BLUEGIGA WI-FI SOFTWARE

API DOCUMENTATION

Tuesday, 28 January 2014

Version 1.7

Table of Contents

1 Version History - WF121 SW API

Version	
0.9	API documentation for SW version v.0.3.0 (Build 25).
0.95	I2C API descriptions added, I2C and SPI end-points added, the power state management API added
1.0	Updated to be compliant with SW version 1.0; ADC read command added, Output Compare command added, e.g., PWM purposes
1.2	Wi-Fi Access point and HTTP server commands and event added and updated (See Wi-Fi and HTTP Server chapters), PS Key Change event added for monitoring PS key changes, UDP Data event added for seeing the source of UDP data, Event for handling the invalid commands added
1.3	Improved the documentation how to use BGAPI over SPI
1.4	Documentation updates for SW v1.2.1 compatibility Added/Changed APIs: • WPS commands and events added under Wi-Fi commands • Ethernet commands and event added • Get Signal Quality command and respective event added under Wi-Fi section • UDP Bind command for defining the source port added under TCP commands • Command setting maximum number of clients for AP mode added • For endpoints, the Error and SyntaxError events added • RTC commands and event added under Hardware section • State event (internal state for Wi-Fi SW) under System removed as not usable by 3rd party SW Editorial modifications: • Possible events for commands are added. • Error and return codes are updated
1.5	Documentation updates for SW v1.2.2 compatibility Added/Changed APIs: • Set DHCP Host Name API added for including host name parameter • Set transmit packet size API added for a TCP/UDP endpoint. • WPS support information added into the scan results event • Connecting to Hidden SSID support added for Connect to SSID command (no API change)
1.6	Improved API documentation
1.7	Improved API documentation

2 Introduction to Bluegiga Wi-Fi software

The Bluegiga Wi-Fi Software contains complete 802.11 MAC and IP networking stacks, providing everything required for creating wireless devices to integrate with existing Wi-Fi infrastructure.

The Wi-Fi Software supports three different modes of use:

- **Standalone architecture**: all software including the Wi-Fi stack and the application software run on the MCU of WF121 module
- **Hosted architecture**: an external MCU runs the application software which controls the WF121 module using BGAPI protocol
- **Mixed architecture**: part of the application run on the MCU of WF121 module and rest on an external MCU

In all above cases, the Bluegiga Wi-Fi Software provides a complete 802.11 MAC and IP networking stacks, so no additional 802.11 or IP stack software is required allowing simple and fast application development

Also a well-defined binary based transport protocol called BGAPI exists between the external host and the WF121 module and also simple and free software development kit is available to aid with development.

Several components make up the Wi-Fi Software Development Kit:

- A 802.11 MAC stack for controlling Wi-Fi functionality
- An IP networking stack for using various networking protocols such as TCP, UDP, DHCP and DNS
- Binary based communication protocol (**BGAPI**) between the host and the module
- A C library (**BGLib**) for the host that implements the BGAPI protocol
- **BGScript** scripting language and interpreter for implementing applications on the WF121 module's internal MCU
- A **WIFIGUI** application to quickly test, prototype and explore the functionality of the module

2.1 Bluegiga Wi-Fi Stack

The integrated Wi-Fi stack provides the necessary functions to scan for access points, configure the encryption and connect to access points. The protocol stack is illustrated below.

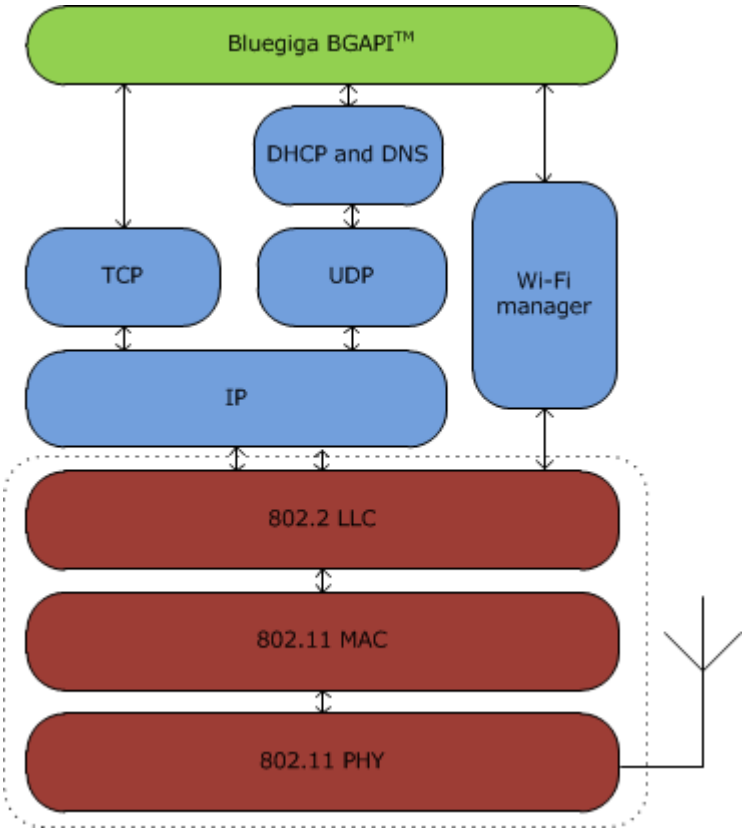


Figure: Bluegiga Wi-Fi Software

2.2 Bluegiga 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 Wi-Fi stack. The 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's a binary based communication protocol designed specifically for ease of implementation within host devices with limited resources.

The BGAPI provides access to the following layers:

- **System**- Various system functions, such as querying the hardware status or reset it
- **Configuration** - Provides access to the devices parameters such as the MAC address
- **TCP stack** - 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 to control the data endpoints
- **Hardware** - An interface to access the various hardware layers such as timers, ADC and other hardware interfaces
- **Persistent Store** - Allows user to read/write data to non-volatile memory
- **Device Firmware Upgrade** - Provides access to firmware update functions

The BGAPI protocol is intended to be used with:

- a serial UART link

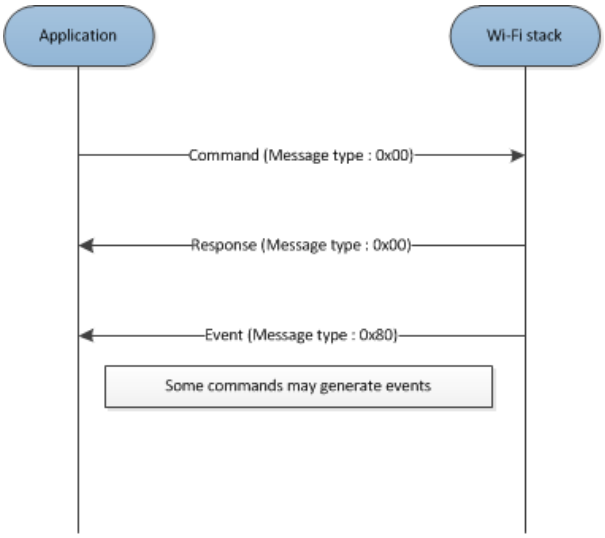


Figure: BGAPI messaging

2.3 Bluegiga BGLib library

For easy implementation of BGAPI protocol, an ANSI C host library is available. The library is easily portable ANSI C code delivered within the Bluegiga Wi-Fi Software Development Kit. The purpose is to simplify the application development to various host environments.

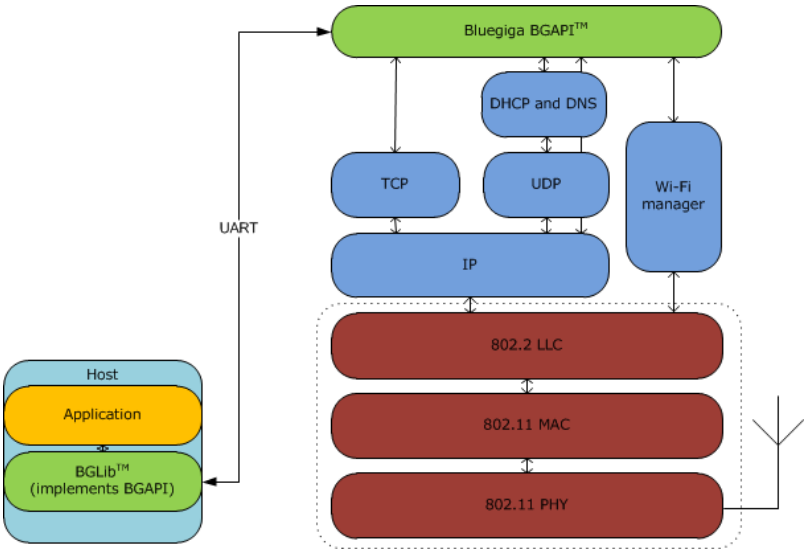


Figure: BGLib library

2.4 Bluegiga BGScript scripting language

Bluegiga's Wi-Fi software allows application developers to create standalone devices without the need of a separate host. The WF121 Wi-Fi module can run simple applications along the Wi-Fi stack and this provides a benefit when one needs to minimize the end product size, cost and current consumption. For developing standalone Wi-Fi applications the development kit provides a simple BGScript scripting language. With BGScript provides access to the same software and hardware interfaces as the BGAPI protocol. The BGScript code can be developed and compiled with free tools provided by Bluegiga.

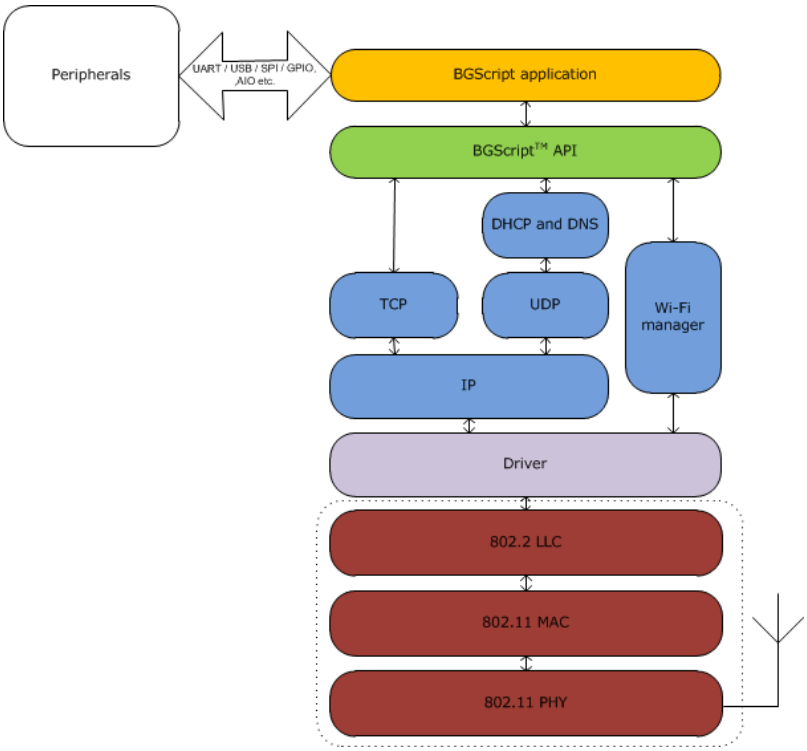


Figure: BGScript architecture

3 Understanding Endpoints

Endpoints play a crucial role in how data is handled and routed inside the Bluegiga Wi-Fi stack. The concept is used to unify the handling of data between different interfaces and peripherals, both externally and internally.

As the name suggests, an endpoint describes where the data ends up going, i.e. the sink for the data. Each data sink has an identifying endpoint number (ID) and the Wi-Fi stack will give each new endpoint the first available number. For example if the module is configured to have a TCP server, upon an incoming TCP connection the Wi-Fi stack will look at what the next available endpoint number is, and give the new connection that number. In the future if any data needs to be written to that socket, it is done by writing to that endpoint number. Since the socket also receives incoming data, the endpoint for where that data is routed, can be configured.

It is important to understand that with bi-directional peripherals, such as a serial port or TCP socket where data can move both in to and out of the Wi-Fi module, the system is configured through configuring the *end*points for each of the peripherals. For example to route the data in both direction 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.

An endpoint can either be configured to be active or inactive. By default endpoints which can receive or send data are active, but in some cases it may be useful to temporarily inhibit the flow of data. This can be done by setting the acitve flag on the endpoint to be false. Server endpoints, such as a TCP server endpoint waiting for a connection, are always inactive, as they will not send nor can they receive data.

A typical active endpoint will automatically stream its data to its configured endpoint. However for UART endpoints it is possible to make the endpoint an API endpoint. This is done by setting streaming to false.

Configuring endpoints individually, instead of simply tying together a TCP socket to a serial port allows for much more flexibility. In a sensor application for example, it might be interesting to have the sensor stream data directly to a server, while anything written by the server to the module can be routed to the BGScript instead of going to the sensor.

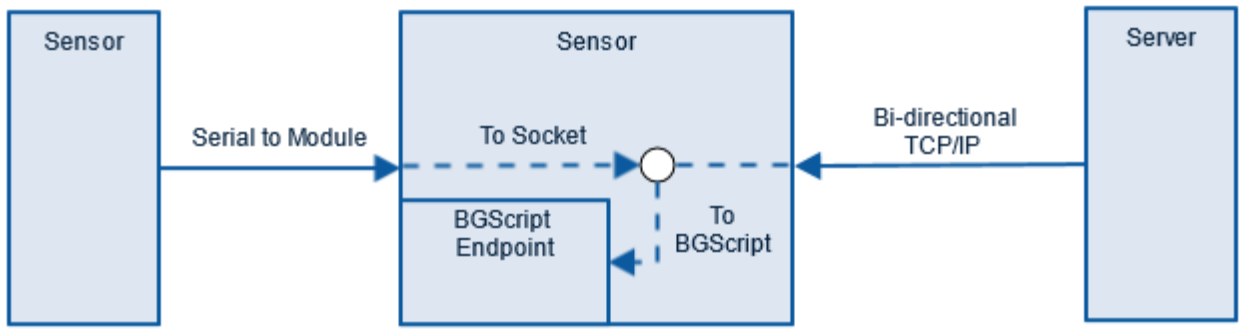


FIGURE: Example of routing a bi-directional stream between different endpoints. The serial interface has the socket configured as its endpoint, while the socket has the BGScript as its endpoint.

If an external microcontroller was used, the setup could look quite similar. In the figure below, the data from the server would get sent to the Microcontroller, and the Sensor would send its data straight to the Server.

