BLUEGIGA WI-FI SOFTWARE

V.1.0 API DOCUMENTATION

Saturday, 9 June 2012

Version 0.8.0

Copyright © 2001 - 2012 Bluegiga Technologies

Bluegiga Technologies reserves the right to alter the hardware, software, and/or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. Bluegiga Technologies assumes no responsibility for any errors which may appear in this manual. Bluegiga Technologies' products are not authorized for use as critical components in life support devices or systems.

Bluegiga Access Server, Access Point, AX4, BSM, iWRAP, BGScript and WRAP THOR are trademarks of Bluegiga Technologies.

The *Bluetooth* trademark and logo are registered trademarks and are owned by the Bluetooth SIG, Inc.

ARM and ARM9 are trademarks of ARM Ltd.

Linux is a trademark of Linus Torvalds.

All other trademarks listed herein belong to their respective owners.

TABLE OF CONTENTS

1. Introduction to Bluegiga Wi-Fi software . . . 6
    1.1 Bluegiga Wi-Fi Stack . . . 6
    1.2 Bluegiga BGAPI protocol . . . 7
    1.3 Bluegiga BGLib library . . . 8
    1.4 Bluegiga BGScript scripting language . . . 9
2. Understanding Endpoints . . . 10
3. API Definition . . . 12
    3.1 BGAPI protocol definition . . . 12
    3.2 BGLIB functions definition . . . 14
    3.3 BGScript API definition . . . 15
    3.4 Data types . . . 16
4. API Reference . . . 17
    4.1 System . . . 18
        4.1.1 Commands . . . 18
            4.1.1.1 Hello . . . 18
            4.1.1.2 Reset . . . 19
            4.1.1.3 Sync . . . 20
        4.1.2 Events . . . 21
            4.1.2.1 Boot . . . 21
            4.1.2.2 State . . . 22
        4.1.3 Enumerations . . . 23
            4.1.3.1 Main state . . . 23
    4.2 Configuration . . . 24
        4.2.1 Commands . . . 24
            4.2.1.1 Get Mac . . . 24
            4.2.1.2 Set Mac . . . 26
        4.2.2 Events . . . 28
            4.2.2.1 Mac Address . . . 28
    4.3 SME . . . 29
        4.3.1 Commands . . . 29
            4.3.1.1 Connect Bssid . . . 29
            4.3.1.2 Connect Ssid . . . 31
            4.3.1.3 Disconnect . . . 32
            4.3.1.4 Set Password . . . 33
            4.3.1.5 Start Scan . . . 34
            4.3.1.6 Stop Scan . . . 35
            4.3.1.7 Wifi Off . . . 36
            4.3.1.8 Wifi On . . . 37
        4.3.2 Events . . . 38
            4.3.2.1 Connected . . . 38
            4.3.2.2 Connect Failed . . . 39
            4.3.2.3 Connect Retry . . . 40
            4.3.2.4 Disconnected . . . 41
            4.3.2.5 Interface Status . . . 42
            4.3.2.6 Scanned . . . 43
            4.3.2.7 Scan Result Drop . . . 44
            4.3.2.8 Scan Result . . . 45
            4.3.2.9 Wifi Is Off . . . 46
            4.3.2.10 Wifi Is On . . . 47
    4.4 TCP stack . . . 48
        4.4.1 Commands . . . 48
            4.4.1.1 Configure . . . 48
            4.4.1.2 Dns Configure . . . 50
            4.4.1.3 Dns Gethostbyname . . . 51
            4.4.1.4 Start Tcp Server . . . 52
            4.4.1.5 Start Udp Server . . . 53
            4.4.1.6 Tcp Connect . . . 54
            4.4.1.7 Udp Connect . . . 56
        4.4.2 Events . . . 58
            4.4.2.1 Configuration . . . 58
            4.4.2.2 Dns Configuration . . . 59
            4.4.2.3 Dns Gethostbyname Result . . . 60

4.4.2.4 Endpoint Status . . . 61
4.5 Endpoint . . . 62
4.5.1 Commands . . . 62
4.5.1.1 Close . . . 62
4.5.1.2 Send . . . 64
4.5.1.3 Set Active . . . 65
4.5.1.4 Set Streaming Destination . . . 67
4.5.1.5 Set Streaming . . . 68
4.5.2 Events . . . 70
4.5.2.1 Closing . . . 70
4.5.2.2 Data . . . 71
4.5.2.3 Status . . . 72
4.5.3 Enumerations . . . 73
4.5.3.1 Endpoint types . . . 73
4.6 Hardware . . . 74
4.6.1 Commands . . . 74
4.6.1.1 Change Notification Config . . . 74
4.6.1.2 Change Notification Pullup . . . 76
4.6.1.3 External Interrupt Config . . . 77
4.6.1.4 Io Port Config Direction . . . 79
4.6.1.5 Io Port Config Open Drain . . . 81
4.6.1.6 Io Port Read . . . 83
4.6.1.7 Io Port Write . . . 85
4.6.1.8 Set Soft Timer . . . 87
4.6.2 Events . . . 89
4.6.2.1 Change Notification . . . 89
4.6.2.2 External Interrupt . . . 90
4.6.2.3 Soft Timer . . . 91
4.7 Persistent Store . . . 92
4.7.1 Commands . . . 92
4.7.1.1 Ps Defrag . . . 92
4.7.1.2 Ps Dump . . . 94
4.7.1.3 Ps Erase All . . . 95
4.7.1.4 Ps Erase . . . 96
4.7.1.5 Ps Load . . . 97
4.7.1.6 Ps Save . . . 98
4.7.2 Events . . . 99
4.7.2.1 Ps Key . . . 99
4.8 Device Firmware Upgrade . . . 100
4.8.1 Commands . . . 100
4.8.1.1 Flash Set Address . . . 100
4.8.1.2 Flash Upload . . . 102
4.8.1.3 Flash Upload Finish . . . 103
4.8.1.4 Reset . . . 104
4.8.2 Events . . . 105
4.8.2.1 Boot . . . 105
4.9 Error codes . . . 106
4.9.1 BGAPI Errors . . . 106
4.9.1.1 Invalid Parameter (0x0180) . . . 106
4.9.1.2 Device in Wrong State (0x0181) . . . 106
4.9.1.3 Out Of Memory (0x0182) . . . 106
4.9.1.4 Feature Not Implemented (0x0183) . . . 106
4.9.1.5 Command Not Recognized (0x0184) . . . 106
4.9.1.6 Timeout (0x0185) . . . 106
4.9.1.7 Unspecified error (0x0186) . . . 106
4.9.2 TCP IP Errors . . . 106
4.9.2.1 Success (0x0200) . . . 106
4.9.2.2 Out of memory (0x0201) . . . 106
4.9.2.3 Buffer error (0x0202) . . . 106
4.9.2.4 Timeout (0x0203) . . . 106
4.9.2.5 Routing (0x0204) . . . 107
4.9.2.6 Address in use (0x0208) . . . 107
4.9.2.7 Already connected (0x0209) . . . 107
4.9.2.8 Connection aborted (0x020A) . . . 107

