`
  - `cmd_tcpip_dns_gethostbyname`
  - `cmd_tcpip_mdns_set_hostname`
  - `cmd_tcpip_mdns_stop`
  - `cmd_tcpip_dnssd_add_service`
  - `cmd_tcpip_dnssd_add_service_instance`
  - `cmd_tcpip_dnssd_add_service_attribute`
  - `cmd_tcpip_dnssd_remove_service`
  - `cmd_tcpip_dnssd_start_service`
  - `cmd_tcpip_dnssd_stop_service`
- *Events*
  - `evt_tcpip_dns_configuration`
  - `evt_tcpip_dns_gethostbyname_result`
  - `evt_tcpip_mdns_started`
  - `evt_tcpip_mdns_stopped`
  - `evt_tcpip_mdns_failed`
  - `evt_tcpip_dnssd_service_started`
  - `evt_tcpip_dnssd_service_failed`
  - `evt_tcpip_dnssd_service_stopped`

DHCP related

- *Commands*
  - `cmd_tcpip_dhcp_set_hostname`
  - `cmd_tcpip_dhcp_enable_routing`
  - `cmd_tcpip_dhcp_configure`
  - `cms_tcpip_dhcp_clients`

- *Events*
  - *evt_tcpip_dhcp_configuration*
  - *evt_tcpip_dhcp_client*

TLS/SSL related

- *Commands*
  - cmd_tcpip_tls_connect
  - cmd_tcpip_tls_set_authmode
  - cmd_tcpip_tls_set_user_certificate
- *Events*
  - evt_tcpip_tls_verify_result

3.11.1 tcpip commands

3.11.1.1 cmd_tcpip_configure

This command is used to configure IP settings of the device.

When enabling DHCP, the IP settings will be stored, but they will be overridden as soon as IP configuration is received from the remote DHCP server.

Table 3.286. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x0d	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x04	method	Message ID
4-7	ipv4	address	Local IP address of the device
8-11	ipv4	netmask	Netmask of the device
12-15	ipv4	gateway	Gateway used by the device
16	uint8	use_dhcp	Defines whether DHCP is used to retrieve IP settings • **0:** DHCP disabled • **1:** DHCP enabled This setting is only applicable in client mode.

Table 3.287. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x04	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_configure(address, netmask, gateway, use_dhcp)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_configure(ipv4 address, ipv4 netmask, ipv4 gateway, uint8 use_dhcp);

/* Response id */
wifi_rsp_tcpip_configure_id

/* Response structure */
struct wifi_msg_tcpip_configure_rsp_t
{
```

```
  uint16 result;
};
```

Table 3.288. Events Generated

Event	Description
tcpip_configuration	Sent when TCP/IP configuration changes

3.11.1.2 cmd_tcpip_dhcp_clients

This command is used to read mac and ip addresses of connected clients in Access Point mode

Table 3.289. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x1a	method	Message ID

Table 3.290. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x1a	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	client_cnt	Count of connected clients

BGScript command

```
call tcpip_dhcp_clients()(result, client_cnt)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_dhcp_clients();

/* Response id */
wifi_rsp_tcpip_dhcp_clients_id

/* Response structure */
struct wifi_msg_tcpip_dhcp_clients_rsp_t
{
  uint16 result;,
  uint8 client_cnt;
};
```

Table 3.291. Events Generated

Event	Description
tcpip_dhcp_client	This event contains a client information. Send for each client

3.11.1.3 cmd_tcpip_dhcp_configure

This command is used to configure the DHCP Server

Table 3.292. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x0c	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x18	method	Message ID
4-7	ipv4	address	First address of the DHCP Server address pool
8-11	ipv4	subnet_mask	Subnetwork mask
12-15	uint32	lease_time	Address lease timeout in seconds **Default:** 86400

Table 3.293. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x18	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_dhcp_configure(address, subnet_mask, lease_time)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_dhcp_configure(ipv4 address, ipv4 subnet_mask, uint32 lease_time);

/* Response id */
wifi_rsp_tcpip_dhcp_configure_id

/* Response structure */
struct wifi_msg_tcpip_dhcp_configure_rsp_t
{
  uint16 result;
};
```

Table 3.294. Events Generated

Event	Description
tcpip_dhcp_configuration	This event contains DHCP Server configuration

3.11.1.4 cmd_tcpip_dhcp_enable_routing

This command is used to set whether the built-in DHCP server responses include gateway and DNS server information.

Gateway and DNS information is included by default.

Table 3.295. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0b	method	Message ID
4	uint8	enable	Routing options • **0:** disabled • **1:** enabled

Table 3.296. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0b	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_dhcp_enable_routing(enable)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_dhcp_enable_routing(uint8 enable);

/* Response id */
wifi_rsp_tcpip_dhcp_enable_routing_id

/* Response structure */
struct wifi_msg_tcpip_dhcp_enable_routing_rsp_t
{
  uint16 result;
};
```

3.11.1.5 cmd_tcpip_dhcp_set_hostname

This command is used to set the DHCP host name parameter (option 12) used in client DHCPDISCOVER and DHCPREQUEST messages.

Table 3.297. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x08	method	Message ID
4	uint8array	hostname	Host name to use

Table 3.298. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x08	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_dhcp_set_hostname(hostname_len, hostname_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_dhcp_set_hostname(uint8 hostname_len, const uint8 *hostname_data);

/* Response id */
wifi_rsp_tcpip_dhcp_set_hostname_id

/* Response structure */
struct wifi_msg_tcpip_dhcp_set_hostname_rsp_t
{
  uint16 result;
};
```

3.11.1.6 cmd_tcpip_dns_configure

This command is used to configure DNS settings of the device.

The primary DNS server is set to 208.67.222.222 (resolver1.opendns.com) by default, the secondary DNS server is zero.

Table 3.299. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x05	method	Message ID
4	uint8	index	Index of the DNS server • **0:** primary DNS server • **1:** secondary DNS server
5-8	ipv4	address	IP address of the DNS server

Table 3.300. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x05	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_dns_configure(index, address)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_dns_configure(uint8 index, ipv4 address);

/* Response id */
wifi_rsp_tcpip_dns_configure_id

/* Response structure */
struct wifi_msg_tcpip_dns_configure_rsp_t
{
  uint16 result;
};
```

Table 3.301. Events Generated

Event	Description
tcpip_dns_configuration	Sent when DNS configuration changes

3.11.1.7 cmd_tcpip_dns_gethostbyname

This command is used to resolve the IP address of a domain name using the configured DNS servers.

Table 3.302. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x06	method	Message ID
4	uint8array	name	The fully qualified domain name to resolve

Table 3.303. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x06	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_dns_gethostbyname(name_len, name_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_dns_gethostbyname(uint8 name_len, const uint8 *name_data);

/* Response id */
wifi_rsp_tcpip_dns_gethostbyname_id

/* Response structure */
struct wifi_msg_tcpip_dns_gethostbyname_rsp_t
{
  uint16 result;
};
```

Table 3.304. Events Generated

Event	Description
tcpip_dns_gethostbyname_result	Sent when DNS resolver query completes

3.11.1.8 cmd_tcpip_dnssd_add_service

This command is used to add a new DNS-SD service.

The maximum amount of DNS-SD services is 3.