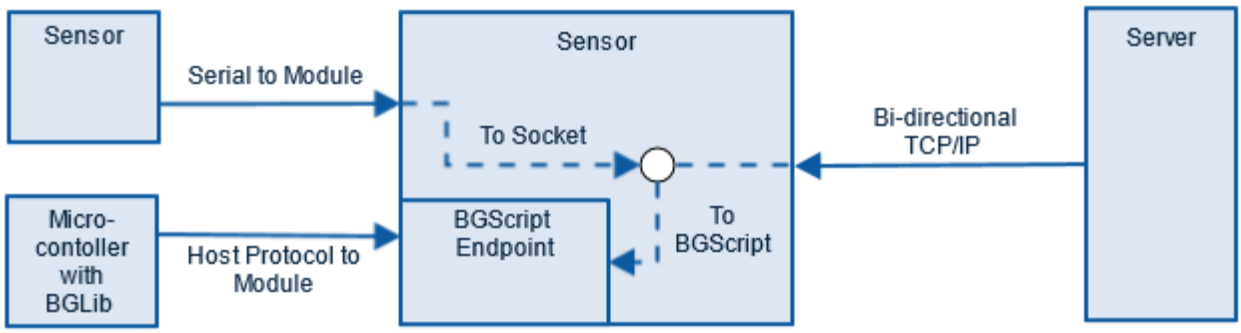


FIGURE: Example of routing a bi-directional stream between different endpoints

3.1 Predefined Endpoints

There are a few endpoints that are predefined upon boot, to provide access to some of the hardware interfaces:

Endpoint	UART	SPI	I2C	other	Note
0	UART1	SPI3	I2C3		
1	UART2	SPI4	I2C5		
2				BGScript	Application code processing. Can be disabled via an API command
3				USB	
4			I2C1		
5-30					Created at runtime by the BGScript code or BGAPI commands.
31				Drop	Anything sent to this endpoint will be dropped. Can be used as "/dev/null" for streaming data.

4 API Definition -- WIFI

This section contains the generic Bluegiga Wi-Fi software API definition. The definition consist of three parts:

- The BGAPI protocol definition
- The BGLib C library description
- The BGScript scriting API description

This section of the document only provides the generic definition and description of the API and the actual commands, responses and event are described in the API reference section.

4.1 BGAPI protocol definition -- WIFI

4.1.1 Packet format

Packets in either direction use the following format.

Table: BGAPI packet format

Octet	Octet bits	Length	Description	Notes
Octet 0	7	1 bit	**Message Type (MT)**	0: Command/Response 1: Event
...	6:3	4 bits	**Technology Type (TT)**	0000: Bluetooth 4.0 single mode 0001: Wi-Fi
...	2:0	3 bits	**Length High (LH)**	Payload length (high bits)
Octet 1	7:0	8 bits	**Length Low (LL)**	Payload length (low bits)
Octet 2	7:0	8 bits	**Class ID (CID)**	Command class ID
Octet 3	7:0	8 bits	**Command ID (CMD)**	Command ID
Octet 4-n	-	0 - 2048 Bytes	**Payload (PL)**	Up to 2048 bytes of payload

4.1.2 Message types

The following message types exist in the BGAPI protocol.

Table: BGAPI message types

Message type	Value	Description
Command	0x00	Command from host to the stack
Response	0x00	Response from stack to the host
Event	0x80	Event from stack to the host

4.1.3 Command Class IDs

The command classes are defined in the API Reference chapter.

4.1.4 Packet Exchange

The BGAPI protocol is a simple command / response protocol similar to AT commands, but instead of ASCII the BGAPI protocol uses binary format.

- The host should wait for the response to a command before issuing another command.

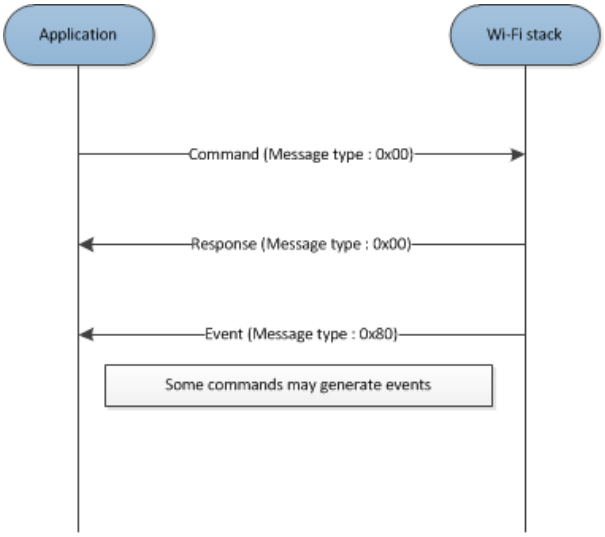


Figure: BGAPI messaging

4.1.5 Introduction to BGAPI over SPI

When using SPI as host interface to Wi-Fi module, host controller is the SPI master and Wi-Fi module is the SPI slave. SPI is synchronous interface so same clock drives both input and output. SPI master sends BGAPI commands and at the same time reads possible events or responses from SPI slave.

Using BGAPI over SPI

The SPI Slave informs master by notify pin (IO port) when there is data to be read. If the SPI slave has no data to send. it sends byte value 0. SPI master must synchronize incoming packets to first nonzero byte. If SPI slave is sending packet, SPI master must clock enough zeros for SPI slave to send a full packet. SPI master must wait response packet for a command before sending new command packet.

Examples

Command & Response

1. SPI master sends IO port read command to the SPI slave. SPI slave will response by sending back 0's.

Master	08	03	06	07	01	FF
Slave	00	00	00	00	00	00

2. SPI slave notifies master with the notify pin (assuming it's configured) that it has data to send.

3. SPI master reads response from the slave.

Master	00	00	00	00	00	00	00	00	00
Slave	08	05	06	07	00	00	01	CD	AB

Event

1. SPI slave notifies SPI master with the notify pin (assuming it's configured) that it has data to send.

2. Master reads event by sending 0's to the SPI slave, until all the data has been received from the slave.

Master	00	00	00	00	00	00	00	00	00
Slave	88	05	06	02	04	78	56	34	12

4.2 BGLIB functions definition -- WIFI

All the BGAPI commands are also available as ANSI C functions as a separate host library called BGLib. The responses and event on the other hand are handled as function call backs. The ANSI C functions are also documented in the API reference section.

The functions and callbacks are documented as follows:

C Functions

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

/* Callback *
void wifi_rsp_system_hello(
    const void *nul
)
```

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

Callback programming

Callback programming is a style of computer programming, which allows lower layer of software to call functions defined on a higher layer. Callback is piece of code or a reference to a piece of code that is passed as an argument. The figure below illustrates the callback architecture used with BGLib.

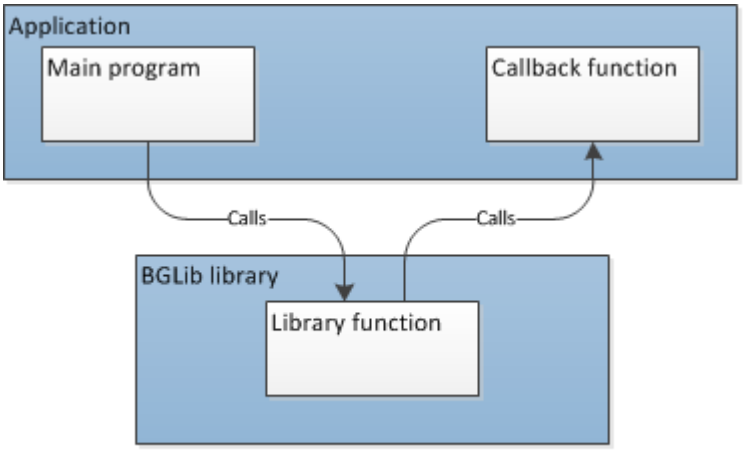


Figure: Callback arhitecture

If you are not familiar with callback programming a basic tutorial can for example be found from here:

http://www.codeguru.com/cpp/cpp/cpp_mfc/callbacks/article.php/c10557

4.3 BGScript API definition -- WIFI

The BGScript functions are also documented in the API reference section. The format of the commands varies slightly from the C-library functions and instead of using call backs 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 and they are not documented separately.

4.4 Data types -- WIFI

Table: Data types used in the documentation

Name	Length	Human readable	Script equivalent	Description
uint8	1 byte	5	Number	A positive integer in the range 0 - 255 (inclusive)
int8	1 byte	-13	Number	An integer in the range -127 - 128 (inclusive)
uint16	2 bytes	1034	Number	A positive integer in little endian format
int16	2 bytes	1232	Number	A signed integer in little endian format
uint32	4 bytes	70000	See note	A positive integer in little endian format
int32	4 bytes	-69876	Number	A signed integer in little endian format
hw_addr	6 bytes	00:07:80:12:34:56	Array of 6 bytes	Also called MAC address
ipv4	4 bytes	192.168.1.1	Array of 4 bytes	IPv4 address in big endian format
uint8array	1 - n	"Hello"	Array	The first byte is the length followed by a variable number of bytes. This means that the maximum size of the payload of a uint8array is 255 bytes.

The script's number type is internally a signed 32 bit integer. This means large unsigned 32-bit integer numbers in API (uint32) will be represented by negative numbers in the script. This is a known limitation.

5 API Reference -- WIFI

This section of the document contains the actual API description, so the description of commands, responses, events and enumerations. The categorization is based on the command classes, which are:

Description	Explanation
System	Various system functions, such as querying the hardware status or reset it
Configuration	Provides access to the devices parameters such as the MAC address
Wi-Fi	Provides access to 802.11 MAC and procedures like access point discovery
TCP stack	Gives access to the TCP/IP stack and various protocols like TCP and UDP
Endpoint	Provides functions to control the data endpoints
Hardware	An interface to access the various hardware layers such as timers, ADC and other hardware interfaces
Persistent store	Allows user to read/write data to non-volatile memory
Device Firmware Upgrade	Provides access to firmware update functions
I2C	Provides the commands for access I2C peripherals
HTTP Server	Provides the commands for activating and managing the embedded Web server

Final section of the API reference contains description of the error codes categorized as follows:

Description
BGAPI errors
TCPIP errors

5.1 System--WIFI

System class provides simple functions to access and query the local device.

5.1.1 Commands--system--WIFI

System commands

Hello--system--WIFI

Hello command can be used to check the communication with the Wi-Fi software (and hardware) works. The command is similar to AT-OK sequence.

Table: 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: RESPONSE

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

C Functions

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

/* Callback *
void wifi_rsp_system_hello(
    const void *nul
)
```

BGScript Functions

```
call system_hello()
```

Reset--system--WIFI

This command resets the Wi-Fi module. The command does not have a response, but it triggers the event called Boot. If you wish the module to enter DFU firmware upgrade mode after boot, for firmware re-flashing over the BGAPI interface, use boot mode 1. Otherwise if you just wish to reset the module use boot mode 0.

Table: 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**	Selects the boot mode **0 : boot to main program** **1 : boot to DFU**

Table: EVENTS

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

C Functions

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

BGScript Functions

```
call system_reset(dfu)
```

Set Max Power Saving State--system--WIFI

This command can be used to set the maximum power saving state allowed for the Wi-Fi module.

Mode 0 means no power saving is in use, but it provides lowest latency and best performance.

Mode 1 means that only the Wi-Fi radio is allowed to sleep and it will automatically go to sleep after 6000ms of inactivity.

<sleep>

Table: 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**	Max allowed power saving state. **0 :** low latency, high performance **1 :** save power, fast wake-up **2 :** deep sleep, requires ext interrupt to wakeup (default)

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x03	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

```
/* Function */
void wifi_cmd_system_set_max_power_saving_state(
    uint8 state
);

/* Callback */
struct wifi_msg_system_set_max_power_saving_state_rsp_t{
    uint16 result
}
void wifi_rsp_system_set_max_power_saving_state(
    const struct wifi_msg_system_set_max_power_saving_state_rsp_t * msg
)
```

BGScript Functions

```
call system_set_max_power_saving_state(state)(result)
```

Sync--system--WIFI

This command can be used to synchronize the system state. When the sync command is received events are output representing the system status. This can be used to synchronize the host software's status with the Wi-Fi software's status.

Table: 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: RESPONSE

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

C Functions

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

/* Callback *
void wifi_rsp_system_sync(
    const void *nul
)
```

BGScript Functions

```
call system_sync()
```

5.1.2 Events--system--WIFI

System events

Boot--system--WIFI

This event indicates the device has started and is ready to receive commands.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hilen	Message type: event
1	0x0E	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x00	method	Message ID
4 - 5	**uint16**	**major**	Major release version
6 - 7	**uint16**	**minor**	Minor release version
8 - 9	**uint16**	**patch**	Patch release version
10 - 11	**uint16**	**build**	Build number
12 - 13	**uint16**	**bootloader_version**	Bootloader version
14 - 15	**uint16**	**tcpip_version**	TCP/IP stack version
16 - 17	**uint16**	**hw**	Hardware version

C Functions

```
/* Callback */
struct wifi_msg_system_boot_evt_t{
    uint16 major,
    uint16 minor,
    uint16 patch,
    uint16 build,
    uint16 bootloader_version,
    uint16 tcpip_version,
    uint16 hw
}
void wifi_evt_system_boot(
    const struct wifi_msg_system_boot_evt_t * msg
)
```

BGScript Functions

```
event system_boot(major, minor, patch, build, bootloader_version, tcpip_version, hw)
```

Power Saving State--system--WIFI

This event indicates the power saving state the module has entered.

Table: 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**	See Set Max Power Saving State

C Functions

```
/* Callback */
struct wifi_msg_system_power_saving_state_evt_t{
    uint8 state
}
void wifi_evt_system_power_saving_state(
    const struct wifi_msg_system_power_saving_state_evt_t * msg
)
```

BGScript Functions

```
event system_power_saving_state(state)
```

SW Exception--system--WIFI

Software exception has occurred.

Table: 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.

C Functions

```
/* Callback */
struct wifi_msg_system_sw_exception_evt_t{
    uint32 address,
    uint8 type
}
void wifi_evt_system_sw_exception(
    const struct wifi_msg_system_sw_exception_evt_t * msg
)
```

BGScript Functions

```
event system_sw_exception(address, type)
```

5.2 Configuration--WIFI

Configuration class command provide functions to change the local devices configuration.

5.2.1 Commands--config--WIFI

Configuration commands

Get Mac--config--WIFI

This command reads the IEEE address from the device.

Table: 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**	The hardware interface to use **0 : Wi-Fi**

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x02	class	Message class: Configuration
3	0x00	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**hw_interface**	Always zero for Wi-Fi client

Table: EVENTS

event	Description
config_mac_address	Device MAC address

C Functions

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

/* Callback */
struct wifi_msg_config_get_mac_rsp_t{
    uint16 result,
    uint8 hw_interface
}
void wifi_rsp_config_get_mac(
    const struct wifi_msg_config_get_mac_rsp_t * msg
)
```

BGScript Functions

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

Set Mac--config--WIFI

This command writes the IEEE address to the device.