4.9.2.9 Connection reset (0x020B) . . . 107
4.9.2.10 Connection closed (0x020C) . . . 107
4.9.2.11 Not connected (0x020D) . . . 107
4.9.2.12 Interface level error (0x020F) . . . 107
5. Contact information . . . 108

1 Introduction to Bluegiga Wi-Fi software

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

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

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

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

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

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

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

1.1 Bluegiga Wi-Fi Stack

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

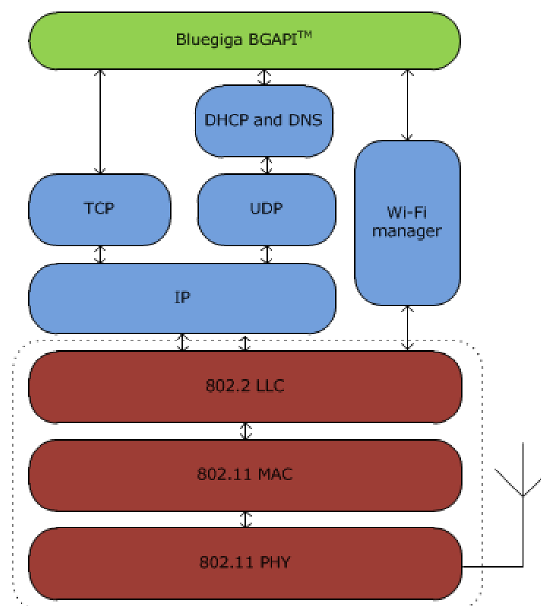

Figure 1: Bluegiga Wi-Fi Software

Copyright © 2000-2011 Bluegiga Technologies

1.2 Bluegiga BGAPI protocol

For applications where a separate host (MCU) is used to implement the end user application, a transport protocol is needed between the host and the Wi-Fi stack. The transport protocol is used to communicate with the Wi-Fi stack as well to transmit and receive data packets. This protocol is called BGAPI and it's a binary based communication protocol designed specifically for ease of implementation within host devices with limited resources.

The BGAPI provides access to the following layers:

- **System** - Various system functions, such as querying the hardware status or reset it
- **Configuration** - Provides access to the devices parameters such as the MAC address
- **TCP stack** - Gives access to the TCP/IP stack and various protocols like TCP and UDP
- **SME** - Provides access to 802.11 MAC and procedures like access point discovery
- **Endpoint** - Provides functions to control the data endpoints
- **Hardware** - An interface to access the various hardware layers such as timers, ADC and other hardware interfaces
- **Persistent Store** - Allows user to read/write data to non-volatile memory
- **Device Firmware Upgrade** - Provides access to firmware update functions

The BGAPI protocol is intended to be used with:

- a serial UART link

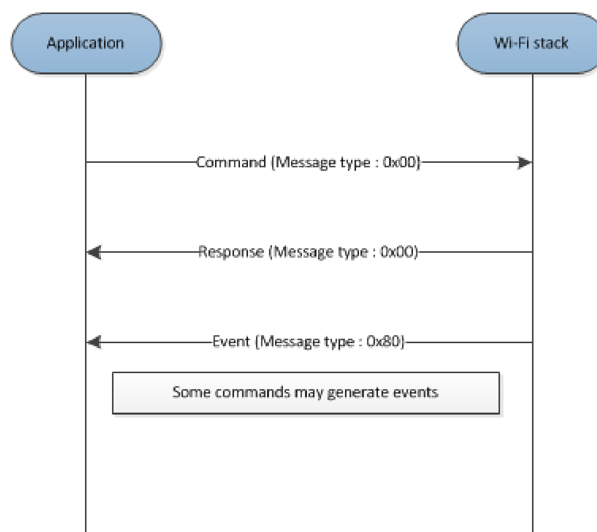

Figure 2: BGAPI messaging

Copyright © 2000-2011 Bluegiga Technologies

1.3 Bluegiga BGLib library

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

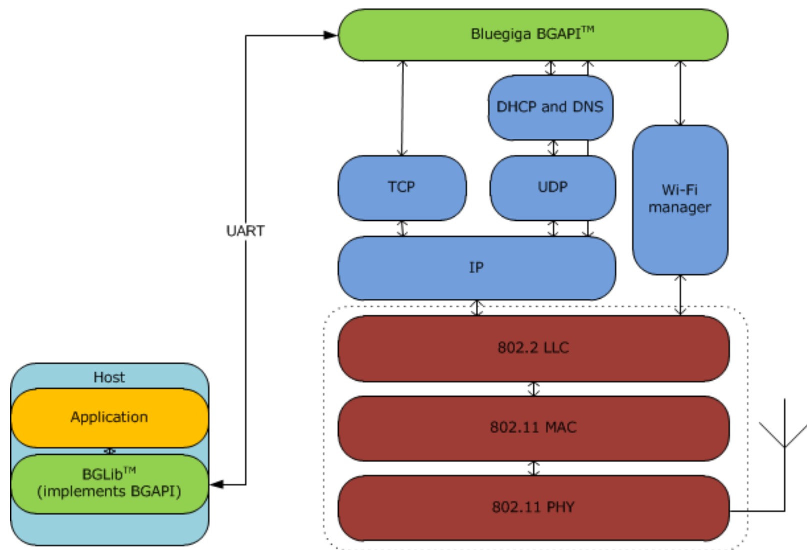

Figure 3: BGLib library

Copyright © 2000-2011 Bluegiga Technologies

1.4 Bluegiga BGScript scripting language

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

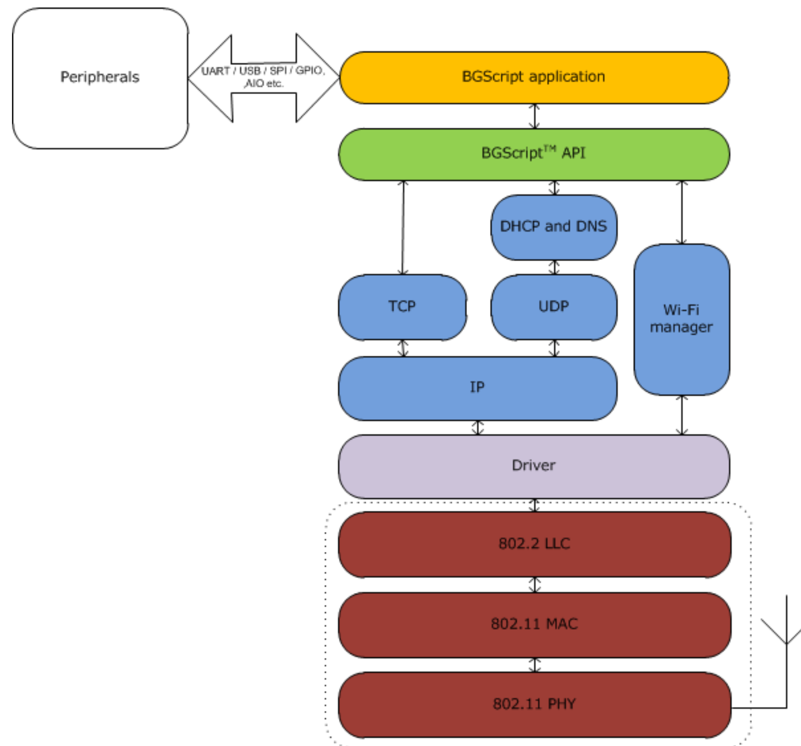

Figure 4: BGScript architecture

2 Understanding Endpoints

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

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

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

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

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

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

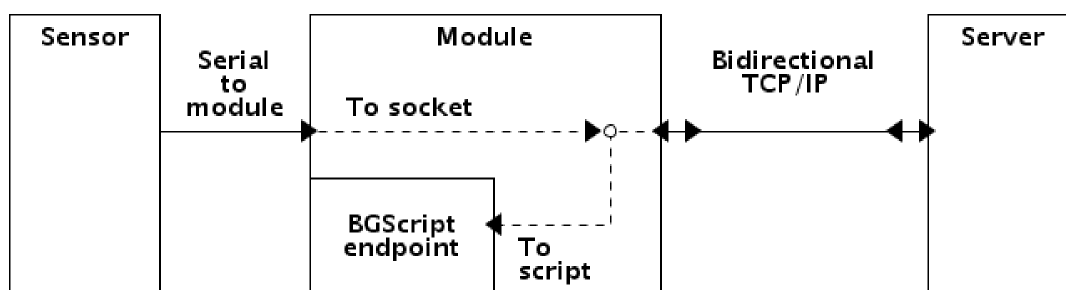

Figure 5: Example of routing a bi-directional stream between different endpoints. The serial interface has the socket configured as its endpoint, while the socket has the BGScript as its endpoint.

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

Copyright © 2000-2011 Bluegiga Technologies

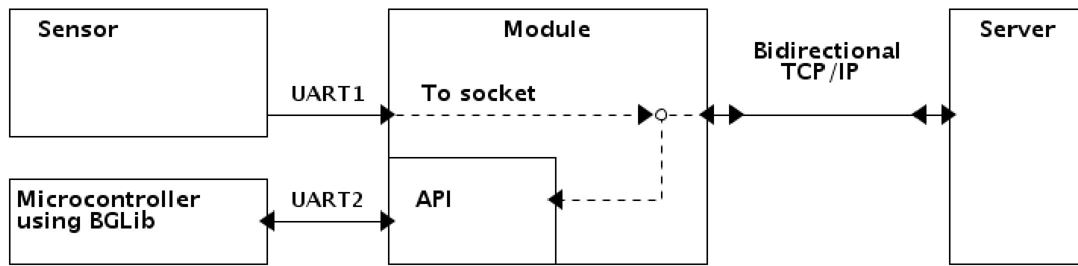

Figure 6: Example of routing a bi-directional stream between different endpoints

Copyright © 2000-2011 Bluegiga Technologies

3 API Definition

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

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

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

3.1 BGAPI protocol definition

Packet format

Packets in either direction use the following format.

Table 1: 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

Message types

The following message types exist in the BGAPI protocol.

Table 2: 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

Command Class IDs

The following command classes exist.

Table 3: BGAPI command classes

Class ID	Description	Explanation
0x00	System	Provides access to system functions
0x01	Persistent Store	Provides access the persistence store (parameters)
0x02	Attribute database	Provides access to local GATT database
0x03	Connection	Provides access to connection management functions
0x04	Attribute client	Functions to access remote devices GATT database
0x05	Security Manager	Bluetooth low energy security functions
0x06	Generic Access Profile	GAP functions
0x07	Hardware	Provides access to hardware such as timers and ADC

Packet Exchange

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

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

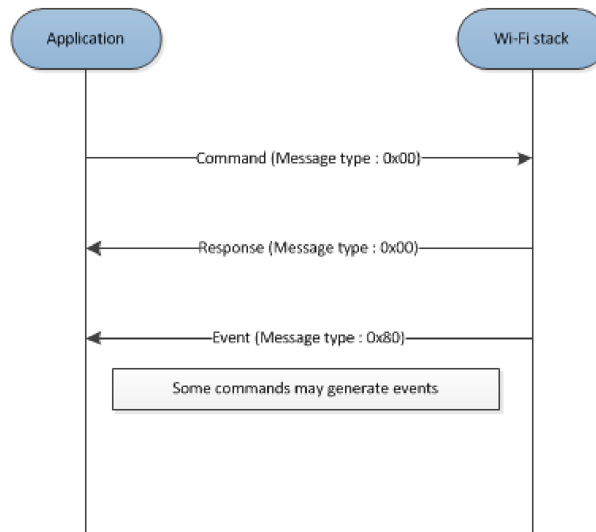

Figure 7: BGAPI messaging

Copyright © 2000-2011 Bluegiga Technologies

3.2 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.

