

WF121 WI-FI SOFTWARE

API DOCUMENTATION

Thursday, 30 September 2021

Version 4.0



Table of Contents

1	Version History - WF121 SW API	4
2.1	Silicon Labs Wi-Fi Stack.....	7
2.2	BGAPI protocol.....	8
2.3	BGLib library.....	9
2.4	BGScript scripting language	10
3	Understanding Endpoints	11
3.1	Predefined Endpoints	12
4	API Definition.....	13
4.1	BGAPI protocol definition	13
4.1.1	Packet format	13
4.1.2	Message types	13
4.1.3	Command Class IDs	14
4.1.4	Packet Exchange	14
4.1.5	Introduction to BGAPI over SPI.....	15
4.2	BGLIB functions definition	16
4.3	BGScript API definition	17
4.4	Data types	18
5	API Reference	20
5.1	System	21
5.1.1	Commands--system	21
5.1.2	Events--system	27
5.2	Configuration	30
5.2.1	Commands--config.....	30
5.2.2	Events--config	33
5.3	Wi-Fi.....	34
5.3.1	Commands--SME.....	34
5.3.2	Events--SME	67
5.4	TCP stack.....	90
5.4.1	Commands--TCP/IP	90
5.4.2	Events--TCP/IP	121
5.5	Endpoint	133
5.5.1	Commands--endpoint.....	133
5.5.2	Events--endpoint	144
5.5.3	Enumerations--endpoint.....	149
5.6	Hardware.....	150
5.6.1	Commands--hardware	150
5.6.2	Enumerations--hardware.....	178
5.6.3	Events--hardware.....	179
5.7	I2C.....	184
5.7.1	Commands--I2C.....	184
5.8	Wired Ethernet	188

5.8.1	Commands--Ethernet.....	188
5.8.2	Events--Ethernet	192
5.9	HTTP Server	193
5.9.1	Commands--HTTPS.....	193
5.9.2	Events--HTTPS	198
5.10	Persistent Store.....	202
5.10.1	Commands--Flash.....	202
5.10.2	Enumerations--Flash.....	210
5.10.3	Events--Flash	212
5.11	Device Firmware Upgrade.....	215
5.11.1	Commands--DFU	215
5.11.2	Events--DFU	219
5.12	Utilities for BGScript	220
5.12.1	Commands--Util	220
5.13	SD card	222
5.13.1	Commands--SDHC	222
5.13.2	Events--SDHC.....	239
5.14	Error codes.....	243
5.14.1	BGAPI Errors	243
5.14.2	Hardware Errors.....	244
5.14.3	TCP/IP Errors.....	245

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 endpoints 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	<p>Wi-Fi Access point and HTTP server commands and event added and updated (see Wi-Fi and HTTP Server chapters)</p> <p>PS Key Change event added for monitoring PS key changes</p> <p>UDP Data event added for seeing the source of UDP data</p> <p>Event for handling invalid commands added</p>
1.3	Improved the documentation regarding how to use BGAPI over SPI
1.4	<p>Documentation updates for SW v1.2.1 compatibility</p> <p>Added/Changed APIs:</p> <ul style="list-style-type: none"> • 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 Syntax Error 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 <p>Editorial modifications:</p> <ul style="list-style-type: none"> • Possible events for commands are added • Error and return codes are updated
1.5	<p>Documentation updates for SW v1.2.2 compatibility</p> <p>Added/Changed APIs:</p> <ul style="list-style-type: none"> • 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

Version	
2.0	<p>Documentation updates for SW v1.3.0 compatibility</p> <p>Added/Changed APIs:</p> <ul style="list-style-type: none"> Set Max Power Saving State includes Deep Sleep Mode parameter EAP commands under WiFi category added for WPA enterprise functionality SSL/TLS commands and event under TCP Stack category added for enabling SSL/TLS communication X.509 command and event category introduced for managing the certificates and certificate storage UART configuration and enabling/disabling commands and event under Hardware category added Ethernet routing options extended for Set Dataroute command under Wired Ethernet category Multiple HTTP commands and events under HTTP server category for enabling more flexible HTTP server usage Low Voltage event under Flash category introduced for detecting an error when writing to Flash and low voltage supplied SD Card command and event category added for managing files on the external SD card BGScript utility command category added for enabling integer and string conversions with BGScript <p>Removed API's</p> <ul style="list-style-type: none"> Button and On Req events under HTTP server category removed.
2.1	Editorial changes
2.2	<p>Documentation updates for SW v1.3.0 compatibility</p> <p>Added/Changed APIs:</p> <ul style="list-style-type: none"> 802.11n mode command added under Wi-Fi category Access point visibility command added under Wi-Fi category A number of new error/reason codes added under Error codes category mDNS commands and events added under TCP/IP category DNS-SD commands and events added under TCP/IP category DHCP routing options command added under TCP/IP category <p>Removed API's</p> <ul style="list-style-type: none"> EAP commands under WiFi category added for WPA enterprise functionality SSL/TLS commands and event under TCP Stack category added for enabling SSL/TLS communication X.509 command and event category introduced for managing the certificates and certificate storage
3.0	<p>Documentation updates for SW v1.4</p> <p>Sections 4.2 and 4.4 updated. Message Class number corrected in sections 5.12 and 5.13. Other minor updates and corrections.</p> <p>Added APIs:</p> <ul style="list-style-type: none"> Client isolation support for AP mode added under Wi-Fi category Multicast support added under TCP/IP category Functionality to list connected clients added under TCP/IP category DHCP configuration functionality added under TCP/IP category

4.0

Documentation updates for SW v1.5.0

Added APIs:

Command `tcpip_set_tcp_client_port_range()`

Updated descriptions:

Event `endpoint_status()`

Command `hardware_uart_conf_set()`

2 Introduction to Silicon Labs Wi-Fi software

The Silicon Labs 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 Silicon Labs 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 as well as a simple and free software development kit 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 Silicon Labs 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.

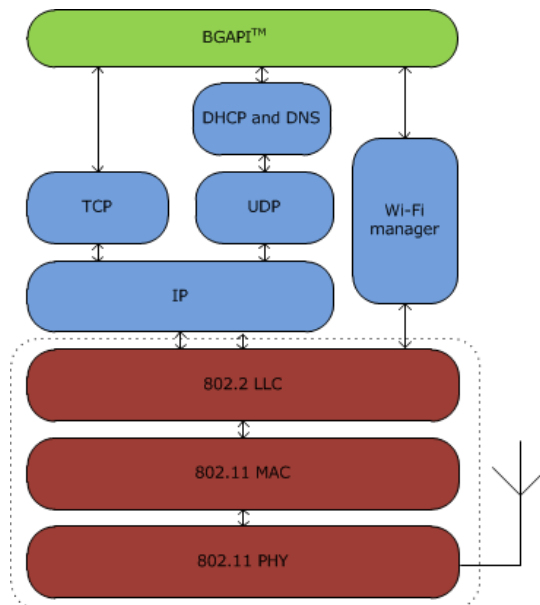


Figure: Silicon Labs Wi-Fi Software

2.2 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

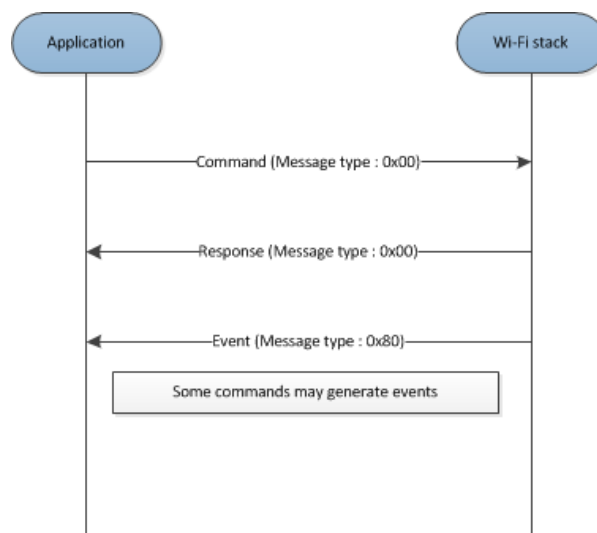


Figure: BGAPI messaging

2.3 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 Silicon Labs Wi-Fi Software Development Kit. The purpose is to simplify the application development to various host environments.

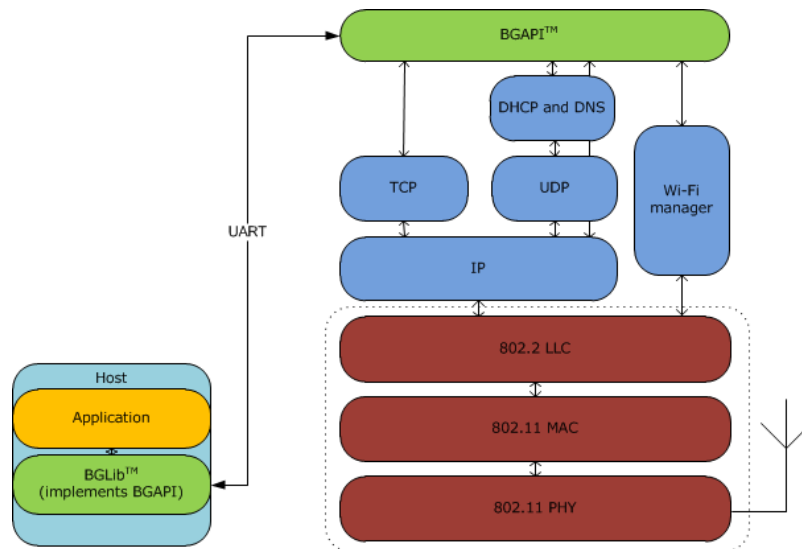


Figure: BGLib library

2.4 BGScript scripting language

Silicon Labs 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 Silicon Labs.

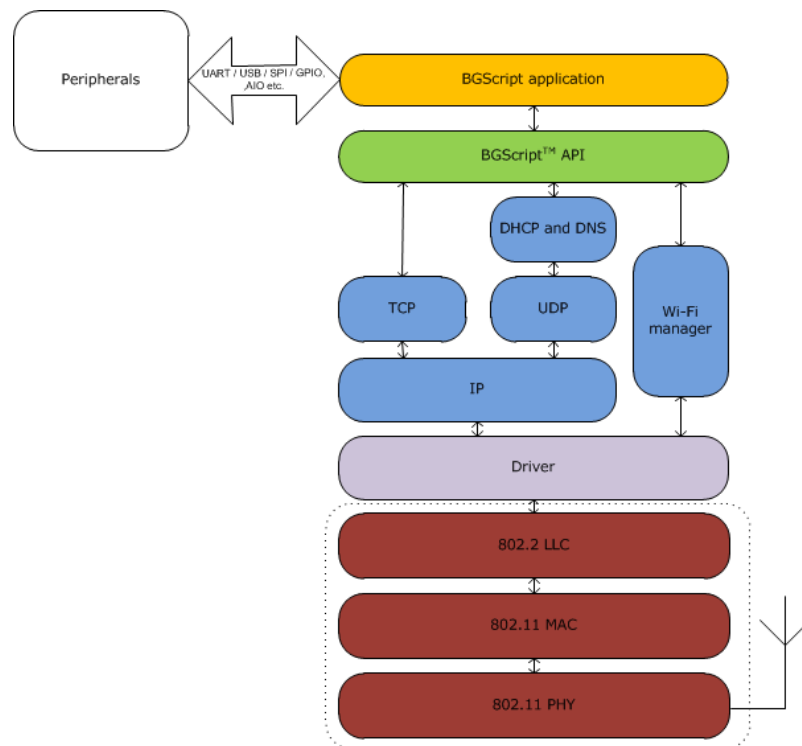


Figure: BGScript architecture

3 Understanding Endpoints

Endpoints play a crucial role in how data is handled and routed inside the Silicon Labs 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 "endpoints" for each of the peripherals. For example, to route the data in both directions between a serial port and a TCP socket, the serial port is configured to have the socket as its endpoint, and the socket is configured to have the serial port as its endpoint.