Table: 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**	The hardware interface to use **0: Wi-Fi**
5 - 10	**hw_addr**	**mac**	The new MAC address to set

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x02	class	Message class: Configuration
3	0x01	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**hw_interface**	Always zero for Wi-Fi client

C Functions

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

/* Callback */
struct wifi_msg_config_set_mac_rsp_t{
    uint16 result,
    uint8 hw_interface
}
void wifi_rsp_config_set_mac(
    const struct wifi_msg_config_set_mac_rsp_t * msg
)
```

BGScript Functions

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

5.2.2 Events--config--WIFI

Configuration events

Mac Address--config--WIFI

The current MAC address

Table: 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**	The hardware interface to use. **0: Wi-Fi**
5 - 10	**hw_addr**	**mac**	The current MAC address

C Functions

```
/* Callback */
struct wifi_msg_config_mac_address_evt_t{
    uint8 hw_interface,
    hw_addr mac
}
void wifi_evt_config_mac_address(
    const struct wifi_msg_config_mac_address_evt_t * msg
)
```

BGScript Functions

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

5.3 Wi-Fi--WIFI

Low-level Wi-Fi control

5.3.1 Commands--sme--WIFI

Wi-Fi commands

Wifi On--sme--WIFI

Turns on the 802.11 radio.

Table: 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: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x00	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table: EVENTS

event	Description
sme_wifi_is_on	Sent after Wi-Fi has been switched on

C Functions

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

/* Callback */
struct wifi_msg_sme_wifi_on_rsp_t{
    uint16 result
}
void wifi_rsp_sme_wifi_on(
    const struct wifi_msg_sme_wifi_on_rsp_t * msg
)
```

BGScript Functions

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

Wifi Off--sme--WIFI

Turns off the 802.11 radio.

Table: 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: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x01	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table: EVENTS

event	Description
sme_wifi_off	Sent after Wi-Fi has been switched off

C Functions

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

/* Callback */
struct wifi_msg_sme_wifi_off_rsp_t{
    uint16 result
}
void wifi_rsp_sme_wifi_off(
    const struct wifi_msg_sme_wifi_off_rsp_t * msg
)
```

BGScript Functions

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

Set Scan Channels--sme--WIFI

Set default scan channel list for commands Start Scan and Connect Ssid.

Table: 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**	The hardware interface to use. **0 : Wi-Fi**
5	**uint8array**	**chList**	List of channels to scan. By default all channels are used if this command is never used. Set a subset of the channels to scan by filling the list in order of priority. (See BGScript examples below.)

Table: RESPONSE

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 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

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

/* Callback */
struct wifi_msg_sme_set_scan_channels_rsp_t{
    uint16 result
}
void wifi_rsp_sme_set_scan_channels(
    const struct wifi_msg_sme_set_scan_channels_rsp_t * msg
)
```

BGScript Functions

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

BGScript - Example 1 - Scan only channels 1, 3 and 13 starting from channel 3, then 13 and finally 1:

```
call sme_set_scan_channels(0, 3, "\x03\x0d\x01")
```

Start Scan--sme--WIFI

Initiates a scan for Access Points. Scanning is not possible once connected.

Table: 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**	The hardware interface to use. **0 : Wi-Fi**
5	**uint8array**	**chList**	The list of channels which will be scanned for Access Points. Empty list means scan channels according to configuration of Set Scan Channels. Fill the list to override the configuration of the Set Scan Channels. (See BGScript examples below.)

Table: RESPONSE

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 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table: EVENTS

event	Description
sme_scan_result	Sent for each Access Point
sme_scan_result_drop	Sent for each dropped Access Point
sme_scanned	Sent after scan completes

C Functions

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

/* Callback */
struct wifi_msg_sme_start_scan_rsp_t{
    uint16 result
}
void wifi_rsp_sme_start_scan(
    const struct wifi_msg_sme_start_scan_rsp_t * msg
)
```

BGScript Functions

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

BGScript - Example 1 - Scan channels according to configuration in Set Scan Channels:

call sme_start_scan(0, 0, "")

BGScript - Example 2 - Scan only channels 2, 3 and 13 starting from channel 3, then 13 and finally 2 (overrides the configuration of Set Scan Channels):

call sme_start_scan(0, 3, "\x03\x0d\x02")

Stop Scan--sme--WIFI

Terminates the active scanning procedure.

Table: 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: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x04	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table: EVENTS

event	Description
sme_scanned	Sent after scan stopped

C Functions

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

/* Callback */
struct wifi_msg_sme_stop_scan_rsp_t{
    uint16 result
}
void wifi_rsp_sme_stop_scan(
    const struct wifi_msg_sme_stop_scan_rsp_t * msg
)
```

BGScript Functions

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

Connect BSSID--sme--WIFI

This command tries to connect to a specific Access Point using its unique BSSID. In order to succeed the command requires a preceding sme_start_scan command where the desired wireless network was found.

Table: 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 that the module should connect to

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x09	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x06	method	Message ID
4 - 5	**uint16**	**result**	Result code **0: success** **Non-zero : an error occurred** Error code 180 is typical if the wireless network was not found in the preceding scan. For other values refer to the error code documentation
6	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**
7 - 12	**hw_addr**	**bssid**	The BSSID of the Access Point that the module will attempt to connect to

Table: EVENTS

event	Description
sme_connect_retry	Sent when a connection attempt fails and is automatically retried
sme_connected	Sent when connection succeeds
sme_connect_failed	Sent when connection fails
sme_interface_status	Sent after Wi-Fi interface is ready

C Functions

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

/* Callback */
struct wifi_msg_sme_connect_bssid_rsp_t{
    uint16 result,
    uint8 hw_interface,
    hw_addr bssid
}
void wifi_rsp_sme_connect_bssid(
    const struct wifi_msg_sme_connect_bssid_rsp_t * msg
)
```

BGScript Functions

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

Disconnect--sme--WIFI

Disconnects from the currently connected Access Point.

Table: 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: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x08	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**

Table: EVENTS

event	Description
sme_disconnected	Sent after disconnected
sme_interface_status	Sent after Wi-Fi interface is down

C Functions

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

/* Callback */
struct wifi_msg_sme_disconnect_rsp_t{
    uint16 result,
    uint8 hw_interface
}
void wifi_rsp_sme_disconnect(
    const struct wifi_msg_sme_disconnect_rsp_t * msg
)
```

BGScript Functions

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

Scan Results Sort Rssi--sme--WIFI

Resend scan results of a previous scan, sorted by RSSI. It can be run only after the command sme_start_scan has been issued at least once in the current session.

Table: 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**	Max number of wireless networks to list, closest on top

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0D	method	Message ID
4 - 5	**uint16**	**result**	Return code **0 : success** **non-zero : An error occurred** For error values refer to the error code documentation

Table: EVENTS

event	Description
sme_scan_sort_result	Sent for each network
sme_scan_sort_finished	Sent after operation completes

C Functions

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

/* Callback */
struct wifi_msg_sme_scan_results_sort_rssi_rsp_t{
    uint16 result
}
void wifi_rsp_sme_scan_results_sort_rssi(
    const struct wifi_msg_sme_scan_results_sort_rssi_rsp_t * msg
)
```

BGScript Functions

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

Set Password--sme--WIFI

Set the network password used when authenticating with an access point.

Table: 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 between 8 and 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: RESPONSE

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	**uint8**	**status**	Result code **0: success** **128: invalid password**

C Functions

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

/* Callback */
struct wifi_msg_sme_set_password_rsp_t{
    uint8 status
}
void wifi_rsp_sme_set_password(
    const struct wifi_msg_sme_set_password_rsp_t * msg
)
```

BGScript Functions

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

Connect SSID--sme--WIFI

This command starts a connection establishment procedure to an Access Point with the given SSID. The command supports both visible and hidden SSIDs.

Executing this command will also launch a transparent scan procedure in order to discover the Access Points, but the results of the scan procedure will not be exposed to the user.The channels used in the scan procedure can be defined with the Set Scan Channels command. If the command had not been executed all channels will be scanned.

Table: 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: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x09	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x07	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**
7 - 12	**hw_addr**	**bssid**	The BSSID of the access point that will be connected to

Table: EVENTS

event	Description
sme_connect_retry	Sent when a connection attempt fails and is automatically retried
sme_connected	Sent when connection succeeds
sme_connect_failed	Sent when connection fails
sme_interface_status	Sent after Wi-Fi interface is up

C Functions

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

/* Callback */
struct wifi_msg_sme_connect_ssid_rsp_t{
    uint16 result,
    uint8 hw_interface,
    hw_addr bssid
}
void wifi_rsp_sme_connect_ssid(
    const struct wifi_msg_sme_connect_ssid_rsp_t * msg
)
```

BGScript Functions

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

Get Signal Quality

Get connection signal quality

Table: 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: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x13	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**

Table: EVENTS

event	Description
sme_signal_quality	Connection signal quality

C Functions

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

/* Callback */
struct wifi_msg_sme_get_signal_quality_rsp_t{
    uint16 result,
    uint8 hw_interface
}
void wifi_rsp_sme_get_signal_quality(
    const struct wifi_msg_sme_get_signal_quality_rsp_t * msg
)
```

BGScript Functions

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

Start WPS

This command starts the Wi-Fi Protected Setup (WPS) session

Table: 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: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x11	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**

Table: EVENTS

event	Description
sme_wps_completed	Sent when WPS successfully completes
sme_wps_failed	Sent when WPS fails
sme_wps_credential_ssid	Received network name credential
sme_wps_credential_password	Received network password credential

C Functions

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

/* Callback */
struct wifi_msg_sme_start_wps_rsp_t{
    uint16 result,
    uint8 hw_interface
}
void wifi_rsp_sme_start_wps(
    const struct wifi_msg_sme_start_wps_rsp_t * msg
)
```

BGScript Functions

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

Stop WPS

This command stops the Wi-Fi Protected Setup (WPS) session.

Table: 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: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x12	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**

Table: EVENTS

event	Description
sme_wps_stopped	Sent after WPS stopped

C Functions

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

/* Callback */
struct wifi_msg_sme_stop_wps_rsp_t{
    uint16 result,
    uint8 hw_interface
}
void wifi_rsp_sme_stop_wps(
    const struct wifi_msg_sme_stop_wps_rsp_t * msg
)
```

BGScript Functions

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

Set Operating Mode--sme--WIFI

This command sets Wi-Fi operating mode either to Wi-Fi client (STA) or Wi-Fi Access Point (AP). Operating mode comes into effect only after the next time the radio is turned on by launching the command sme_wifi_on

Table: 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**	**1:** Station (default) **2:** Access Point

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0A	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

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

/* Callback */
struct wifi_msg_sme_set_operating_mode_rsp_t{
    uint16 result
}
void wifi_rsp_sme_set_operating_mode(
    const struct wifi_msg_sme_set_operating_mode_rsp_t * msg
)
```

BGScript Functions

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

Set AP Max Clients

Set the maximum amount of stations that can be associated to the Access Point at the same time.

Table: 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 stations from 0 to 5.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x10	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**

C Functions

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

/* Callback */
struct wifi_msg_sme_set_ap_max_clients_rsp_t{
    uint16 result,
    uint8 hw_interface
}
void wifi_rsp_sme_set_ap_max_clients(
    const struct wifi_msg_sme_set_ap_max_clients_rsp_t * msg
)
```

BGScript Functions

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

Set AP Password--sme--WIFI

This command sets the Wi-Fi password for an Access Point.

Table: 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 Access Point

Table: RESPONSE

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	**uint8**	**status**	Result code **0: success** **128: invalid password**

C Functions

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

/* Callback */
struct wifi_msg_sme_set_ap_password_rsp_t{
    uint8 status
}
void wifi_rsp_sme_set_ap_password(
    const struct wifi_msg_sme_set_ap_password_rsp_t * msg
)
```

BGScript Functions

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

Start AP Mode--sme--WIFI

This command starts the Wi-Fi Access Point mode.

Table: 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.
5	**uint8**	**security**	Security mode to use. **0:** Open security **1:** WPA security **2:** WPA2 security **3:** WEP security
6	**uint8array**	**ssid**	SSID of the network

Table: RESPONSE

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 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**

Table: EVENTS

event	Description
sme_ap_mode_started	Sent after AP mode successfully started
sme_ap_mode_failed	Sent after AP mode fails
sme_interface_status	Sent after Wi-Fi interface is up

C Functions

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

/* Callback */
struct wifi_msg_sme_start_ap_mode_rsp_t{
    uint16 result,
    uint8 hw_interface
}
void wifi_rsp_sme_start_ap_mode(
    const struct wifi_msg_sme_start_ap_mode_rsp_t * msg
)
```

BGScript Functions

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

Stop AP Mode--sme--WIFI

This command stops the Wi-Fi Access Point mode.

Table: 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: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0C	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**

Table: EVENTS

event	Description
sme_ap_mode_stopped	Sent after AP mode stopped
sme_interface_status	Sent after Wi-Fi interface is down

C Functions

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

/* Callback */
struct wifi_msg_sme_stop_ap_mode_rsp_t{
    uint16 result,
    uint8 hw_interface
}
void wifi_rsp_sme_stop_ap_mode(
    const struct wifi_msg_sme_stop_ap_mode_rsp_t * msg
)
```

BGScript Functions

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

AP Client Disconnect--sme--WIFI

This command disconnects a station from the network (access point).

Table: 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 station

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x0E	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**

Table: EVENTS

event	Description
sme_ap_client_left	Send when the station has been disconnected

C Functions

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

/* Callback */
struct wifi_msg_sme_ap_client_disconnect_rsp_t{
    uint16 result,
    uint8 hw_interface
}
void wifi_rsp_sme_ap_client_disconnect(
    const struct wifi_msg_sme_ap_client_disconnect_rsp_t * msg
)
```

BGScript Functions

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

5.3.2 Events--sme--WIFI

Wi-Fi events

Wifi Is On--sme--WIFI

Event indicates that the 802.11 radio is powered up and ready to receive commands.

Table: 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**	Return code **0 : success** **non-zero : An error occurred** For error values refer to the error code documentation

C Functions

```
/* Callback */
struct wifi_msg_sme_wifi_is_on_evt_t{
    uint16 result
}
void wifi_evt_sme_wifi_is_on(
    const struct wifi_msg_sme_wifi_is_on_evt_t * msg
)
```

BGScript Functions

```
event sme_wifi_is_on(result)
```

Wifi Is Off--sme--WIFI

An event indicating the 802.11 radio has been powered off. This event can also indicate that the 802.11 radio had an internal error and was shut down, and the user application will need to re-initialize it by turning it on.