Table 3.305. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0f	method	Message ID
4-5	uint16	port	Service port **Range:** 1 - 65535
6	uint8	protocol	Service protocol • **0:** TCP • **1:** UDP
7	uint8array	service	Service name **Length:** 1 - 15 bytes

Table 3.306. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0f	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	index	Index of the DNS-SD service

BGScript command

```
call tcpip_dnssd_add_service(port, protocol, service_len, service_data)(result, index)
```

BGLIB C API

```
/* Function */
void  wifi_cmd_tcpip_dnssd_add_service(uint16  port,  uint8  protocol,  uint8  service_len,  const  uint8
*service_data);

/* Response id */
wifi_rsp_tcpip_dnssd_add_service_id

/* Response structure */
struct wifi_msg_tcpip_dnssd_add_service_rsp_t
{
```

```
  uint16 result;,
  uint8 index;
};
```

3.11.1.9 cmd_tcpip_dnssd_add_service_attribute

This command is used to add a service attribute to a DNS-SD service.

One DNS-SD service can contain multiple service attributes. The maximum combined length of service attributes is 64 bytes.

Table 3.307. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x11	method	Message ID
4	uint8	index	Index of the DNS-SD service
5	uint8array	attribute	Service attribute **Length:** 1 - 64 bytes

Table 3.308. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x11	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_dnssd_add_service_attribute(index, attribute_len, attribute_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_dnssd_add_service_attribute(uint8 index, uint8 attribute_len, const uint8 *attribute_data);

/* Response id */
wifi_rsp_tcpip_dnssd_add_service_attribute_id

/* Response structure */
struct wifi_msg_tcpip_dnssd_add_service_attribute_rsp_t
{
  uint16 result;
};
```

3.11.1.10 cmd_tcpip_dnssd_add_service_instance

This command is used to set the instance name of a DNS-SD service.

Table 3.309. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x10	method	Message ID
4	uint8	index	Index of the DNS-SD service
5	uint8array	instance	Service instance name **Length:** 1 - 63 bytes

Table 3.310. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x10	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_dnssd_add_service_instance(index, instance_len, instance_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_dnssd_add_service_instance(uint8 index, uint8 instance_len, const uint8 *instance_data);

/* Response id */
wifi_rsp_tcpip_dnssd_add_service_instance_id

/* Response structure */
struct wifi_msg_tcpip_dnssd_add_service_instance_rsp_t
{
  uint16 result;
};
```

3.11.1.11 cmd_tcpip_dnssd_remove_service

This command is used to remove a DNS-SD service.

Table 3.311. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x12	method	Message ID
4	uint8	index	Index of the DNS-SD service

Table 3.312. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x12	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_dnssd_remove_service(index)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_dnssd_remove_service(uint8 index);

/* Response id */
wifi_rsp_tcpip_dnssd_remove_service_id

/* Response structure */
struct wifi_msg_tcpip_dnssd_remove_service_rsp_t
{
  uint16 result;
};
```

3.11.1.12 cmd_tcpip_dnssd_start_service

This command is used to start a DNS-SD service.

The DNS-SD service cannot be started until the instance name has been set using the tcpip_dnssd_add_service_instance command.

Table 3.313. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x13	method	Message ID
4	uint8	index	Index of the DNS-SD service

Table 3.314. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x13	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_dnssd_start_service(index)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_dnssd_start_service(uint8 index);

/* Response id */
wifi_rsp_tcpip_dnssd_start_service_id

/* Response structure */
struct wifi_msg_tcpip_dnssd_start_service_rsp_t
{
  uint16 result;
};
```

Table 3.315. Events Generated

Event	Description
tcpip_dnssd_service_started	Sent when the DNS-SD service has been successfully started
tcpip_dnssd_service_failed	Sent when the DNS-SD service startup fails

3.11.1.13 cmd_tcpip_dnssd_stop_service

This command is used to stop a DNS-SD service.

Table 3.316. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x14	method	Message ID
4	uint8	index	Index of the DNS-SD service

Table 3.317. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x14	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_dnssd_stop_service(index)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_dnssd_stop_service(uint8 index);

/* Response id */
wifi_rsp_tcpip_dnssd_stop_service_id

/* Response structure */
struct wifi_msg_tcpip_dnssd_stop_service_rsp_t
{
  uint16 result;
};
```

Table 3.318. Events Generated

Event	Description
tcpip_dnssd_service_stopped	Sent when the DNS-SD service has been stopped

3.11.1.14 cmd_tcpip_force_tcp_server_endpoint_close

This command is used to close tcp server endpoint even when tcp clients are connected.

Table 3.319. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x1c	method	Message ID
4	uint8	endpoint	Index of the endpoint to close **Range:** 0 - 30

Table 3.320. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x1c	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	endpoint	Index of the endpoint closed **Range:** 0 - 30

BGScript command

```
call tcpip_force_tcp_server_endpoint_close(endpoint)(result, endpoint)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_force_tcp_server_endpoint_close(uint8 endpoint);

/* Response id */
wifi_rsp_tcpip_force_tcp_server_endpoint_close_id

/* Response structure */
struct wifi_msg_tcpip_force_tcp_server_endpoint_close_rsp_t
{
  uint16 result;,
  uint8 endpoint;
};
```

Table 3.321. Events Generated

Event	Description
endpoint_status	Sent when the endpoint status changes

3.11.1.15 cmd_tcpip_mdns_gethostbyname

This command is used to resolve the IP address of a domain name using Multicast DNS (MDNS).

Table 3.322. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x19	method	Message ID
4	uint8array	name	The fully qualified domain name to resolve. For example: host123.local

Table 3.323. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x19	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_mdns_gethostbyname(name_len, name_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_mdns_gethostbyname(uint8 name_len, const uint8 *name_data);

/* Response id */
wifi_rsp_tcpip_mdns_gethostbyname_id

/* Response structure */
struct wifi_msg_tcpip_mdns_gethostbyname_rsp_t
{
  uint16 result;
};
```

Table 3.324. Events Generated

Event	Description
tcpip_mdns_gethostbyname_result	Sent when MDNS resolver query completes

3.11.1.16 cmd_tcpip_mdns_set_hostname

This command is used to set the mDNS host name.

Table 3.325. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0c	method	Message ID
4	uint8array	hostname	The mDNS host name **Length:** 1 - 63 bytes The top-level domain .local is added automatically.

Table 3.326. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0c	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_mdns_set_hostname(hostname_len, hostname_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_mdns_set_hostname(uint8 hostname_len, const uint8 *hostname_data);

/* Response id */
wifi_rsp_tcpip_mdns_set_hostname_id

/* Response structure */
struct wifi_msg_tcpip_mdns_set_hostname_rsp_t
{
  uint16 result;
};
```

3.11.1.17 cmd_tcpip_mdns_start

This command is used to start the mDNS service.

The mDNS service cannot be started until the host name has been set using the tcpip_mdns_set_hostname command.

Table 3.327. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0d	method	Message ID

Table 3.328. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0d	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_mdns_start()(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_mdns_start();

/* Response id */
wifi_rsp_tcpip_mdns_start_id

/* Response structure */
struct wifi_msg_tcpip_mdns_start_rsp_t
{
  uint16 result;
};
```

Table 3.329. Events Generated

Event	Description
tcpip_mdns_started	Sent when the mDNS service has been successfully started
tcpip_mdns_failed	Sent when the mDNS service startup fails

3.11.1.18 cmd_tcpip_mdns_stop

This command is used to stop the mDNS service.