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 active 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.

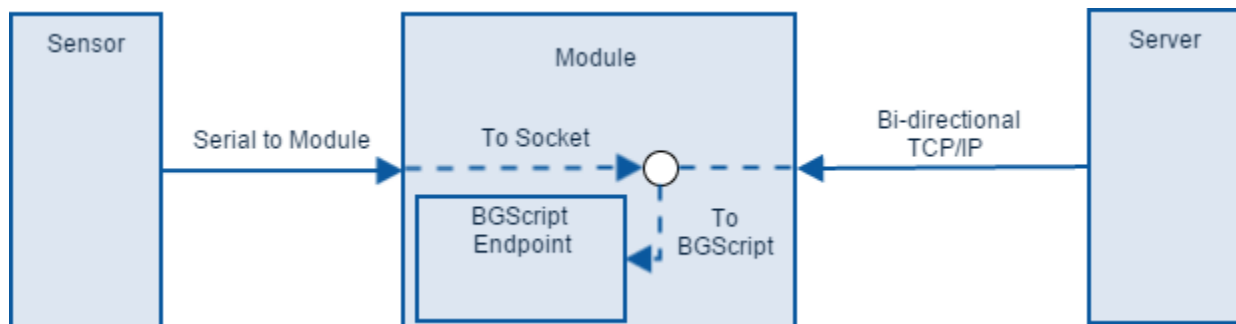


FIGURE: Example of routing a bi-directional stream between different endpoints. The serial interface has the socket configured as its endpoint, while the socket has the BGScript as its endpoint.

If an external micro-controller is used, the setup could look quite similar. In the figure below, the data from the server would get sent to the micro-controller, and the sensor would send its data straight to the Server.

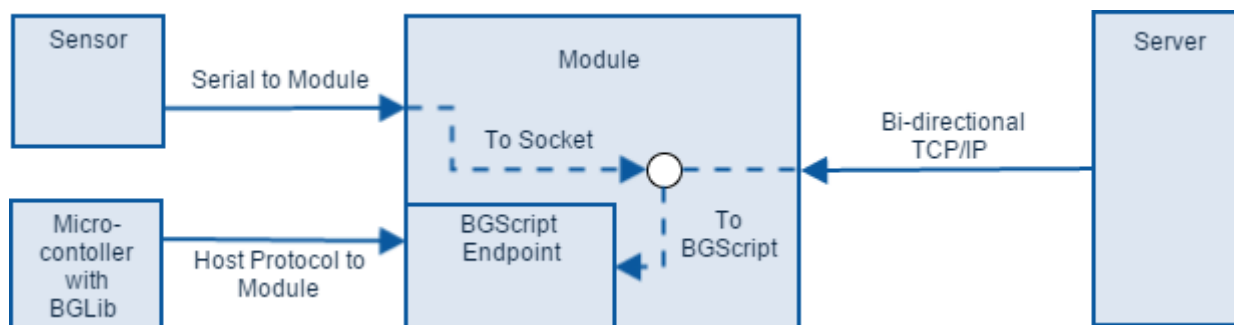


FIGURE: Example of routing a bi-directional stream between different endpoints.

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 to selectively prevent a BGScript application from catching events and therefore running any of its code.
3				USB	
4			I2C1		
5-30					Created at run-time 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

This section contains the generic Silicon Labs Wi-Fi software API definition. The definition consists of three

- parts: The BGAPI protocol definition
- The BGLib C library description
- The BGScript scripting API description

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

4.1 BGAPI protocol definition

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.

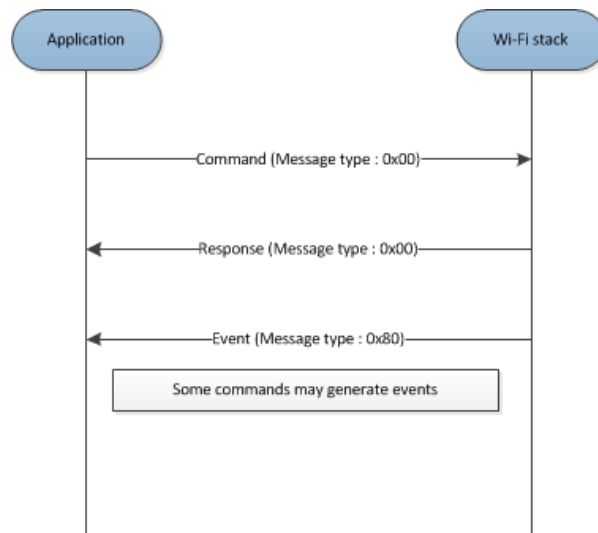


Figure: BGAPI messaging



BGAPI messaging

When sending commands over BGAPI you always have to wait for the BGAPI response before sending a new command to the module.

4.1.5 Introduction to BGAPI over SPI

When using SPI as the host interface to the Wi-Fi module, the host controller is the SPI Master and Wi-Fi module is the SPI Slave. SPI is a synchronous interface which means that the same clock drives both the input and the output. SPI Master sends BGAPI commands and at the same time reads possible events or responses from the SPI Slave.

Using BGAPI over SPI

The SPI Slave informs the master by using the notify pin (I/O 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 the SPI Slave is sending a packet, the SPI Master must clock enough zeros for the SPI Slave to be able to send a full packet. Then the SPI Master must wait for a response packet for the command that was sent before sending a new command packet.

Using notify pin

Notify pin must be connected to a GPIO on the host, which is used as an interrupt. This is because the notification sent using this pin can be relatively short and if the host just polls the GPIO status it could miss the notification and a BGAPI message. Also, the host application should keep count on the interrupts received via the notification pin and read the BGAPI messages over the SPI until the notification counter reaches zero.

Examples

Command & Response processing

1. SPI Master sends I/O port read command to the SPI Slave. SPI Slave responds by sending back 0's.

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

2. SPI Slave notifies the SPI Master with the notify pin (assuming that the pin has been configured properly) that it has data to send.

3. SPI master detects an IO interrupt and increases a notify counter value by one (1)

4. SPI Master reads the response from the SPI Slave.

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

5. SPI Master reduces the value of notify counter by one (1)

Event processing

1. SPI Slave notifies the SPI Master with the notify pin that it has data to send.

2. SPI master detects an IO interrupt and increases a notify counter value by one (1)

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

4. SPI Master reduces the value of notify counter by one (1)

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

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.

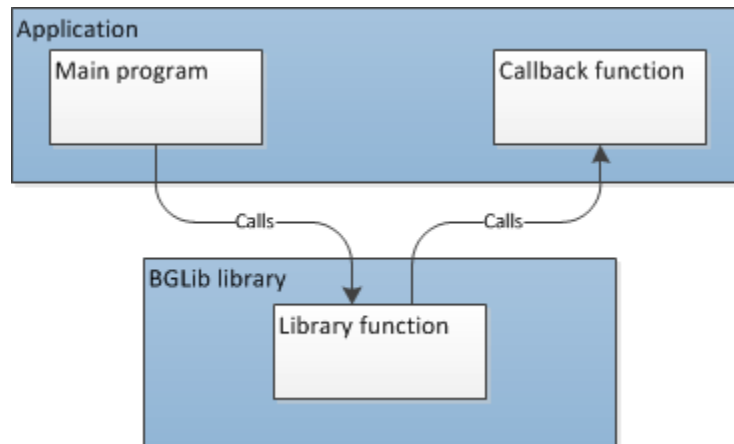


Figure: Callback architecture

Optional way to use BGLib

The optional way to use BGLib is to utilize predefined macros for commands and user defined functions for response and event handling. Usage of BGLib, macros and data structures are defined in *wifi_bglib.h* file in SDK's *api* -directory. Detailed usage of BGLib can be discovered from *tcp_example* in SDK's *src_example* directory.

4.3 BGScript API definition

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

Data types used in the documentation is shown in the table below.

Table: Data types used in the documentation

Name	Length (bytes)	Description	Example
hw_addr	6	MAC Address	Human readable 00:07:80:1A:2B:3C Packet data (hex) 00 07 80 1A 2B 3C
ipv4	4	IPv4 address in big endian format	Human readable 192.168.1.1 Packet data (hex) C0 A8 01 01
uint8array	1-256	Variable length byte array. The first byte defines the length n of the data that follows, n having the value 0 n 255.	Human readable "Hello" Packet data (hex) 05 48 65 6c 6c 6f
uint16array	2-2047	Variable length byte array. The first two bytes define the length n of the payload data that follows, n having the value 0 n 2045. Length field is in little endian format. Note: The payload data length is in bytes (not in multiples of 2)! Note: The upper limit 2045 is the theoretical maximum value for n because the actual maximum value depends on the other data in the message array (excluding the message header).	Human readable "World!" Packet data (hex) 06 00 57 6f 72 6c 64 21
uint8	1	8-bit unsigned integer	Human readable 167 Packet data (hex) a7
int8	1	8-bit signed integer, 2's complement	Human readable -22 Packet data (hex) ea
uint16	2	16-bit unsigned integer	Human readable 4567 Packet data (hex) d7 11
int16	2	16-bit signed integer, 2's complement	Human readable -4567 Packet data (hex) 29 ee
uint32	4	32-bit unsigned integer	Human readable 2864434397 Packet data (hex) dd cc bb aa
int32	4	32-bit signed integer, 2's complement	Human readable -45678 Packet data (hex) 92 4d ff ff

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

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

Description	Explanation
System	This class contains commands related to various system functions <i>Examples:</i> Wi-Fi module system reset, power saving state selection
Configuration	This class contains commands related to device parameter configuration <i>Examples:</i> Read MAC address, set MAC address
Wi-Fi	This class contains commands related to 802.11 MAC <i>Examples:</i> Wi-Fi on, Wi-Fi off, start scan, stop scan
TCP stack	This class contains commands related to TCP/IP stack and various protocols such as TCP and UDP and IP address setup <i>Examples:</i> Configure local IP address, netmask, and gateway, configure DNS client settings
Endpoint	This class contains commands related to data endpoint control including creation and deletion of endpoints and data routing <i>Examples:</i> Activate or deactivate an endpoint, set desired transmit packet size
Hardware	This class contains commands related to hardware peripherals and interfaces <i>Examples:</i> Read Wi-Fi module internal AD converter, I/O port read and write
I2C	This class contains commands related to I2C peripherals <i>Example:</i> Start read, stop read
Wired Ethernet	This class contains commands related to the RMII (Ethernet) interface of the Wi-Fi module <i>Examples:</i> Test wired Ethernet connection, close wired Ethernet connection
HTTP Server	This class contains commands related to Web server activation and management <i>Example:</i> Add mapping between an HTTP server URL and storage device
Persistent store (PS)	This class contains commands related to non-volatile memory <i>Examples:</i> Store a value into defined PS key, defragment persistent store
Device Firmware Upgrade	This class contains commands related to firmware update <i>Examples:</i> Reset system into DFU mode, upload firmware update file into Wi-Fi module
Utilities for BGScript	This class contains commands related to BGScript programming language utilities <i>Example:</i> Convert decimal value from ASCII string to a 32-bit signed integer
SD card	This class contains commands related to the Wi-Fi module SD memory card interface <i>Example:</i> Close file, delete file

The final section of the API reference contains descriptions of the error codes which are categorized as follows:

Description
BGAPI errors
TCPIP errors
Hardware errors

5.1 System

This class contains commands related to various system functions.

