AN915: BGM111 API Reference Guide

This document contains the full API reference for the Blue Gecko BGM111 Bluetooth Smart module.

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

The ultra-low power operating modes and fast wake-up times of the Silicon Labs' energy friendly 32-bit MCUs, combined with the low transmit and receive power consumption of the Bluetooth radio, result in a solution optimized for battery powered applications.

The Silicon Labs' fully certified Bluetooth Smart modules and software are designed to help developers accelerate time to market and reduce development costs and compliance risks by providing a versatile, plug-and-play Bluetooth solution.

1. API Reference

The following sections contain the full API reference for the Blue Gecko BGM111 Bluetooth Smart module.

1.1 Device Firmware Upgrade (dfu)

DFU class commands can be used to perform a firmware update over the configured host interface. These commands and events are only available when the module has been booted into DFU mode. **The DFU process is as follows:**

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

1.1.1 cmd_dfu_flash_set_address

After re-booting the local device into DFU mode, this command can be used to define the starting address on the flash where the new firmware will be written in.

Table 1.1. Command

Byte	Type	Name	Description
0	0x20	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 the new firmware is uploaded to. • **0x0000:** Always use the value 0x0000

Table 1.2. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x01	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_dfu_flash_set_address(uint32 address);

/* Callback */
struct gecko_msg_dfu_flash_set_address_rsp_t
{
  uint16 result
}
void gecko_rsp_dfu_flash_set_address(
  const struct gecko_msg_dfu_flash_set_address_rsp_t *msg
)
```

1.1.2 cmd_dfu_flash_upload

This command is used to upload the firmware update file to the Bluetooth module. The payload of the command is 128 bytes, so multiple commands need to be used to upload the full firmware image file.

Table 1.3. Command

Byte	Type	Name	Description
0	0x20	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 up to 128 bytes which will be written into the flash.

Table 1.4. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x02	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_dfu_flash_upload(uint8 data_len, const uint8 *data_data);

/* Callback */
struct gecko_msg_dfu_flash_upload_rsp_t
{
  uint16 result
}
void gecko_rsp_dfu_flash_upload(
  const struct gecko_msg_dfu_flash_upload_rsp_t *msg
)
```

1.1.3 cmd_dfu_flash_upload_finish

This command can be used to tell to the device that the DFU file has been fully uploaded. To return the device back to normal mode the command cmd_dfu_reset must be issued next.

Table 1.5. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x03	method	Message ID

Table 1.6. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x03	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_dfu_flash_upload_finish();

/* Callback */
struct gecko_msg_dfu_flash_upload_finish_rsp_t
{
  uint16 result
}
void gecko_rsp_dfu_flash_upload_finish(
  const struct gecko_msg_dfu_flash_upload_finish_rsp_t *msg
)
```

1.1.4 cmd_dfu_reset

This command can be used to reset the system. This command does not have a response, but it triggers one of the boot events (normal reset or boot to DFU mode) after re-boot.

Table 1.7. Command

Byte	Type	Name	Description
0	0x20	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

BGScript command

```
call dfu_reset(dfu)
```

BGLIB C API

```
/* Function */

void gecko_cmd_dfu_reset(uint8 dfu);

/* Command does not have callback */
```

Table 1.8. Events generated

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

1.1.5 evt_dfu_boot

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

Table 1.9. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x04	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x00	method	Message ID
4-7	uint32	version	The version of the bootloader

BGScript event

```
event dfu_boot(version)
```

C Functions

```
/* Callback */
struct gecko_msg_dfu_boot_evt_t
{
  uint32 version
}
void gecko_rsp_dfu_boot(
  const struct gecko_msg_dfu_boot_evt_t *msg
)
```

1.2 Endpoint (endpoint)

Endpoint class functions are used to control endpoints. They allow the creation and deletion of endpoints as well as data routing configuration.

1.2.1 cmd_endpoint_close

This command can be used to close an endpoint.

Table 1.10. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0b	class	Message class: Endpoint
3	0x02	method	Message ID
4	uint8	endpoint	The index of the endpoint to closeEndpoint index is provided by events which indicate Bluetooth connection establishment.

Table 1.11. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x0b	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 codes
6	uint8	endpoint	The endpoint that was closed

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_endpoint_close(uint8 endpoint);

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

Table 1.12. Events generated

Event	Description
endpoint_status	Sent when endpoint status changes

1.2.2 cmd_endpoint_clr_flags

This command can be used to clear endpoint flags.

Table 1.13. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x0b	class	Message class: Endpoint
3	0x04	method	Message ID
4	uint8	endpoint	Endpoint of the connection
5-8	uint32	flags	Flags to be cleared

Table 1.14. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x0b	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 codes
6	uint8	endpoint	

BGScript command

```
call endpoint_clr_flags(endpoint,flags)(result,endpoint)
```

BGLIB C API

```
/* Function */

void gecko_cmd_endpoint_clr_flags(uint8 endpoint, uint32 flags);

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

Table 1.15. Events generated

Event	Description
endpoint_status	Sent when endpoint status changes

1.2.3 cmd_endpoint_read_counters

This command can be used to read the data performance counters (data sent counter and data received counter) of an endpoint.

Table 1.16. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0b	class	Message class: Endpoint
3	0x05	method	Message ID
4	uint8	endpoint	Endpoint of the connection

Table 1.17. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x0b	lolen	Minimum payload length
2	0x0b	class	Message class: Endpoint
3	0x05	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes
6	uint8	endpoint	
7-10	uint32	tx	Amount of data sent to this endpoint
11-14	uint32	rx	Amount of data received from this endpoint

BGScript command

```
call endpoint_read_counters(endpoint)(result,endpoint,tx,rx)
```

BGLIB C API

```
/* Function */

void gecko_cmd_endpoint_read_counters(uint8 endpoint);

/* Callback */
struct gecko_msg_endpoint_read_counters_rsp_t
{
  uint16 result,
  uint8 endpoint,
  uint32 tx,
  uint32 rx
}
void gecko_rsp_endpoint_read_counters(
  const struct gecko_msg_endpoint_read_counters_rsp_t *msg
)
```

1.2.4 cmd_endpoint_send

This command can be used to send data to the defined endpoint.

Table 1.18. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x0b	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 1.19. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x0b	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 codes
6	uint8	endpoint	The endpoint to which the data was written

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_endpoint_send(uint8 endpoint, uint8 data_len, const uint8 *data_data);

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

1.2.5 cmd_endpoint_set_flags

This command can be used to set endpoint flags to control and/or indicate in which mode the endpoint connection is operating.