Table: 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**	Return code **0 : success** **non-zero : An error occurred** For error values refer to the error code documentation

C Functions

```
/* Callback */
struct wifi_msg_sme_wifi_is_off_evt_t{
    uint16 result
}
void wifi_evt_sme_wifi_is_off(
    const struct wifi_msg_sme_wifi_is_off_evt_t * msg
)
```

BGScript Functions

```
event sme_wifi_is_off(result)
```

Scan Result--sme--WIFI

This event indicates Access Point scan results. After a scan has been started these events are 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: 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 an Access Point which was found
10	**int8**	**channel**	The channel on which an Access Point was seen
11 - 12	**int16**	**rssi**	The received signal strength indication of the found Access Point, in dBm. The RSSI values are reported in 1dBm steps. This event is triggered only if the RSSI value corresponds to -88dBm or above.
13	**int8**	**snr**	The signal to noise ratio of an Access Point. The values are reported in 1dB steps.
14	**uint8**	**secure**	Access Point security status as a bitmask. **bit 0: whether the AP supports secure connections** **bit 1: whether the AP supports WPS**
15	**uint8array**	**ssid**	The SSID of the network the Access Point belongs to

C Functions

```
/* Callback */
struct wifi_msg_sme_scan_result_evt_t{
    hw_addr bssid,
    int8 channel,
    int16 rssi,
    int8 snr,
    uint8 secure,
    uint8 ssid_len,
    const uint8* ssid_data
}
void wifi_evt_sme_scan_result(
    const struct wifi_msg_sme_scan_result_evt_t * msg
)
```

BGScript Functions

```
event sme_scan_result(bssid, channel, rssi, snr, secure, ssid_len, ssid_data)
```

Scan Result Drop--sme--WIFI

Event indicates that the Access Point was dropped from the internal scan list. It is not possible to connect to it anymore.

Table: 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

C Functions

```
/* Callback */
struct wifi_msg_sme_scan_result_drop_evt_t{
    hw_addr bssid
}
void wifi_evt_sme_scan_result_drop(
    const struct wifi_msg_sme_scan_result_drop_evt_t * msg
)
```

BGScript Functions

```
event sme_scan_result_drop(bssid)
```

Scanned--sme--WIFI

This event indicates the Access Point scan has finished.

Table: 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 normally**

C Functions

```
/* Callback */
struct wifi_msg_sme_scanned_evt_t{
    int8 status
}
void wifi_evt_sme_scanned(
    const struct wifi_msg_sme_scanned_evt_t * msg
)
```

BGScript Functions

```
event sme_scanned(status)
```

Scan Sort Result--sme--WIFI

scan result

Table: 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 which was found
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. The values are reported in 1dBm steps.
13	**int8**	**snr**	The signal to noise ratio of the Access Point. The values are reported in 1dB steps.
14	**uint8**	**secure**	Access Point security status as a bitmask. **bit 0: whether the AP supports secure connections** **bit 1: whether the AP supports WPS**
15	**uint8array**	**ssid**	The SSID of the network the Access Point belongs to.

C Functions

```
/* Callback */
struct wifi_msg_sme_scan_sort_result_evt_t{
    hw_addr bssid,
    int8 channel,
    int16 rssi,
    int8 snr,
    uint8 secure,
    uint8 ssid_len,
    const uint8* ssid_data
}
void wifi_evt_sme_scan_sort_result(
    const struct wifi_msg_sme_scan_sort_result_evt_t * msg
)
```

BGScript Functions

```
event sme_scan_sort_result(bssid, channel, rssi, snr, secure, ssid_len, ssid_data)
```

Scan Sort Finished--sme--WIFI

scan result sort finished

Table: 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

C Functions

```
/* Callback *
void wifi_evt_sme_scan_sort_finished(
    const void *nul
)
```

BGScript Functions

```
event sme_scan_sort_finished()
```

Connected--sme--WIFI

Event indicates a successful connection to an Access point.

Table: 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**	Wi-Fi connection status **0**: Wi-Fi is connected **1:** Wi-Fi is not connected
5	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**
6 - 11	**hw_addr**	**bssid**	The BSSID of the device that the module connected to

C Functions

```
/* Callback */
struct wifi_msg_sme_connected_evt_t{
    int8 status,
    uint8 hw_interface,
    hw_addr bssid
}
void wifi_evt_sme_connected(
    const struct wifi_msg_sme_connected_evt_t * msg
)
```

BGScript Functions

```
event sme_connected(status, hw_interface, bssid)
```

Connect Retry--sme--WIFI

Informative event indicating a connection attempt failed, and an automatic retry will take place.

Table: 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**

C Functions

```
/* Callback */
struct wifi_msg_sme_connect_retry_evt_t{
    uint8 hw_interface
}
void wifi_evt_sme_connect_retry(
    const struct wifi_msg_sme_connect_retry_evt_t * msg
)
```

BGScript Functions

```
event sme_connect_retry(hw_interface)
```

Connect Failed--sme--WIFI

Indicates a failed connection to an Access Point. This 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: 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**	Failure reason For values refer to the error code documentation
6	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**

C Functions

```
/* Callback */
struct wifi_msg_sme_connect_failed_evt_t{
    uint16 reason,
    uint8 hw_interface
}
void wifi_evt_sme_connect_failed(
    const struct wifi_msg_sme_connect_failed_evt_t * msg
)
```

BGScript Functions

```
event sme_connect_failed(reason, hw_interface)
```

Disconnected--sme--WIFI

Event indicates a disconnections from an Access Point.

Table: 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 code documentation
6	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**

C Functions

```
/* Callback */
struct wifi_msg_sme_disconnected_evt_t{
    uint16 reason,
    uint8 hw_interface
}
void wifi_evt_sme_disconnected(
    const struct wifi_msg_sme_disconnected_evt_t * msg
)
```

BGScript Functions

```
event sme_disconnected(reason, hw_interface)
```

WPS Credential SSID

This event is produced during Wi-Fi Protected Setup (WPS) and contains the SSID of the network.

Table: 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

C Functions

```
/* Callback */
struct wifi_msg_sme_wps_credential_ssid_evt_t{
    uint8 hw_interface,
    uint8 ssid_len,
    const uint8* ssid_data
}
void wifi_evt_sme_wps_credential_ssid(
    const struct wifi_msg_sme_wps_credential_ssid_evt_t * msg
)
```

BGScript Functions

```
event sme_wps_credential_ssid(hw_interface, ssid_len, ssid_data)
```

WPS Credential Password

This event is produced during Wi-Fi Protected Setup (WPS) and contains the password of the network.

Table: 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.

C Functions

```
/* Callback */
struct wifi_msg_sme_wps_credential_password_evt_t{
    uint8 hw_interface,
    uint8 password_len,
    const uint8* password_data
}
void wifi_evt_sme_wps_credential_password(
    const struct wifi_msg_sme_wps_credential_password_evt_t * msg
)
```

BGScript Functions

```
event sme_wps_credential_password(hw_interface, password_len, password_data)
```

WPS Completed

This event indicates the Wi-Fi Protected Setup (WPS) session was completed successfully.

Table: 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**

C Functions

```
/* Callback */
struct wifi_msg_sme_wps_completed_evt_t{
    uint8 hw_interface
}
void wifi_evt_sme_wps_completed(
    const struct wifi_msg_sme_wps_completed_evt_t * msg
)
```

BGScript Functions

```
event sme_wps_completed(hw_interface)
```

WPS Failed

This event indicates the Wi-Fi Protected Setup (WPS) session failed.

Table: 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**	Failure reason For values refer to the error code documentation
6	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**

C Functions

```
/* Callback */
struct wifi_msg_sme_wps_failed_evt_t{
    uint16 reason,
    uint8 hw_interface
}
void wifi_evt_sme_wps_failed(
    const struct wifi_msg_sme_wps_failed_evt_t * msg
)
```

BGScript Functions

```
event sme_wps_failed(reason, hw_interface)
```

WPS Stopped

This event indicates the Wi-Fi Protected Setup (WPS) session was stopped.

Table: 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**

C Functions

```
/* Callback */
struct wifi_msg_sme_wps_stopped_evt_t{
    uint8 hw_interface
}
void wifi_evt_sme_wps_stopped(
    const struct wifi_msg_sme_wps_stopped_evt_t * msg
)
```

BGScript Functions

```
event sme_wps_stopped(hw_interface)
```

Signal Quality

Connection signal quality

Table: 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**

C Functions

```
/* Callback */
struct wifi_msg_sme_signal_quality_evt_t{
    int8 rssi,
    uint8 hw_interface
}
void wifi_evt_sme_signal_quality(
    const struct wifi_msg_sme_signal_quality_evt_t * msg
)
```

BGScript Functions

```
event sme_signal_quality(rssi, hw_interface)
```

AP Mode Started--sme--WIFI

This event indicates the Wi-Fi Access Point mode has been successfully started.

Table: 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**

C Functions

```
/* Callback */
struct wifi_msg_sme_ap_mode_started_evt_t{
    uint8 hw_interface
}
void wifi_evt_sme_ap_mode_started(
    const struct wifi_msg_sme_ap_mode_started_evt_t * msg
)
```

BGScript Functions

```
event sme_ap_mode_started(hw_interface)
```

AP Mode Stopped--sme--WIFI

This event indicates the Wi-Fi Access Point mode has been stopped.

Table: 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**

C Functions

```
/* Callback */
struct wifi_msg_sme_ap_mode_stopped_evt_t{
    uint8 hw_interface
}
void wifi_evt_sme_ap_mode_stopped(
    const struct wifi_msg_sme_ap_mode_stopped_evt_t * msg
)
```

BGScript Functions

```
event sme_ap_mode_stopped(hw_interface)
```

Ap Mode Failed--sme--WIFI

This event indicates the Wi-Fi Access Point mode has failed.

Table: 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**	Failure reason For values refer to the error code documentation
6	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**

C Functions

```
/* Callback */
struct wifi_msg_sme_ap_mode_failed_evt_t{
    uint16 reason,
    uint8 hw_interface
}
void wifi_evt_sme_ap_mode_failed(
    const struct wifi_msg_sme_ap_mode_failed_evt_t * msg
)
```

BGScript Functions

```
event sme_ap_mode_failed(reason, hw_interface)
```

AP Client Joined--sme--WIFI

This event indicates a Wi-Fi client has joined the network.

Table: 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 station
10	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**

C Functions

```
/* Callback */
struct wifi_msg_sme_ap_client_joined_evt_t{
    hw_addr mac_address,
    uint8 hw_interface
}
void wifi_evt_sme_ap_client_joined(
    const struct wifi_msg_sme_ap_client_joined_evt_t * msg
)
```

BGScript Functions

```
event sme_ap_client_joined(mac_address, hw_interface)
```

AP Client Left--sme--WIFI

This event indicates a Wi-Fi client (client), has left the access point.

This happens for example in a case when a station goes out of range or is abruptly powered off. It might take from 180s to 270s for this event to be issued at the module operating as the access point.

Table: 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 station
10	**uint8**	**hw_interface**	Hardware interface **0 : Wi-Fi**

C Functions

```
/* Callback */
struct wifi_msg_sme_ap_client_left_evt_t{
    hw_addr mac_address,
    uint8 hw_interface
}
void wifi_evt_sme_ap_client_left(
    const struct wifi_msg_sme_ap_client_left_evt_t * msg
)
```

BGScript Functions

```
event sme_ap_client_left(mac_address, hw_interface)
```

Interface Status--sme--WIFI

This event indicates the current network status. Once for example DHCP has successfully finished and the module has an IP address, it will send this with status = 1.

Table: 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**	Network status **0 : network down** **1 : network up**

C Functions

```
/* Callback */
struct wifi_msg_sme_interface_status_evt_t{
    uint8 hw_interface,
    uint8 status
}
void wifi_evt_sme_interface_status(
    const struct wifi_msg_sme_interface_status_evt_t * msg
)
```

BGScript Functions

```
event sme_interface_status(hw_interface, status)
```

5.4 TCP stack--WIFI

TCP/IP class commands provide access to the TCP/IP stack and functions like DHCP or IP address setup. The class also allows TCP and UDP clients and servers to be created.

5.4.1 Commands--tcpip--WIFI

TCP stack commands

Configure--tcpip--WIFI

This command configure TCP/IP settings.

When using static IP addresses, this command can be used to configure the local IP address, netmask and gateway.

When enabling DHCP Client the settings for the static IP will be stored, but they will overridden as soon as the remote DHCP Server will assign to the module its IP configuration.

Table: 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**	The local IP address of the device
8 - 11	**ipv4**	**netmask**	The netmask of the device
12 - 15	**ipv4**	**gateway**	The gateway
16	**uint8**	**use_dhcp**	Use DHCP **0 : Use static IP settings** **1 : DHCP Client is used to obtain an IP configuration**

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x04	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table: EVENTS

event	Description
tcpip_configuration	Sent when TCP/IP configuration changes

C Functions

```
/* Function */
void wifi_cmd_tcpip_configure(
    ipv4 address,
    ipv4 netmask,
    ipv4 gateway,
    uint8 use_dhcp
);

/* Callback */
struct wifi_msg_tcpip_configure_rsp_t{
    uint16 result
}
void wifi_rsp_tcpip_configure(
    const struct wifi_msg_tcpip_configure_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_configure(address, netmask, gateway, use_dhcp)(result)
```

DHCP Set Hostname--tcpip--WIFI

Set DHCP hostname parameter (option 12) used in client DHCPDISCOVER and DHCPREQUEST messages.

Table: 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**	Hostname

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x08	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

```
/* Function */
void wifi_cmd_tcpip_dhcp_set_hostname(
    uint8 hostname_len,
    const uint8* hostname_data
);

/* Callback */
struct wifi_msg_tcpip_dhcp_set_hostname_rsp_t{
    uint16 result
}
void wifi_rsp_tcpip_dhcp_set_hostname(
    const struct wifi_msg_tcpip_dhcp_set_hostname_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_dhcp_set_hostname(hostname_len, hostname_data)(result)
```

DNS Configure--tcpip--WIFI

This command configures DNS client settings.

Table: 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**	Two different DNS servers can be stored. Index indicates which of the two this is. **0 : primary DNS server** **1 : secondary DNS server**
5 - 8	**ipv4**	**address**	The IP address of the DNS server

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x05	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table: EVENTS

event	Description
tcpip_dns_configuration	Sent when DNS configuration changes

C Functions

```
/* Function */
void wifi_cmd_tcpip_dns_configure(
    uint8 index,
    ipv4 address
);

/* Callback */
struct wifi_msg_tcpip_dns_configure_rsp_t{
    uint16 result
}
void wifi_rsp_tcpip_dns_configure(
    const struct wifi_msg_tcpip_dns_configure_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_dns_configure(index, address)(result)
```

DNS Gethostbyname--tcpip--WIFI

This command starts a hostname to an IP address resolve procedure using the configured DNS servers.

Table: 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 name of the server whose IP address should be resolved, for example www.bluegiga.com.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x06	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table: EVENTS

event	Description
tcpip_dns_gethostbyname_result	Sent when DNS resolver query completes

C Functions

```
/* Function */
void wifi_cmd_tcpip_dns_gethostbyname(
    uint8 name_len,
    const uint8* name_data
);

/* Callback */
struct wifi_msg_tcpip_dns_gethostbyname_rsp_t{
    uint16 result
}
void wifi_rsp_tcpip_dns_gethostbyname(
    const struct wifi_msg_tcpip_dns_gethostbyname_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_dns_gethostbyname(name_len, name_data)(result)
```

TCP Connect--tcpip--WIFI

This command attempts to create a new TCP socket to a TCP server.