5.1.1 Commands--system

These commands are related to the system.

Sync--system

This command can be used to synchronize the system state. When the sync command is sent, multiple events are output representing the system status. This command 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	hlen	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	hlen	Message type: command
1	0x00	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x00	method	Message ID

Table: EVENTS

event	Description
sme_wifi_is_on	Sent if Wi-Fi has been switched on
sme_wifi_off	Sent if Wi-Fi has been switched off
sme_scan_result	Sent for each cached scan result
sme_connected	Device connection status
tcpip_endpoint_status	Sent for each TCP/IP endpoint
endpoint_status	Sent for each endpoint
config_mac_address	Device MAC address
tcpip_configuration	Device TCP/IP configuration
tcpip_dns_configuration	Device DNS configuration

C Functions

```
/* Function */  
void wifi_cmd_system_sync(  
    void  
);  
  
/* Callback */  
void wifi_rsp_system_sync(  
    const void *nul  
)
```

BGScript Functions

```
call system_sync()
```

Reset--system

This command can be used to reset the Wi-Fi module. This command does not have a response, but it triggers the [Boot](#) event.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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 into normal mode
dfu_boot	Sent after the device has booted into DFU mode

C Functions

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

BGScript Functions

```
call system_reset(dfu)
```

Hello--system

This command can be used to check whether communication between the Wi-Fi software and hardware functions.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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()
```


Set Max Power Saving State--system

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



Mode 0: No power saving is in use. Use this mode for the lowest latency and best performance.

Mode 1: The Wi-Fi radio is allowed to sleep and it will automatically go to sleep after 6000 ms of inactivity.

Mode 2: Both MCU and Wi-Fi radio are allowed to go to sleep after an inactivity timeout defined in the hardware configuration file. The module wakes up automatically every eight (8) seconds to check for scheduled tasks and it also generates [Power Saving State](#) event to notify the host of the scheduled wake up.

If the **<sleep>** configuration is not used in the hardware configuration file then only states 0 and 1 are possible.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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 external interrupt to wake up Default: 2

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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)
```

5.1.2 Events--system

These events are related to the system.

Boot--system

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

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

SW Exception--system

This event indicates that a software exception has occurred.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

Power Saving State--system

This event indicates the power saving state into which 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)
```

5.2 Configuration

This class contains commands related to device parameter configuration.

5.2.1 Commands--config

These commands are related to configuring the device.

Get MAC--config--WIFI

This command can be used to read the IEEE address of the device.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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

This command can be used to write an IEEE address into the device.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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

These events are related to configuration of the device.

MAC Address--config

This event indicates the current MAC address of the device.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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

This class contains commands related to 802.11 MAC.

5.3.1 Commands--SME

These commands are related to Wi-Fi.

Wi-Fi On--SME

This command can be used to turn on the 802.11 radio.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

Wi-Fi Off--SME

This command can be used to turn off the 802.11 radio.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

Start Scan--SME

This command initiates a scan for Access Points. Scanning is not possible once connected.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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

This command can be used to terminate the active scanning procedure.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

Set Password--SME

This command can be used to set the network password used when authenticating with an Access Point.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x05	method	Message ID
4	uint8array	password	WEP/WPA/WPA2 pre-shared password to be used when connecting to an Access Point. WPA/WPA2-PSK is either a hash of 64 hexadecimal characters, or a passphrase of 8 to 63 ASCII-encoded characters. 64-bit WEP key is either 5 ASCII-encoded characters or 10 HEX characters. 128-bit WEP key is either 13 ASCII-encoded characters or 26 HEX characters. Out of the two methods of authentication that can be used by WEP only the Open System authentication is supported.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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 BSSID--SME

This command can be used to try to connect to a specific Access Point using its unique BSSID. In order to succeed, this command requires a preceding [sme_start_scan](#) command and that the desired wireless network was found during that scan. If the Access Point is using channel 12 or 13, for the connection to be successful at least one of the Access Points found within radio coverage range must advertise the use of channels up to 13.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

Connect SSID--SME

This command can be used to start a connection establishment procedure with an Access Point with the given SSID. This command supports both visible and hidden SSIDs.

Executing this command will also launch a transparent scan procedure in order to discover the Access Points in range, 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 has not been executed all channels (1 to 13) will be scanned. If the Access Point is using channel 12 or 13, for the connection to be successful, at least one of the Access Points found in range must advertise the use of channels up to 13.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

Disconnect--SME

This command can be used to disconnect from the currently connected Access Point.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

Set Scan Channels--SME

This command can be used to set the default scan channel list for commands [Start Scan](#) and [Connect Ssid](#).

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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 (1 to 13) 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 example below).

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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")
```

Set Operating Mode--SME

This command can be used to set the Wi-Fi operating mode either to Wi-Fi client (STA) or Wi-Fi Access Point (AP). The selected operating mode will become effective after the next time the radio is turned on by launching the command `sme_wifi_on`.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```


Start AP Mode--SME

This command can be used to start the Wi-Fi Access Point mode.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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. Possible channels are: 1...11 (default) 1...13 (only after a scan is performed beforehand and only if at least one access point found in range is advertising channel use up to 13)
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. Maximum length of the SSID is 32 characters.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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

This command can be used to stop the Wi-Fi Access Point mode.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

Scan Results Sort RSSI--SME

This command can be used to resend scan results of a previous scan, sorted according to RSSI value. This command can be run only after the command `sme_start_scan` has been issued at least once during the current session.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

AP Client Disconnect--SME

This command can be used to disconnect a station from the network (from an Access Point).

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

Set AP Password--SME

This command can be used to set the Wi-Fi password for an Access Point.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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 WPA/WPA2-PSK is either a string of 64 hexadecimal characters or a passphrase of 8 to 63 ASCII-encoded characters. 64-bit WEP key is either 5 ASCII-encoded characters or 10 HEX characters. 128-bit WEP key is either 13 ASCII-encoded characters or 26 HEX characters.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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)
```

Set AP Max Clients--SME

This command can be used to 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	hlen	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 allowed stations, from 0 to 5. Default is 5.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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)
```

Start WPS--SME

This command can be used to start the Wi-Fi Protected Setup (WPS) session. Only WPS PUSH mode for the Wi-Fi client side is available.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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--SME

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

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

Get Signal Quality--SME

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

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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 SSID Scan--SME

This command can be used to initiate an active scan for Access Points. Scanning is not possible once connected.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x14	method	Message ID
4	uint8array	ssid	The SSID to scan for. Zero-length if all SSID's are to be scanned.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x14	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_ssid_scan(
    uint8 ssid_len,
    const uint8* ssid_data
);

/* Callback */
struct wifi_msg_sme_start_ssid_scan_rsp_t{
    uint16 result
}
void wifi_rsp_sme_start_ssid_scan(
    const struct wifi_msg_sme_start_ssid_scan_rsp_t * msg
)
```

BGScript Functions

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

Set AP Hidden--SME

This command can be used to set whether the Access Point is hidden or visible. The Access Point is set visible by default.

Table: COMMAND

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

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x15	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_hidden(
    uint8 hidden
);

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

BGScript Functions

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


Set 11n Mode--SME

This command can be used to select whether 802.11n mode is enabled or disabled. The mode is enabled by default.

Table: COMMAND

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

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x16	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_11n_mode(
    uint8 mode
);

/* Callback */
struct wifi_msg_sme_set_11n_mode_rsp_t{
    uint16 result,
    uint8 hw_interface
}
void wifi_rsp_sme_set_11n_mode(
    const struct wifi_msg_sme_set_11n_mode_rsp_t * msg
)
```

BGScript Functions

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

Set AP Client Isolation--SME

This command can be used to isolate clients from each other in Access Point mode. Multiple clients can be connected to the AP and communicate with the AP but not with each other. The isolation is disabled by default. Note that in this mode no multicast traffic is re-transmitted by the AP.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x17	method	Message ID
4	uint8	isolation	AP client isolation 0 : disabled 1 : enabled

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x17	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_client_isolation(
    uint8 isolation
);

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

BGScript Functions

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

5.3.2 Events--SME

This events are related to Wi-Fi.

Wi-Fi is On--SME

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

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

Wi-Fi is Off--SME

This event indicates that 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, in which case the user application must re-initialize the radio by turning it on.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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

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

This 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	hlen	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

This event indicates the Access Point scan is finished.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

Connected--SME

This event indicates a successful connection to an Access Point.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```


Disconnected--SME

This event indicates a disconnection from an Access Point. The event occurs either because [sme_disconnect](#) command was issued or connection to an Access Point was lost. Connection loss could occur because the Access Point has been switched off or the user has moved out of coverage. The timeout for detecting a lost connection is 50 beacons which under typical network configuration translates to roughly 5 seconds.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

Interface Status--SME

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

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

Connect Failed--SME

This event 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	hlen	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)
```

Connect Retry--SME

This event indicates that a connection attempt failed, which will eventually lead to an automatic retry. This event appears typically when the module is commanded to connect to a wireless network but the given password is wrong. The amount of retries is fixed to 10 and the retries can be stopped with the command `wifi_cmd_sme_disconnect`.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

AP Mode Started--SME

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

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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

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

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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

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

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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

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

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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

This event indicates that a Wi-Fi client (client), has left the Access Point.

In case a station moves out of range or is abruptly powered off, it might take from 180 to 270 seconds for this event to be issued at the module operating as the Access Point. This timeout range is due to the fact that the remote station is not sending any message to indicate that it is going to disconnect, while at the module side the conditions to declare that a client has left are the following: considering slots of 90 seconds from the moment the client connected, if in one slot there has been no message from the client, then the next slot is used to send empty frames to the client (at the interval of 15 seconds) and if there is still no response from the client then the connection is considered down, at which time this event is generated.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

Scan Sort Result--SME

This event indicates the scan result.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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

This event indicates that the scan result sort is finished.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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()
```

WPS Stopped--SME

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

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

WPS Completed--SME

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

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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--SME

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

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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 Credential SSID--SME

This event indicates the SSID of the network in relation to Wi-Fi Protected Setup (WPS).

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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--SME

This event indicates the password of then network in relation to Wi-Fi Protected Setup (WPS).