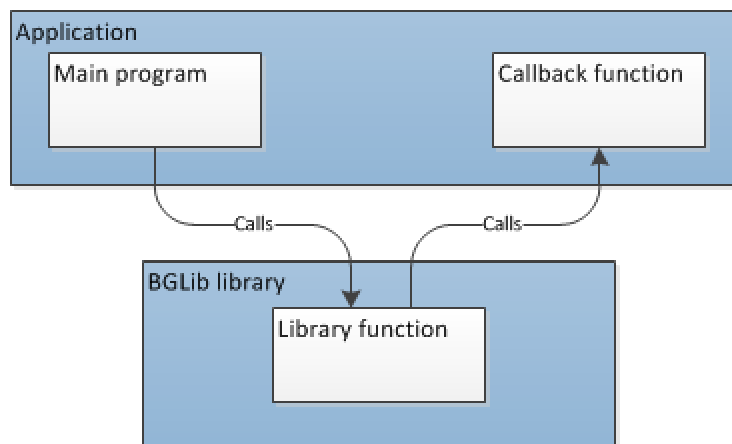

Figure 8: Callback arhitecture

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

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

Copyright © 2000-2011 Bluegiga Technologies

3.3 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.

3.4 Data types

Table 4: Data types used in the documentation

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

Copyright © 2000-2011 Bluegiga Technologies

4 API Reference

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

Description	Explanation
System	Various system functions, such as querying the hardware status or reset it
Configuration	Provides access to the devices parameters such as the MAC address
SME	Gives access to the TCP/IP stack and various protocols like TCP and UDP
TCP stack	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

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

Description
BGAPI errors
TCPIP errors

4.1 System

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

4.1.1 Commands

System commands

Hello

Hello command can be used to check the communication with the Wi-Fi software works. It's similar to AT-OK sequence.

Table 5: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x00	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x02	method	Message ID

Table 6: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x00	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x02	method	Message ID

C Functions

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

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

BGScript Functions

```
call system_hello()
```

Copyright © 2000-2011 Bluegiga Technologies

Reset

This command resets the local device immediately. The command does not have a response.

Table 7: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x01	method	Message ID
4	**uint8**	**dfu**	Selects the boot mode **0 : boot to main program** **1 : boot to DFU**

C Functions

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

BGScript Functions

```
call system_reset(dfu)
```

Copyright © 2000-2011 Bluegiga Technologies

Sync

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

Table 8: COMMAND

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

Table 9: RESPONSE

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

C Functions

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

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

BGScript Functions

```
call system_sync()
```

Copyright © 2000-2011 Bluegiga Technologies

4.1.2 Events

System events

Boot

Boot event tells the device has started and is ready to receive commands.

Table 10: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x04	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)
```

Copyright © 2000-2011 Bluegiga Technologies

State

These event are generated to expose the current status of the system.

Table 11: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x02	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x01	method	Message ID
4 - 5	**uint16**	**state**	Describes the current state of the module. See Main state enumeration for values.

C Functions

```
/* Callback */
struct wifi_msg_system_state_evt_t{
    uint16 state
}
void wifi_evt_system_state(
    const struct wifi_msg_system_state_evt_t * msg
)
```

BGScript Functions

```
event system_state(state)
```

4.1.3 Enumerations

System commands

Main state

States

Table 12: VALUES

Value	Name	Description
1	system_idle	Idle
2	system_powered	Wi-Fi powered and initialized
4	system_connecting	Connecting to AP
8	system_connected	Connected to AP

Copyright © 2000-2011 Bluegiga Technologies

4.2 Configuration

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

4.2.1 Commands

Configuration commands

Get Mac

This command reads the IEEE address from the device.

Table 13: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x02	class	Message class: Configuration
3	0x00	method	Message ID
4	**uint8**	**hw_interface**	The hardware interface to use **0 : Wi-Fi**

Table 14: RESPONSE

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

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

This command writes the IEEE address to the device.

Table 15: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x07	lolen	Minimum payload length
2	0x02	class	Message class: Configuration
3	0x01	method	Message ID
4	**uint8**	**hw_interface**	The hardware interface to use **0: Wi-Fi**
5 - 10	**hw_addr**	**mac**	The new MAC address to set

Table 16: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x02	class	Message class: Configuration
3	0x01	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**hw_interface**	Always zero for Wi-Fi client

C Functions

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

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

Copyright © 2000-2011 Bluegiga Technologies

BGScript Functions

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

4.2.2 Events

Configuration events

Mac Address

The current MAC address

Table 17: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x07	lolen	Minimum payload length
2	0x02	class	Message class: Configuration
3	0x00	method	Message ID
4	**uint8**	**hw_interface**	The hardware interface to use. **0: Wi-Fi**
5 - 10	**hw_addr**	**mac**	The current MAC address

C Functions

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

BGScript Functions

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

Copyright © 2000-2011 Bluegiga Technologies

4.3 SME

SME class commands provide functions to 802.11 operations like access point discovery and association.

4.3.1 Commands

Wi-Fi commands

Connect Bssid

Tries to connect to a specific Access Point using its unique BSSID.

Table 18: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x06	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x06	method	Message ID
4 - 9	**hw_addr**	**bssid**	The BSSID of the Access Point that the module should connect to

Table 19: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x09	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x06	method	Message ID
4 - 5	**uint16**	**result**	Result code **0: success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**hw_interface**	Zero for Wi-Fi client
7 - 12	**hw_addr**	**bssid**	The BSSID of the Access Point that the module will attempt to connect to

Copyright © 2000-2011 Bluegiga Technologies

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

Connects to an Access Point which belongs to the network identified with the given SSID.

Table 20: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x07	method	Message ID
4	**uint8array**	**ssid**	The SSID of the network to connect to

Table 21: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x09	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x07	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**hw_interface**	Zero for Wi-Fi client
7 - 12	**hw_addr**	**bssid**	The BSSID of the access point that will be connected to

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

Disconnects from the currently connected Access Point.

Table 22: COMMAND

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

Table 23: RESPONSE

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

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Set Password

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

Table 24: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x05	method	Message ID
4	**uint8array**	**password**	The password to be used when connecting to an Access Point

Table 25: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x05	method	Message ID
4	**uint8**	**status**	

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Start Scan

Initiates the access point scan procedure.

Table 26: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x03	method	Message ID
4	**uint8**	**handle**	The hardware interface to use. **0 : Wi-Fi**
5	**uint8array**	**chList**	The list of channels which will be scanned for Access Points

Table 27: RESPONSE

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

C Functions