Table 3.330. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0e	method	Message ID

Table 3.331. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0e	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_mdns_stop()(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_mdns_stop();

/* Response id */
wifi_rsp_tcpip_mdns_stop_id

/* Response structure */
struct wifi_msg_tcpip_mdns_stop_rsp_t
{
  uint16 result;
};
```

Table 3.332. Events Generated

Event	Description
tcpip_mdns_stopped	Sent when the mDNS service has been stopped

3.11.1.19 cmd_tcpip_multicast_join

This command is used to join a multicast group.

Maximum number of additional multicast groups that can be joined is 4. Use 224.0.0.2 - 239.255.255.255 as address range. Note that 224.0.0.1 is automatically joined.

Table 3.333. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x15	method	Message ID
4-7	ipv4	address	IP address of the multicast group

Table 3.334. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x15	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_multicast_join(address)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_multicast_join(ipv4 address);

/* Response id */
wifi_rsp_tcpip_multicast_join_id

/* Response structure */
struct wifi_msg_tcpip_multicast_join_rsp_t
{
  uint16 result;
};
```

3.11.1.20 cmd_tcpip_multicast_leave

This command is used to leave a multicast group.

Table 3.335. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x16	method	Message ID
4-7	ipv4	address	IP address of the multicast group

Table 3.336. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x16	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_multicast_leave(address)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_multicast_leave(ipv4 address);

/* Response id */
wifi_rsp_tcpip_multicast_leave_id

/* Response structure */
struct wifi_msg_tcpip_multicast_leave_rsp_t
{
  uint16 result;
};
```

3.11.1.21 cmd_tcpip_set_tcp_client_port_range

This command is used to set range for tcp client random port generation

Table 3.337. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x1b	method	Message ID
4-5	uint16	tcp_local_port_range_start	Start of local port range (min. 0xC000). Default: 0xC000
6-7	uint16	tcp_local_port_range_end	End of local port range (max. 0xFFFF). Default: 0xFFFF

Table 3.338. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x1b	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_set_tcp_client_port_range(tcp_local_port_range_start, tcp_local_port_range_end)(result)
```

BGLIB C API

```
/* Function */
void       wifi_cmd_tcpip_set_tcp_client_port_range(uint16       tcp_local_port_range_start,       uint16
tcp_local_port_range_end);

/* Response id */
wifi_rsp_tcpip_set_tcp_client_port_range_id

/* Response structure */
struct wifi_msg_tcpip_set_tcp_client_port_range_rsp_t
{
  uint16 result;
};
```

3.11.1.22 cmd_tcpip_start_tcp_server

This command is used to start a TCP server.

Table 3.339. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x00	method	Message ID
4-5	uint16	port	Local TCP port that the server listens to
6	int8	default_destination	The endpoint where the received data will be routed to • **-1:** route to all BGAPI/BGScript endpoints • **0 - 30:** route to a specific endpoint • **31:** discard the received data

Table 3.340. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x00	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	endpoint	Index of the TCP server endpoint

BGScript command

```
call tcpip_start_tcp_server(port, default_destination)(result, endpoint)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_start_tcp_server(uint16 port, int8 default_destination);

/* Response id */
wifi_rsp_tcpip_start_tcp_server_id

/* Response structure */
struct wifi_msg_tcpip_start_tcp_server_rsp_t
{
  uint16 result;,
  uint8 endpoint;
};
```

Table 3.341. Events Generated

Event	Description
tcpip_endpoint_status	Sent when TCP/IP endpoint status changes
endpoint_status	Sent when the endpoint status changes

3.11.1.23 cmd_tcpip_start_udp_server

This command is used to start an UDP server.

Table 3.342. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x02	method	Message ID
4-5	uint16	port	Local UDP port that the server listens to
6	int8	default_destination	Index of the endpoint where the received data will be routed to • **-1:** route to all BGAPI/BGScript endpoints • **0 - 30:** route to a specific endpoint • **31:** discard the received data

Table 3.343. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x02	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	endpoint	Index of the UDP server endpoint

BGScript command

```
call tcpip_start_udp_server(port, default_destination)(result, endpoint)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_start_udp_server(uint16 port, int8 default_destination);

/* Response id */
wifi_rsp_tcpip_start_udp_server_id

/* Response structure */
struct wifi_msg_tcpip_start_udp_server_rsp_t
{
  uint16 result;,
  uint8 endpoint;
};
```

Table 3.344. Events Generated

Event	Description
tcpip_endpoint_status	Sent when TCP/IP endpoint status changes
endpoint_status	Sent when the endpoint status changes

3.11.1.24 cmd_tcpip_tcp_connect

This command is used to create a new TCP connection to a TCP server.

Table 3.345. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x07	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x01	method	Message ID
4-7	ipv4	address	IP address of the remote server
8-9	uint16	port	TCP port of the remote server
10	int8	routing	Index of the endpoint where the received data will be routed to • **-1:** route to all BGAPI/BGScript endpoints • **0 - 30:** route to a specific endpoint • **31:** discard the received data

Table 3.346. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x01	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	endpoint	Index of the TCP connection endpoint

BGScript command

```
call tcpip_tcp_connect(address, port, routing)(result, endpoint)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_tcp_connect(ipv4 address, uint16 port, int8 routing);

/* Response id */
wifi_rsp_tcpip_tcp_connect_id

/* Response structure */
struct wifi_msg_tcpip_tcp_connect_rsp_t
{
  uint16 result;,
  uint8 endpoint;
};
```

Table 3.347. Events Generated

Event	Description
tcpip_endpoint_status	Sent when TCP/IP endpoint status changes
endpoint_status	Sent when the endpoint status changes

3.11.1.25 cmd_tcpip_tls_connect

This command is used to create a new TLS connection to a TLS server.

Only one TLS connection at a time is feasible

Table 3.348. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x07	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x09	method	Message ID
4-7	ipv4	address	IP address of the remote server
8-9	uint16	port	TLS port of the remote server
10	int8	routing	Index of the endpoint where the received data will be routed to • **-1:** route to all BGAPI/BGScript endpoints • **0 - 30:** route to a specific endpoint • **31:** discard the received data

Table 3.349. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x09	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	endpoint	Index of the TLS connection endpoint

BGScript command

```
call tcpip_tls_connect(address, port, routing)(result, endpoint)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_tls_connect(ipv4 address, uint16 port, int8 routing);

/* Response id */
wifi_rsp_tcpip_tls_connect_id

/* Response structure */
struct wifi_msg_tcpip_tls_connect_rsp_t
{
  uint16 result;,
  uint8 endpoint;
};
```

Table 3.350. Events Generated

Event	Description
tcpip_endpoint_status	Sent when TCP/IP endpoint status changes
endpoint_status	Sent when the endpoint status changes
tcpip_tls_verify_result	Sent during TLS handshake if the verification mode enabled

3.11.1.26 cmd_tcpip_tls_set_authmode

This command is used to set the TLS certificate verification mode.

The mode must be set before calling the tcpip_tls_connect command. If the verification mode has been set to optional or mandatory, the server certificate is verified against a root certificate from the built-in certificate store during TLS connection setup. The verification result is indicated with the evt_tcpip_tls_verify_result event.