Table 1.20. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x0b	class	Message class: Endpoint
3	0x03	method	Message ID
4	uint8	endpoint	The endpoint which to control
5-8	uint32	flags	Flags to be set

Table 1.21. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x0b	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 codes
6	uint8	endpoint	

BGScript command

```
call endpoint_set_flags(endpoint,flags)(result,endpoint)
```

BGLIB C API

```
/* Function */

void gecko_cmd_endpoint_set_flags(uint8 endpoint, uint32 flags);

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

Table 1.22. Events generated

Event	Description
endpoint_status	Sent when endpoint status changes

1.2.6 cmd_endpoint_set_streaming_destination

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

Table 1.23. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x0b	class	Message class: Endpoint
3	0x01	method	Message ID
4	uint8	endpoint	The endpoint which to control
5	uint8	destination_endpoint	The destination for the data

Table 1.24. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x0b	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 codes
6	uint8	endpoint	Endpoint of the connection

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_endpoint_set_streaming_destination(uint8 endpoint, uint8 destination_endpoint);

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

Table 1.25. Events generated

Event	Description
endpoint_status	Sent when endpoint status changes

1.2.7 evt_endpoint_closing

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

Table 1.26. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x0b	class	Message class: Endpoint
3	0x03	method	Message ID
4-5	uint16	reason	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes
6	uint8	endpoint	Endpoint handleValue0xff: connection failed without associated endpoint

BGScript event

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

C Functions

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

1.2.8 evt_endpoint_data

This event indicates incoming data from an endpoint.

Table 1.27. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x02	lolen	Minimum payload length
2	0x0b	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

BGScript event

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

C Functions

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

1.2.9 evt_endpoint_status

This event indicates an endpoint's status.

Table 1.28. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x07	lolen	Minimum payload length
2	0x0b	class	Message class: Endpoint
3	0x02	method	Message ID
4	uint8	endpoint	The index of the endpoint whose status this event describes
5-8	uint32	type	Unsigned 32bit integer
9	int8	destination_endpoint	The index of the endpoint to which the incoming data goes.
10	uint8	flags	Flags which indicate the mode of endpoint

BGScript event

```
event endpoint_status(endpoint,type,destination_endpoint,flags)
```

C Functions

```
/* Callback */
struct gecko_msg_endpoint_status_evt_t
{
  uint8 endpoint,
  uint32 type,
  int8 destination_endpoint,
  uint8 flags
}
void gecko_rsp_endpoint_status(
  const struct gecko_msg_endpoint_status_evt_t *msg
)
```

1.2.10 evt_endpoint_syntax_error

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

Table 1.29. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x0b	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 codes
6	uint8	endpoint	Endpoint of the connection

BGScript event

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

C Functions

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

1.2.11 enum_endpoint_types

These values define the endpoint types. Predefined endpoint ID's in BT121 are:

- **0:** UART
- **1:** SCRIPT
- **3:** SPI1
- **4:** SPI2
- **31:** DROP

Table 1.30. Enumerations

Value	Name	Description
0	endpoint_free	Endpoint is not in use
1	endpoint_uart	UART
2	endpoint_script	Scripting
4	endpoint_reserved	Reserved for future use
16	endpoint_drop	Drop all data sent to this endpoint
32	endpoint_rfcomm	RFCOMM channel
64	endpoint_spi	SPI
128	endpoint_connection	Connection
256	endpoint_native	Endpoint used for native interface
512	endpoint_iap	iAP

1.2.12 define_endpoint_endpoint_flags

Table 1.31. Defines

Value	Name	Description
1	endpoint_FLAG_UPDATED	endpoint status has been changed since last indication
2	endpoint_FLAG_ACTIVE	endpoint is active and can send and receive data
4	endpoint_FLAG_STREAMING	endpoint is in streaming mode. Data is sent and received from endpoint without framing
8	endpoint_FLAG_BGAPI	endpoint is configured for BGAPI. Data received is parsed as BGAPI commands. Also all BGAPI events and responses are sent to this endpoint
16	endpoint_FLAG_WAIT_CLOSE	endpoint is closed from the device side. Host needs to acknowledge by sending endpoint_close command to device with this endpoint as parameter
32	endpoint_FLAG_CLOSING	endpoint is closed from the host side and is waiting for the device to acknowledge closing of the endpoint

1.3 Persistent Store (flash)

Persistent Store commands can be used to manage the user data in the flash memory of the Bluetooth module. User data stored in PS keys within the flash memory is persistent across reset and power cycling of the module.

1.3.1 cmd_flash_ps_dump

This command can be used to retrieve all PS keys and their current values. For each existing PS key a flash_pskey event will be generated which includes the corresponding PS key value.

Table 1.32. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x0d	class	Message class: Persistent Store
3	0x00	method	Message ID

Table 1.33. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0d	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 codes

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_flash_ps_dump();

/* Callback */
struct gecko_msg_flash_ps_dump_rsp_t
{
  uint16 result
}
void gecko_rsp_flash_ps_dump(
  const struct gecko_msg_flash_ps_dump_rsp_t *msg
)
```

Table 1.34. Events generated

Event	Description
flash_ps_key	PS Key contents

1.3.2 cmd_flash_ps_erase

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

Table 1.35. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x0d	class	Message class: Persistent Store
3	0x04	method	Message ID
4-5	uint16	key	PS key to erase

Table 1.36. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0d	class	Message class: Persistent Store
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 codes

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_flash_ps_erase(uint16 key);

/* Callback */
struct gecko_msg_flash_ps_erase_rsp_t
{
  uint16 result
}
void gecko_rsp_flash_ps_erase(
  const struct gecko_msg_flash_ps_erase_rsp_t *msg
)
```

1.3.3 cmd_flash_ps_erase_all

This command can be used to erase all PS keys and their corresponding value.

Table 1.37. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x0d	class	Message class: Persistent Store
3	0x01	method	Message ID

Table 1.38. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0d	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 codes

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_flash_ps_erase_all();

/* Callback */
struct gecko_msg_flash_ps_erase_all_rsp_t
{
  uint16 result
}
void gecko_rsp_flash_ps_erase_all(
  const struct gecko_msg_flash_ps_erase_all_rsp_t *msg
)
```

1.3.4 cmd_flash_ps_load

This command can be used for retrieving the value of the specified PS key.

Table 1.39. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x0d	class	Message class: Persistent Store
3	0x03	method	Message ID
4-5	uint16	key	PS key of the value to be retrieved

Table 1.40. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x0d	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 codes
6	uint8array	value	The returned value of the specified PS key.