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	Message type: event
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x15	method	Message ID
4	uint8	hw_interface	Hardware interface 0 : Wi-Fi
5	uint8array	password	Password of the network. WPA/WPA2-PSK is either a string of 64 hexadecimal characters or a passphrase of 8 to 63 ASCII-encoded characters. 64-bit WEP key is either 5 ASCII-encoded characters or 10 HEX characters. 128-bit WEP key is either 13 ASCII-encoded characters or 26 HEX characters.

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)
```


Signal Quality--SME

This event indicates the signal quality (RSSI value) of the connection in dBm units.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

5.4 TCP stack

This class contains commands related to TCP/IP stack and various protocols such as TCP and UDP and IP address setup.

5.4.1 Commands--TCP/IP

These commands are related to the TCP stack.

Start TCP Server--TCP/IP

This command can be used to start 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	hlen	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	hlen	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

event	Description
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)

```

TCP Connect--TCP/IP

This command can be used to attempt the creation of a new TCP socket to a TCP server.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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 UDP Server--TCP/IP

This command can be used to start an UDP server.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

UDP Connect--TCP/IP

This command can be used to attempt the creation of a new UDP connection.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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. Notice that in the current firmware there cannot be data coming from the endpoint assigned to this UDP connection, due to the connectionless nature of the UDP protocol. So, any index can be used here, and no practical effect should be expected.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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 assigned to the newly created UDP connection. Use this index for example to send data to the remote UDP server using the command endpoint_send

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)
```

Configure--TCP/IP

This command can be used to 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 a DHCP Client the settings for the static IP will be stored, but they will be overridden as soon as the remote DHCP Server will assign its IP configuration to the module.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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 used by the device when in station (client) mode. In access point mode this is not relevant, because the local DHCP server will automatically use the local IP address above as the gateway to pass to remote clients.
16	uint8	use_dhcp	Use DHCP 0 : Use static IP settings 1 : DHCP Client is used to obtain the dynamic IP configuration

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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)
```

DNS Configure--TCP/IP

This command can be used to configure DNS client settings.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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--TCP/IP

This command can be used to start a procedure to resolve the hostname related to an IP address using the configured DNS servers.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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.Silicon Labs.com.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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)
```

UDP Bind--TCP/IP

In case an UDP endpoint exists, this command can be used to change the currently used local source port (which is otherwise pseudo-randomly generated by the firmware) to a desired specific source port. Use this command after the command `tcpip_udp_connect` is issued and the UDP endpoint assigned.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

DHCP Set Hostname--TCP/IP

This command can be used to set the DHCP host name parameter (option 12) used in client DHCPDISCOVER and DHCPREQUEST messages.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x08	method	Message ID
4	uint8array	hostname	Host name

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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)
```


DHCP Enable Routing--TCP/IP

This command can be used to enable or disable gateway and DNS router options in DHCP server offer and ack. Options are enabled by default.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x09	method	Message ID
4	uint8	enable	Routing options 0 : disabled 1 : enabled

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
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_tcpip_dhcp_enable_routing(
    uint8 enable
);

/* Callback */
struct wifi_msg_tcpip_dhcp_enable_routing_rsp_t{
    uint16 result
}
void wifi_rsp_tcpip_dhcp_enable_routing(
    const struct wifi_msg_tcpip_dhcp_enable_routing_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_dhcp_enable_routing(enable) (result)
```

mDNS Set Hostname--TCP/IP

This command can be used to set the mDNS hostname. mDNS service cannot be started until the hostname is set. The maximum length of hostname is 63 bytes.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0A	method	Message ID
4	uint8array	hostname	Multicast DNS hostname. The top-level domain .local is added automatically.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
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_tcpip_mdns_set_hostname(
    uint8 hostname_len,
    const uint8* hostname_data
);

/* Callback */
struct wifi_msg_tcpip_mdns_set_hostname_rsp_t{
    uint16 result
}
void wifi_rsp_tcpip_mdns_set_hostname(
    const struct wifi_msg_tcpip_mdns_set_hostname_rsp_t * msg
)
```

BGScript Functions

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

mDNS Start--TCP/IP

This command can be used to start the mDNS service.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x00	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0B	method	Message ID

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
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

Table: EVENTS

event	Description
tcpip_mdns_started	Sent when mDNS service is successfully started
tcpip_mdns_failed	Sent when mDNS service startup fails

C Functions

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

/* Callback */
struct wifi_msg_tcpip_mdns_start_rsp_t{
    uint16 result
}
void wifi_rsp_tcpip_mdns_start(
    const struct wifi_msg_tcpip_mdns_start_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_mdns_start() (result)
```

mDNS Stop--TCP/IP

This command can be used to stop a mDNS service.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x00	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0C	method	Message ID

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
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

C Functions

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

/* Callback */
struct wifi_msg_tcpip_mdns_stop_rsp_t{
    uint16 result
}
void wifi_rsp_tcpip_mdns_stop(
    const struct wifi_msg_tcpip_mdns_stop_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_mdns_stop() (result)
```

DNS-SD Add Service--TCP/IP

This command can be used to add a new DNS-SD service. The maximum length of the service name is 15 bytes.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x04	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0D	method	Message ID
4 - 5	uint16	port	Service port
6	uint8	protocol	Service protocol 0 : TCP 1 : UDP
7	uint8array	service	Service name

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
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
6	uint8	index	Service index

C Functions

```
/* Function */
void wifi_cmd_tcpip_dnssd_add_service(
    uint16 port,
    uint8 protocol,
    uint8 service_len,
    const uint8* service_data
);

/* Callback */
struct wifi_msg_tcpip_dnssd_add_service_rsp_t{
    uint16 result,
    uint8 index
}
void wifi_rsp_tcpip_dnssd_add_service(
    const struct wifi_msg_tcpip_dnssd_add_service_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_dnssd_add_service(port, protocol, service_len, service_data)(result, index)
```

DNS-SD Add Service Instance--TCP/IP

This command can be used to add a DNS-SD service instance name. The maximum length of the service instance name is 63 bytes. The DNS-SD service cannot be started until the instance name is set.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0E	method	Message ID
4	uint8	index	Service index
5	uint8array	instance	Service instance name

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
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

C Functions

```
/* Function */
void wifi_cmd_tcpip_dnssd_add_service_instance(
    uint8 index,
    uint8 instance_len,
    const uint8* instance_data
);

/* Callback */
struct wifi_msg_tcpip_dnssd_add_service_instance_rsp_t{
    uint16 result
}
void wifi_rsp_tcpip_dnssd_add_service_instance(
    const struct wifi_msg_tcpip_dnssd_add_service_instance_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_dnssd_add_service_instance(index, instance_len, instance_data)(result)
```

DNS-SD Add Service Attribute--TCP/IP

This command can be used to add a DNS-SD service attribute. The maximum length of the service attribute is 63 bytes.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0F	method	Message ID
4	uint8	index	Service index
5	uint8array	attribute	Service attribute

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0F	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_dnssd_add_service_attribute(
    uint8 index,
    uint8 attribute_len,
    const uint8* attribute_data
);

/* Callback */
struct wifi_msg_tcpip_dnssd_add_service_attribute_rsp_t{
    uint16 result
}
void wifi_rsp_tcpip_dnssd_add_service_attribute(
    const struct wifi_msg_tcpip_dnssd_add_service_attribute_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_dnssd_add_service_attribute(index, attribute_len, attribute_data)(result)
```


DNS-SD Remove Service--TCP/IP

This command can be used to remove a DNS-SD service.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x10	method	Message ID
4	uint8	index	Service index

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
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

C Functions

```
/* Function */
void wifi_cmd_tcpip_dnssd_remove_service(
    uint8 index
);

/* Callback */
struct wifi_msg_tcpip_dnssd_remove_service_rsp_t{
    uint16 result
}
void wifi_rsp_tcpip_dnssd_remove_service(
    const struct wifi_msg_tcpip_dnssd_remove_service_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_dnssd_remove_service(index) (result)
```

DNS-SD Start Service--TCP/IP

This command can be used to start a DNS-SD service.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x11	method	Message ID
4	uint8	index	Service index

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
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

Table: EVENTS

event	Description
tcpip_dnssd_service_started	Sent when a DNS-SD service is successfully started
tcpip_dnssd_service_failed	Sent when a DNS-SD service startup fails

C Functions

```
/* Function */
void wifi_cmd_tcpip_dnssd_start_service(
    uint8 index
);

/* Callback */
struct wifi_msg_tcpip_dnssd_start_service_rsp_t{
    uint16 result
}
void wifi_rsp_tcpip_dnssd_start_service(
    const struct wifi_msg_tcpip_dnssd_start_service_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_dnssd_start_service(index) (result)
```

DNS-SD Stop Service--TCP/IP

This command can be used to stop a DNS-SD service.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x12	method	Message ID
4	uint8	index	Service index

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
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

C Functions

```
/* Function */
void wifi_cmd_tcpip_dnssd_stop_service(
    uint8 index
);

/* Callback */
struct wifi_msg_tcpip_dnssd_stop_service_rsp_t{
    uint16 result
}
void wifi_rsp_tcpip_dnssd_stop_service(
    const struct wifi_msg_tcpip_dnssd_stop_service_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_dnssd_stop_service(index) (result)
```

Multicast Join--TCP/IP

This command is used to join a multicast group.

Maximum number of multicast groups that can be joined is 4. Use 224.0.0.2 - 224.0.0.254 as address range. Note that 224.0.0.1 is automatically joined.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x04	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x13	method	Message ID
4-7	ipv4	address	IP address of the multicast group

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
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

Table: EVENTS

C Functions
<pre>/* Function */ void wifi_cmd_tcpip_multicast_join(ipv4 address); /* Callback */ struct wifi_msg_tcpip_multicast_join_rsp_t{ uint16 result } void wifi_rsp_tcpip_multicast_join(const struct wifi_msg_multicast_join_rsp_t * msg)</pre>

Multicast Leave--TCP/IP

This command is used to leave a multicast group.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x04	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x14	method	Message ID
4-7	ipv4	address	IP address of the multicast group

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x14	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

C Functions
<pre> /* Function */ void wifi_cmd_tcpip_multicast_leave(ipv4 address); /* Callback */ struct wifi_msg_tcpip_multicast_leave_rsp_t{ uint16 result } void wifi_rsp_tcpip_multicast_leave(const struct wifi_msg_multicast_leave_rsp_t * msg) </pre>

DHCP Configure--TCP/IP

This command can be used to configure DHCP Server subnetwork mask and address lease time.

Values are saved in PS-keys called FLASH_PS_KEY_DHCPSPACE, FLASH_PS_KEY_DHCPMASK and FLASH_PS_KEY_DHCPLEASETIME and default values are 192.168.1.2, 255.255.255.0 and 86400 seconds. Parameters are taken in use on DHCP server startup.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x0c	lolen	Minimum payload length

Byte	Type	Name	Description
2	0x04	class	Message class: TCP stack
3	0x15	method	Message ID
4-7	ipv4	address	First IPv4 address of address leasing pool
8-11	ipv4	subnet_mask	Subnetwork mask
12-15	uint32	lease_time	DHCP address lease timeout in seconds

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x015	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_dhcp_configuration	This event contains DHCP Server configuration

C Functions

```

/* Function */
void wifi_cmd_tcpip_dhcp_configure(
    ipv4 address,
    ipv4 subnet_mask,
    uint32 lease_time
);