The verification mode is set to mandatory by default.

Table 3.351. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0a	method	Message ID
4	uint8	auth_mode	Verification mode • **0:** no verification • **1:** optional verification, verification failure is ignored • **2:** mandatory verification, verification failure causes connection to fail

Table 3.352. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x00	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0a	method	Message ID

BGScript command

```
call tcpip_tls_set_authmode(auth_mode)()
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_tls_set_authmode(uint8 auth_mode);

/* Response id */
wifi_rsp_tcpip_tls_set_authmode_id

/* Response structure */
struct wifi_msg_tcpip_tls_set_authmode_rsp_t
{
};
```

3.11.1.27 cmd_tcpip_tls_set_hostname

Sets hostname to check against common name in certificate.

Table 3.353. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x1d	method	Message ID
4	uint8array	hostname	Variable length byte array. The first byte defines the length of the data that follows, 0 - 255 bytes. • **Example:** • Human readable: "Hello" • Packet data (hex): 05 48 65 6C 6C 6F

Table 3.354. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x1d	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_tls_set_hostname(hostname_len, hostname_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_tls_set_hostname(uint8 hostname_len, const uint8 *hostname_data);

/* Response id */
wifi_rsp_tcpip_tls_set_hostname_id

/* Response structure */
struct wifi_msg_tcpip_tls_set_hostname_rsp_t
{
  uint16 result;
};
```

3.11.1.28 cmd_tcpip_tls_set_user_certificate

This command is used to set the client certificate for a TLS connection.

If set, the certificate will be sent to the TLS server if requested by the server during authentication.

Table 3.355. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x17	method	Message ID
4	uint8array	fingerprint	Fingerprint of the client certificate **Length:** 16 bytes

Table 3.356. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x17	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_tls_set_user_certificate(fingerprint_len, fingerprint_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_tls_set_user_certificate(uint8 fingerprint_len, const uint8 *fingerprint_data);

/* Response id */
wifi_rsp_tcpip_tls_set_user_certificate_id

/* Response structure */
struct wifi_msg_tcpip_tls_set_user_certificate_rsp_t
{
  uint16 result;
};
```

3.11.1.29 cmd_tcpip_udp_bind

This command is used to change the source port of an existing UDP endpoint. There is no following event after this command. Positive response for this command is confirmation of successful port change.

Table 3.357. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x07	method	Message ID
4	uint8	endpoint	Index of the UDP endpoint to change
5-6	uint16	port	New UDP source port

Table 3.358. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x07	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call tcpip_udp_bind(endpoint, port)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_udp_bind(uint8 endpoint, uint16 port);

/* Response id */
wifi_rsp_tcpip_udp_bind_id

/* Response structure */
struct wifi_msg_tcpip_udp_bind_rsp_t
{
  uint16 result;
};
```

3.11.1.30 cmd_tcpip_udp_connect

This command is used to create a new UDP connection.

Table 3.359. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x07	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x03	method	Message ID
4-7	ipv4	address	IP address of the remote server to which the packets will be sent
8-9	uint16	port	UDP port of the remote server
10	int8	routing	Index of the endpoint where the received data will be routed to. Due to the unidirectional nature of UDP endpoints, this parameter has no effect on this command. Set to zero.

Table 3.360. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x03	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8	endpoint	Index of the UDP connection endpoint

BGScript command

```
call tcpip_udp_connect(address, port, routing)(result, endpoint)
```

BGLIB C API

```
/* Function */
void wifi_cmd_tcpip_udp_connect(ipv4 address, uint16 port, int8 routing);

/* Response id */
wifi_rsp_tcpip_udp_connect_id

/* Response structure */
struct wifi_msg_tcpip_udp_connect_rsp_t
{
  uint16 result;,
  uint8 endpoint;
};
```

Table 3.361. Events Generated

Event	Description
tcpip_endpoint_status	Sent when TCP/IP endpoint status changes
endpoint_status	Sent when the endpoint status changes

3.11.2 tcpip events

3.11.2.1 evt_tcpip_configuration

This event indicates TCP/IP configuration status.

Table 3.362. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x0d	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x00	method	Message ID
4-7	ipv4	address	Local IP address of the device
8-11	ipv4	netmask	Netmask of the device
12-15	ipv4	gateway	Gateway of the device
16	uint8	use_dhcp	Defines whether DHCP is used to retrieve IP settings • **0:** DHCP disabled • **1:** DHCP enabled

BGScript event

```
event tcpip_configuration(address, netmask, gateway, use_dhcp)
```

C Functions

```
/* Event id */
wifi_evt_tcpip_configuration_id

/* Event structure */
struct wifi_msg_tcpip_configuration_evt_t
{
  ipv4 address;,
  ipv4 netmask;,
  ipv4 gateway;,
  uint8 use_dhcp;
};
```

3.11.2.2 evt_tcpip_dhcp_client

This event contains a client information. Send for each client

Table 3.363. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x0a	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0e	method	Message ID
4-7	ipv4	ip_address	IPv4 address of the client.Zero value means that client is connected, but IPv4 address is not offered by the module's DHCP server.
8-13	hw_addr	mac_address	MAC address of the client

BGScript event

```
event tcpip_dhcp_client(ip_address, mac_address)
```

C Functions

```
/* Event id */
wifi_evt_tcpip_dhcp_client_id

/* Event structure */
struct wifi_msg_tcpip_dhcp_client_evt_t
{
  ipv4 ip_address;,
  hw_addr mac_address;
};
```

3.11.2.3 evt_tcpip_dhcp_configuration

This event contains DHCP Server configuration

Table 3.364. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x0d	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0c	method	Message ID
4	uint8	routing_enabled	DHCP server routing enabled. • **0:** DHCP server responses **don't** include gateway and DNS server information • **1:** Gateway and DNS server information are included in responses
5-8	ipv4	address	First address of DHCP Server address pool
9-12	ipv4	subnet_mask	Subnetwork mask
13-16	uint32	lease_time	DHCP address lease timeout in seconds

BGScript event

```
event tcpip_dhcp_configuration(routing_enabled, address, subnet_mask, lease_time)
```

C Functions

```
/* Event id */
wifi_evt_tcpip_dhcp_configuration_id

/* Event structure */
struct wifi_msg_tcpip_dhcp_configuration_evt_t
{
  uint8 routing_enabled;,
  ipv4 address;,
  ipv4 subnet_mask;,
  uint32 lease_time;
};
```

3.11.2.4 evt_tcpip_dns_configuration

This event indicates DNS configuration status.

Table 3.365. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x05	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x01	method	Message ID
4	uint8	index	Index of the DNS server • **0:** primary DNS server • **1:** secondary DNS server
5-8	ipv4	address	IP address of the DNS server

BGScript event

```
event tcpip_dns_configuration(index, address)
```

C Functions

```
/* Event id */
wifi_evt_tcpip_dns_configuration_id

/* Event structure */
struct wifi_msg_tcpip_dns_configuration_evt_t
{
  uint8 index;,
  ipv4 address;
};
```

3.11.2.5 evt_tcpip_dns_gethostbyname_result

This event indicates that DNS resolver query has been completed.

If successful, the address field contains the corresponding IP address.

Table 3.366. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x07	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x03	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6-9	ipv4	address	The resolved IP address of the domain name
10	uint8array	name	The domain name that was resolved

BGScript event