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_flash_ps_load(uint16 key);

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

1.3.5 cmd_flash_ps_save

This command can be used to store a value into the specified PS key.

Table 1.41. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x0d	class	Message class: Persistent Store
3	0x02	method	Message ID
4-5	uint16	key	PS key
6	uint8array	value	Value to store into the specified PS key.

Table 1.42. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0d	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 codes

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_flash_ps_save(uint16 key, uint8 value_len, const uint8 *value_data);

/* Callback */
struct gecko_msg_flash_ps_save_rsp_t
{
  uint16 result
}
void gecko_rsp_flash_ps_save(
  const struct gecko_msg_flash_ps_save_rsp_t *msg
)
```

1.3.6 evt_flash_ps_key

This event indicates that the flash_ps_dump command was given. It returns a single PS key with it's corresponding value. There can be multiple events if multiple PS keys exist.

Table 1.43. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x0d	class	Message class: Persistent Store
3	0x00	method	Message ID
4-5	uint16	key	PS key
6	uint8array	value	Current value of the PS key specified in this event.

BGScript event

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

C Functions

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

1.4 Generic Attribute Profile (gatt)

1.4.1 cmd_gatt_discover_characteristics

This command can be used to discover all characteristics of the defined GATT service from a remote GATT database. This command generates a unique gatt_characteristic event for every discovered characteristic. Received gatt_procedure_completed event indicates that this GATT procedure has succesfully completed or failed with error.

Table 1.44. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x03	method	Message ID
4	uint8	connection	Connection handle
5-8	uint32	service	GATT service handleThis value is normally received from the gatt_service event.

Table 1.45. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
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 codes

BGScript command

```
call gatt_discover_characteristics(connection,service)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_discover_characteristics(uint8 connection, uint32 service);

/* Callback */
struct gecko_msg_gatt_discover_characteristics_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_discover_characteristics(
  const struct gecko_msg_gatt_discover_characteristics_rsp_t *msg
)
```

Table 1.46. Events generated

Event	Description
gatt_characteristic	Discovered characteristic from remote GATT database.

Event	Description
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

1.4.2 cmd_gatt_discover_characteristics_by_uuid

This command can be used to discover all the characteristics of the specified GATT service in a remote GATT database having the specified UUID. This command generates a unique gatt_characteristic event for every discovered characteristic having the specified UUID. Received gatt_procedure_completed event indicates that this GATT procedure has successfully completed or failed with error.

Table 1.47. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x06	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x04	method	Message ID
4	uint8	connection	Connection handle
5-8	uint32	service	GATT service handleThis value is normally received from the gatt_service event.
9	uint8array	uuid	Characteristic UUID

Table 1.48. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
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 codes

BGScript command

```
call gatt_discover_characteristics_by_uuid(connection,service,uuid_len, uuid_data)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_discover_characteristics_by_uuid(uint8 connection, uint32 service, uint8 uuid_len, const ui
nt8 *uuid_data);

/* Callback */
struct gecko_msg_gatt_discover_characteristics_by_uuid_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_discover_characteristics_by_uuid(
  const struct gecko_msg_gatt_discover_characteristics_by_uuid_rsp_t *msg
)
```

Table 1.49. Events generated

Event	Description
gatt_characteristic	Discovered characteristic from remote GATT database.
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

1.4.3 cmd_gatt_discover_descriptors

This command can be used to discover all the descriptors of the specified remote GATT characteristics in a remote GATT database. This command generates a unique gatt_descriptor event for every discovered descriptor. Received gatt_procedure_completed event indicates that this GATT procedure has succesfully completed or failed with error.

Table 1.50. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x06	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event

Table 1.51. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
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 codes

BGScript command

```
call gatt_discover_descriptors(connection,characteristic)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_discover_descriptors(uint8 connection, uint16 characteristic);

/* Callback */
struct gecko_msg_gatt_discover_descriptors_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_discover_descriptors(
  const struct gecko_msg_gatt_discover_descriptors_rsp_t *msg
)
```

Table 1.52. Events generated

Event	Description
gatt_descriptor	Discovered descriptor from remote GATT database.

Event	Description
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

1.4.4 cmd_gatt_discover_primary_services

This command can be used to discover all the primary services of a remote GATT database. This command generates a unique gatt_service event for every discovered primary service. Received gatt_procedure_completed event indicates that this GATT procedure has succesfully completed or failed with error.

Table 1.53. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x01	method	Message ID
4	uint8	connection	Connection handle

Table 1.54. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
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 codes

BGScript command

```
call gatt_discover_primary_services(connection)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_discover_primary_services(uint8 connection);

/* Callback */
struct gecko_msg_gatt_discover_primary_services_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_discover_primary_services(
  const struct gecko_msg_gatt_discover_primary_services_rsp_t *msg
)
```

Table 1.55. Events generated

Event	Description
gatt_service	Discovered service from remote GATT database
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

1.4.5 cmd_gatt_discover_primary_services_by_uuid

This command can be used to discover primary services with the specified UUID in a remote GATT database. This command generates unique gatt_service event for every discovered primary service. Received gatt_procedure_completed event indicates that this GATT procedure has succesfully completed or failed with error.

Table 1.56. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x02	method	Message ID
4	uint8	connection	Connection handle
5	uint8array	uuid	Characteristic UUID

Table 1.57. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
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 codes

BGScript command

```
call gatt_discover_primary_services_by_uuid(connection,uuid_len, uuid_data)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_discover_primary_services_by_uuid(uint8 connection, uint8 uuid_len, const uint8 *uuid_data)
;

/* Callback */
struct gecko_msg_gatt_discover_primary_services_by_uuid_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_discover_primary_services_by_uuid(
  const struct gecko_msg_gatt_discover_primary_services_by_uuid_rsp_t *msg
)
```

Table 1.58. Events generated

Event	Description
gatt_service	Discovered service from remote GATT database.

Event	Description
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

1.4.6 cmd_gatt_execute_characteristic_value_write

This command can be used to commit or cancel previously queued writes to a long characteristic of a remote GATT server. Writes are sent to queue with prepare_characteristic_value_write command. Content, offset and length of queued values are validated by this procedure. A received gatt_procedure_completed event indicates that all data has been written succesfully or that an error response has been received.

Table 1.59. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x0c	method	Message ID
4	uint8	connection	Connection handle
5	uint8	flags	Unsigned 8bit integer

Table 1.60. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x0c	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call gatt_execute_characteristic_value_write(connection,flags)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_execute_characteristic_value_write(uint8 connection, uint8 flags);

/* Callback */
struct gecko_msg_gatt_execute_characteristic_value_write_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_execute_characteristic_value_write(
  const struct gecko_msg_gatt_execute_characteristic_value_write_rsp_t *msg
)
```

Table 1.61. Events generated

Event	Description
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

1.4.7 cmd_gatt_find_included_services

This command can be used to find out if a service of a remote GATT database includes one or more other services. This command generates a unique gatt_service_completed event for each included service. This command generates a unique gatt_service event for every discovered service. Received gatt_procedure_completed event indicates that this GATT procedure has successfully completed or failed with error.