/* Callback */
struct wifi_msg_tcpip_dhcp_configure_rsp_t{
    uint16 result
}
void wifi_rsp_tcpip_dhcp_configure(
    const struct wifi_msg_dhcp_configure_rsp_t * msg
)

```

BGScript Functions

```

call tcpip_dhcp_configure(address, subnet_mask, lease_time)(result)

```

DHCP Clients--TCP/IP

This command can be used to get IPv4 address and MAC address of each client connected to WF121 access point.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x00	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x16	method	Message ID

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x16	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	client_cnt	Count of connected clients

Table: EVENTS

event	Description
tcpip_dhcp_client	This event contains a client information. Sent for each client.

C Functions

```

/* Function */
void wifi_cmd_tcpip_dhcp_clients(
    void
);

/* Callback */
struct wifi_msg_tcpip_dhcp_clients_rsp_t{
    uint16 result,
    uint8  client_cnt
}
void wifi_rsp_tcpip_dhcp_clients(
    const struct wifi_msg_dhcp_clients_rsp_t * msg
)

```

BGScript Functions

```

call tcpip_dhcp_clients() (result,client_cnt)

```

Set TCP Client Port Range--TCP/IP

This command is used to set range for TCP client random port generation

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x04	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x17	method	Message ID
4-5	uint16	tcp_local_port_range_start	Start of local port range (min. 0xC000). Default: 0xC000
6-7	uint16	tcp_local_port_range_end	End of local port range (max. 0xFFFF). Default: 0xFFFF

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x17	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_set_tcp_client_port_range (
    uint16 tcp_local_port_range_start,
    uint16 tcp_local_port_range_end
);

/* Callback */
struct wifi_msg_tcpip_set_tcp_client_port_range_rsp_t{
    uint16 result
}
void wifi_rsp_tcpip_set_tcp_client_port_range(
    const struct wifi_msg_set_tcp_client_port_range_rsp_t * msg
)
```

BGScript Functions

```
call tcpip_set_tcp_client_port_range(tcp_local_port_range_start, tcp_local_port_range_end) (result)
```


5.4.2 Events--TCP/IP

These events are related to the TCP stack.

Configuration--TCP/IP

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--TCP/IP

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	hlen	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)
```

Endpoint Status--TCP/IP

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

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

DNS Gethostbyname Result--TCP/IP

This 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	hlen	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)
```

UDP Data--TCP/IP

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)
```

mDNS Started--TCP/IP

This event indicates that a mDNS service has been successfully started.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	Message type: event
1	0x00	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x05	method	Message ID

C Functions

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

BGScript Functions

```
event tcpip_mdns_started()
```

mDNS Failed--TCP/IP

This event indicates that a mDNS service has failed.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	Message type: event
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x06	method	Message ID
4 - 5	uint16	reason	Failure reason For values refer to the error code documentation

C Functions

```
/* Callback */
struct wifi_msg_tcpip_mdns_failed_evt_t{
    uint16 reason
}
void wifi_evt_tcpip_mdns_failed(
    const struct wifi_msg_tcpip_mdns_failed_evt_t * msg
)
```

BGScript Functions

```
event tcpip_mdns_failed(reason)
```

mDNS Stopped--TCP/IP

This event indicates that a mDNS service has been stopped.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	Message type: event
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x07	method	Message ID
4 - 5	uint16	reason	Stop reason For values refer to the error code documentation

C Functions

```
/* Callback */
struct wifi_msg_tcpip_mdns_stopped_evt_t{
    uint16 reason
}
void wifi_evt_tcpip_mdns_stopped(
    const struct wifi_msg_tcpip_mdns_stopped_evt_t * msg
)
```

BGScript Functions

```
event tcpip_mdns_stopped(reason)
```


DNS-SD Service Started--TCP/IP

This event indicates that a DNS-SD service has been successfully started.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hilen	Message type: event
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x08	method	Message ID
4	uint8	index	Service index

C Functions

```
/* Callback */
struct wifi_msg_tcpip_dnssd_service_started_evt_t{
    uint8 index
}
void wifi_evt_tcpip_dnssd_service_started(
    const struct wifi_msg_tcpip_dnssd_service_started_evt_t * msg
)
```

BGScript Functions

```
event tcpip_dnssd_service_started(index)
```

DNS-SD Service Failed--TCP/IP

This event indicates that a DNS-SD service has failed.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hilen	Message type: event
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x09	method	Message ID
4 - 5	uint16	reason	Failure reason For values refer to the error code documentation
6	uint8	index	Service index

C Functions

```
/* Callback */
struct wifi_msg_tcpip_dnssd_service_failed_evt_t{
    uint16 reason,
    uint8 index
}
void wifi_evt_tcpip_dnssd_service_failed(
    const struct wifi_msg_tcpip_dnssd_service_failed_evt_t * msg
)
```

BGScript Functions

```
event tcpip_dnssd_service_failed(reason, index)
```

DNS-SD Service Stopped--TCP/IP

This event indicates that a DNS-SD service has been stopped.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hilen	Message type: event
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0A	method	Message ID
4 - 5	uint16	reason	Stop reason For values refer to the error code documentation
6	uint8	index	Service index

C Functions

```
/* Callback */
struct wifi_msg_tcpip_dnssd_service_stopped_evt_t{
    uint16 reason,
    uint8 index
}
void wifi_evt_tcpip_dnssd_service_stopped(
    const struct wifi_msg_tcpip_dnssd_service_stopped_evt_t * msg
)
```

BGScript Functions

```
event tcpip_dnssd_service_stopped(reason, index)
```

DHCP Configuration--TCP/IP

This event contains DHCP Server configuration.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	Message type: event
1	0x0D	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x0B	method	Message ID
4	uint8	routing_enabled	DHCP server routing enabled <ul style="list-style-type: none">• 0 = DHCP server responses don't include gateway and DNS server information• 1 = Gateway and DNS server information are included in responses
5-8	ipv4	address	First address of DHCP Server address pool
9-12	ipv4	subnet_mask	Subnetwork mask
13-16	uint32	lease_time	DHCP address lease timeout

C Functions

```
/* Callback */
struct wifi_msg_tcpip_dhcp_configuration_evt_t{
    uint8 routing_enabled,
    ipv4 address,
    ipv4 subnet_mask,
    uint32 lease_time
}
void wifi_evt_tcpip_configuration(
    const struct wifi_msg_tcpip_dhcp_configuration_evt_t * msg
)
```

BGScript Functions

```
event tcpip_dhcp_configuration(routing_enabled,address,subnet_mask,lease_time)
```

DHCP Client--TCP/IP

This event contains IPv4 address and MAC address of one client connected to WF121 Access Point. Message is sent for each client one after another.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	Message type: event
1	0x0A	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack

Byte	Type	Name	Description
3	0x0C	method	Message ID
4-7	ipv4	address	Client IPv4 address Zero value means that client is connected, but IPv4 address is not offered by the module's DHCP server.
8-13	hw_addr	mac	Client MAC address

C Functions

```

/* Callback */
struct wifi_msg_tcpip_dhcp_client_evt_t{
    ipv4 address,
    hw_addr mac
}
void wifi_evt_tcpip_dhcp_client(
    const struct wifi_msg_tcpip_dhcp_client_evt_t * msg
)

```

BGScript Functions

```
event tcpip_dhcp_client(address, mac)
```

5.5 Endpoint

This class contains commands related to data endpoint control including creation and deletion of endpoints and data routing.

5.5.1 Commands--endpoint

These commands are related to endpoints.

Send--endpoint

This command can be used to send data to a given endpoint.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint

Byte	Type	Name	Description
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 Streaming--endpoint

This command can be used to configure an UART into a streaming or BGAPI mode. When an 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	hlen	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	hlen	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 Active--endpoint

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 (but host can still send BGAPI commands to it). 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	hlen	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	hlen	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)
```

Set Streaming Destination--endpoint

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	hlen	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	hlen	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

This command can be used to close an endpoint. This command is valid only for UDP or TCP endpoints.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

Set Transmit Size--endpoint

This command can be used to set the desired transmit packet size: endpoint will buffer outgoing data until packet size is reached and then transmit it to remote end. **This only applies to UDP endpoints and should not be used with any other type of endpoint, including TCP.** When using packet size 0, the data will be sent out without a fixed size. If the transmit packet size is set to a higher value than 255, then multiple [endpoint_send](#) command need to be issued to fill the transmit buffer and to effectively send the data to the remote end, due to the fact that the [endpoint_send](#) command can carry at most 255 payload bytes.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

Disable--endpoint

This command can be used to disable an UART type endpoint. This command effectively turns down an UART interface until the module is reset or power-cycled. When an UART interface is disabled its pins go to high-impedance state.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x06	method	Message ID
4	uint8	endpoint	Index of the endpoint to disable 0: UART0 1: UART1

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
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
6	uint8	endpoint	The index of the endpoint

C Functions

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

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

BGScript Functions

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

5.5.2 Events--endpoint

These events are related to endpoints.

Syntax Error--endpoint

This event indicates that a protocol error was detected in BGAPI command parser. This event is triggered if a BGAPI command from the host contains syntax error(s), or if a command is only partially sent. Then the BGAPI parser has a 1 second command timeout and if a valid command is not transmitted within this timeout an error is raised and the partial or wrong command will be ignored.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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 Typical errors are: - 0x0184 Command Not Recognized - 0x0185 Timeout For these and other error codes refer to the Error codes documentation
6	uint8	endpoint	The BGAPI 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)
```


Data--endpoint

This event indicates incoming data from an endpoint.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

Status--endpoint

This event indicates an endpoint's status. Additionally, this event will be sent after connecting and disconnecting the USB cable.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

Closing--endpoint

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	hlen	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--endpoint

This event indicates an error in an endpoint.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

5.5.3 Enumerations--endpoint

These enumerations are related to endpoints.

Endpoint types--endpoint

These enumerations define the endpoint types. Not to be confused with the dynamic endpoint indexes used e.g. with command [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

This class contains commands related to hardware peripherals and interfaces.

5.6.1 Commands--hardware

Hardware commands

Set Soft Timer--hardware

This command can be used to enable the software timer. Multiple concurrent timers can be running at the same time.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

External Interrupt Config--hardware

This command can be used to configure 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 WF121's internal use and cannot be used for other purposes. Interrupts can be triggered either on the rising edge or the falling edge.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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 INT0 : 0x01 INT2 : 0x04 INT3 : 0x08 INT4 : 0x10 Example: interrupts INT0 and INT4 are enabled with value of 0x5
5	uint8	polarity	External interrupt polarity bits, rising edge if set, falling edge otherwise INT0 : 0x01 INT2 : 0x04 INT3 : 0x08 INT4 : 0x10 Example: INT0 as falling and INT2 as rising are set with value of 0x4

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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)

```

Change Notification Config--hardware

This command can be used to configure change notifications (CN). The PIC32 micro controller has a limited number of standard GPIO interrupts. Change notifications can be used in a similar way to GPIO interrupts in most cases but they are not identical with each other and operate on different pins. This command can be used to configure for which pins the change notification interrupts are enabled. A list of pins and corresponding change notification source see WF121 Datasheet *page 9, Table 2: Multifunction pad descriptions*. More detailed information can be found in the PIC32 Datasheet in the 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	hlen	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	hlen	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. The CN command must be issued again before getting the next event.