Table: 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**	The IP address of the remote server to which the module should connect.
8 - 9	**uint16**	**port**	The TCP port on the remote server
10	**int8**	**routing**	The endpoint where the incoming data from the TCP server should be routed to. **-1** : data received is not automatically routed to other endpoint, but received as endpoint_data event.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x01	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint id of the newly created TCP connection

Table: EVENTS

event	Description
tcpip_endpoint_status	Sent when TCP/IP endpoint status changes
endpoint_status	Sent when endpoint status changes

C Functions

```
/* Function */
void wifi_cmd_tcpip_tcp_connect(
    ipv4 address,
    uint16 port,
    int8 routing
);

/* Callback */
struct wifi_msg_tcpip_tcp_connect_rsp_t{
    uint16 result,
    uint8 endpoint
}
```

```
void wifi_rsp_tcpip_tcp_connect(
    const struct wifi_msg_tcpip_tcp_connect_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_tcp_connect(address, port, routing)(result, endpoint)
```

Start TCP Server--tcpip--WIFI

This command starts a TCP server.

Once the TCP server is started, and a remote client establishes a new connection, the data coming from this client will be routed by default to the endpoint specified in the **default_destination** parameter. If such endpoint, say the UART interface, is configured to communicate with host via the BGAPI, then data will be carried via the endpoint_data event, otherwise raw data is sent out of the specified interface. When **-1** is used, data received from the client is passed to BGScript via endpoint_data event, and/or endpoint_data event containing the data is sent out of the interfaces over which BGAPI is enabled.

Table: 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**	The local port which to listen on.
6	**int8**	**default_destination**	Endpoint where data from connected client will be routed to. Use **-1** to generate endpoint_data events instead.

Table: RESPONSE

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**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**endpoint**	The number of the endpoint the newly created TCP server uses

Table: EVENTS

event	Description
tcpip_endpoint_status	Sent when TCP/IP endpoint status changes
endpoint_status	Sent when endpoint status changes

C Functions

```
/* Function */
void wifi_cmd_tcpip_start_tcp_server(
    uint16 port,
    int8 default_destination
);

/* Callback */
```

```
struct wifi_msg_tcpip_start_tcp_server_rsp_t{
    uint16 result,
    uint8 endpoint
}
void wifi_rsp_tcpip_start_tcp_server(
    const struct wifi_msg_tcpip_start_tcp_server_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_start_tcp_server(port, default_destination)(result, endpoint)
```

UDP Connect--tcpip--WIFI

This command attempts to create a new UDP connection.

Table: 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**	The IP address of the remote server to which the packets should be sent
8 - 9	**uint16**	**port**	The UDP port of the remote server
10	**int8**	**routing**	The endpoint index where the data from this connection should be routed to

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x03	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint index of the newly created UDP connection

Table: EVENTS

event	Description
tcpip_endpoint_status	Sent when TCP/IP endpoint status changes
endpoint_status	Sent when endpoint status changes

C Functions

```
/* Function */
void wifi_cmd_tcpip_udp_connect(
    ipv4 address,
    uint16 port,
    int8 routing
);

/* Callback */
struct wifi_msg_tcpip_udp_connect_rsp_t{
    uint16 result,
    uint8 endpoint
}
void wifi_rsp_tcpip_udp_connect(
    const struct wifi_msg_tcpip_udp_connect_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_udp_connect(address, port, routing)(result, endpoint)
```

UDP Bind

This command binds an existing UDP connection to a specific source port.

Table: 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**	The endpoint which source port to change
5 - 6	**uint16**	**port**	The UDP port of source

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x07	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

```
/* Function */
void wifi_cmd_tcpip_udp_bind(
    uint8 endpoint,
    uint16 port
);

/* Callback */
struct wifi_msg_tcpip_udp_bind_rsp_t{
    uint16 result
}
void wifi_rsp_tcpip_udp_bind(
    const struct wifi_msg_tcpip_udp_bind_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_udp_bind(endpoint, port)(result)
```

Start UDP Server--tcpip--WIFI

This command starts an UDP server.

All WF121 software profiles other than **wifi_http_serv**, the software currently has four (4) UDP control blocks. Out of these four blocks DNS and DHCP clients always reserve one block each. Also in Wi-Fi Access Point mode DNS and DHCP servers both reserve one block each. Now this typically leaves two (2) blocks available for the user, e.g. enough for one bidirectional UDP connection (server + client) or two unidirectional UDP connections.

Table: 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**	the local UDP port that the server listens on
6	**int8**	**default_destination**	The endpoint to which incoming UDP packets should be written. -1 for destination means incoming data is notified with Udp Data event which is the event carrying in addition the source IP address and port.

Table: RESPONSE

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**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint number of the newly created UDP server

Table: EVENTS

event	Description
tcpip_endpoint_status	Sent when TCP/IP endpoint status changes
endpoint_status	Sent when endpoint status changes

C Functions

```
/* Function */
void wifi_cmd_tcpip_start_udp_server(
    uint16 port,
    int8 default_destination
);

/* Callback */
```

```
struct wifi_msg_tcpip_start_udp_server_rsp_t{
    uint16 result,
    uint8 endpoint
}
void wifi_rsp_tcpip_start_udp_server(
    const struct wifi_msg_tcpip_start_udp_server_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_start_udp_server(port, default_destination)(result, endpoint)
```

5.4.2 Events--tcpip--WIFI

TCP stack events

Configuration--tcpip--WIFI

This event indicates TCP/IP configuration status.

Table: 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**	The IP address of the local device
8 - 11	**ipv4**	**netmask**	Netmask of the local device
12 - 15	**ipv4**	**gateway**	The gateway of the local device
16	**uint8**	**use_dhcp**	DHCP used **0 = DHCP Client not being used** **1 = DHCP Client being used**

C Functions

```
/* Callback */
struct wifi_msg_tcpip_configuration_evt_t{
    ipv4 address,
    ipv4 netmask,
    ipv4 gateway,
    uint8 use_dhcp
}
void wifi_evt_tcpip_configuration(
    const struct wifi_msg_tcpip_configuration_evt_t * msg
)
```

BGScript Functions

```
event tcpip_configuration(address, netmask, gateway, use_dhcp)
```

DNS Configuration--tcpip--WIFI

This event indicates DNS configuration status.

Note that if static IP configuration is in use and no DNS configuration has been provided by the user, the primary DNS server defaults to 208.67.222.222 (resolver1.opendns.com).

Table: 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**	DNS server ID **0:** primary DNS server **1:** secondary DNS server
5 - 8	**ipv4**	**address**	The IP address of the DNS server

C Functions

```
/* Callback */
struct wifi_msg_tcpip_dns_configuration_evt_t{
    uint8 index,
    ipv4 address
}
void wifi_evt_tcpip_dns_configuration(
    const struct wifi_msg_tcpip_dns_configuration_evt_t * msg
)
```

BGScript Functions

```
event tcpip_dns_configuration(index, address)
```

DNS Gethostbyname Result--tcpip--WIFI

The event is generated as a response to a DNS Gethostbyname command. If the procedure is successful, this message contains the IP address of the queried address.

Table: 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
6 - 9	**ipv4**	**address**	The resolved IP address of the server
10	**uint8array**	**name**	Name of the server whose IP address was resolved

C Functions

```
/* Callback */
struct wifi_msg_tcpip_dns_gethostbyname_result_evt_t{
    uint16 result,
    ipv4 address,
    uint8 name_len,
    const uint8* name_data
}
void wifi_evt_tcpip_dns_gethostbyname_result(
    const struct wifi_msg_tcpip_dns_gethostbyname_result_evt_t * msg
)
```

BGScript Functions

```
event tcpip_dns_gethostbyname_result(result, address, name_len, name_data)
```

Endpoint Status--tcpip--WIFI

This event indicates the current status of a TCP/IP endpoint.

Table: 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**	The endpoint index this message describes
5 - 8	**ipv4**	**local_ip**	The local IP address of this endpoint
9 - 10	**uint16**	**local_port**	The local port of this endpoint
11 - 14	**ipv4**	**remote_ip**	The remote IP address of this endpoint. For server endpoints (which have no client), this will not contain any valid value.
15 - 16	**uint16**	**remote_port**	The port of the remote device. For server endpoints (which have no client), this will not contain any valid value.

C Functions

```
/* Callback */
struct wifi_msg_tcpip_endpoint_status_evt_t{
    uint8 endpoint,
    ipv4 local_ip,
    uint16 local_port,
    ipv4 remote_ip,
    uint16 remote_port
}
void wifi_evt_tcpip_endpoint_status(
    const struct wifi_msg_tcpip_endpoint_status_evt_t * msg
)
```

BGScript Functions

```
event tcpip_endpoint_status(endpoint, local_ip, local_port, remote_ip, remote_port)
```

UDP Data--tcpip--WIFI

This event indicates incoming data from an UDP endpoint. 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: 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**	The endpoint which received this data, i.e. to which it was sent
5 - 8	**ipv4**	**source_address**	IP address of the client that sent this data
9 - 10	**uint16**	**source_port**	Client UDP port where this data was sent from.
11 - 12	**uint16array**	**data**	The raw data

C Functions

```
/* Callback */
struct wifi_msg_tcpip_udp_data_evt_t{
    uint8 endpoint,
    ipv4 source_address,
    uint16 source_port,
    uint16 data_len,
    const uint8* data_data
}
void wifi_evt_tcpip_udp_data(
    const struct wifi_msg_tcpip_udp_data_evt_t * msg
)
```

BGScript Functions

```
event tcpip_udp_data(endpoint, source_address, source_port, data_len, data_data)
```

5.5 Endpoint--WIFI

Endpoint class functions provide the control of endpoints and allow them to be created, deleted and also allows the data routing to be configured.

5.5.1 Commands--endpoint--WIFI

Endpoint commands

Set Active--endpoint--WIFI

This command can be used to activate or deactivate endpoints. By default endpoints are active, 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 UART endpoint over which BGAPI is carried, will prevent BGAPI events and responses to go out of that UART interface. Similarly, deactivating the BGScript endpoint will prevent events to be passed to the script, thus preventing the calls in it to be executed. Server endpoints however are never active, as they can neither send nor receive data.

Table: 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**	The endpoint to control
5	**uint8**	**active**	Endpoint status **0 : inactive** **1 : active**

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x02	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint that was controlled

Table: EVENTS

event	Description
endpoint_status	Sent when endpoint status changes

C Functions

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

/* Callback */
struct wifi_msg_endpoint_set_active_rsp_t{
    uint16 result,
    uint8 endpoint
}
void wifi_rsp_endpoint_set_active(
    const struct wifi_msg_endpoint_set_active_rsp_t * msg
)
```

BGScript Functions

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

Send--endpoint--WIFI

This command sends data to a given endpoint.

Table: 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**	The index of the endpoint to which the data will be sent.
5	**uint8array**	**data**	The RAW data which will be written or sent.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x00	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint to which the data was written.

C Functions

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

/* Callback */
struct wifi_msg_endpoint_send_rsp_t{
    uint16 result,
    uint8 endpoint
}
void wifi_rsp_endpoint_send(
    const struct wifi_msg_endpoint_send_rsp_t * msg
)
```

BGScript Functions

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

Set Transmit Size--endpoint--WIFI

Set transmit packet size of a TCP/UDP endpoint. Endpoint will buffer outgoing data until packet size is reached and then transmit it. When using packet size 0 the data will be sent out without a fixed size.

Table: 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**	The index of the endpoint
5 - 6	**uint16**	**size**	Size of data packet to transmit **0:** No defined packet size

Table: RESPONSE

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 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint that was configured

C Functions

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

/* Callback */
struct wifi_msg_endpoint_set_transmit_size_rsp_t{
    uint16 result,
    uint8 endpoint
}
void wifi_rsp_endpoint_set_transmit_size(
    const struct wifi_msg_endpoint_set_transmit_size_rsp_t * msg
)
```

BGScript Functions

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

Set Streaming--endpoint--WIFI

This command configures a UART into a streaming or BGAPI mode. When a UART endpoint is in a streaming mode, the data gets transparently routed to another endpoint like TCP. In BGAPI mode the data is exposed via BGAPI.

This setting currently only operates on UART endpoints.

Table: 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**	The endpoint whose streaming mode should be changed
5	**uint8**	**streaming**	Endpoint mode **0 : Use as BGAPI interface** **1 : Streaming to another endpoint**

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x01	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint this message refers to

Table: EVENTS

event	Description
endpoint_status	Sent when endpoint status changes

C Functions

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

/* Callback */
struct wifi_msg_endpoint_set_streaming_rsp_t{
    uint16 result,
    uint8 endpoint
}
void wifi_rsp_endpoint_set_streaming(
```

```
    const struct wifi_msg_endpoint_set_streaming_rsp_t * msg
)
```

BGScript Functions

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

Set Streaming Destination--endpoint--WIFI

This command can be used to set the destination where data from an endpoint will be routed to.

Table: 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**	The endpoint which to control
5	**int8**	**streaming_destination**	The destination for the data

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x03	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint that was controlled

Table: EVENTS

event	Description
endpoint_status	Sent when endpoint status changes

C Functions

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

/* Callback */
struct wifi_msg_endpoint_set_streaming_destination_rsp_t{
    uint16 result,
    uint8 endpoint
}
void wifi_rsp_endpoint_set_streaming_destination(
    const struct wifi_msg_endpoint_set_streaming_destination_rsp_t * msg
)
```

BGScript Functions

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

Close--endpoint--WIFI

This command can be used to close an endpoint.

Table: 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**	The index of the endpoint to close

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x04	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint that was closed

Table: EVENTS

event	Description
endpoint_status	Sent when endpoint status changes

C Functions

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

/* Callback */
struct wifi_msg_endpoint_close_rsp_t{
    uint16 result,
    uint8 endpoint
}
void wifi_rsp_endpoint_close(
    const struct wifi_msg_endpoint_close_rsp_t * msg
)
```

BGScript Functions

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

5.5.2 Events--endpoint--WIFI

Endpoint events

Status--endpoint--WIFI

This event indicates an endpoint's status.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hilen	Message type: event
1	0x08	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x02	method	Message ID
4	**uint8**	**endpoint**	The index of the endpoint whose status this event describes
5 - 8	**uint32**	**type**	The type of endpoint, see the endpoint type enumeration
9	**uint8**	**streaming**	Endpoint mode **0 : Endpoint is connected to BGAPI** **1 : Endpoint is streaming to another endpoint**
10	**int8**	**destination**	The index of the endpoint to which the incoming data goes
11	**uint8**	**active**	Endpoint status **0 : receiving and sending of data is blocked** **1 : receiving and sending is allowed.**

C Functions

```
/* Callback */
struct wifi_msg_endpoint_status_evt_t{
    uint8 endpoint,
    uint32 type,
    uint8 streaming,
    int8 destination,
    uint8 active
}
void wifi_evt_endpoint_status(
    const struct wifi_msg_endpoint_status_evt_t * msg
)
```

BGScript Functions

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

Data--endpoint--WIFI

This event indicates incoming data from an endpoint.