```
/* Function */
void wifi_cmd_sme_start_scan(
    uint8 handle,
    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(handle, chList_len, chList_data)(result)
```

Copyright © 2000-2011 Bluegiga Technologies

Stop Scan

Terminates the active scanning procedure.

Table 28: COMMAND

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

Table 29: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x04	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Wifi Off

Turns off the 802.11 radio.

Table 30: COMMAND

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

Table 31: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x01	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Wifi On

Turns on the 802.11 radio.

Table 32: COMMAND

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

Table 33: RESPONSE

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

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)
```

Copyright © 2000-2011 Bluegiga Technologies

4.3.2 Events

Wi-Fi events

Connected

Event indicates a successful connection to an Access point.

Table 34: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x08	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x05	method	Message ID
4	**int8**	**status**	Zero indicates success
5	**uint8**	**hw_interface**	The hardware interface to use. **0 : Wi-Fi**
6 - 11	**hw_addr**	**bssid**	The BSSID of the device that the module connected to

C Functions

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

BGScript Functions

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

Connect Failed

Indicates a failed connection to an Access Point.

Table 35: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x08	method	Message ID
4 - 5	**uint16**	**reason**	
6	**uint8**	**hw_interface**	The hardware interface to use. **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)
```

Copyright © 2000-2011 Bluegiga Technologies

Connect Retry

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

Table 36: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x09	method	Message ID
4	**uint8**	**hw_interface**	The hardware interface to use. **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)
```

Copyright © 2000-2011 Bluegiga Technologies

Disconnected

Event indicates a disconnections from an Access Point.

Table 37: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x06	method	Message ID
4 - 5	**uint16**	**reason**	See error codes
6	**uint8**	**hw_interface**	The hardware interface to use. **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)
```

Copyright © 2000-2011 Bluegiga Technologies

Interface Status

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

Table 38: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x07	method	Message ID
4	**uint8**	**hw_interface**	The hardware interface to use **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)
```

Copyright © 2000-2011 Bluegiga Technologies

Scanned

This event indicates the Access Point scan has finished.

Table 39: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x04	method	Message ID
4	**int8**	**status**	Scan status **0 : scan finished normally**

C Functions

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

BGScript Functions

```
event sme_scanned(status)
```

Copyright © 2000-2011 Bluegiga Technologies

Scan Result Drop

Event indicates that the Access Point was dropped from the modules BSSID filtering list.

Table 40: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x06	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x03	method	Message ID
4 - 9	**hw_addr**	**bssid**	The BSSID of the Access Point that was dropped

C Functions

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

BGScript Functions

```
event sme_scan_result_drop(bssid)
```

Copyright © 2000-2011 Bluegiga Technologies

Scan Result

This event indicates Access Point scan results. After a scan has been started these events are printed every time an Access Point is discovered.

Table 41: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x0F	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
13	**int8**	**snr**	The signal to noise ratio of an Access Point
14	**uint8**	**secure**	Access Point security status **0 : open** **1 : secure**
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)
```

Copyright © 2000-2011 Bluegiga Technologies

Wifi Is Off

An event indicating the 802.11 radio has been powered off. This event can also indicate that the 802.11 radio had an internal error and was shut down.

Table 42: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x01	method	Message ID
4 - 5	**uint16**	**result**	Return code **0 : success** **non-zero : An error occurred** For error values refer to the error code documentation

C Functions

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

BGScript Functions

```
event sme_wifi_is_off(result)
```

Copyright © 2000-2011 Bluegiga Technologies

Wifi Is On

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

Table 43: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Wi-Fi
3	0x00	method	Message ID
4 - 5	**uint16**	**result**	Return code **0 : success** **non-zero : An error occurred** For error values refer to the error code documentation

C Functions

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

BGScript Functions