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

This command can be used to configure change notification pull-up settings. For a detailed discussion concerning change notifications, see the [Change Notification Config](#) command.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

IO Port Config Direction--hardware

This command can be used to configure the data flow direction of I/O-port(s).

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	Bit mask of which pins on the port this command affects.
7 - 8	uint16	direction	The bit mask describing which are inputs and which are outputs. 0 : Output 1 : Input

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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

This command can be used to configure I/O-port open drain functionality. Open drain means that when the pin is in high state, it is in high impedance state and when low it is able to sink current. Open drain is sometimes also called [Open Collector](#).

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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 Write--hardware

This command can be used to write the pins of an I/O-port.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	Bit mask of which pins on the port this command affects. For each bit in the bit mask: 0 = Don't modify/write, 1 = modify/write
7 - 8	uint16	data	Bit mask of which pins to set. For each bit in the bit mask: 0 : low 1 : high

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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)
```

IO Port Read--hardware

This command can be used to read the status of pins of an I/O-port.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

Output Compare--hardware

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

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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 32-bit mode selected. 0: 16-bit 1: 32-bit, 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 16-bit 0-0xFFFFFFFF for 32-bit

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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)
```

ADC Read--hardware

This command is used to read the module's A/D converter. The hardware configuration file (normally hardware.xml) should also contain the <adc ... /> tag, which is meant to enable the pins to use for the ADC readings.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

RTC Init--hardware

This command can be used to initialize the internal Real Time Clock (RTC).

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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--hardware

This command can be used to set the internal Real Time Clock (RTC) time.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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--hardware

This command can be used to read the internal Real Time Clock (RTC) value.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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--hardware

This command can be used to set an alarm for the internal Real Time Clock (RTC).

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)

```

UART Conf Set--hardware

This command can be used to re-configure an UART interface.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x09	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0E	method	Message ID
4	uint8	id	UART ID
5 - 8	uint32	rate	Baud rate e.g. 115200, up to 10 Mbps
9	uint8	data_bits	Data bits Values: 8 or 9
10	uint8	stop_bits	Stop bits Values: 1 or 2
11	uint8	parity	Parity 0: none 1: odd 2: even
12	uint8	flow_ctrl	Flow control 0: none 1: RTS/CTS 2: RTS

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
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

C Functions

```
/* Function */  
void wifi_cmd_hardware_uart_conf_set (
```

```

uint8 id,
uint32 rate,
uint8 data_bits,
uint8 stop_bits,
uint8 parity,
uint8 flow_ctrl
);

/* Callback */
struct wifi_msg_hardware_uart_conf_set_rsp_t{
    uint16 result
}
void wifi_rsp_hardware_uart_conf_set(
    const struct wifi_msg_hardware_uart_conf_set_rsp_t * msg
)

```

BGScript Functions

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


UART Conf Get--hardware

This command can be used to read the current configuration of an UART interface.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0F	method	Message ID
4	uint8	id	UART id

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x0F	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_uart_conf	UART configuration

C Functions

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

/* Callback */
struct wifi_msg_hardware_uart_conf_get_rsp_t{
    uint16 result
}
void wifi_rsp_hardware_uart_conf_get(
    const struct wifi_msg_hardware_uart_conf_get_rsp_t * msg
)
```

BGScript Functions

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

5.6.2 Enumerations--hardware

These enumerations are related to hardware.

RTC Alarm repeat--hardware

These enumerations define how often an 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

These events are related to hardware.

Soft Timer--hardware

This event indicates that a software timer has reached the defined count (elapsed).

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

Change Notification--hardware

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	hlen	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

This event is generated when an external interrupt occurs and provides a time stamp and IRQ. The IRQ's 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	Time stamp 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--hardware

This event indicates and alarm generated from the internal Real Time Clock (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()
```

UART Conf--hardware

This event reports the current configuration of a UART interface. It follows the command `hardware_uart_conf_get`.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hilen	Message type: event
1	0x09	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x04	method	Message ID
4	uint8	id	UART id
5 - 8	uint32	rate	Baud rate in bps e.g. 115200
9	uint8	data_bits	Data bits 8 or 9
10	uint8	stop_bits	Stop bits 1 or 2
11	uint8	parity	Parity 0=none, 1=odd, 2=even
12	uint8	flow_ctrl	Flow control 0=none, 1=rts/cts 2=rts

C Functions

```
/* Callback */
struct wifi_msg_hardware_uart_conf_evt_t{
    uint8 id,
    uint32 rate,
    uint8 data_bits,
    uint8 stop_bits,
    uint8 parity,
    uint8 flow_ctrl
}
void wifi_evt_hardware_uart_conf(
    const struct wifi_msg_hardware_uart_conf_evt_t * msg
)
```

BGScript Functions

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

5.7 I2C

This class contains commands related to I2C peripherals.

5.7.1 Commands--I2C

These commands are related to I2C interface.

Start Read--I2C

This command can be used to start I2C transmission for reading data. The data is transferred via [Endpoint Data](#) events.



Only I2C master functionality is supported.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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

This command can be used to prepare an I2C endpoint for data transmission. The data is sent using the [Endpoint Send](#) command.


 Only I2C master functionality is supported.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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

This command can be used to stop the I2C transmission.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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

This class contains commands related to the RMII (Ethernet) interface of the Wi-Fi module.

5.8.1 Commands--Ethernet

These commands are related to wired Ethernet.

Set Dataroute--Ethernet

This command can be used to configure the Ethernet interface's data route and functionality mode.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x0A	class	Message class: Wired Ethernet
3	0x00	method	Message ID
4	uint8	route	0 : Off Ethernet interface is off and the link is down. 1 : Bridge Module is transparent Ethernet - Wi-Fi bridge mode and all data is routed from Ethernet to Wi-Fi and vice versa and it will bypass the built in IP stack. Before enabling this setting you must first connect to a Wi-Fi network in client (STA) mode or create an access point. It is suggested to have only one client connected in AP mode. 2 : Ethernet server Ethernet is connected to the module's built-in IP stack. And the built-in DHCP and HTTP servers, as well as TCP and UDP endpoints accessible via Ethernet. Before enabling this this setting, module must be configured into a Wi-Fi access point (AP) mode. However when this command is entered Wi-Fi is disabled and Ethernet used instead. 3 : Ethernet device In this mode the Ethernet can be used as a client to connect to a network instead of Wi-Fi. When this command is entered the Wi-Fi radio is disabled.

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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
Byte	Type	Name	Description

			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--Ethernet

This command can be used to close the wired Ethernet connection.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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()
```

Connected--Ethernet--W-Fi

This command can be used to test wired Ethernet connection.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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 not connected 1: Cable is 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)
```

5.8.2 Events--Ethernet

These events are related to the wired Ethernet.

Link Status--Ethernet

This event indicates the status changes of Ethernet link state.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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 is down 1: link is 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

This class contains commands related to Web server activation and management.

5.9.1 Commands--HTTPS

These commands are related to the HTTP server.

Enable--HTTPS

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

When the DHCP server is started, the IP address pool for the clients will start with the IP address set with [DHCP Configure--TCP/IP](#) command.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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 0: Disable 1: Enable
5	uint8	dhcps	Enable/disable DHCP server 0: Disable 1: Enable
6	uint8	dnss	Enable/disable DNS server 0: Disable 1: Enable

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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 dhcp,
    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, dhcp, dnss)(result)
```

Add Path--HTTPS

This command can be used to add a mapping between an HTTP server URL and a storage device from where a resource will be served.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x01	method	Message ID
4	uint8	device	Storage device type 0 : Built-in flash 1 : BGAPI / BGScript 2 : SD card
5	uint8array	path	Server path (URL)

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
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_https_add_path(
    uint8 device,
    uint8 path_len,
    const uint8* path_data
);

/* Callback */
struct wifi_msg_https_add_path_rsp_t{
    uint16 result
}
void wifi_rsp_https_add_path(
    const struct wifi_msg_https_add_path_rsp_t * msg
)
```

BGScript Functions

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

API Response--HTTPS

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

Table: COMMAND

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

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
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_https_api_response(
    uint32 request,
    uint8 data_len,
    const uint8* data_data
);

/* Callback */
struct wifi_msg_https_api_response_rsp_t{
    uint16 result
}
void wifi_rsp_https_api_response(
    const struct wifi_msg_https_api_response_rsp_t * msg
)
```

BGScript Functions

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

API Response Finish--HTTPS

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

Table: COMMAND

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

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
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_https_api_response_finish(
    uint32 request
);

/* Callback */
struct wifi_msg_https_api_response_finish_rsp_t{
    uint16 result
}
void wifi_rsp_https_api_response_finish(
    const struct wifi_msg_https_api_response_finish_rsp_t * msg
)
```

BGScript Functions

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

5.9.2 Events--HTTPS

These events are related to the HTTP server.

API Request--HTTPS

This event is always generated when an HTTP request from a remote client is asking for a resource which is defined to be provided by a BGScript, or by the host MCU using the BGAPI, in accordance to the configuration entered with the command [https_add_path](#). Request number is a unique identification that is used for all subsequent events related to this particular request, namely [https_api_request_header](#) then [https_api_request_data](#) (generated only if HTTP request indeed contains data) then [https_api_request_finished](#). Any response command to this request, such as [https_api_response](#) followed by [https_api_response_finish](#), must include the same number.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	Message type: event
1	0x06	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x02	method	Message ID
4 - 7	uint32	request	Request number
8	uint8	method	Request method 0: GET 1: PUT 2: POST 3: DELETE
9	uint8array	resource	Request resource

C Functions

```
/* Callback */
struct wifi_msg_https_api_request_evt_t{
    uint32 request,
    uint8 method,
    uint8 resource_len,
    const uint8* resource_data
}
void wifi_evt_https_api_request(
    const struct wifi_msg_https_api_request_evt_t * msg
)
```

BGScript Functions

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

API Request Header--HTTPS

This event includes the HTTP header data of a particular HTTP request. At least one event is expected to be generated, but multiple events may be also generated.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	Message type: event
1	0x05	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x03	method	Message ID
4 - 7	uint32	request	Request number
8	uint8array	data	Request header data

C Functions

```
/* Callback */
struct wifi_msg_https_api_request_header_evt_t{
    uint32 request,
    uint8 data_len,
    const uint8* data_data
}
void wifi_evt_https_api_request_header(
    const struct wifi_msg_https_api_request_header_evt_t * msg
)
```

BGScript Functions

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

API Request Data--HTTPS

This event includes HTTP payload data of a particular HTTP request. Multiple events may be generated, or none if the request does not carry data at all.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	Message type: event
1	0x05	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x04	method	Message ID
4 - 7	uint32	request	Request number
8	uint8array	data	Request data

C Functions

```
/* Callback */
struct wifi_msg_https_api_request_data_evt_t{
    uint32 request,
    uint8 data_len,
    const uint8* data_data
}
void wifi_evt_https_api_request_data(
    const struct wifi_msg_https_api_request_data_evt_t * msg
)
```

BGScript Functions

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


API Request Finished--HTTPS

This event indicates that all HTTP header data, and payload data if any, have been fully delivered for a particular HTTP request. The HTTP request is left pending until [API Response Finish](#) has been called or a timeout occurs.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	Message type: event
1	0x04	lolen	Minimum payload length
2	0x09	class	Message class: HTTP Server
3	0x05	method	Message ID
4 - 7	uint32	request	Request number

C Functions