Table 1.62. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x10	method	Message ID
4	uint8	connection	Connection handle
5-8	uint32	service	GATT service handleThis value is normally received from the gatt_service event.

Table 1.63. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x10	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call gatt_find_included_services(connection,service)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_find_included_services(uint8 connection, uint32 service);

/* Callback */
struct gecko_msg_gatt_find_included_services_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_find_included_services(
  const struct gecko_msg_gatt_find_included_services_rsp_t *msg
)
```

Table 1.64. Events generated

Event	Description
gatt_service	Discovered service from remote GATT database.
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

1.4.8 cmd_gatt_prepare_characteristic_value_write

This command can be used to add a characteristic value to the write queue of a remote GATT server. More specifically, this command can be used in those special cases where very long attributes need to be written or values need to be written atomically such as in a case when there is a need to send the values of multiple different characteristics before sending the execute command. In all cases when the amount of data to transfer fits into the BGAPI payload the command gatt_write_characteristic_value is recommended also for writing long values since it transparently performs prepare_write and execute_write commands. A received gatt_characteristic_value event can be used to verify that the data has been transmitted. Writes are executed or canceled by execute_characteristic_value_write command. Content, offset and length of given value is verified by the server when execute_characteristic_value_write is executed.

Table 1.65. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x06	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x0b	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event
7-8	uint16	offset	Offset of characteristic value
9	uint8array	value	Value to write into the specified characteristic of the remote GATT database

Table 1.66. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x0b	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call gatt_prepare_characteristic_value_write(connection,characteristic,offset,value_len, value_data)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_prepare_characteristic_value_write(uint8 connection, uint16 characteristic, uint16 offset,
uint8 value_len, const uint8 *value_data);

/* Callback */
struct gecko_msg_gatt_prepare_characteristic_value_write_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_prepare_characteristic_value_write(
```

```
  const struct gecko_msg_gatt_prepare_characteristic_value_write_rsp_t *msg
)
```

Table 1.67. Events generated

Event	Description
gatt_characteristic_value	Received characteristic value which has been added to the write queue of remote GATT server. Received value can be used to verify that the data has not been corrupted during transmission.
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

1.4.9 cmd_gatt_read_characteristic_value

This command can be used to read the value of a characteristic from a remote GATT database. A single gatt_characteristic_value event is generated if the length of the characteristic value returned by the remote GATT server is less than or equal to the size of the GATT MTU. If the length of the value exceeds the size of the GATT MTU more than one gatt_characteristic_value event is generated because the firmware will automatically use the "read long" GATT procedure. Received gatt_procedure_completed event indicates that all data has been read successfully or that an error response has been received.

Table 1.68. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x07	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event

Table 1.69. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
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 codes

BGScript command

```
call gatt_read_characteristic_value(connection,characteristic)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_read_characteristic_value(uint8 connection, uint16 characteristic);

/* Callback */
struct gecko_msg_gatt_read_characteristic_value_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_read_characteristic_value(
  const struct gecko_msg_gatt_read_characteristic_value_rsp_t *msg
)
```

Table 1.70. Events generated

Event	Description
gatt_characteristic_value	Received characteristic value from remote GATT server.
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

1.4.10 cmd_gatt_read_characteristic_value_by_uuid

This command can be used to read the characteristic value of a service from a remote GATT database by giving the UUID of the characteristic and the handle of the service containing this characteristic. A single gatt_characteristic_value event is generated if the length of the characteristic value returned by the remote GATT server is less than or equal to the size of the GATT MTU. If the length of the value exceeds the size of the GATT MTU more than one gatt_characteristic_value event is generated because the firmware will automatically use the "read long" GATT procedure. Received gatt_procedure_completed event indicates that all data has been read successfully or that an error response has been received.

Table 1.71. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x06	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x08	method	Message ID
4	uint8	connection	Connection handle
5-8	uint32	service	GATT service handleThis value is normally received from the gatt_service event.
9	uint8array	uuid	Characteristic UUID

Table 1.72. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
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 codes

BGScript command

```
call gatt_read_characteristic_value_by_uuid(connection,service,uuid_len, uuid_data)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_read_characteristic_value_by_uuid(uint8 connection, uint32 service, uint8 uuid_len, const u
int8 *uuid_data);

/* Callback */
struct gecko_msg_gatt_read_characteristic_value_by_uuid_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_read_characteristic_value_by_uuid(
  const struct gecko_msg_gatt_read_characteristic_value_by_uuid_rsp_t *msg
)
```

Table 1.73. Events generated

Event	Description
gatt_characteristic_value	Received characteristic value from remote GATT server.
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

1.4.11 cmd_gatt_read_descriptor_value

This command can be used to read the descriptor value of a characteristic in a remote GATT database. A single gatt_descriptor_value event is generated if the length of the descriptor value returned by the remote GATT server is less than or equal to the size of the GATT MTU. If the length of the value exceeds the size of the GATT MTU more than one gatt_descriptor_value event is generated because the firmware will automatically use the "read long" GATT procedure. Received gatt_procedure_completed event indicates that all data has been read successfully or that an error response has been received.

Table 1.74. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x0e	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	descriptor	GATT characteristic descriptor handle

Table 1.75. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x0e	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call gatt_read_descriptor_value(connection,descriptor)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_read_descriptor_value(uint8 connection, uint16 descriptor);

/* Callback */
struct gecko_msg_gatt_read_descriptor_value_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_read_descriptor_value(
  const struct gecko_msg_gatt_read_descriptor_value_rsp_t *msg
)
```

Table 1.76. Events generated

Event	Description
gatt_descriptor_value	Received descriptor value from remote GATT server.
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

1.4.12 cmd_gatt_read_multiple_characteristic_values

With this single command it is possible read the values of multiple characteristics from a remote GATT database at once. gatt_characteristic_value events are generated as the values are returned by the remote GATT server. Received gatt_procedure_completed event indicates that data has been read successfully or that an error response has been received.

Table 1.77. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x11	method	Message ID
4	uint8	connection	Connection handle
5	uint8array	characteristic_list	Little endian encoded uint16 list of characteristics to be read.

Table 1.78. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x11	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call gatt_read_multiple_characteristic_values(connection,characteristic_list_len, characteristic_list_data)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_read_multiple_characteristic_values(uint8 connection, uint8 characteristic_list_len, const uint8 *characteristic_list_data);

/* Callback */
struct gecko_msg_gatt_read_multiple_characteristic_values_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_read_multiple_characteristic_values(
  const struct gecko_msg_gatt_read_multiple_characteristic_values_rsp_t *msg
)
```

Table 1.79. Events generated

Event	Description
gatt_characteristic_value	Received characteristic values from remote GATT server.
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

1.4.13 cmd_gatt_send_characteristic_confirmation

This command must be used to send a characteristic confirmation to a remote GATT server after receiving an indication. The gatt_characteristic_value_event carries the att_opcode containing handle_value_indication (0x1e) which reveals that an indication has been received and this must be confirmed with this command. Confirmation needs to be sent within 30 seconds, otherwise the GATT transactions between the client and the server are discontinued.