Table: 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**	The endpoint which received this data, i.e. to which it was sent.
5	**uint8array**	**data**	The raw data

C Functions

```
/* Callback */
struct wifi_msg_endpoint_data_evt_t{
    uint8 endpoint,
    uint8 data_len,
    const uint8* data_data
}
void wifi_evt_endpoint_data(
    const struct wifi_msg_endpoint_data_evt_t * msg
)
```

BGScript Functions

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

Closing--endpoint--WIFI

This event indicates an endpoint is closing or indicates that the remote end has terminated the connection. The event should be acknowledged by calling the endpoint close command or otherwise the software will not re-use the endpoint index.

Table: 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**	Zero indicates success. For other values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint which is closing

C Functions

```
/* Callback */
struct wifi_msg_endpoint_closing_evt_t{
    uint16 reason,
    uint8 endpoint
}
void wifi_evt_endpoint_closing(
    const struct wifi_msg_endpoint_closing_evt_t * msg
)
```

BGScript Functions

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

Error

This event indicated an error in an endpoint.

Table: 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 code documentation
6	**uint8**	**endpoint**	The endpoint where the error occurred

C Functions

```
/* Callback */
struct wifi_msg_endpoint_error_evt_t{
    uint16 reason,
    uint8 endpoint
}
void wifi_evt_endpoint_error(
    const struct wifi_msg_endpoint_error_evt_t * msg
)
```

BGScript Functions

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

Syntax Error

This event indicates a syntax error.

Table: 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**	Error reason For values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint where the error occurred

C Functions

```
/* Callback */
struct wifi_msg_endpoint_syntax_error_evt_t{
    uint16 result,
    uint8 endpoint
}
void wifi_evt_endpoint_syntax_error(
    const struct wifi_msg_endpoint_syntax_error_evt_t * msg
)
```

BGScript Functions

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

5.5.3 Enumerations--endpoint--WIFI

Endpoint commands

Endpoint types--endpoint--WIFI

Endpoint types. Note that these are not to be confused with the dynamic endpoint indexes used for example in the endpoint_send.

Table: VALUES

Value	Name	Description
0	endpoint_free	Endpoint is not in use
1	endpoint_uart	Endpoint of type hardware UART
2	endpoint_usb	USB
4	endpoint_tcp	TCP client
8	endpoint_tcp_server	TCP server
16	endpoint_udp	UDP client
32	endpoint_udp_server	UDP server
64	endpoint_script	Scripting
128	endpoint_wait_close	Waiting for closing
256	endpoint_spi	SPI
512	endpoint_i2c	I2C
1024	endpoint_drop	Drop all data sent to this

5.6 Hardware--WIFI

The Hardware class provides methods to access the local devices hardware interfaces such asIO and timers etc.

5.6.1 Commands--hardware--WIFI

Hardware commands

ADC Read--hardware--WIFI

This command is used to read the devices A/D converter.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x09	method	Message ID
4	**uint8**	**input**	Adc input.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x05	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x09	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**input**	ADC input.
7 - 8	**uint16**	**value**	ADC value

C Functions

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

/* Callback */
struct wifi_msg_hardware_adc_read_rsp_t{
    uint16 result,
    uint8 input,
    uint16 value
}
void wifi_rsp_hardware_adc_read(
    const struct wifi_msg_hardware_adc_read_rsp_t * msg
)
```

BGScript Functions

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

Change Notification Config--hardware--WIFI

This command configures change notifications (CN). The PIC32 micro controller has a limited number of standard GPIO interrupts. Change notifications can be used in a very similar way to GPIO interrupts in many cases but they are not identical and operate on different pins. This configures for which pins the change notification interrupts are enabled. For a list of what pin is what change notification source, please refer to the Datasheet *page 9, Table 2: Multifunction pad descriptions.* Even more information can be found in the PIC32 datasheet chapter discussing change notifications.

WF121 pin #	PIC32 pin	Change notification bit
2	RB15	12
14	RB1	3
15	RB0	2
24	RB5	7
33	RC13	1
34	RC14	0
35	RF4	17
36	RF5	18
42	RD6	15
43	RD7	16

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x04	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x02	method	Message ID
4 - 7	**uint32**	**enable**	Change notification bits to enable

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x02	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table: EVENTS

Event	Description
hardware change_notification	Sent after a pin state change has been detected, and the pin change notification for that pin is enabled

C Functions

```
/* Function */
void wifi_cmd_hardware_change_notification_config(
    uint32 enable
);

/* Callback */
struct wifi_msg_hardware_change_notification_config_rsp_t{
    uint16 result
}
void wifi_rsp_hardware_change_notification_config(
    const struct wifi_msg_hardware_change_notification_config_rsp_t * msg
)
```

BGScript Functions

```
call hardware_change_notification_config(enable)(result)
```

Change Notification Pullup--hardware--WIFI

This command configures change notification pull-ups. For a detailed discussion about change notifications, see the Change Notification Config command.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x04	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x03	method	Message ID
4 - 7	**uint32**	**pullup**	Bitmask for which of the change notification pins have pull-ups enabled

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x03	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

```
/* Function */
void wifi_cmd_hardware_change_notification_pullup(
    uint32 pullup
);

/* Callback */
struct wifi_msg_hardware_change_notification_pullup_rsp_t{
    uint16 result
}
void wifi_rsp_hardware_change_notification_pullup(
    const struct wifi_msg_hardware_change_notification_pullup_rsp_t * msg
)
```

BGScript Functions

```
call hardware_change_notification_pullup(pullup)(result)
```

External Interrupt Config--hardware--WIFI

This command configures pins which will generate interrupts.

In the WF121 Wi-Fi module there are four pins which support interrupts: RD0/INT0, RD9/INT2, RD10/INT3, RD11/INT4. INT1 is reserved for the WF121s internal use only.

For example to enable the interrupt on pin RD11 which is identified by bit 4 (fifth bit from right) of bitmask, the *enable* bitfield should be set 0x10 (that is, binary 10000)

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x01	method	Message ID
4	**uint8**	**enable**	External interrupt bits to enable
5	**uint8**	**polarity**	External interrupt polarity, 1=rising edge, 0=falling edge

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x01	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table: EVENTS

Event	Description
hardware external_interrupt	Sent after external interrupt detected.

C Functions

```
/* Function */
void wifi_cmd_hardware_external_interrupt_config(
    uint8 enable,
    uint8 polarity
);

/* Callback */
struct wifi_msg_hardware_external_interrupt_config_rsp_t{
    uint16 result
}
void wifi_rsp_hardware_external_interrupt_config(
    const struct wifi_msg_hardware_external_interrupt_config_rsp_t * msg
)
```

BGScript Functions

```
call hardware_external_interrupt_config(enable, polarity)(result)
```

IO Port Config Direction--hardware--WIFI

This command configures I/O-port(s) direction.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x05	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x04	method	Message ID
4	**uint8**	**port**	Port index **0 : A** **1 : B** **2 : C** **3 : D** **4 : E** **5 : F** **6 : G**
5 - 6	**uint16**	**mask**	Bitmask of which pins on the port this command affects.
7 - 8	**uint16**	**direction**	The bitmask describing which are inputs and which are outputs. **0 : Output** **1 : Input**

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x04	method	Message ID
4 - 5	**uint16**	**result**	Return code. **0 : Success** **non-zero : An error occurred** For other values refer to the error code documentation

C Functions

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

/* Callback */
struct wifi_msg_hardware_io_port_config_direction_rsp_t{
    uint16 result
}
void wifi_rsp_hardware_io_port_config_direction(
    const struct wifi_msg_hardware_io_port_config_direction_rsp_t * msg
)
```

BGScript Functions

```
call hardware_io_port_config_direction(port, mask, direction)(result)
```

IO Port Config Open Drain--hardware--WIFI

This command configure I/O-port open drain. Open drain means that the pin when high, is in high impedance state and when low is in able to sink current. Open drain is sometimes also called Open Collector.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x05	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x05	method	Message ID
4	**uint8**	**port**	Port index **0 : A** **1 : B** **2 : C** **3 : D** **4 : E** **5 : F** **6 : G**
5 - 6	**uint16**	**mask**	Bitmask of which pins on the port this command affects
7 - 8	**uint16**	**open_drain**	Bitmask of which pins are configured to be open drain. For each bit this means: **0 : Open drain disabled** **1 : Open drain enabled**

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x05	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

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

/* Callback */
struct wifi_msg_hardware_io_port_config_open_drain_rsp_t{
    uint16 result
}
```

```
void wifi_rsp_hardware_io_port_config_open_drain(
    const struct wifi_msg_hardware_io_port_config_open_drain_rsp_t * msg
)
```

BGScript Functions

```
call hardware_io_port_config_open_drain(port, mask, open_drain)(result)
```

IO Port Read--hardware--WIFI

This command reads the status of pins in an I/O-port.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x07	method	Message ID
4	**uint8**	**port**	Port index: **0 : A** **1 : B** **2 : C** **3 : D** **4 : E** **5 : F** **6 : G**
5 - 6	**uint16**	**mask**	Bitmask of which pins on the port should be read. For each bit in the bitmask: **0 : Don't read** **1 : Read**

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x05	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x07	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**port**	Port index
7 - 8	**uint16**	**data**	port data

C Functions

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

/* Callback */
struct wifi_msg_hardware_io_port_read_rsp_t{
    uint16 result,
    uint8 port,
    uint16 data
}
```

```
void wifi_rsp_hardware_io_port_read(
    const struct wifi_msg_hardware_io_port_read_rsp_t * msg
)
```

BGScript Functions

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

IO Port Write--hardware--WIFI

This command writes the pins of an I/O-port.

Table: 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**	**port**	Port index: **0 : A** **1 : B** **2 : C** **3 : D** **4 : E** **5 : F** **6 : G**
5 - 6	**uint16**	**mask**	Bitmask of which pins on the port this command affects. For each bit in the bitmask: 0 = Don't modify/write, 1 = modify/write
7 - 8	**uint16**	**data**	Bitmask of which pins to set. For each bit in the bitmask: **0 : low** **1 : high**

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x06	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

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

/* Callback */
struct wifi_msg_hardware_io_port_write_rsp_t{
    uint16 result
}
void wifi_rsp_hardware_io_port_write(
    const struct wifi_msg_hardware_io_port_write_rsp_t * msg
)
```

BGScript Functions

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

Output Compare--hardware--WIFI

Set output compare settings, e.g., using for PWM purposes. Output compare output is disabled when module enters sleep mode. The <timer> tag in the hardware.xml must be configured properly if using this command.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x08	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x08	method	Message ID
4	**uint8**	**index**	Index of output compare module. (1-5)
5	**uint8**	**bit32**	Is 32bit mode selected. **0:** 16bit **1:** 32bit, Requires timer to be configured for 32.
6	**uint8**	**timer**	Timer to use for comparison. **2:** Timer 2 **3:** Timer 3
7	**uint8**	**mode**	comparison mode **0:** Output compare peripheral is disabled but continues to draw current **1:** Initialize OCx pin low; compare event forces OCx pin high **2:** Initialize OCx pin high; compare event forces OCx pin low **3:** Compare event toggles OCx pin **4:** Initialize OCx pin low; generate single output pulse on OCx pin **5:** Initialize OCx pin low; generate continuous output pulses on OCx pin **6:** PWM mode on OCx; Fault pin disabled **7:** PWM mode on OCx; Fault pin enabled
8 - 11	**uint32**	**compare_value**	0-0xFFFF for 16bit 0-0xFFFFFFFF for 32 bit

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x08	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

```
/* Function */
void wifi_cmd_hardware_output_compare(
    uint8 index,
    uint8 bit32,
    uint8 timer,
```

```
    uint8 mode,
    uint32 compare_value
);

/* Callback */
struct wifi_msg_hardware_output_compare_rsp_t{
    uint16 result
}
void wifi_rsp_hardware_output_compare(
    const struct wifi_msg_hardware_output_compare_rsp_t * msg
)
```

BGScript Functions

```
call hardware_output_compare(index, bit32, timer, mode, compare_value)(result)
```

RTC Init

This command initializes the internal Real Time Clock.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0A	method	Message ID
4	**uint8**	**enable**	Enable/Disable RTC.
5 - 6	**int16**	**drift**	Drift of clock. Added to 32.768kHz SOSC every minute.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0A	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

```
/* Function */
void wifi_cmd_hardware_rtc_init(
    uint8 enable,
    int16 drift
);

/* Callback */
struct wifi_msg_hardware_rtc_init_rsp_t{
    uint16 result
}
void wifi_rsp_hardware_rtc_init(
    const struct wifi_msg_hardware_rtc_init_rsp_t * msg
)
```

BGScript Functions

```
call hardware_rtc_init(enable, drift)(result)
```

RTC Set Time

This command sets the internal Real Time Clock time.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x08	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0B	method	Message ID
4 - 5	**int16**	**year**	Current year of RTC time (2000-2099)
6	**int8**	**month**	Current month of RTC time (1-12)
7	**int8**	**day**	Current day of RTC time (1-31)
8	**int8**	**weekday**	Current weekday of RTC time (0-6 sunday-saturday)
9	**int8**	**hour**	Current hour of RTC time (0-23)
10	**int8**	**minute**	Current minute of RTC time (0-59)
11	**int8**	**second**	Current second of RTC time (0-59)

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0B	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

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

/* Callback */
struct wifi_msg_hardware_rtc_set_time_rsp_t{
    uint16 result
}
void wifi_rsp_hardware_rtc_set_time(
    const struct wifi_msg_hardware_rtc_set_time_rsp_t * msg
)
```

BGScript Functions

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

RTC Get Time

This commands reads the internal Real Time Clock value.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x00	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0C	method	Message ID

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x0A	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0C	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6 - 7	**int16**	**year**	Current year of RTC time (2000-2099)
8	**int8**	**month**	Current month of RTC time (1-12)
9	**int8**	**day**	Current day of RTC time (1-31)
10	**int8**	**weekday**	Current weekday of RTC time (0-6 sunday-saturday)
11	**int8**	**hour**	Current hour of RTC time (0-23)
12	**int8**	**minute**	Current minute of RTC time (0-59)
13	**int8**	**second**	Current second of RTC time (0-59)