```
event sme_wifi_is_on(result)
```

Copyright © 2000-2011 Bluegiga Technologies

4.4 TCP stack

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

4.4.1 Commands

TCP stack commands

Configure

This command configure TCP/IP settings.

When using static IP addresses, this can be used to configure the local IP address, netmask and gateway. When using DHCP the settings for the static IP will be stored, but overridden once the dhcp server gives the module its address.

Table 44: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x0D	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x04	method	Message ID
4 - 7	**ipv4**	**address**	The local IP address of the device
8 - 11	**ipv4**	**netmask**	The netmask of the device
12 - 15	**ipv4**	**gateway**	The gateway
16	**uint8**	**use_dhcp**	Use DHCP **0 : Use static IP settings** **1 : DHCP is used to obtain an IP configuration**

Table 45: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x04	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Dns Configure

This command configure DNS settings.

Table 46: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x05	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x05	method	Message ID
4	**uint8**	**index**	Two different DNS servers can be stored. Index indicates which of the two this is. **0 : primary DNS server** **1 : secondary DNS server**
5 - 8	**ipv4**	**address**	The IP address of the DNS server

Table 47: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x05	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Dns Gethostbyname

This command start resolving a hostname into an IP address using the configured DNS servers.

Table 48: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x06	method	Message ID
4	**uint8array**	**name**	The name of the server whose IP address should be resolved, for example www.bluegiga.com.

Table 49: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x06	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Start Tcp Server

This command starts a TCP server.

Table 50: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x00	method	Message ID
4 - 5	**uint16**	**port**	The local port which to listen on.
6	**int8**	**default_destination**	When a new connection is made, that new connection will get its endpoint set to match the default specified here.

Table 51: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x00	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**endpoint**	The number of the endpoint the newly created TCP server uses

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Start Udp Server

This command starts a UDP server.

Table 52: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x02	method	Message ID
4 - 5	**uint16**	**port**	the local UDP port that the server listens on
6	**int8**	**default_destination**	The endpoint to which incoming UDP packets should be written

Table 53: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x02	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint number of the newly created UDP server

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Tcp Connect

This command attempt to create a new TCP connection.

Table 54: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x07	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x01	method	Message ID
4 - 7	**ipv4**	**address**	The IP address of the remote server to which the module should connect.
8 - 9	**uint16**	**port**	The TCP port on the remote server
10	**int8**	**routing**	The endpoint where the data from this TCP connection should be routed

Table 55: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x01	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint id of the newly created TCP connection

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
)
```

Copyright © 2000-2011 Bluegiga Technologies

BGScript Functions

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

Udp Connect

This command attempt to create a new UDP connection.

Table 56: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x07	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x03	method	Message ID
4 - 7	**ipv4**	**address**	The IP address of the remote server to which the packets should be sent
8 - 9	**uint16**	**port**	The UDP port of the remote server
10	**int8**	**routing**	The endpoint index where the data from this connection should be routed to

Table 57: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x04	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 - 7	**uint16**	**endpoint**	The endpoint index of the newly created UDP connection

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
)
```

Copyright © 2000-2011 Bluegiga Technologies

BGScript Functions

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

Copyright © 2000-2011 Bluegiga Technologies

4.4.2 Events

TCP stack events

Configuration

This event indicates TCP/IP configuration status.

Table 58: EVENT

Byte	Type	Name	Description
0	0x80	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 is not used** **1 = DHCP is 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)
```

Copyright © 2000-2011 Bluegiga Technologies

Dns Configuration

This event indicates DNS configuration status.

Table 59: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x05	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x01	method	Message ID
4	**uint8**	**index**	DNS server ID **0 = primary DNS server** **1 = secondary DNS server**
5 - 8	**ipv4**	**address**	The IP address of the DNS server

C Functions

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

BGScript Functions

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

Copyright © 2000-2011 Bluegiga Technologies

Dns Gethostbyname Result

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

Table 60: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x07	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x03	method	Message ID
4 - 5	**uint16**	**result**	Result code
6 - 9	**ipv4**	**address**	The resolved IP address of the server
10	**uint8array**	**name**	Name of the server whose IP address was resolved

C Functions

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

BGScript Functions

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

Endpoint Status

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

Table 61: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x0D	lolen	Minimum payload length
2	0x04	class	Message class: TCP stack
3	0x02	method	Message ID
4	**uint8**	**endpoint**	The endpoint index this message describes
5 - 8	**ipv4**	**local_ip**	The local IP address of this endpoint
9 - 10	**uint16**	**local_port**	The local port of this endpoint
11 - 14	**ipv4**	**remote_ip**	The remote IP address of this endpoint. For server endpoints (which have no client), this will not contain any valid value.
15 - 16	**uint16**	**remote_port**	The port of the remote device. For server endpoints (which have no client), this will not contain any valid value.

C Functions

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

BGScript Functions

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

Copyright © 2000-2011 Bluegiga Technologies

4.5 Endpoint

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

4.5.1 Commands

Endpoint commands

Close

This command can be used to close an endpoint.

Table 62: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x04	method	Message ID
4	**uint8**	**endpoint**	The index of the endpoint to close

Table 63: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x04	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint that the was closed

Copyright © 2000-2011 Bluegiga Technologies

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Send

This command sends data to a given endpoint.

Table 64: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x00	method	Message ID
4	**uint8**	**endpoint**	The index of the endpoint to which the data will be sent
5	**uint8array**	**data**	The RAW data which will be written or sent

Table 65: RESPONSE

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

C Functions

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

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

BGScript Functions

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

Set Active

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 incoming data from an endpoint by deactivating it. Server endpoints however are never active, as they can neither send nor receive data.

This currently only works on TCP and UDP endpoints.

Table 66: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x02	method	Message ID
4	**uint8**	**endpoint**	The endpoint to control
5	**uint8**	**active**	Endpoint status **0 : inactive** **1 : active**

Table 67: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x02	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint that was controlled

Copyright © 2000-2011 Bluegiga Technologies

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

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

Table 68: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x03	method	Message ID
4	**uint8**	**endpoint**	The endpoint which to control
5	**int8**	**streaming_destination**	The destination for the data

Table 69: RESPONSE

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

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Set Streaming

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

This setting currently only operates on UART endpoints.

Table 70: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x01	method	Message ID
4	**uint8**	**endpoint**	The endpoint whose streaming mode should be changed
5	**uint8**	**streaming**	Endpoint mode **0 : Use as BGAPI interface** **1 : Streaming to another endpoint**

Table 71: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x01	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint this message refers to

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
)
```

Copyright © 2000-2011 Bluegiga Technologies

BGScript Functions

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

Copyright © 2000-2011 Bluegiga Technologies

4.5.2 Events

Endpoint events

Closing

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 72: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x03	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x03	method	Message ID
4 - 5	**uint16**	**reason**	Zero indicates success. For other values refer to the error code documentation
6	**uint8**	**endpoint**	The endpoint which is closing

C Functions

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

BGScript Functions

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

Copyright © 2000-2011 Bluegiga Technologies

Data

This event indicates incoming data from an endpoint.

Table 73: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x02	lolen	Minimum payload length
2	0x05	class	Message class: Endpoint
3	0x01	method	Message ID
4	**uint8**	**endpoint**	The endpoint which received this data, i.e. to which it was sent
5	**uint8array**	**data**	The raw data

C Functions

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

BGScript Functions

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

Status

This event indicates an endpoint's status.

Table 74: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x05	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	**uint8**	**type**	The type of endpoint, see the endpoint type enumeration
6	**uint8**	**streaming**	Endpoint mode **0 : Endpoint is connected to BGAPI** **1 : Endpoint is streaming to another endpoint**
7	**int8**	**destination**	The index of the endpoint to which the incoming data goes
8	**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,
    uint8 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)
```