Table 1.80. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x0d	method	Message ID
4	uint8	connection	Connection handle

Table 1.81. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x0d	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call gatt_send_characteristic_confirmation(connection)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_send_characteristic_confirmation(uint8 connection);

/* Callback */
struct gecko_msg_gatt_send_characteristic_confirmation_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_send_characteristic_confirmation(
  const struct gecko_msg_gatt_send_characteristic_confirmation_rsp_t *msg
)
```

1.4.14 cmd_gatt_server_read_attribute_type

This command can be used to read the type of an attribute from a local GATT database. Type is usually given as 16-bit or 128-bit UUID.

Table 1.82. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x01	method	Message ID
4-5	uint16	attribute	Attribute handle

Table 1.83. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x01	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes
6	uint8array	type	Variable length byte array

BGScript command

```
call gatt_server_read_attribute_type(attribute)(result,type_len, type_data)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_server_read_attribute_type(uint16 attribute);

/* Callback */
struct gecko_msg_gatt_server_read_attribute_type_rsp_t
{
  uint16 result,
  uint8 type_len,
  const uint8 *type_data
}
void gecko_rsp_gatt_server_read_attribute_type(
  const struct gecko_msg_gatt_server_read_attribute_type_rsp_t *msg
)
```

1.4.15 cmd_gatt_server_read_attribute_value

This command can be used to read the value of an attribute from a local GATT database.

Table 1.84. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x00	method	Message ID
4-5	uint16	attribute	Attribute handle
6-7	uint16	offset	Value offset

Table 1.85. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x00	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes
6	uint8array	value	Variable length byte array

BGScript command

```
call gatt_server_read_attribute_value(attribute,offset)(result,value_len, value_data)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_server_read_attribute_value(uint16 attribute, uint16 offset);

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

1.4.16 cmd_gatt_server_send_characteristic_notification

This command can be used to send notifications and indications to a remote GATT client. Notification or indication is sent only if the client has enabled them by setting the corresponding flag to the Client Characteristic Configuration descriptor. A new notification or indication cannot be sent before a confirmation from the GATT client is first received. The confirmation is indicated by the event called gatt_server_characteristic_status_event.

Table 1.86. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x05	method	Message ID
4	uint8	connection	Handle of the connection over which the notification or indication is sent.Values0xff: Sends notification or indication to all connected devices.Other: Connection handle
5-6	uint16	characteristic	Characteristic handle
7	uint8array	value	Value to be notified or indicated

Table 1.87. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
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 codes

BGScript command

```
call gatt_server_send_characteristic_notification(connection,characteristic,value_len, value_data)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_server_send_characteristic_notification(uint8 connection, uint16 characteristic, uint8 valu
e_len, const uint8 *value_data);

/* Callback */
struct gecko_msg_gatt_server_send_characteristic_notification_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_server_send_characteristic_notification(
  const struct gecko_msg_gatt_server_send_characteristic_notification_rsp_t *msg
)
```

1.4.17 cmd_gatt_server_send_user_read_response

This command must be used to send a response to a user_read_request event. The response needs to be sent within 30 seconds. otherwise no more GATT transactions are allowed by the remote side. If attr_errorcode is set to 0 the characteristic value is sent to the remote GATT client in the normal way. Other values will cause the local GATT server to send an attribute protocol error response instead of the actual data.

Table 1.88. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x03	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event
7	uint8	att_errorcode	Attribute protocol error codeValues0: No errorOther: see link
8	uint8array	value	Characteristic value

Table 1.89. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x03	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call gatt_server_send_user_read_response(connection,characteristic,att_errorcode,value_len, value_data)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_server_send_user_read_response(uint8 connection, uint16 characteristic, uint8 att_errorcode
, uint8 value_len, const uint8 *value_data);

/* Callback */
struct gecko_msg_gatt_server_send_user_read_response_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_server_send_user_read_response(
  const struct gecko_msg_gatt_server_send_user_read_response_rsp_t *msg
)
```

1.4.18 cmd_gatt_server_send_user_write_response

This command must be used to send a response to a user_write_request event. The response needs to be sent within 30 seconds. otherwise no more GATT transactions are allowed by the remote side. If attr_errorcode is set to 0 the ATT protocol's write response is sent to indicate to the remote GATT client that the write operation was processed successfully. Other values will cause the local GATT server to send an ATT protocol error response.

Table 1.90. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x04	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event
7	uint8	att_errorcode	Attribute protocol error codeValues0: No errorOther: see link

Table 1.91. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
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 codes

BGScript command

```
call gatt_server_send_user_write_response(connection,characteristic,att_errorcode)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_server_send_user_write_response(uint8 connection, uint16 characteristic, uint8 att_errorcod
e);

/* Callback */
struct gecko_msg_gatt_server_send_user_write_response_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_server_send_user_write_response(
  const struct gecko_msg_gatt_server_send_user_write_response_rsp_t *msg
)
```

1.4.19 cmd_gatt_server_write_attribute_value

This command can be used to write the value of an attribute in the local GATT database. Writing the value of a characteristic of the local GATT database will not trigger notifications or indications to the remote GATT client in case such characteristic has property of indicate or notify and the client has enabled notification or indication. Notifications and indications are sent to the remote GATT client using send_characteristic_notification command.

Table 1.92. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x02	method	Message ID
4-5	uint16	attribute	Attribute handle
6-7	uint16	offset	Value offset
8	uint8array	value	Value

Table 1.93. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x02	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call gatt_server_write_attribute_value(attribute,offset,value_len, value_data)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_server_write_attribute_value(uint16 attribute, uint16 offset, uint8 value_len, const uint8
*value_data);

/* Callback */
struct gecko_msg_gatt_server_write_attribute_value_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_server_write_attribute_value(
  const struct gecko_msg_gatt_server_write_attribute_value_rsp_t *msg
)
```

1.4.20 cmd_gatt_set_characteristic_notification

This command can be used to enable or disable the notifications and indications being sent from a remote GATT server. This procedure discovers a characteristic client configuration descriptor and writes the related configuration flags to a remote GATT database. A received gatt_procedure_completed event indicates that this GATT procedure has successfully completed or that is has failed with an error.