C Functions

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

/* Callback */
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
}
void wifi_rsp_hardware_rtc_get_time(
    const struct wifi_msg_hardware_rtc_get_time_rsp_t * msg
)
```

BGScript Functions

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

RTC Set Alarm

This commands sets an alarm for the internal Real Time Clock.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x09	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0D	method	Message ID
4	**uint8**	**month**	Month for alarm (1-12), -1 for don't care
5	**uint8**	**day**	Day for alarm (1-31), -1 for don't care
6	**int8**	**weekday**	Weekday for alarm (0-6 sunday-saturday)
7	**uint8**	**hour**	Hour for alarm (0-23), -1 for don't care
8	**uint8**	**minute**	Minute for alarm (0-59), -1 for don't care
9	**uint8**	**second**	Second for alarm (0-59), -1 for don't care
10	**uint8**	**repeat_mask**	repeat mask for matching alarm time
11 - 12	**int16**	**repeat_count**	How often alarm repeats (0=no limit)

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0D	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table: EVENTS

event	Description
hardware_rtc_alarm	Sent when RTC alarm is triggered

C Functions

```
/* Function */
void wifi_cmd_hardware_rtc_set_alarm(
    uint8 month,
    uint8 day,
    int8 weekday,
    uint8 hour,
    uint8 minute,
    uint8 second,
    uint8 repeat_mask,
```

```
    int16 repeat_count
);

/* Callback */
struct wifi_msg_hardware_rtc_set_alarm_rsp_t{
    uint16 result
}
void wifi_rsp_hardware_rtc_set_alarm(
    const struct wifi_msg_hardware_rtc_set_alarm_rsp_t * msg
)
```

BGScript Functions

```
call hardware_rtc_set_alarm(month, day, weekday, hour, minute, second, repeat_mask,
repeat_count)(result)
```

Set Soft Timer--hardware--WIFI

This command enables the software timer. Multiple concurrent timers can be running at the same time.

Table: 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 how often to send events, in milliseconds. If time is 0, removes the scheduled timer
8	**uint8**	**handle**	Handle that is returned with event
9	**uint8**	**single_shot**	**0 : false** (Timer is repeating) **1 : true** (Timer runs only once)

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x00	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table: EVENTS

Event	Description
hardware soft_timer	Sent after specified interval

C Functions

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

/* Callback */
struct wifi_msg_hardware_set_soft_timer_rsp_t{
    uint16 result
}
void wifi_rsp_hardware_set_soft_timer(
    const struct wifi_msg_hardware_set_soft_timer_rsp_t * msg
)
```

BGScript Functions

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

5.6.2 Enumerations--hardware--WIFI

Hardware commands

RTC Alarm repeat--hardware--WIFI

How often alarm repeats

Table: VALUES

Value	Name	Description
0	hardware_alarm_every_half_second	Repeat every second
1	hardware_alarm_every_second	Repeat every second
2	hardware_alarm_every_ten_seconds	Repeat every ten seconds
3	hardware_alarm_every_minute	Repeat every minute
4	hardware_alarm_every_ten_minutes	Repeat every ten minutes
5	hardware_alarm_every_hour	Repeat every hour
6	hardware_alarm_every_day	Repeat every day
7	hardware_alarm_every_week	Repeat every week
8	hardware_alarm_every_month	Repeat every month
9	hardware_alarm_every_year	Repeat every year

5.6.3 Events--hardware--WIFI

Hardware events

Change Notification--hardware--WIFI

This event indicates an IO port status change and provides a time stamp when the change occurred.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hilen	Message type: event
1	0x04	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x01	method	Message ID
4 - 7	**uint32**	**timestamp**	Timestamp of when the change occurred

C Functions

```
/* Callback */
struct wifi_msg_hardware_change_notification_evt_t{
    uint32 timestamp
}
void wifi_evt_hardware_change_notification(
    const struct wifi_msg_hardware_change_notification_evt_t * msg
)
```

BGScript Functions

```
event hardware_change_notification(timestamp)
```

External Interrupt--hardware--WIFI

This event is generated when external interrupt occurs and provides a timestamp and IRQ. The IRQs and their corresponding pins are documented in the WF121 datasheet, page 9, table 2.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hilen	Message type: event
1	0x05	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x02	method	Message ID
4	**uint8**	**irq**	IRQ index
5 - 8	**uint32**	**timestamp**	Timestamp of when the interrupt occurred

Table: GPIO interrupts

WF121 pin number	IRQ	IRQ index
37	INT 4	4
38	INT 0	0
44	INT 2	2
46	INT 3	3

C Functions

```
/* Callback */
struct wifi_msg_hardware_external_interrupt_evt_t{
    uint8 irq,
    uint32 timestamp
}
void wifi_evt_hardware_external_interrupt(
    const struct wifi_msg_hardware_external_interrupt_evt_t * msg
)
```

BGScript Functions

```
event hardware_external_interrupt(irq, timestamp)
```

RTC Alarm

This event indicates and alarm from the internal RTC.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hilen	Message type: event
1	0x00	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x03	method	Message ID

C Functions

```
/* Callback *
void wifi_evt_hardware_rtc_alarm(
    const void *nul
)
```

BGScript Functions

```
event hardware_rtc_alarm()
```

Soft Timer--hardware--WIFI

This event indicates software timer has elapsed.

Table: 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**	Timer handle

C Functions

```
/* Callback */
struct wifi_msg_hardware_soft_timer_evt_t{
    uint8 handle
}
void wifi_evt_hardware_soft_timer(
    const struct wifi_msg_hardware_soft_timer_evt_t * msg
)
```

BGScript Functions

```
event hardware_soft_timer(handle)
```

5.7 I2C--WIFI

Inter-Integrated Circuit

5.7.1 Commands--i2c--WIFI

I2C commands

Start Read--i2c--WIFI

Start I2C transmission for reading. Actual data is received via the endpoint_data even. Only I2C Master is supported.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x04	lolen	Minimum payload length
2	0x08	class	Message class: I2C
3	0x00	method	Message ID
4	**uint8**	**endpoint**	I2C endpoint to use.
5 - 6	**uint16**	**slave_address**	Slave address to use
7	**uint8**	**length**	Amount of data to move

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x08	class	Message class: I2C
3	0x00	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

```
/* Function */
void wifi_cmd_i2c_start_read(
    uint8 endpoint,
    uint16 slave_address,
    uint8 length
);

/* Callback */
struct wifi_msg_i2c_start_read_rsp_t{
    uint16 result
}
void wifi_rsp_i2c_start_read(
    const struct wifi_msg_i2c_start_read_rsp_t * msg
)
```

BGScript Functions

```
call i2c_start_read(endpoint, slave_address, length)(result)
```

Start Write--i2c--WIFI

Prepare an I2C endpoint for data transmission. Actual data is sent via the endpoint_send command. Only I2C Master is supported.

Table: 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**	**endpoint**	I2C endpoint to use.
5 - 6	**uint16**	**slave_address**	Slave address to use

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x08	class	Message class: I2C
3	0x01	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

```
/* Function */
void wifi_cmd_i2c_start_write(
    uint8 endpoint,
    uint16 slave_address
);

/* Callback */
struct wifi_msg_i2c_start_write_rsp_t{
    uint16 result
}
void wifi_rsp_i2c_start_write(
    const struct wifi_msg_i2c_start_write_rsp_t * msg
)
```

BGScript Functions

```
call i2c_start_write(endpoint, slave_address)(result)
```

Stop--i2c--WIFI

Stop I2C transmission

Table: 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**	**endpoint**	I2C endpoint to use.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x08	class	Message class: I2C
3	0x02	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

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

/* Callback */
struct wifi_msg_i2c_stop_rsp_t{
    uint16 result
}
void wifi_rsp_i2c_stop(
    const struct wifi_msg_i2c_stop_rsp_t * msg
)
```

BGScript Functions

```
call i2c_stop(endpoint)(result)
```

5.8 Wired Ethernet--WIFI

Wired Ethernet

5.8.1 Commands--ethernet--WIFI

Wired Ethernet commands

Connected

Test wired Ethernet cable connection

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x00	lolen	Minimum payload length
2	0x0A	class	Message class: Wired Ethernet
3	0x02	method	Message ID

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x0A	class	Message class: Wired Ethernet
3	0x02	method	Message ID
4	**uint8**	**state**	Ethernet cable status **0**: Cable is connected **1:** Cable is not connected

C Functions

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

/* Callback */
struct wifi_msg_ethernet_connected_rsp_t{
    uint8 state
}
void wifi_rsp_ethernet_connected(
    const struct wifi_msg_ethernet_connected_rsp_t * msg
)
```

BGScript Functions

```
call ethernet_connected()(state)
```

Set Dataroute

Set wired Ethernet data route

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x0A	class	Message class: Wired Ethernet
3	0x00	method	Message ID
4	**uint8**	**route**	Ethernet data route **0 : not routed** **1 : routed to Wi-Fi** **2 : routed to local stack**

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x0A	class	Message class: Wired Ethernet
3	0x00	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table: EVENTS

event	Description
ethernet_link_status	Sent when Ethernet link status changes

C Functions

```
/* Function */
void wifi_cmd_ethernet_set_dataroute(
    uint8 route
);

/* Callback */
struct wifi_msg_ethernet_set_dataroute_rsp_t{
    uint16 result
}
void wifi_rsp_ethernet_set_dataroute(
    const struct wifi_msg_ethernet_set_dataroute_rsp_t * msg
)
```

BGScript Functions

```
call ethernet_set_dataroute(route)(result)
```

Close

Close wired Ethernet connection

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x00	lolen	Minimum payload length
2	0x0A	class	Message class: Wired Ethernet
3	0x01	method	Message ID

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x00	lolen	Minimum payload length
2	0x0A	class	Message class: Wired Ethernet
3	0x01	method	Message ID

C Functions

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

/* Callback *
void wifi_rsp_ethernet_close(
    const void *nul
)
```

BGScript Functions

```
call ethernet_close()
```

5.8.2 Events--ethernet--WIFI

Wired Ethernet events

Link Status

Wired Ethernet link state changed

Table: EVENT

Byte	Type	Name	Description
0	0x88	hilen	Message type: event
1	0x01	lolen	Minimum payload length
2	0x0A	class	Message class: Wired Ethernet
3	0x00	method	Message ID
4	**uint8**	**state**	Link state **0 : link down** **1 : link up**

C Functions

```
/* Callback */
struct wifi_msg_ethernet_link_status_evt_t{
    uint8 state
}
void wifi_evt_ethernet_link_status(
    const struct wifi_msg_ethernet_link_status_evt_t * msg
)
```

BGScript Functions

```
event ethernet_link_status(state)
```

5.9 HTTP Server--WIFI

5.9.1 Commands--https--WIFI

http server commands

Enable--https--WIFI

Enable/disable HTTP, DHCP and DNS servers.

When the DHCP server is started, the IP address pool for the clients will start with the IP address previously defined in the PSKey called FLASH_PS_KEY_DHCPS_SPACE, while the subnet mask will always be configured to 255.255.255.0

Table: 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**	Enable/disable HTTP server
5	**uint8**	**dhcps**	Enable/disable DHCP server
6	**uint8**	**dnss**	Enable/disable DNS server

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: http server
3	0x00	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

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

/* Callback */
struct wifi_msg_https_enable_rsp_t{
    uint16 result
}
```

```
void wifi_rsp_https_enable(
    const struct wifi_msg_https_enable_rsp_t * msg
)
```

BGScript Functions

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

5.9.2 Events--https--WIFI

http server events

Button--https--WIFI

Button clicked

Table: 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	0x01	method	Message ID
4	**uint8**	**number**	Button number

C Functions

```
/* Callback */
struct wifi_msg_https_button_evt_t{
    uint8 number
}
void wifi_evt_https_button(
    const struct wifi_msg_https_button_evt_t * msg
)
```

BGScript Functions

```
event https_button(number)
```

On Req--https--WIFI

Request to set module service on

Table: 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	0x00	method	Message ID
4	**uint8**	**service**	Service mode **1 : client mode** **2 : Access Point mode**

C Functions

```
/* Callback */
struct wifi_msg_https_on_req_evt_t{
    uint8 service
}
void wifi_evt_https_on_req(
    const struct wifi_msg_https_on_req_evt_t * msg
)
```

BGScript Functions

```
event https_on_req(service)
```

5.10 Persistent Store--WIFI

The Persistent Store (PS) class provides methods to read, write and dump the local devices parameters (PS keys).

4KiB of flash is reserved for Persistent Store.

PS Key ids over 0x8000 can be used for user storage.

Flash endurance limits PS total written data to 4MiB.

5.10.1 Commands--flash--WIFI

Persistent Store commands

PS Save--flash--WIFI

This command stores a value into the given PS (persistent store) key. This can be used to store user data in the flash memory, so that the data remains available across resets and power cycles. The max size of a single PS-key is 255 bytes and a total of 128 keys are available. There is 4KB of reserved space in total for all PS-Keys.

Table: 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 Keys 8000 to 807F can be used for persistent storage of user data.
6	**uint8array**	**value**	Key value

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x03	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table: EVENTS

event	Description
flash_ps_key_changed	Sent when a PS key changes

C Functions

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

/* Callback */
struct wifi_msg_flash_ps_save_rsp_t{
    uint16 result
}
void wifi_rsp_flash_ps_save(
    const struct wifi_msg_flash_ps_save_rsp_t * msg
)
```

BGScript Functions

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

PS Load--flash--WIFI

This command retrieves the value of the given PS key from the persistent store.

Table: 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 to load

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x04	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** ie: **0x0186 :Unspecified error, when trying to read a non-existent key** For other values refer to the error code documentation
6	**uint8array**	**value**	Key value, i.e. the data stored

C Functions

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

/* Callback */
struct wifi_msg_flash_ps_load_rsp_t{
    uint16 result,
    uint8 value_len,
    const uint8* value_data
}
void wifi_rsp_flash_ps_load(
    const struct wifi_msg_flash_ps_load_rsp_t * msg
)
```

BGScript Functions

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

PS Dump--flash--WIFI

This command dumps 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 having the key index 65535, indicating the dump has finished.

Table: 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: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x01	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table: EVENTS

Event	Description
flash ps_key	PS Key contents

C Functions

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

/* Callback */
struct wifi_msg_flash_ps_dump_rsp_t{
    uint16 result
}
void wifi_rsp_flash_ps_dump(
    const struct wifi_msg_flash_ps_dump_rsp_t * msg
)
```

BGScript Functions

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

PS Defrag--flash--WIFI

This command manually defragments the persistent store. Persistent store is also automatically defragmented if there is not enough space.

Table: 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: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x00	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

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

/* Callback */
struct wifi_msg_flash_ps_defrag_rsp_t{
    uint16 result
}
void wifi_rsp_flash_ps_defrag(
    const struct wifi_msg_flash_ps_defrag_rsp_t * msg
)
```

BGScript Functions

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

PS Erase--flash--WIFI

This command erases a single PS key and its value from the persistent store.