Copyright © 2000-2011 Bluegiga Technologies

4.5.3 Enumerations

Endpoint commands

Endpoint types

Endpoint types

Table 75: 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

Copyright © 2000-2011 Bluegiga Technologies

4.6 Hardware

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

4.6.1 Commands

Hardware commands

Change Notification Config

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

Table 76: Change notification bits

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 77: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x04	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x02	method	Message ID
4 - 7	**uint32**	**enable**	Change notification bits to enable

Table 78: RESPONSE

Copyright © 2000-2011 Bluegiga Technologies

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

Table 79: EVENTS

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

C Functions

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

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

BGScript Functions

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

Copyright © 2000-2011 Bluegiga Technologies

Change Notification Pullup

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

Table 80: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x04	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x03	method	Message ID
4 - 7	**uint32**	**pullup**	Bitmask for which of the change notification pins have pull-ups enabled

Table 81: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x03	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

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

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

BGScript Functions

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

External Interrupt Config

This command configures pins which will generate interrupts.

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

For example to enable the interrupt on pin RD11, the bit should be set 0x10 in the enable bitfield.

Table 82: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x01	method	Message ID
4	**uint8**	**enable**	External interrupt bits to enable
5	**uint8**	**polarity**	External interrupt polarity. Each bit in the bitmask have the following meansing: **0 : falling edge** **1 : rising edge**

Table 83: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x01	method	Message ID
4 - 5	**uint16**	**result**	ZResult code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table 84: EVENTS

Event	Description
hardware external_interrupt	Sent after external interrupt detected.

C Functions

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

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

BGScript Functions

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

Io Port Config Direction

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

Table 85: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x05	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x04	method	Message ID
4	**uint8**	**port**	Port index **0 : A** **1 : B** **2 : C** **3 : D** **4 : E** **5 : F** **6 : G**
5 - 6	**uint16**	**mask**	Bitmask of which pins on the port this command affects.
7 - 8	**uint16**	**direction**	The bitmask describing which are inputs and which are outputs. **0 : Output** **1 : Input**

Table 86: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x04	method	Message ID
4 - 5	**uint16**	**result**	Return code. **0 : Success** **non-zero : An error occurred** For other values refer to the error code documentation

Copyright © 2000-2011 Bluegiga Technologies

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Io Port Config Open Drain

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

Table 87: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x05	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x05	method	Message ID
4	**uint8**	**port**	Port index **0 : A** **1 : B** **2 : C** **3 : D** **4 : E** **5 : F** **6 : G**
5 - 6	**uint16**	**mask**	Bitmask of which pins on the port this command affects
7 - 8	**uint16**	**open_drain**	Bitmask of which pins are configured to be open drain. For each bit this means: **0 : Open drain disabled** **1 : Open drain enabled**

Table 88: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x05	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Copyright © 2000-2011 Bluegiga Technologies

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Io Port Read

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

Table 89: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x07	method	Message ID
4	**uint8**	**port**	Port index: **0 : A** **1 : B** **2 : C** **3 : D** **4 : E** **5 : F** **6 : G**
5 - 6	**uint16**	**mask**	Bitmask of which pins on the port should be read. For each bit in the bitmask: **0 : Don't read** **1 : Read**

Table 90: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x05	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x07	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation
6	**uint8**	**port**	Port index
7 - 8	**uint16**	**data**	port data

Copyright © 2000-2011 Bluegiga Technologies

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Io Port Write

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

Table 91: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x05	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x06	method	Message ID
4	**uint8**	**port**	Port index: **0 : A** **1 : B** **2 : C** **3 : D** **4 : E** **5 : F** **6 : G**
5 - 6	**uint16**	**mask**	Bitmask of which pins on the port this command affects. For each bit in the bitmask: 0 = Don't modify/write, 1 = modify/write
7 - 8	**uint16**	**data**	Bitmask of which pins to set. For each bit in the bitmask: **0 : low** **1 : high**

Table 92: RESPONSE

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

Copyright © 2000-2011 Bluegiga Technologies

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)
```

Set Soft Timer

This command enables the software timer. Only one timer can be running at a time.

Table 93: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x06	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x00	method	Message ID
4 - 7	**uint32**	**time**	Interval between how often to send events, in milliseconds. The resolution is approximately 10ms. **Range:**
8	**uint8**	**handle**	Handle that is returned with event
9	**uint8**	**single_shot**	Continuous timer **0 : false** **1 : true**

Table 94: RESPONSE

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

Table 95: EVENTS

Event	Description
hardware soft_timer	Sent after specified interval

Copyright © 2000-2011 Bluegiga Technologies

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)
```

4.6.2 Events

Hardware events

Change Notification

This is a change notification event.

Table 96: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x04	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x01	method	Message ID
4 - 7	**uint32**	**timestamp**	Timestamp of when the change occurred

C Functions

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

BGScript Functions

```
event hardware_change_notification(timestamp)
```

External Interrupt

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

Table 97: GPIO interrupts

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

Table 98: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x05	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x02	method	Message ID
4	**uint8**	**irq**	IRQ index
5 - 8	**uint32**	**timestamp**	Timestamp of when the interrupt ocurred

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Soft Timer

This event indicates software timer has elapsed.

Table 99: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x01	lolen	Minimum payload length
2	0x06	class	Message class: Hardware
3	0x00	method	Message ID
4	**uint8**	**handle**	Timer handle

C Functions

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

BGScript Functions

```
event hardware_soft_timer(handle)
```

Copyright © 2000-2011 Bluegiga Technologies

4.7 Persistent Store

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

4.7.1 Commands

Persistent Store commands

Ps Defrag

This command defragments the persistent store.