Table 1.94. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x05	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event
7	uint8	flags	Characteristic client configuration flags

Table 1.95. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
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 codes

BGScript command

```
call gatt_set_characteristic_notification(connection,characteristic,flags)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_set_characteristic_notification(uint8 connection, uint16 characteristic, uint8 flags);

/* Callback */
struct gecko_msg_gatt_set_characteristic_notification_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_set_characteristic_notification(
  const struct gecko_msg_gatt_set_characteristic_notification_rsp_t *msg
)
```

Table 1.96. Events generated

Event	Description
gatt_characteristic_value	Informs that a characteristic value has been indicated or notified by remote GATT server.
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

1.4.21 cmd_gatt_set_max_mtu

This command can be used to set the maximum number of GATT Message Transfer Units (MTU). If max_mtu is non-default, MTU is exchanged automatically after Bluetooth LE connection has been established.

Table 1.97. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x00	method	Message ID
4-5	uint16	max_mtu	Maximum number of Message Transfer Units (MTU) allowe-dRange: 23 to 64Default: 23

Table 1.98. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
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 codes

BGScript command

```
call gatt_set_max_mtu(max_mtu)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_set_max_mtu(uint16 max_mtu);

/* Callback */
struct gecko_msg_gatt_set_max_mtu_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_set_max_mtu(
  const struct gecko_msg_gatt_set_max_mtu_rsp_t *msg
)
```

1.4.22 cmd_gatt_write_characteristic_value

This command can be used to write the value of a characteristic in a remote GATT database. If the length of the given value is greater than the exchanged GATT MTU (Message Transfer Unit), "write long" GATT procedure is used automatically. Received gatt_procedure_completed event indicates that all data has been written successfully or that an error response has been received.

Table 1.99. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x09	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event
7	uint8array	value	Characteristic value

Table 1.100. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x09	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call gatt_write_characteristic_value(connection,characteristic,value_len, value_data)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_write_characteristic_value(uint8 connection, uint16 characteristic, uint8 value_len, const
uint8 *value_data);

/* Callback */
struct gecko_msg_gatt_write_characteristic_value_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_write_characteristic_value(
  const struct gecko_msg_gatt_write_characteristic_value_rsp_t *msg
)
```

Table 1.101. Events generated

Event	Description
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

1.4.23 cmd_gatt_write_characteristic_value_without_response

This command can be used to write the value of a characteristic in a remote GATT database. This command does not generate any event. All failures on the server are ignored silently. For example, if an error is generated in the remote GATT server and the given value is not written into database no error message willl be reported to the local GATT client.

Table 1.102. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x0a	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event
7	uint8array	value	Characteristic value

Table 1.103. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x0a	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call gatt_write_characteristic_value_without_response(connection,characteristic,value_len, value_data)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_write_characteristic_value_without_response(uint8 connection, uint16 characteristic, uint8
value_len, const uint8 *value_data);

/* Callback */
struct gecko_msg_gatt_write_characteristic_value_without_response_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_write_characteristic_value_without_response(
  const struct gecko_msg_gatt_write_characteristic_value_without_response_rsp_t *msg
)
```

1.4.24 cmd_gatt_write_descriptor_value

This command can be used to write the value of a characteristic descriptor in a remote GATT database. If the length of the given value is greater than the exchanged GATT MTU size, "write long" GATT procedure is used automatically. Received gatt_procedure_completed event indicates that all data has been written succesfully or that an error response has been received.

Table 1.104. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x0f	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	descriptor	GATT characteristic descriptor handle
7	uint8array	value	Descriptor value

Table 1.105. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x0f	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call gatt_write_descriptor_value(connection,descriptor,value_len, value_data)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_write_descriptor_value(uint8 connection, uint16 descriptor, uint8 value_len, const uint8 *v
alue_data);

/* Callback */
struct gecko_msg_gatt_write_descriptor_value_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_write_descriptor_value(
  const struct gecko_msg_gatt_write_descriptor_value_rsp_t *msg
)
```

Table 1.106. Events generated

Event	Description
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

1.4.25 evt_gatt_characteristic

This event indicates that a GATT characteristic was discovered from a remote GATT database by gatt_discover_characteristics or gatt_discover_primary_characteristics_by_uuid command.

Table 1.107. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x05	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x02	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event
7	uint8	properties	Characteristic properties
8	uint8array	uuid	Characteristic UUID

BGScript event

```
event gatt_characteristic(connection,characteristic,properties,uuid_len, uuid_data)
```

C Functions

```
/* Callback */
struct gecko_msg_gatt_characteristic_evt_t
{
  uint8 connection,
  uint16 characteristic,
  uint8 properties,
  uint8 uuid_len,
  const uint8 *uuid_data
}
void gecko_rsp_gatt_characteristic(
  const struct gecko_msg_gatt_characteristic_evt_t *msg
)
```

1.4.26 evt_gatt_characteristic_value

This event indicates that the value of a characteristic at the remote GATT server was received. This event is triggered in the following cases: after a read command, prepare_write command, when the remote GATT server sends indications or notifications. The parameter att_opcode reveals which GATT transaction triggered this event. In particular for indications the att_opcode type is handle_value_indication (0x1d) and the application needs to confirm indication with send_characteristic_confirmation command if att_opcode type is handle_value_indication (0x1d).

Table 1.108. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x07	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x04	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event
7	uint8	att_opcode	Attribute opcode which informs the GATT transaction used
8-9	uint16	offset	Value offset
10	uint8array	value	Characteristic value

BGScript event

```
event gatt_characteristic_value(connection,characteristic,att_opcode,offset,value_len, value_data)
```

C Functions

```
/* Callback */
struct gecko_msg_gatt_characteristic_value_evt_t
{
  uint8 connection,
  uint16 characteristic,
  uint8 att_opcode,
  uint16 offset,
  uint8 value_len,
  const uint8 *value_data
}
void gecko_rsp_gatt_characteristic_value(
  const struct gecko_msg_gatt_characteristic_value_evt_t *msg
)
```

1.4.27 evt_gatt_descriptor

This event indicates that a GATT characteristic descriptor was discovered from a remote GATT database by using the gatt_discover_descriptor command.

Table 1.109. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x04	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x03	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	descriptor	GATT characteristic descriptor handle
7	uint8array	uuid	Characteristic UUID

BGScript event