```
event tcpip_dns_gethostbyname_result(result, address, name_len, name_data)
```

C Functions

```
/* Event id */
wifi_evt_tcpip_dns_gethostbyname_result_id

/* Event structure */
struct wifi_msg_tcpip_dns_gethostbyname_result_evt_t
{
  uint16 result;,
  ipv4 address;,
  uint8array name;
};
```

3.11.2.6 evt_tcpip_dnssd_service_failed

This event indicates that a DNS-SD service has failed.

Table 3.367. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0a	method	Message ID
4-5	uint16	reason	Reason of fail For values refer to the error codes.
6	uint8	index	Index of the DNS-SD service

BGScript event

```
event tcpip_dnssd_service_failed(reason, index)
```

C Functions

```
/* Event id */
wifi_evt_tcpip_dnssd_service_failed_id

/* Event structure */
struct wifi_msg_tcpip_dnssd_service_failed_evt_t
{
  uint16 reason;,
  uint8 index;
};
```

3.11.2.7 evt_tcpip_dnssd_service_started

This event indicates that a DNS-SD service has been successfully started.

Table 3.368. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x09	method	Message ID
4	uint8	index	Index of the DNS-SD service

BGScript event

```
event tcpip_dnssd_service_started(index)
```

C Functions

```
/* Event id */
wifi_evt_tcpip_dnssd_service_started_id

/* Event structure */
struct wifi_msg_tcpip_dnssd_service_started_evt_t
{
  uint8 index;
};
```

3.11.2.8 evt_tcpip_dnssd_service_stopped

This event indicates that a DNS-SD service has been stopped.

Table 3.369. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0b	method	Message ID
4-5	uint16	reason	Reason of service stop For values refer to the error codes.
6	uint8	index	Index of the DNS-SD service

BGScript event

```
event tcpip_dnssd_service_stopped(reason, index)
```

C Functions

```
/* Event id */
wifi_evt_tcpip_dnssd_service_stopped_id

/* Event structure */
struct wifi_msg_tcpip_dnssd_service_stopped_evt_t
{
  uint16 reason;,
  uint8 index;
};
```

3.11.2.9 evt_tcpip_endpoint_status

This event indicates the current status of a TCP/IP endpoint.

Table 3.370. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x0d	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x02	method	Message ID
4	uint8	endpoint	Index of the endpoint **Range:** 0 - 30
5-8	ipv4	local_ip	Local IP address of the endpoint
9-10	uint16	local_port	Local port of the endpoint
11-14	ipv4	remote_ip	Remote IP address of the endpoint
15-16	uint16	remote_port	Remote port of the endpoint

BGScript event

```
event tcpip_endpoint_status(endpoint, local_ip, local_port, remote_ip, remote_port)
```

C Functions

```
/* Event id */
wifi_evt_tcpip_endpoint_status_id

/* Event structure */
struct wifi_msg_tcpip_endpoint_status_evt_t
{
  uint8 endpoint;,
  ipv4 local_ip;,
  uint16 local_port;,
  ipv4 remote_ip;,
  uint16 remote_port;
};
```

3.11.2.10 evt_tcpip_mdns_failed

This event indicates that the mDNS service has failed.

Table 3.371. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x07	method	Message ID
4-5	uint16	reason	Reason of fail For values refer to the error codes.

BGScript event

```
event tcpip_mdns_failed(reason)
```

C Functions

```
/* Event id */
wifi_evt_tcpip_mdns_failed_id

/* Event structure */
struct wifi_msg_tcpip_mdns_failed_evt_t
{
  uint16 reason;
};
```

3.11.2.11 evt_tcpip_mdns_gethostbyname_result

This event indicates that MDNS resolver query has been completed.

If successful, the address field contains the corresponding IP address.

Table 3.372. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x07	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0d	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6-9	ipv4	address	The resolved IP address of the domain name
10	uint8array	name	The domain name that was resolved

BGScript event

```
event tcpip_mdns_gethostbyname_result(result, address, name_len, name_data)
```

C Functions

```
/* Event id */
wifi_evt_tcpip_mdns_gethostbyname_result_id

/* Event structure */
struct wifi_msg_tcpip_mdns_gethostbyname_result_evt_t
{
  uint16 result;,
  ipv4 address;,
  uint8array name;
};
```

3.11.2.12 evt_tcpip_mdns_started

This event indicates that the mDNS service has been successfully started.

Table 3.373. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x00	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x06	method	Message ID

BGScript event

```
event tcpip_mdns_started()
```

C Functions

```
/* Event id */
wifi_evt_tcpip_mdns_started_id

/* Event structure */
struct wifi_msg_tcpip_mdns_started_evt_t
{
};
```

3.11.2.13 evt_tcpip_mdns_stopped

This event indicates that the mDNS service has been stopped.

Table 3.374. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x08	method	Message ID
4-5	uint16	reason	Reason of service stop For values refer to the error codes.

BGScript event

```
event tcpip_mdns_stopped(reason)
```

C Functions

```
/* Event id */
wifi_evt_tcpip_mdns_stopped_id

/* Event structure */
struct wifi_msg_tcpip_mdns_stopped_evt_t
{
  uint16 reason;
};
```

3.11.2.14 evt_tcpip_tls_verify_result

This event indicates the result of a TLS certificate verification.

Table 3.375. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x05	method	Message ID
4	uint8	depth	Position of the certificate in the certificate chain. Server certificate is always at position 0. Certificate that signed the server certificate is at position 1. Certificate that signed the certificate at position 1 is at position 2 and so on.
5-6	uint16	flags	Verification result • **0:** verification success • **non-zero:** verification failure

BGScript event

```
event tcpip_tls_verify_result(depth, flags)
```

C Functions

```
/* Event id */
wifi_evt_tcpip_tls_verify_result_id

/* Event structure */
struct wifi_msg_tcpip_tls_verify_result_evt_t
{
  uint8 depth;,
  uint16 flags;
};
```

3.11.2.15 evt_tcpip_udp_data

This event contains received data from an UDP endpoint. **Please note that due to limitations in the BGscript engine, if length of the payload data exceeds 256 bytes, this event is dropped in BGscript.**

In order to receive this event, instead of the endpoint data event, use -1 as the default destination in the start udp server command.

Table 3.376. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x09	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x04	method	Message ID
4	uint8	endpoint	Index of the endpoint which received the data **Range:** 0 - 30
5-8	ipv4	source_address	IP address that sent the data
9-10	uint16	source_port	UDP port where the data was sent from
11-12	uint16array	data	The received data

BGScript event

```
event tcpip_udp_data(endpoint, source_address, source_port, data_len, data_data)
```

C Functions