```
/* Callback */
struct wifi_msg_https_api_request_finished_evt_t{
    uint32 request
}
void wifi_evt_https_api_request_finished(
    const struct wifi_msg_https_api_request_finished_evt_t * msg
)
```

BGScript Functions

```
event https_api_request_finished(request)
```

5.10 Persistent Store

This class contains commands related to non-volatile memory. These commands can be used to read, write, and dump the local devices parameters (PS keys).



4kB of flash is reserved for Persistent Store.

PS Key IDs over 0x8000 can be used for storing user data.

Flash endurance limits the total written data into PS to 4MB.

5.10.1 Commands--Flash

These commands are related to the Persistent Store.

PS Defrag--Flash

This command can be used to manually initiate the defragmentation of the Persistent Store. Persistent store is also automatically defragmented if there is not enough space.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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 Dump--Flash

This command can be used to dump all the PS keys from the Persistent Store. The command will generate a series of PS key events. The last PS key event is identified by the key index value 65535, indicating that the dump has finished listing all PS keys.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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 Erase All--Flash

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


 This command will erase the MAC address of the device!

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

PS Save--Flash

This command can be used to store a value into the given PS (Persistent Store) key. This command can be used to store user data into the Wi-Fi module flash memory, so that the data remains available across resets and power cycles.

The maximum 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	hlen	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	hlen	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

This command can be used to retrieve the value of the given PS key from the Persistent Store.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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 Example: 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 Erase--Flash

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

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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)
```

5.10.2 Enumerations--Flash

These enumerations are related to Persistent Store.

PS_KEYS--Flash

These enumerations 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 Wi-Fi 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_DHCP_ENABLE	DHCP server enable
31	FLASH_PS_KEY_DHCP_SPACE	DHCP server first address of IPv4 address space, as 4 byte integer
32	FLASH_PS_KEY_DHCP_DISABLE_ROUTING	DHCP server 'routing' enabled
33	FLASH_PS_KEY_DHCP_MASK	DHCP server subnet mask, as 4 byte integer
34	FLASH_PS_KEY_DHCP_LEASETIME	DHCP server address lease time, as 4 byte integer
35	FLASH_PS_KEY_DNSS_ENABLE	DNS server enable

Value (dec)	Name	Description
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
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

These events are related to Persistent Store.

PS Key--Flash

This event is generated when PS keys are dumped from the Persistent Store.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

PS Key Changed--Flash

This event indicates that a PS key has been changed.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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)
```

Low Voltage--Flash

These events indicate that a low voltage state was detected during Flash operation. Flash writing is not possible.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hilen	Message type: event
1	0x00	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x02	method	Message ID

C Functions

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

BGScript Functions

```
event flash_low_voltage()
```

5.11 Device Firmware Upgrade

This class contains commands related to firmware update. The commands and corresponding events in this class are available only when the device has been booted into DFU mode.

5.11.1 Commands--DFU

These commands relate to Device Firmware Upgrade.

Reset--DFU

This command resets the Wi-Fi module. This command does not have a response, but triggers one of the boot events after re-boot.

The DFU process:

1. Boot device to DFU mode with [Reset](#) command.
2. Wait for [DFU boot](#) event.
3. Send command [Flash Set Address](#) to start the firmware update.
4. Upload the firmware with [Flash Upload](#) commands until all the data has been uploaded.
5. Send [Flash Upload Finish](#) to when all the data has been uploaded.
6. Finalize the DFU firmware update with [Reset](#) command.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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

This command can be used after re-booting the Wi-Fi module into DFU mode and it sets the starting address on the Flash from where onwards the new firmware data will be written into.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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. Use the value 0x0000

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	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

This command can be used to upload the firmware update file to the Wi-Fi Module. The payload of the data in this command is 128 bytes, so multiple commands need to be used to upload the full firmware update file.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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

This command can be used to tell the device that the DFU file has been fully uploaded. Next the user should issue the [DFU Reset](#) command to return the device back into normal mode.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	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	hlen	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

These events are related to Device Firmware Upgrade.

Boot--DFU

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

Table: EVENT

Byte	Type	Name	Description
0	0x88	hlen	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 Utilities for BGScript

This class contains commands related to BGScript programming language utilities.

5.12.1 Commands--Util

These commands are utility functions for using BGScript commands.

Atol--Util

This command can be used to convert a decimal value from an ASCII string format into a 32-bit signed integer format.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x0C	class	Message class: Utilities for BGScript
3	0x00	method	Message ID
4	uint8array	string	String to convert

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x04	lolen	Minimum payload length
2	0x0C	class	Message class: Utilities for BGScript
3	0x00	method	Message ID
4 - 7	int32	value	32-bit integer value

C Functions

```
/* Function */
void wifi_cmd_util_atoi(
    uint8 string_len,
    const uint8* string_data
);

/* Callback */
struct wifi_msg_util_atoi_rsp_t{
    int32 value
}
void wifi_rsp_util_atoi(
    const struct wifi_msg_util_atoi_rsp_t * msg
)
```

BGScript Functions

```
call util_atoi(string_len, string_data)(value)
```

ItoA--Util

This command can be used to convert a number from signed 32-bit integer format into decimal ASCII value format.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x04	lolen	Minimum payload length
2	0x0C	class	Message class: Utilities for BGScript
3	0x01	method	Message ID
4 - 7	int32	value	32-bit integer value to convert

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x0C	class	Message class: Utilities for BGScript
3	0x01	method	Message ID
4	uint8array	string	Converted ASCII string

C Functions

```
/* Function */
void wifi_cmd_util_itoa(
    int32 value
);

/* Callback */
struct wifi_msg_util_itoa_rsp_t{
    uint8 string_len,
    const uint8* string_data
}
void wifi_rsp_util_itoa(
    const struct wifi_msg_util_itoa_rsp_t * msg
)
```

BGScript Functions

```
call util_itoa(value)(string_len, string_data)
```

5.13 SD card

This class contains commands related to the Wi-Fi module SD memory card interface.

5.13.1 Commands--SDHC

These commands are related with SD/SDHC memory card file system.

Fopen--SDHC

This command can be used to open a file. When you open a file you can set the mode to either read (0x1), write (0x2) or read/write(0x03).

To create a new file, the new bit (0x08) must be set.



You must open a file before it can be read, written or renamed.

Maximum total length of file name (path + file name) is 254 bytes.

Maximum 10 files can be opened concurrently.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
3	0x00	method	Message ID
4	uint8	mode	Open mode 0x1 : read 0x2 : write 0x8 : create new file
5	uint8array	fname	File name

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
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
sdhc_ffile	File information. In the event fattrib value is file open attribute. E.g. read only when file opened as read only, even file attribute in disk is different.

C Functions

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

/* Callback */
struct wifi_msg_sdhc_fopen_rsp_t{
    uint16 result
}
void wifi_rsp_sdhc_fopen(
    const struct wifi_msg_sdhc_fopen_rsp_t * msg
)
```

BGScript Functions

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

Fclose--SDHC

This command can be used to close a file.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
3	0x01	method	Message ID
4	uint8	fhandle	Handle of the open file

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
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_sdhc_fclose(
    uint8 fhandle
);

/* Callback */
struct wifi_msg_sdhc_fclose_rsp_t{
    uint16 result
}
void wifi_rsp_sdhc_fclose(
    const struct wifi_msg_sdhc_fclose_rsp_t * msg
)
```

BGScript Functions

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


Fdir--sdhc--WIFI

This command can be used to list files of the current directory. The command also lists files and folders with the hidden attribute set.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
3	0x02	method	Message ID
4	uint8	mode	File list mode 0: normal 1: recursive
5	uint8array	path	Path of directory to list

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
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
sdhc_file	File info
sdhc_ready	Operation ready

C Functions

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

/* Callback */
struct wifi_msg_sdhc_fdir_rsp_t{
    uint16 result
}
void wifi_rsp_sdhc_fdir(
```

```
const struct wifi_msg_sdhc_fdir_rsp_t * msg  
)
```

BGScript Functions

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

Fread--SDHC

This command can be used to read data from a file. The read data is sent in multiple `sdhc_fdata` events each one containing up to 512 bytes of data.



- When a whole file is read, the reading always starts from the beginning of the file.
- After a file has been written you must close and reopen it, before you can read it.
- To restart a file read process from the beginning you need to close and reopen the file.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x05	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
3	0x03	method	Message ID
4	uint8	fhandle	Handle of opened file
5 - 8	uint32	fsize	Bytes to read 0 : read the whole file

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
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
<code>sdhc_fdata</code>	Data
<code>sdhc_ready</code>	Operation ready

C Functions

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

```

);

/* Callback */
struct wifi_msg_sdhc_fread_rsp_t{
    uint16 result
}
void wifi_rsp_sdhc_fread(
    const struct wifi_msg_sdhc_fread_rsp_t * msg
)

```

BGScript Functions

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

Fwrite--sdhc--WIFI

Write data to a file.



You can write more data to a file by calling this command again, but only after you have received **Ready** event. Data will be appended to the end of the file.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
3	0x04	method	Message ID
4	uint8	fhandle	Handle of opened file
5 - 6	uint16array	fdata	Data to write

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
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
sdhc_ready	Operation ready

C Functions

```
/* Function */
void wifi_cmd_sdhc_fwrite(
    uint8 fhandle,
    uint16 fdata_len,
    const uint8* fdata_data
);

/* Callback */
struct wifi_msg_sdhc_fwrite_rsp_t{
    uint16 result
}
void wifi_rsp_sdhc_fwrite(
    const struct wifi_msg_sdhc_fwrite_rsp_t * msg
```

)

BGScript Functions

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

Fdelete--SDHC

This command can be used to delete a file or empty directory.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
3	0x05	method	Message ID
4	uint8array	fname	File name with path

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
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_sdhc_fdelete(
    uint8 fname_len,
    const uint8* fname_data
);

/* Callback */
struct wifi_msg_sdhc_fdelete_rsp_t{
    uint16 result
}
void wifi_rsp_sdhc_fdelete(
    const struct wifi_msg_sdhc_fdelete_rsp_t * msg
)
```

BGScript Functions

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

Fmkdir--SDHC

This command can be used to create a directory.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
3	0x06	method	Message ID
4	uint8array	dir_name	Directory to create

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
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_sdhc_fmkdir(
    uint8 dir_name_len,
    const uint8* dir_name_data
);

/* Callback */
struct wifi_msg_sdhc_fmkdir_rsp_t{
    uint16 result
}
void wifi_rsp_sdhc_fmkdir(
    const struct wifi_msg_sdhc_fmkdir_rsp_t * msg
)
```

BGScript Functions

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


Fchdir--SDHC

This command can be used to change the active directory.



To get the current active directory, use this command with path value ".".