```
event gatt_descriptor(connection,descriptor,uuid_len, uuid_data)
```

C Functions

```
/* Callback */
struct gecko_msg_gatt_descriptor_evt_t
{
  uint8 connection,
  uint16 descriptor,
  uint8 uuid_len,
  const uint8 *uuid_data
}
void gecko_rsp_gatt_descriptor(
  const struct gecko_msg_gatt_descriptor_evt_t *msg
)
```

1.4.28 evt_gatt_descriptor_value

This event indicates that the value of a descriptor at the remote GATT server was received. This event is generated by the read_descriptor_value command.

Table 1.110. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x06	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x05	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	descriptor	GATT characteristic descriptor handle
7-8	uint16	offset	Value offset
9	uint8array	value	Descriptor value

BGScript event

```
event gatt_descriptor_value(connection,descriptor,offset,value_len, value_data)
```

C Functions

```
/* Callback */
struct gecko_msg_gatt_descriptor_value_evt_t
{
  uint8 connection,
  uint16 descriptor,
  uint16 offset,
  uint8 value_len,
  const uint8 *value_data
}
void gecko_rsp_gatt_descriptor_value(
  const struct gecko_msg_gatt_descriptor_value_evt_t *msg
)
```

1.4.29 evt_gatt_mtu_exchanged

This event indicates that a GATT MTU exchange procedure has been completed.

Table 1.111. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x00	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	mtu	Exchanged GATT MTU

BGScript event

```
event gatt_mtu_exchanged(connection,mtu)
```

C Functions

```
/* Callback */
struct gecko_msg_gatt_mtu_exchanged_evt_t
{
  uint8 connection,
  uint16 mtu
}
void gecko_rsp_gatt_mtu_exchanged(
  const struct gecko_msg_gatt_mtu_exchanged_evt_t *msg
)
```

1.4.30 evt_gatt_procedure_completed

This event indicates that the previous GATT procedure has been completed succesfully or that is has failed with an error.

Table 1.112. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x06	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript event

```
event gatt_procedure_completed(connection,result)
```

C Functions

```
/* Callback */
struct gecko_msg_gatt_procedure_completed_evt_t
{
  uint8 connection,
  uint16 result
}
void gecko_rsp_gatt_procedure_completed(
  const struct gecko_msg_gatt_procedure_completed_evt_t *msg
)
```

1.4.31 evt_gatt_server_attribute_value

This event indicates that the value of an attribute in the local GATT database has been changed by a remote GATT client. Parameter att_opcode describes which attribute procedure was used to change the value.

Table 1.113. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x07	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x00	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	attribute	Attribute Handle
7	uint8	att_opcode	Attribute opcode which informs the procedure from which attribute the value was received from
8-9	uint16	offset	Value offset
10	uint8array	value	Value

BGScript event

```
event gatt_server_attribute_value(connection,attribute,att_opcode,offset,value_len, value_data)
```

C Functions

```
/* Callback */
struct gecko_msg_gatt_server_attribute_value_evt_t
{
  uint8 connection,
  uint16 attribute,
  uint8 att_opcode,
  uint16 offset,
  uint8 value_len,
  const uint8 *value_data
}
void gecko_rsp_gatt_server_attribute_value(
  const struct gecko_msg_gatt_server_attribute_value_evt_t *msg
)
```

1.4.32 evt_gatt_server_characteristic_status

This event states either that a local Client Characteristic Configuration descriptor has been changed by the remote GATT client, or that a confirmation from the remote GATT client was received upon a successful reception of the indication. Confirmation by the remote GATT client should be received within 30 seconds after an indication has been sent with the send_characteristic_notification command, otherwise GATT transactions are disabled by the stack.

Table 1.114. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x06	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x03	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event
7	uint8	status_flags	Describes whether Client Characteristic Configuration was changed or if confirmation was received.
8-9	uint16	client_config_flags	This entry carries the new value of the Client Characteristic Configuration.

BGScript event

```
event gatt_server_characteristic_status(connection,characteristic,status_flags,client_config_flags)
```

C Functions

```
/* Callback */
struct gecko_msg_gatt_server_characteristic_status_evt_t
{
  uint8 connection,
  uint16 characteristic,
  uint8 status_flags,
  uint16 client_config_flags
}
void gecko_rsp_gatt_server_characteristic_status(
  const struct gecko_msg_gatt_server_characteristic_status_evt_t *msg
)
```

1.4.33 evt_gatt_server_user_read_request

This event indicates that a remote GATT client is attempting to read a value of an attribute from the local GATT database, where the attribute was defined in the GATT XML firmware configuration file to have type="user". Parameter att_opcode informs which attribute procedure was used to read the value. The application needs to respond to this request by using the send_user_read_response command within 30 seconds, otherwise this GATT connection is dropped by remote side.

Table 1.115. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x06	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x01	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event
7	uint8	att_opcode	Attribute opcode which informs the procedure from which attribute the value was received from
8-9	uint16	offset	Value offset

BGScript event

```
event gatt_server_user_read_request(connection,characteristic,att_opcode,offset)
```

C Functions

```
/* Callback */
struct gecko_msg_gatt_server_user_read_request_evt_t
{
  uint8 connection,
  uint16 characteristic,
  uint8 att_opcode,
  uint16 offset
}
void gecko_rsp_gatt_server_user_read_request(
  const struct gecko_msg_gatt_server_user_read_request_evt_t *msg
)
```

1.4.34 evt_gatt_server_user_write_request

This event indicates that a remote GATT client is attempting to write a value of an attribute in to the local GATT database, where the attribute was defined in the GATT XML firmware configuration file to have type="user". Parameter att_opcode informs which attribute procedure was used to write the value. The application needs to respond to this request by using the send_user_write_response command within 30 seconds, otherwise otherwise this GATT connection is dropped by remote side.

Table 1.116. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x07	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x02	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event
7	uint8	att_opcode	Attribute opcode which informs the procedure from which attribute the value was received from
8-9	uint16	offset	Value offset
10	uint8array	value	Value

BGScript event

```
event gatt_server_user_write_request(connection,characteristic,att_opcode,offset,value_len, value_data)
```

C Functions