```
/* Event id */
wifi_evt_tcpip_udp_data_id

/* Event structure */
struct wifi_msg_tcpip_udp_data_evt_t
{
  uint8 endpoint;,
  ipv4 source_address;,
  uint16 source_port;,
  uint16array data;
};
```

3.12 Utilities for BGScript Command Class

The commands in this class are helper functions called from BGScript.

This class consists of the following items:

- *Commands*
  - `cmd_util_atoi` converts an ASCII string into an integer value.
  - `cmd_util itoa` converts an integer value into an ASCII string value.

3.12.1 util commands

3.12.1.1 cmd_util_atoi

This command is used to convert a decimal value from an ASCII string format into a signed 32-bit integer format.

Table 3.377. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0d	class	Message class: Utilities for BGScript
3	0x00	method	Message ID
4	uint8array	string	String to convert

Table 3.378. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x04	lolen	Minimum payload length
2	0x0d	class	Message class: Utilities for BGScript
3	0x00	method	Message ID
4-7	int32	value	32-bit integer value

BGScript command

```
call util_atoi(string_len, string_data)(value)
```

BGLIB C API

```
/* Function */
void wifi_cmd_util_atoi(uint8 string_len, const uint8 *string_data);

/* Response id */
wifi_rsp_util_atoi_id

/* Response structure */
struct wifi_msg_util_atoi_rsp_t
{
  int32 value;
};
```

3.12.1.2 cmd_util_atou

This command is used to convert a decimal value from an ASCII string format into an unsigned 32-bit integer format.

Table 3.379. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0d	class	Message class: Utilities for BGScript
3	0x02	method	Message ID
4	uint8array	string	String to convert

Table 3.380. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x04	lolen	Minimum payload length
2	0x0d	class	Message class: Utilities for BGScript
3	0x02	method	Message ID
4-7	uint32	value	32-bit unsigned integer value

BGScript command

```
call util_atou(string_len, string_data)(value)
```

BGLIB C API

```
/* Function */
void wifi_cmd_util_atou(uint8 string_len, const uint8 *string_data);

/* Response id */
wifi_rsp_util_atou_id

/* Response structure */
struct wifi_msg_util_atou_rsp_t
{
  uint32 value;
};
```

3.12.1.3 cmd_util_base64decode

This command is used to convert data in base64 format to binary format.

Table 3.381. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0d	class	Message class: Utilities for BGScript
3	0x05	method	Message ID
4	uint8array	input_data	Data to decode.

Table 3.382. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x0d	class	Message class: Utilities for BGScript
3	0x05	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8array	decoded_data	Decoded data to binary representation.

BGScript command

```
call util_base64decode(input_data_len, input_data_data)(result, decoded_data_len, decoded_data_data)
```

BGLIB C API

```
/* Function */
void wifi_cmd_util_base64decode(uint8 input_data_len, const uint8 *input_data_data);

/* Response id */
wifi_rsp_util_base64decode_id

/* Response structure */
struct wifi_msg_util_base64decode_rsp_t
{
  uint16 result;,
  uint8array decoded_data;
};
```

3.12.1.4 cmd_util_base64encode

This command is used to convert binary data to a base64 format.

Table 3.383. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0d	class	Message class: Utilities for BGScript
3	0x04	method	Message ID
4	uint8array	input_data	Data to encode. The maximum length is 189 due to maximum uint8array length equal 255 bytes.

Table 3.384. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x0d	class	Message class: Utilities for BGScript
3	0x04	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8array	encoded_data	Encoded data to Base64 representation.

BGScript command

```
call util_base64encode(input_data_len, input_data_data)(result, encoded_data_len, encoded_data_data)
```

BGLIB C API

```
/* Function */
void wifi_cmd_util_base64encode(uint8 input_data_len, const uint8 *input_data_data);

/* Response id */
wifi_rsp_util_base64encode_id

/* Response structure */
struct wifi_msg_util_base64encode_rsp_t
{
  uint16 result;,
  uint8array encoded_data;
};
```

3.12.1.5 cmd_util_itoa

This command is used to convert an integer from a signed 32-bit integer format into decimal ASCII value format.

Table 3.385. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x0d	class	Message class: Utilities for BGScript
3	0x01	method	Message ID
4-7	int32	value	32-bit integer value to convert

Table 3.386. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x01	lolen	Minimum payload length
2	0x0d	class	Message class: Utilities for BGScript
3	0x01	method	Message ID
4	uint8array	string	Converted ASCII string

BGScript command

```
call util_itoa(value)(string_len, string_data)
```

BGLIB C API

```
/* Function */
void wifi_cmd_util_itoa(int32 value);

/* Response id */
wifi_rsp_util_itoa_id

/* Response structure */
struct wifi_msg_util_itoa_rsp_t
{
  uint8array string;
};
```

3.12.1.6 cmd_util_utoa

This command is used to convert an integer from an unsigned 32-bit integer format into decimal ASCII value format.

Table 3.387. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x0d	class	Message class: Utilities for BGScript
3	0x03	method	Message ID
4-7	uint32	value	32-bit unsigned integer value to convert

Table 3.388. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x01	lolen	Minimum payload length
2	0x0d	class	Message class: Utilities for BGScript
3	0x03	method	Message ID
4	uint8array	string	Converted ASCII string

BGScript command

```
call util_utoa(value)(string_len, string_data)
```

BGLIB C API

```
/* Function */
void wifi_cmd_util_utoa(uint32 value);

/* Response id */
wifi_rsp_util_utoa_id

/* Response structure */
struct wifi_msg_util_utoa_rsp_t
{
  uint8array string;
};
```

3.13 X.509 Command Class

The commands in this class are used to manage X.509 (ITU-T Standard for a Public Key Infrastucture and Privilege Management Infrastucture) cryptography certificates and to the related keys.

This class consists of the following items:

Certificate store related

- *Commands*
  - `cmd_x509_reset_store`
  - `cmd_x509_list_certificates`
- *Events*
  - `evt_x509_certificates_listed`
  - `evt_x509_certificate`
  - `evt_x509_certificate_subject`

Certificate processing related

- *Commands*
  - `cmd_x509_add_certificate`
  - `cmd_x509_add_certificate_data`
  - `cmd_x509_certificate_finish`
  - `cmd_x509_delete_certificate`
- *Events*
  - `evt_x509_certificates_listed`

Private key operation related

- *Commands*
  - `cmd_x509_add_private_key`
  - `cmd_x509_add_private_key_data`
  - `cmd_x509_add_private_key_finish`
- *Events*

Examples

1. **Adding a certificate into certificate store**
   a. Reserve space for the certificate in the certificate store using the `cmd_x509_add_certificate` command.
   b. Load the certificate by repeating the `cmd_x509_add_certificate_data` command until the whole certificate is loaded.

      Note: The size of the loaded data must not exceed the size of the data space reserved for the certificate with *cmd_x509_add_certificate* command

   c. Finish the loading process by using the `cmd_x509_add_certificate_finish` command.

3.13.1 x509 commands

3.13.1.1 cmd_x509_add_certificate

This command is used to add a certificate to a certificate store.

Table 3.389. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x01	method	Message ID
4	uint8	store	Store used
5-6	uint16	size	Size of certificate data

Table 3.390. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x01	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call x509_add_certificate(store, size)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_x509_add_certificate(uint8 store, uint16 size);

/* Response id */
wifi_rsp_x509_add_certificate_id

/* Response structure */
struct wifi_msg_x509_add_certificate_rsp_t
{
  uint16 result;
};
```

3.13.1.2 cmd_x509_add_certificate_data

This command is used to upload a block of certificate data.