Table: 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 to erase

Table: RESPONSE

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**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table: EVENTS

event	Description
flash_ps_key_changed	Sent when a PS key changes

C Functions

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

/* Callback */
struct wifi_msg_flash_ps_erase_rsp_t{
    uint16 result
}
void wifi_rsp_flash_ps_erase(
    const struct wifi_msg_flash_ps_erase_rsp_t * msg
)
```

BGScript Functions

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

PS Erase All--flash--WIFI

This command erases all PS keys from the persistent store.

> This will erase the device's MAC address!

Table: 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: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x02	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table: EVENTS

event	Description
flash_ps_key_changed	Sent when a PS key changes

C Functions

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

/* Callback */
struct wifi_msg_flash_ps_erase_all_rsp_t{
    uint16 result
}
void wifi_rsp_flash_ps_erase_all(
    const struct wifi_msg_flash_ps_erase_all_rsp_t * msg
)
```

BGScript Functions

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

5.10.2 Enumerations--flash--WIFI

Persistent Store commands

PS_KEYS--flash--WIFI

Define keys

Table: VALUES

Value (dec)	Name	Description
1	FLASH_PS_KEY_MAC	MAC address
2	FLASH_PS_KEY_IPV4_SETTINGS	
3	FLASH_PS_KEY_DNS0_SETTINGS	
4	FLASH_PS_KEY_DNS1_SETTINGS	
5	FLASH_PS_KEY_MODULE_SERVICE	
10	FLASH_PS_KEY_APPL_NUM1	Integer type parameter for application
11	FLASH_PS_KEY_APPL_NUM2	Integer type parameter for application
12	FLASH_PS_KEY_APPL_NUM3	Integer type parameter for application
13	FLASH_PS_KEY_APPL_NUM4	Integer type parameter for application
14	FLASH_PS_KEY_APPL_STR1	String type parameter for application
15	FLASH_PS_KEY_APPL_STR2	String type parameter for application
16	FLASH_PS_KEY_APPL_STR3	String type parameter for application
17	FLASH_PS_KEY_APPL_STR4	String type parameter for application
18	FLASH_PS_KEY_APPL_TITLE	Web page title
20	FLASH_PS_KEY_AP_SSID	Access point SSID
21	FLASH_PS_KEY_AP_CHANNEL	Access point wifi channel number
22	FLASH_PS_KEY_AP_PW	Access point encryption password
23	FLASH_PS_KEY_AP_WIFI_N	Access point IEEE 802.11n enabled
24	FLASH_PS_KEY_AP_SECURITY	Access point secure mode 0=Open, 1=WPA, 2=WPA2, 3=WEP, 4=WPA2 mixed
25	FLASH_PS_KEY_CLIENT_SSID	Client SSID
26	FLASH_PS_KEY_CLIENT_PW	Client encryption password
30	FLASH_PS_KEY_DHCPS_ENABLE	DHCP server enable
31	FLASH_PS_KEY_DHCPS_SPACE	DHCP server first address of IPv4 address space
35	FLASH_PS_KEY_DNSS_ENABLE	DNS server enable
36	FLASH_PS_KEY_DNSS_URL	DNS server URL
37	FLASH_PS_KEY_DNSS_ANY_URL	DNS server reply to any URL enabled
40	FLASH_PS_KEY_AP_SCANLIST_ITEM_1	Access point found in client scan
41	FLASH_PS_KEY_AP_SCANLIST_ITEM_2	Access point found in client scan
42	FLASH_PS_KEY_AP_SCANLIST_ITEM_3	Access point found in client scan

Value (dec)	Name	Description
43	FLASH_PS_KEY_AP_SCANLIST_ITEM_4	Access point found in client scan
44	FLASH_PS_KEY_AP_SCANLIST_ITEM_5	Access point found in client scan
45	FLASH_PS_KEY_AP_SCANLIST_ITEM_6	Access point found in client scan
46	FLASH_PS_KEY_AP_SCANLIST_ITEM_7	Access point found in client scan
47	FLASH_PS_KEY_AP_SCANLIST_ITEM_8	Access point found in client scan
48	FLASH_PS_KEY_AP_SCANLIST_ITEM_9	Access point found in client scan
49	FLASH_PS_KEY_AP_SCANLIST_ITEM_10	Access point found in client scan
50	FLASH_PS_KEY_AP_LABEL1	Text of label in webpage
51	FLASH_PS_KEY_AP_LABEL2	Text of label in webpage
52	FLASH_PS_KEY_AP_LABEL3	Text of label in webpage
53	FLASH_PS_KEY_AP_LABEL4	Text of label in webpage
54	FLASH_PS_KEY_AP_LABEL5	Text of label in webpage
55	FLASH_PS_KEY_AP_LABEL6	Text of label in webpage
56	FLASH_PS_KEY_AP_LABEL7	Text of label in webpage
57	FLASH_PS_KEY_AP_LABEL8	Text of label in webpage
58	FLASH_PS_KEY_AP_LABEL9	Text of label in webpage
59	FLASH_PS_KEY_AP_LABEL10	Text of label in webpage

5.10.3 Events--flash--WIFI

Persistent Store events

PS Key Changed--flash--WIFI

This event indicates a PS key has been changed.

Table: 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

C Functions

```
/* Callback */
struct wifi_msg_flash_ps_key_changed_evt_t{
    uint16 key
}
void wifi_evt_flash_ps_key_changed(
    const struct wifi_msg_flash_ps_key_changed_evt_t * msg
)
```

BGScript Functions

```
event flash_ps_key_changed(key)
```

PS Key--flash--WIFI

This event is generated when PS keys are dumped from the persistent store.

Table: 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 **65535 : Last key**
6	**uint8array**	**value**	Key value

C Functions

```
/* Callback */
struct wifi_msg_flash_ps_key_evt_t{
    uint16 key,
    uint8 value_len,
    const uint8* value_data
}
void wifi_evt_flash_ps_key(
    const struct wifi_msg_flash_ps_key_evt_t * msg
)
```

BGScript Functions

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

5.11 Device Firmware Upgrade--WIFI

DFU class commands can be used to perform a firmware update. The commands and events are only available when the module has been booted into DFU mode.

5.11.1 Commands--dfu--WIFI

Device Firmware Upgrade commands

Reset--dfu--WIFI

This command resets the module. This command does not have a response, but triggers the event called Boot. Use boot mode 1 if you wish the module to enter DFU mode after the reset.

Table: 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 : Normal reset** **1 : Boot to DFU mode**

Table: EVENTS

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

C Functions

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

BGScript Functions

```
call dfu_reset(dfu)
```

Flash Set Address--dfu--WIFI

This command sets an address on the flash where the data will be written.

Table: 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 offset in the flash where to start flashing.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
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 **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

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

/* Callback */
struct wifi_msg_dfu_flash_set_address_rsp_t{
    uint16 result
}
void wifi_rsp_dfu_flash_set_address(
    const struct wifi_msg_dfu_flash_set_address_rsp_t * msg
)
```

BGScript Functions

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

Flash Upload--dfu--WIFI

This command uploads binary data to the device for flashing. Flash address will be updated automatically.

Table: 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**	An array of 128B of data which will be written into the flash.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
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 **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

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

/* Callback */
struct wifi_msg_dfu_flash_upload_rsp_t{
    uint16 result
}
void wifi_rsp_dfu_flash_upload(
    const struct wifi_msg_dfu_flash_upload_rsp_t * msg
)
```

BGScript Functions

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

Flash Upload Finish--dfu--WIFI

This command tells to the device the uploading of DFU data has been finished.

Table: 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: RESPONSE

Byte	Type	Name	Description
0	0x08	hilen	Message type: command
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 **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

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

/* Callback */
struct wifi_msg_dfu_flash_upload_finish_rsp_t{
    uint16 result
}
void wifi_rsp_dfu_flash_upload_finish(
    const struct wifi_msg_dfu_flash_upload_finish_rsp_t * msg
)
```

BGScript Functions

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

5.11.2 Events--dfu--WIFI

Device Firmware Upgrade events

Boot--dfu--WIFI

This event indicates that the module booted in DFU mode, and is ready to receive commands related to DFU firmware upgade.

Table: 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**	The version of the bootloader

C Functions

```
/* Callback */
struct wifi_msg_dfu_boot_evt_t{
    uint32 version
}
void wifi_evt_dfu_boot(
    const struct wifi_msg_dfu_boot_evt_t * msg
)
```

BGScript Functions

```
event dfu_boot(version)
```

5.12 Error codes -- WIFI

This section of the document describes the different error codes the Wi-Fi software can produce.

5.12.1 BGAPI Errors--WIFI

Errors related to BGAPI protocol

Invalid Parameter (0x0180)--WIFI

Command contained invalid parameter

Device in Wrong State (0x0181)--WIFI

Device is in wrong state to accept command

Out Of Memory (0x0182)--WIFI

Device has run out of memory

Feature Not Implemented (0x0183)--WIFI

Feature is not implemented

Command Not Recognized (0x0184)--WIFI

Command was not recognized

Timeout (0x0185)--WIFI

Command or Procedure failed due to timeout

Unspecified error (0x0186)--WIFI

Unspecified error

Hardware failure (0x0187)--WIFI

Hardware failure

Internal buffers are full (0x0188)--WIFI

Command not accepted, because internal buffers are full

Disconnected (0x0189)--WIFI

Command or Procedure failed due to disconnection

Too many requests (0x018A)--WIFI

Too many Simultaneous Requests

Access Point not in scanlist (0x018B)--WIFI

Access Point not found from scanlist

Invalid password (0x018C)--WIFI

Password is invalid or missing

Authentication failure (0x018D)--WIFI

WPA/WPA2 authentication has failed

Overflow (0x018E)--WIFI

Overflow detected

Multiple PBC sessions (0x018F)--WIFI

Multiple PBC sessions detected

Ethernet not connected (0x0190)--WIFI

Ethernet cable not connected

Ethernet route not set (0x0191)--WIFI

Ethernet route not set

Wrong operating mode (0x0192)--WIFI

Wrong operating mode for this command

5.12.2 Hardware Errors--WIFI

Errors related to hardware

PS Store is full (0x0301)--WIFI

Flash reserved for PS store is full

PS key not found (0x0302)--WIFI

PS key not found

i2c write already in progress (0x0303)--WIFI

Tried to start i2c write transmission, but it is already in progress.

i2c ack missing (0x0304)--WIFI

Acknowledge for i2c was not received.

5.12.3 TCP IP Errors--WIFI

Errors related to TCP/IP stack

Success (0x0200)--WIFI

No error

Out of memory (0x0201)--WIFI

Out of memory

Buffer error (0x0202)--WIFI

Buffer handling failed

Timeout (0x0203)--WIFI

Timeout

Routing (0x0204)--WIFI

Could not find route

In progress (0x0205)--WIFI

Operation in progress

Illegal_value (0x0206)--WIFI

Illegal value

would_block (0x0207)--WIFI

Operation would block

Address in use (0x0208)--WIFI

Address in use

Already connected (0x0209)--WIFI

Already connected

Connection aborted (0x020A)--WIFI

Connection aborted

Connection reset (0x020B)--WIFI

Connection reset

Connection closed (0x020C)--WIFI

Connection closed

Not connected (0x020D)--WIFI

Not connected

Illegal argument (0x020E)--WIFI

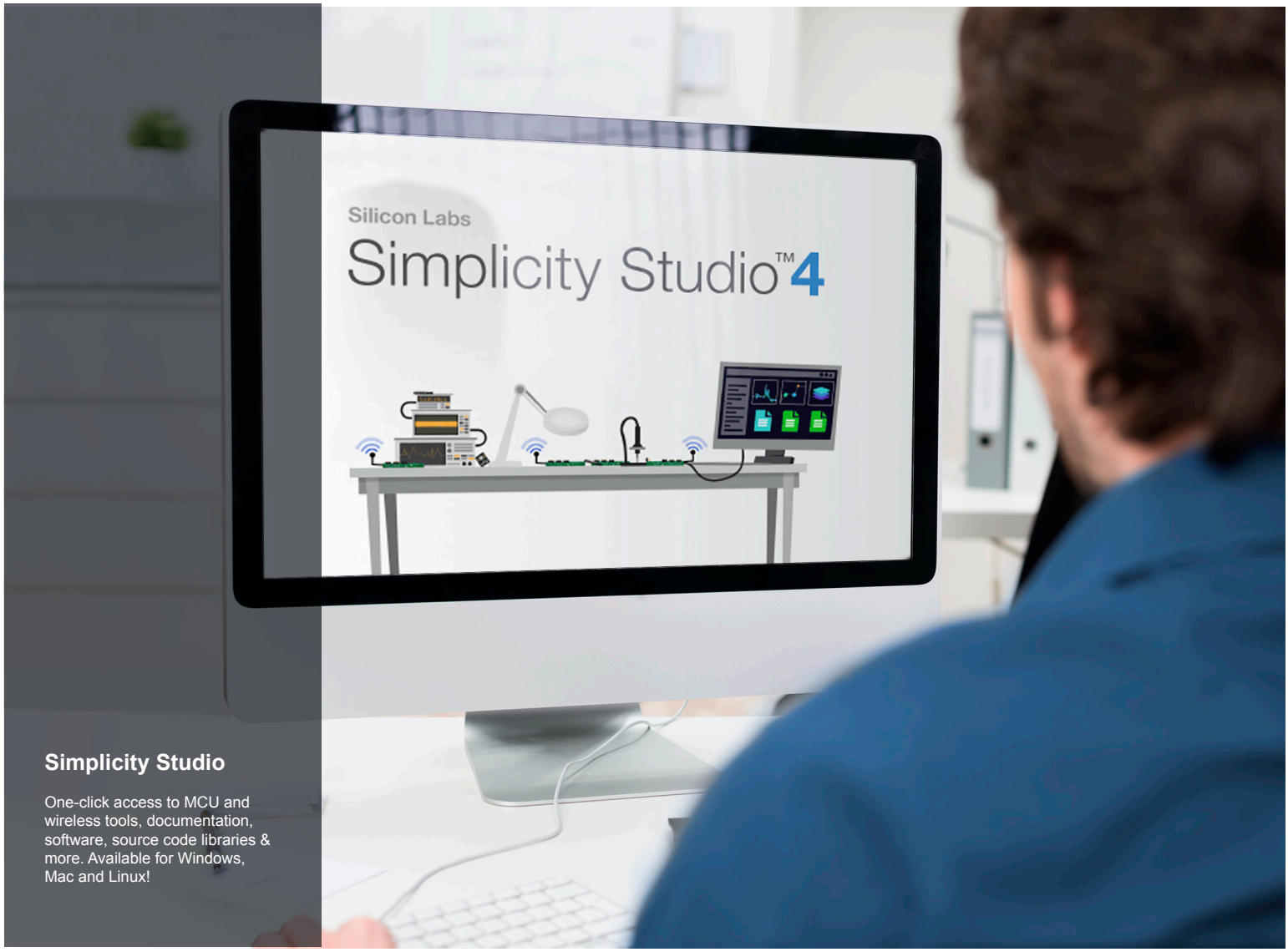
Illegal argument

Interface level error (0x020F)--WIFI

Interface error

Unknown host (0x0280)--WIFI

Unknown host



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com