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
3	0x07	method	Message ID
4	uint8array	dir_name	Full path of directory to change

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
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

Table: EVENTS

Event	Description
sdhc_fpwd	Path of active directory

C Functions

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

/* Callback */
struct wifi_msg_sdhc_fchdir_rsp_t{
    uint16 result
}
void wifi_rsp_sdhc_fchdir(
    const struct wifi_msg_sdhc_fchdir_rsp_t * msg
)
```

BGScript Functions

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

Frename--SDHC

This command can be used to rename a file in the current directory.



To rename a file:

1. Open the file in the current directory.
2. Rename the file, which will also close the file.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
3	0x08	method	Message ID
4	uint8	fhandle	Handle of opened file
5	uint8array	new_name	File name to change

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
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_sdhc_frename(
    uint8 fhandle,
    uint8 new_name_len,
    const uint8* new_name_data
);

/* Callback */
struct wifi_msg_sdhc_frename_rsp_t{
    uint16 result
}
void wifi_rsp_sdhc_frename(
    const struct wifi_msg_sdhc_frename_rsp_t * msg
)
```

BGScript Functions

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

Fchmode--SDHC

This command can be used to change the attributes of a file.

Table: COMMAND

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
3	0x09	method	Message ID
4	uint8	value	File or directory attributes. Values: 0x01: read only 0x02: hidden 0x04: system
5	uint8array	fname	File name

Table: RESPONSE

Byte	Type	Name	Description
0	0x08	hlen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x0B	class	Message class: SD/SDHC memory card filesystem
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_sdhc_fchmode(
    uint8 value,
    uint8 fname_len,
    const uint8* fname_data
);

/* Callback */
struct wifi_msg_sdhc_fchmode_rsp_t{
    uint16 result
}
void wifi_rsp_sdhc_fchmode(
    const struct wifi_msg_sdhc_fchmode_rsp_t * msg
)
```

BGScript Functions

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

5.13.2 Events--SDHC

These events are related to the SD/SDHC memory card file system.

Ready--SDHC

This event indicates that an SD operation has finished.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hilen	Message type: event
1	0x04	lolen	Minimum payload length
2	0x0C	class	Message class: SD/SDHC memory card filesystem
3	0x00	method	Message ID
4	uint8	fhandle	Handle of opened file
5	uint8	operation	Operation which caused the event. 0: card state 1: file read 2: file write 3: file listing
6 - 7	uint16	result	Result code 0: success Non-zero: an error occurred For other values refer to the error code documentation

C Functions

```
/* Callback */
struct wifi_msg_sdhc_ready_evt_t{
    uint8 fhandle,
    uint8 operation,
    uint16 result
}
void wifi_evt_sdhc_ready(
    const struct wifi_msg_sdhc_ready_evt_t * msg
)
```

BGScript Functions

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

Fdata--SDHC

This event is generated when the content of the data is read with the command [Fread](#). Multiple events might be generated based on the file size.

Table: EVENT

Byte	Type	Name	Description
0	0x88	hilen	Message type: event
1	0x03	lolen	Minimum payload length
2	0x0C	class	Message class: SD/SDHC memory card filesystem
3	0x01	method	Message ID
4	uint8	fhandle	Handle of opened file
5 - 6	uint16array	data	Content

C Functions

```
/* Callback */
struct wifi_msg_sdhc_fdata_evt_t{
    uint8 fhandle,
    uint16 data_len,
    const uint8* data_data
}
void wifi_evt_sdhc_fdata(
    const struct wifi_msg_sdhc_fdata_evt_t * msg
)
```

BGScript Functions

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


Ffile--SDHC

This event contains the file information and is generated in response to commands [Fdir](#) or [Fopen](#).

Table: EVENT

Byte	Type	Name	Description
0	0x88	hilen	Message type: event
1	0x07	lolen	Minimum payload length
2	0x0C	class	Message class: SD/SDHC memory card filesystem
3	0x02	method	Message ID
4	uint8	fhandle	Handle of opened file, 255=file not opened
5 - 8	uint32	fsize	File size
9	uint8	fattrib	File attributes 0x01 : read only 0x02 : hidden 0x04 : system 0x10 : directory
10	uint8array	fname	File name

C Functions

```
/* Callback */
struct wifi_msg_sdhc_ffile_evt_t{
    uint8 fhandle,
    uint32 fsize,
    uint8 fattrib,
    uint8 fname_len,
    const uint8* fname_data
}
void wifi_evt_sdhc_ffile(
    const struct wifi_msg_sdhc_ffile_evt_t * msg
)
```

BGScript Functions

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

Fpwd--SDHC

This event is generated when active directory is changed with command [Fchdir](#).

Table: EVENT

Byte	Type	Name	Description
0	0x88	hilen	Message type: event
1	0x01	lolen	Minimum payload length
2	0x0C	class	Message class: SD/SDHC memory card filesystem
3	0x03	method	Message ID
4	uint8array	fdir	Full path of active directory

C Functions

```
/* Callback */
struct wifi_msg_sdhc_fpwd_evt_t{
    uint8 fdir_len,
    const uint8* fdir_data
}
void wifi_evt_sdhc_fpwd(
    const struct wifi_msg_sdhc_fpwd_evt_t * msg
)
```

BGScript Functions

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

5.14 Error codes

This section of the document describes the error codes produced by the Wi-Fi software.

5.14.1 BGAPI Errors

These errors are related to BGAPI protocol issues.

Invalid Parameter (0x0180)

This error code indicates that a command contained an invalid parameter

Device in Wrong State (0x0181)

This error code indicates that the device is in wrong state to accept commands.

Out Of Memory (0x0182)

This error indicates that the device has run out of memory.

Feature Not Implemented (0x0183)

This error indicates that the feature in question has not been implemented.

Command Not Recognized (0x0184)

This error indicates that the issued command was not recognized.

Timeout (0x0185)

This error indicates that a command or procedure failed due to timeout.

This error code is generated e.g. if you send an incomplete command to the Wi-Fi module - after the timeout of 1 second this error code is sent to the host transported by the event [endpoint_syntax_error](#).

This error code is generated also e.g. when the maximum number of retry attempts (10) to try to connect to a wireless network have been executed. A typical example of such a case might be when the issued password is invalid, in which case the error code is transported by the event [wifi_evt_sme_connect_failed](#).

Unspecified error (0x0186)

This error code is generated when an unspecified error is detected.

Hardware failure (0x0187)

This error code is generated when a hardware failure is detected.

Internal buffers are full (0x0188)

This error code is generated the a command was not accepted due to full internal buffers.

Disconnected (0x0189)

This error code is generated when a command or procedure has failed due to disconnection.

Too many requests (0x018A)

This error code is generated when there are too many simultaneous requests.

Access Point not in scanlist (0x018B)

This error code is generated when the defined Access Point is not found from the scanlist.

Invalid password (0x018C)

This error code is generated in the following cases:

- 1) you try to connect to a secured network without setting the password or the password is too short
- 2) you try to start a secured AP without setting the password or the password is too short
- 3) you try to set a 64-character PSK with non-hex characters
- 4) you try to set an invalid WEP key (invalid characters or invalid length)

Notice that WPA does not contain any standard way for the Access Point to communicate to the station that the password is invalid. The Access Point just disconnects the client during authentication if the password is found invalid. Some stations take an educated guess that this probably means the password is incorrect. WF121 simply retries the authentication until it exceeds the maximum amount of retries (10) which then causes the 0x0185 Timeout Error.

Authentication failure (0x018D)

This error code is generated when the WPA/WPA2 authentication has failed.

Overflow (0x018E)

This error code is generated when an overflow has been detected.

Multiple PBC sessions (0x018F)

This error code is generated when multiple PBC (Push Button Configuration) sessions have been detected.

Ethernet not connected (0x0190)

This error code is generated when the Ethernet cable is not connected.

Ethernet route not set (0x0191)

This error code is generated if the Ethernet route is not set.

Wrong operating mode (0x0192)

This error code is generated if the operating mode is wrong for the issued command.

Not found (0x0193)

This error code is generated if the requested resource was not found.

Already exists (0x0194)

This error is generated if the requested resource already exists.

Invalid configuration (0x0195)

This error code is generated if the current configuration is invalid.

Access Point lost (0x0196)

This error code is generated if the connection to an Access Point is lost.

5.14.2 Hardware Errors

These errors are related to hardware issues.

PS Store is full (0x0301)

This error is generated if the Flash which is reserved for the PS (Persistent Store) is full.

PS key not found (0x0302)

This error is generated if the PS key (Persistent Store key) was not found.

I2C write already in progress (0x0303)

This error is generated if I2C transmission was initiated while a transmission was already in progress.

I2C ack missing (0x0304)

This error is generated if an acknowledgement for I2C was not received.

Flash write failed (0x0308)

This error code is generated if writing to Flash failed.

File not opened (0x0305)

This error code is generated if an access was attempted to an unopened file.

File not found (0x0306)

This error code is generated if the requested file was not found in the SD card.

Disk error (0x0307)

This error code is generated if an SD card error was detected or if the SD card is full.

5.14.3 TCP/IP Errors

These errors are related to TCP/IP stack issues.

Success (0x0200)

This code indicates that no error was detected.

Out of memory (0x0201)

This error code is generated when the system has run out of memory.

Buffer error (0x0202)

This error code is generated when handling of buffers has failed.

Timeout (0x0203)

This error code is generated when a timeout has been detected.

Routing (0x0204)

This error code is generated when a route could not be found.

In progress (0x0205)

This error code is generated when an operation is in progress.

Illegal_value (0x0206)

This error code is generated when the issued value is deemed illegal.

Would_block (0x0207)--WIFI

This error code is generated when an operation blocks.

Address in use (0x0208)

This error code is generated when the issued address is already in use.

Already connected (0x0209)

This error code is generated when the Wi-Fi module is already connected.

Connection aborted (0x020A)

This error code is generated when a connection is aborted.

Connection reset (0x020B)

This error code is generated when a connection has been reset.

Connection closed (0x020C)

This error code is generated when a connection has been closed.

Not connected (0x020D)

This error code is generated when the Wi-Fi module is not connected.

Illegal argument (0x020E)

This error code is generated if an illegal argument was issued.

Interface level error (0x020F)

This error code is generated if an interface error was detected.

Unknown host (0x0280)

This error is generated if an unknown host is detected.

Service not running (0x0210)

This error code is generated if the specified service is not running.

Service running (0x0211)

This error code is generated if the specified service is already running.

Hostname not set (0x0212)

This error code is generated if the hostname has not been set.

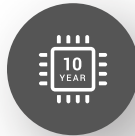
Hostname conflict (0x0213)

This error code is generated if a hostname conflict was detected.

Smart. Connected. Energy-Friendly.



IoT Portfolio
www.silabs.com/products



Quality
www.silabs.com/quality



Support & Community
www.silabs.com/community

Disclaimer

Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications.

Note: This content may contain offensive terminology that is now obsolete. Silicon Labs is replacing these terms with inclusive language wherever possible. For more information, visit www.silabs.com/about-us/inclusive-lexicon-project

Trademark Information

Silicon Laboratories Inc.®, Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, EFM®, EFM32®, EFR, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Redpine Signals®, WiSeConnect, n-Link, ThreadArch®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, Gecko OS, Gecko OS Studio, Precision32®, Simplicity Studio®, Telegesis, the Telegesis Logo®, USBXpress®, Zentri, the Zentri logo and Zentri DMS, Z-Wave®, and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. Wi-Fi is a registered trademark of the Wi-Fi Alliance. All other products or brand names mentioned herein are trademarks of their respective holders.



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

www.silabs.com