Table 3.391. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x02	method	Message ID
4	uint8array	data	Certificate data to store

Table 3.392. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x02	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call x509_add_certificate_data(data_len, data_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_x509_add_certificate_data(uint8 data_len, const uint8 *data_data);

/* Response id */
wifi_rsp_x509_add_certificate_data_id

/* Response structure */
struct wifi_msg_x509_add_certificate_data_rsp_t
{
  uint16 result;
};
```

3.13.1.3 cmd_x509_add_certificate_finish

This command is used to finish adding a certificate.

The command must be called once all the certificate data has been uploaded.

Table 3.393. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x03	method	Message ID

Table 3.394. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x03	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8array	fingerprint	Certificate fingerprint **Length:** 16 bytes

BGScript command

```
call x509_add_certificate_finish()(result, fingerprint_len, fingerprint_data)
```

BGLIB C API

```
/* Function */
void wifi_cmd_x509_add_certificate_finish();

/* Response id */
wifi_rsp_x509_add_certificate_finish_id

/* Response structure */
struct wifi_msg_x509_add_certificate_finish_rsp_t
{
  uint16 result;,
  uint8array fingerprint;
};
```

3.13.1.4 cmd_x509_add_private_key

This command is used to add a private key to the RAM certificate store.

Table 3.395. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x04	method	Message ID
4-5	uint16	size	Size of private key data
6	uint8array	fingerprint	Fingerprint of the corresponding user certificate **Length:** 16 bytes

Table 3.396. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x04	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call x509_add_private_key(size, fingerprint_len, fingerprint_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_x509_add_private_key(uint16 size, uint8 fingerprint_len, const uint8 *fingerprint_data);

/* Response id */
wifi_rsp_x509_add_private_key_id

/* Response structure */
struct wifi_msg_x509_add_private_key_rsp_t
{
  uint16 result;
};
```

3.13.1.5 cmd_x509_add_private_key_data

This command is used to upload a block of private key data.

Table 3.397. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x05	method	Message ID
4	uint8array	data	Private key data to store

Table 3.398. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x05	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call x509_add_private_key_data(data_len, data_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_x509_add_private_key_data(uint8 data_len, const uint8 *data_data);

/* Response id */
wifi_rsp_x509_add_private_key_data_id

/* Response structure */
struct wifi_msg_x509_add_private_key_data_rsp_t
{
  uint16 result;
};
```

3.13.1.6 cmd_x509_add_private_key_finish

This command is used to finish adding a private key.

The command must be called once all the private key data has been uploaded.

Table 3.399. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x06	method	Message ID
4	uint8array	password	Password used to encrypt the private key data **Length:** 0 - 64 bytes

Table 3.400. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x06	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.
6	uint8array	fingerprint	Private key fingerprint **Length:** 16 bytes

BGScript command

```
call x509_add_private_key_finish(password_len, password_data)(result, fingerprint_len, fingerprint_data)
```

BGLIB C API

```
/* Function */
void wifi_cmd_x509_add_private_key_finish(uint8 password_len, const uint8 *password_data);

/* Response id */
wifi_rsp_x509_add_private_key_finish_id

/* Response structure */
struct wifi_msg_x509_add_private_key_finish_rsp_t
{
  uint16 result;,
  uint8array fingerprint;
};
```

3.13.1.7 cmd_x509_delete_certificate

This command is used to delete a certificate from the certificate store.

Table 3.401. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x07	method	Message ID
4	uint8array	fingerprint	Certificate fingerprint **Length:** 16 bytes

Table 3.402. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x07	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call x509_delete_certificate(fingerprint_len, fingerprint_data)(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_x509_delete_certificate(uint8 fingerprint_len, const uint8 *fingerprint_data);

/* Response id */
wifi_rsp_x509_delete_certificate_id

/* Response structure */
struct wifi_msg_x509_delete_certificate_rsp_t
{
  uint16 result;
};
```

3.13.1.8 cmd_x509_list_certificates

This command is used to retrieve information about added certificates.

Table 3.403. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x08	method	Message ID

Table 3.404. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x08	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call x509_list_certificates()(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_x509_list_certificates();

/* Response id */
wifi_rsp_x509_list_certificates_id

/* Response structure */
struct wifi_msg_x509_list_certificates_rsp_t
{
  uint16 result;
};
```

Table 3.405. Events Generated

Event	Description
x509_certificate	Sent for each certificate
x509_certificate_subject	Sent for each certificate subject
x509_certificates_listed	Sent after all the certificates have been listed

3.13.1.9 cmd_x509_reset_store

This command is used to reset the certificate stores.

Flash and RAM stores are erased and the modifications to the compile-time store are reset.

Table 3.406. Command

Byte	Type	Name	Description
0	0x08	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x00	method	Message ID

Table 3.407. Response

Byte	Type	Name	Description
0	0x08	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x00	method	Message ID
4-5	uint16	result	Result code, unsigned 16-bit integer in **little endian** format • **0:** success • **non-zero:** an error occurred For other values refer to the Error codes.

BGScript command

```
call x509_reset_store()(result)
```

BGLIB C API

```
/* Function */
void wifi_cmd_x509_reset_store();

/* Response id */
wifi_rsp_x509_reset_store_id

/* Response structure */
struct wifi_msg_x509_reset_store_rsp_t
{
  uint16 result;
};
```

3.13.2 x509 events

3.13.2.1 evt_x509_certificate

This event contains information about a certificate.

Table 3.408. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x04	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x00	method	Message ID
4	uint8	index	Unique certificate listing index
5	uint8	type	Certificate type
6	uint8	store	Store certificate is in
7	uint8array	fingerprint	Certificate fingerprint **Length:** 16 bytes

BGScript event

```
event x509_certificate(index, type, store, fingerprint_len, fingerprint_data)
```

C Functions

```
/* Event id */
wifi_evt_x509_certificate_id

/* Event structure */
struct wifi_msg_x509_certificate_evt_t
{
  uint8 index;,
  uint8 type;,
  uint8 store;,
  uint8array fingerprint;
};
```

3.13.2.2 evt_x509_certificate_subject

This event contains the subject field of a certificate.

Table 3.409. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x02	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x01	method	Message ID
4	uint8	index	Unique certificate listing index
5	uint8array	subject	Certificate subject

BGScript event

```
event x509_certificate_subject(index, subject_len, subject_data)
```

C Functions

```
/* Event id */
wifi_evt_x509_certificate_subject_id

/* Event structure */
struct wifi_msg_x509_certificate_subject_evt_t
{
  uint8 index;,
  uint8array subject;
};
```

3.13.2.3 evt_x509_certificates_listed

This event indicates all the certificates have been listed.

Table 3.410. Event

Byte	Type	Name	Description
0	0x88	hilen	Message type: Event
1	0x00	lolen	Minimum payload length
2	0x0b	class	Message class: X.509
3	0x02	method	Message ID

BGScript event