Table 100: COMMAND

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

Table 101: RESPONSE

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

C Functions

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

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

Copyright © 2000-2011 Bluegiga Technologies

BGScript Functions

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

Ps Dump

This command dumps all the PS keys from the persistent store. The command will generate a series of PS key events. The last PS Key event is identified by having the key index 65535, indicating the dump has finished.

Table 102: COMMAND

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

Table 103: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x01	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

Table 104: 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

This command erases all PS keys from the persistent store.

This will erase the device's MAC address!

Table 105: COMMAND

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

Table 106: RESPONSE

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

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Ps Erase

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

Table 107: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x05	method	Message ID
4 - 5	**uint16**	**key**	Key index to erase

Table 108: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x05	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

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)
```

Copyright © 2000-2011 Bluegiga Technologies

Ps Load

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

Table 109: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x04	method	Message ID
4 - 5	**uint16**	**key**	Key index to load

Table 110: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x04	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** ie: **0x0186 : Unspecified error, when trying to read a non-existent key** For other values refer to the error code documentation
6	**uint8array**	**value**	Key value, i.e. the data stored

C Functions

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

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

BGScript Functions

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

Copyright © 2000-2011 Bluegiga Technologies

Ps Save

This command stores a value to the given PS key into the persistent store.

Table 111: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x03	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x03	method	Message ID
4 - 5	**uint16**	**key**	Key index
6	**uint8array**	**value**	Key value

Table 112: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x03	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

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)
```

Copyright © 2000-2011 Bluegiga Technologies

4.7.2 Events

Persistent Store events

Ps Key

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

Table 113: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x03	lolen	Minimum payload length
2	0x07	class	Message class: Persistent Store
3	0x00	method	Message ID
4 - 5	**uint16**	**key**	Key index **65535 : Last key**
6	**uint8array**	**value**	Key value

C Functions

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

BGScript Functions

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

Copyright © 2000-2011 Bluegiga Technologies

4.8 Device Firmware Upgrade

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

4.8.1 Commands

Device Firmware Upgrade commands

Flash Set Address

This command set address on the flash memory for writing data.

Table 114: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x04	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x01	method	Message ID
4 - 7	**uint32**	**address**	The offset in the flash where to start flashing.

Table 115: RESPONSE

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x02	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x01	method	Message ID
4 - 5	**uint16**	**result**	Result code **0 : success** **Non-zero : an error occurred** For other values refer to the error code documentation

C Functions

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

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

Copyright © 2000-2011 Bluegiga Technologies

BGScript Functions

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

Copyright © 2000-2011 Bluegiga Technologies

Flash Upload

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

Table 116: COMMAND

Byte	Type	Name	Description
0	0x00	hilen	Message type: command
1	0x01	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x02	method	Message ID
4	**uint8array**	**data**	An array of data which will be written into the flash.

Table 117: RESPONSE

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

C Functions

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

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

BGScript Functions

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

Flash Upload Finish

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

Table 118: COMMAND

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

Table 119: RESPONSE

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

C Functions

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

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

BGScript Functions

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

Copyright © 2000-2011 Bluegiga Technologies

Reset

This command resets the device.

Table 120: COMMAND

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

C Functions

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

BGScript Functions

```
call dfu_reset(dfu)
```

Copyright © 2000-2011 Bluegiga Technologies

4.8.2 Events

Device Firmware Upgrade events

Boot

This event indicates the device booted in DFU mode, and is ready to receive commands.

Table 121: EVENT

Byte	Type	Name	Description
0	0x80	hilen	Message type: event
1	0x04	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x00	method	Message ID
4 - 7	**uint32**	**version**	The vesion 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)
```

Copyright © 2000-2011 Bluegiga Technologies

4.9 Error codes

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

4.9.1 BGAPI Errors

Errors related to BGAPI protocol

Invalid Parameter (0x0180)

Command contained invalid parameter

Device in Wrong State (0x0181)

Device is in wrong state to accept command

Out Of Memory (0x0182)

Device has run out of memory

Feature Not Implemented (0x0183)

Feature is not implemented

Command Not Recognized (0x0184)

Command was not recognized

Timeout (0x0185)

Command or Procedure failed due to timeout

Unspecified error (0x0186)

Unspecified error

4.9.2 TCP IP Errors

Errors related to TCP/IP stack

Success (0x0200)

No error

Out of memory (0x0201)

Out of memory

Buffer error (0x0202)

Buffer handling failed

Timeout (0x0203)

Timeout

Copyright © 2000-2011 Bluegiga Technologies

Routing (0x0204)

Could not find route

Address in use (0x0208)

Address in use

Already connected (0x0209)

Already connected

Connection aborted (0x020A)

Connection aborted

Connection reset (0x020B)

Connection reset

Connection closed (0x020C)

Connection closed

Not connected (0x020D)

Not connected

Interface level error (0x020F)

Interface error

Copyright © 2000-2011 Bluegiga Technologies

5 Contact information

Sales:	sales@bluegiga.com
Technical support:	support@bluegiga.com http://techforum.bluegiga.com
Orders:	orders@bluegiga.com
WWW:	http://www.bluegiga.com http://www.bluegiga.hk
Head Office / Finland:	Phone: +358-9-4355 060 Fax: +358-9-4355 0660 Sinikalliontie 5 A 02630 ESPOO FINLAND
Head address / Finland:	P.O. Box 120 02631 ESPOO FINLAND
Sales Office / USA:	Phone: +1 770 291 2181 Fax: +1 770 291 2183 Bluegiga Technologies, Inc. 3235 Satellite Boulevard, Building 400, Suite 300 Duluth, GA, 30096, USA
Sales Office / Hong-Kong:	Phone: +852 3182 7321 Fax: +852 3972 5777 Bluegiga Technologies, Inc. 19/F Silver Fortune Plaza, 1 Wellington Street, Central Hong Kong

Copyright © 2000-2011 Bluegiga Technologies