```
event x509_certificates_listed()
```

C Functions

```
/* Event id */
wifi_evt_x509_certificates_listed_id

/* Event structure */
struct wifi_msg_x509_certificates_listed_evt_t
{
};
```

3.13.3 x509 enumerations

3.13.3.1 enum_x509_store

This enumeration defines certificate stores.

Table 3.411. Enumerations

Value	Name	Description
0	x509_store_flash	Flash
1	x509_store_ram	RAM

3.13.3.2 enum_x509_type

This enumeration defines certificate types.

Table 3.412. Enumerations

Value	Name	Description
0	x509_type_ca	CA certificate
1	x509_type_user	User/client certificate

3.14 Error codes

This chapter describes all BGAPI error codes.

- **Hardware Error Codes indicate errors related to the hardware, such as failure of the microSD memory card, I2C interface, etc.**

Code	Name	Description
0x0301	ps_store_full	Flash reserved for Persistent Store is full
0x0302	ps_key_not_found	Persistent Store key not found
0x0303	i2c_write_already_started	Tried to start an I2C write transaction, but it is already in progress
0x0304	i2c_ack_missing	Acknowledge for an I2C operation was not received due to bus fail or operation timeout
0x0308	flash_failed	Writing to flash failed
0x0305	sdhc_not_opened	Tried to access an unopened file
0x0306	sdhc_not_found	File not in SD card
0x0307	sdhc_disk_error	Disk error or disk full

- **BGAPI Error Codes are common error codes. Typical examples are an error code indicating an invalid command or parameter, or that the module is in a wrong state for the sent command.**

Code	Name	Description
0x0100	success	No error
0x0180	invalid_param	Command contained an invalid parameter
0x0181	wrong_state	Device is in wrong state to accept command
0x0182	out_of_memory	Device has run out of memory
0x0183	not_implemented	Feature is not implemented
0x0184	invalid_command	Command was not recognized
0x0185	timeout	Command or procedure failed due to timeout
0x0186	unspecified	Unspecified error
0x0187	hardware	Hardware failure. Possible causes of this error: Radio chip indicates unspecified error, is unresponsive or last request to radio chip has failed. To recover from this error, you can try to turn chip on or reset the module.
0x0188	buffers_full	Command not accepted, because internal buffers are full. Request won't be processed until sent again after some time. This error might also occur when buffers are deallocated for example when TCP connection is in closing state.
0x0189	disconnected	Command or procedure failed due to disconnection
0x018a	too_many_requests	Too many simultaneous requests
0x018b	ap_not_in_scanlist	Access Point not found from scanlist
0x018c	invalid_password	Password is invalid or missing
0x018d	authentication_failure	WPA/WPA2 authentication has failed
0x018e	overflow	Overflow detected
0x018f	multiple_pbc_sessions	Multiple PBC sessions detected
0x0190	eth_not_connected	Ethernet cable not connected

Code	Name	Description
0x0191	eth_route_not_set	Ethernet route not set
0x0192	wrong_operating_mode	Wrong operating mode for this command
0x0193	not_found	Requested resource not found
0x0194	already_exists	Requested resource already exists
0x0195	invalid_configuration	Current configuration is invalid
0x0196	ap_lost	Connection to Access Point lost

- **TCP/IP Error Codes indicate errors related to the internal TCP/IP stack, such as a closed connection, host name not set, etc.**

Code	Name	Description
0x0200	success	No error
0x0201	out_of_memory	Out of memory
0x0202	buffer	Buffer handling failed
0x0203	timeout	Timeout
0x0204	routing	Could not find route
0x0205	in_progress	Operation in progress
0x0206	illegal_value	Illegal value
0x0207	would_block	Operation would block
0x0208	in_use	Address in use
0x0209	already_connected	Already connected
0x020a	abort	Connection aborted
0x020b	reset	Connection reset
0x020c	closed	Connection closed
0x020d	not_connected	Not connected
0x020e	illegal_argument	Illegal argument
0x020f	interface	Interface error
0x0210	service_not_running	Service not running
0x0211	service_running	Service already running
0x0212	hostname_not_set	Hostname not set
0x0213	hostname_conflict	Hostname conflict detected
0x0280	unknown_host	Unknown host

4. Document Revision History

Table 4.1. Document Revision History

Revision Number	Effective Date	Change Description
1.4	September 15th, 2021	**Updated chapter *Introduction to Wizard Gecko Software Architecture* with more details on mixed mode usage** **Updated chapter *Error codes*** • Updated buffers_full error message description **Updated chapter *Using Endpoints*** • Added endpoint streaming mode more detailed explanation • Updated *Figure 1.5* and *Figure 1.6* with more details **Added commands** • *cmd_hardware_dbg_interface_set_active* • *cmd_https_enable_detailed_error* • *cmd_sme_add_scan_result_filter* • *cmd_sme_remove_scan_result_filter* • *cmd_sme_set_ap_custom_ie* • *cmd_tcpip_force_tcp_server_endpoint_close* • *cmd_tcpip_set_tcp_client_port_range* • *cmd_tcpip_tls_set_hostname* • *cmd_util_atou* • *cmd_util_base64encode* • *cmd_util_base64decode* • *cmd_util_utoa* **Added events** • *evt_https_error_detailed* • *evt_sme_scan_result_filter* **Changed commands** • *cmd_https_enable* **Changed events** • *evt_endpoint_status*

Revision Number	Effective Date	Change Description
1.3	March 9th, 2017	**Added commands** • *cmd_hardware_uart_conf_get* • *cmd_hardware_uart_conf_set* • *cmd_sme_p2p_accept_client* • *cmd_sme_start_p2p_group* • *cmd_sme_stop_p2p_group* • *cmd_sme_ap_client_config* • *cmd_sme_set_ap_client_isolation* • *cmd_tcpip_dhcp_configure* • *cmd_tcpip_mdns_gethostbyname* • *cmd_tcpip_dhcp_clients* **Added events** • *evt_hardware_uart_conf* • *evt_sme_p2p_group_started* • *evt_sme_p2p_group_stopped* • *evt_sme_p2p_group_failed* • *evt_sme_p2p_client_wants_to_join* • *evt_tcpip_dhcp_configuration* • *evt_tcpip_mdns_gethostbyname_result* • *evt_tcpip_dhcp_client* • *evt_https_error* **Updated command and event descriptions** • *cmd_sme_start_ap_mode* • *cmd_sme_set_ap_max_cli*ents • *cmd_tcpip_tls_connect* • *evt_tcpip_udp_data* • *cmd_endpoint_close* • *evt_flash_ps_key_changed* **Updated chapter *BGLib Functions Definition* to BGLib v2**. **Text changes in command class descriptions and enumerations.**
1.2	September 16th, 2016	**Updated command descriptions** • *evt_sme_scan_result*

Revision Number	Effective Date	Change Description
1.1	May 18th, 2016	**Added commands** • *cmd_tcpip_multicast_join* • *cmd_tcpip_multicast_leave* • *cmd_hardware_spi_transfer* • *cmd_tcpip_tls_set_user_certificate* **Renamed commands** • *cmd_tcpip_ssl_connect* has been renamed to *cmd_tcpip_tls_connect* • *cmd_tcpip_ssl_set_authmode* has been renamed to *cmd_tcpip_tls_set_authmode* **Renamed events** • *evt_tcpip_ssl_verify_result* has been renamed to *evt_tcpip_tls_verify_result* **Updated command descriptions** • *cmd_endpoint_set_streaming* • *cmd_sme_get_signal_quality* **Updated event descriptions** • *evt_endpoint_closing* • *evt_endpoint_error* **Updated command class descriptions** • *sme class* • *tcpip class* • *hardware*
1.0	February 22nd, 2016	Initial release.

Smart. Connected. Energy-Friendly.

IoT Portfolio
www.silabs.com/products

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com