```
/* Callback */
struct gecko_msg_gatt_server_user_write_request_evt_t
{
  uint8 connection,
  uint16 characteristic,
  uint8 att_opcode,
  uint16 offset,
  uint8 value_len,
  const uint8 *value_data
}
void gecko_rsp_gatt_server_user_write_request(
  const struct gecko_msg_gatt_server_user_write_request_evt_t *msg
)
```

1.4.35 evt_gatt_service

This event indicates that a GATT service at the remote GATT database was discovered. This event follows the commands issued by the local GATT client.

Table 1.117. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x06	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x01	method	Message ID
4	uint8	connection	Connection handle
5-8	uint32	service	GATT service handleThis value is normally received from the gatt_service event.
9	uint8array	uuid	Characteristic UUID

BGScript event

```
event gatt_service(connection,service,uuid_len, uuid_data)
```

C Functions

```
/* Callback */
struct gecko_msg_gatt_service_evt_t
{
  uint8 connection,
  uint32 service,
  uint8 uuid_len,
  const uint8 *uuid_data
}
void gecko_rsp_gatt_service(
  const struct gecko_msg_gatt_service_evt_t *msg
)
```

1.4.36 enum_gatt_att_opcode

These values indicate which attribute request or response has caused the event.

Table 1.118. Enumerations

Value	Name	Description
8	gatt_read_by_type_request	Read by type request
9	gatt_read_by_type_response	Read by type response
10	gatt_read_request	Read request
11	gatt_read_response	Read response
12	gatt_read_blob_request	Read blob request
13	gatt_read_blob_response	Read blob response
14	gatt_read_multiple_request	Read multiple request
15	gatt_read_multiple_response	Read multiple response
18	gatt_write_request	Write request
19	gatt_write_response	Write response
82	gatt_write_command	Write command
22	gatt_prepare_write_request	Prepare write request
23	gatt_prepare_write_response	Prepare write response
24	gatt_execute_write_request	Excute write request
25	gatt_execute_write_response	Excute write response
27	gatt_handle_value_notification	Notfication
29	gatt_handle_value_indication	Indication

1.4.37 enum_gatt_client_config_flag

These values define whether the client is to receive notifications or indications from a remote GATT server.

Table 1.119. Enumerations

Value	Name	Description
1	gatt_notification	Notification
2	gatt_indication	Indication

1.4.38 enum_gatt_execute_write_flag

These values define whether the GATT server is to cancel all queued writes or commit all queued writes to a remote database.

Table 1.120. Enumerations

Value	Name	Description
0	gatt_cancel	Cancel all queued writes
1	gatt_commit	Commit all queued writes

1.4.39 enum_gatt_server_characteristic_status_flag

These values describe whether characteristic client configuration was changed or whether a characteristic confirmation was received.

Table 1.121. Enumerations

Value	Name	Description
1	gatt_server_client_config	Characteristic client configuration has been changed.
2	gatt_server_confirmation	Characteristic confirmation has been received.

1.5 Generic Attribute Profile Server (gatt_server)

1.5.1 cmd_gatt_server_read_attribute_type

This command can be used to read the type of an attribute from a local GATT database. Type is usually given as 16-bit or 128-bit UUID.

Table 1.122. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x01	method	Message ID
4-5	uint16	attribute	Attribute handle

Table 1.123. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x01	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes
6	uint8array	type	Variable length byte array

BGScript command

```
call gatt_server_read_attribute_type(attribute)(result,type_len, type_data)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_server_read_attribute_type(uint16 attribute);

/* Callback */
struct gecko_msg_gatt_server_read_attribute_type_rsp_t
{
  uint16 result,
  uint8 type_len,
  const uint8 *type_data
}
void gecko_rsp_gatt_server_read_attribute_type(
  const struct gecko_msg_gatt_server_read_attribute_type_rsp_t *msg
)
```

1.5.2 cmd_gatt_server_read_attribute_value

This command can be used to read the value of an attribute from a local GATT database.

Table 1.124. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x00	method	Message ID
4-5	uint16	attribute	Attribute handle
6-7	uint16	offset	Value offset

Table 1.125. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x00	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes
6	uint8array	value	Variable length byte array

BGScript command

```
call gatt_server_read_attribute_value(attribute,offset)(result,value_len, value_data)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_server_read_attribute_value(uint16 attribute, uint16 offset);

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

1.5.3 cmd_gatt_server_send_characteristic_notification

This command can be used to send notifications and indications to a remote GATT client. Notification or indication is sent only if the client has enabled them by setting the corresponding flag to the Client Characteristic Configuration descriptor. A new notification or indication cannot be sent before a confirmation from the GATT client is first received. The confirmation is indicated by the event called gatt_server_characteristic_status_event.

Table 1.126. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x05	method	Message ID
4	uint8	connection	Handle of the connection over which the notification or indication is sent.Values0xff: Sends notification or indication to all connected devices.Other: Connection handle
5-6	uint16	characteristic	Characteristic handle
7	uint8array	value	Value to be notified or indicated

Table 1.127. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
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 codes

BGScript command

```
call gatt_server_send_characteristic_notification(connection,characteristic,value_len, value_data)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_server_send_characteristic_notification(uint8 connection, uint16 characteristic, uint8 valu
e_len, const uint8 *value_data);

/* Callback */
struct gecko_msg_gatt_server_send_characteristic_notification_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_server_send_characteristic_notification(
  const struct gecko_msg_gatt_server_send_characteristic_notification_rsp_t *msg
)
```

1.5.4 cmd_gatt_server_send_user_read_response

This command must be used to send a response to a user_read_request event. The response needs to be sent within 30 seconds. otherwise no more GATT transactions are allowed by the remote side. If attr_errorcode is set to 0 the characteristic value is sent to the remote GATT client in the normal way. Other values will cause the local GATT server to send an attribute protocol error response instead of the actual data.

Table 1.128. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x03	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event
7	uint8	att_errorcode	Attribute protocol error codeValues0: No errorOther: see link
8	uint8array	value	Characteristic value

Table 1.129. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x03	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call gatt_server_send_user_read_response(connection,characteristic,att_errorcode,value_len, value_data)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_server_send_user_read_response(uint8 connection, uint16 characteristic, uint8 att_errorcode
, uint8 value_len, const uint8 *value_data);

/* Callback */
struct gecko_msg_gatt_server_send_user_read_response_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_server_send_user_read_response(
  const struct gecko_msg_gatt_server_send_user_read_response_rsp_t *msg
)
```

1.5.5 cmd_gatt_server_send_user_write_response

This command must be used to send a response to a user_write_request event. The response needs to be sent within 30 seconds. otherwise no more GATT transactions are allowed by the remote side. If attr_errorcode is set to 0 the ATT protocol's write response is sent to indicate to the remote GATT client that the write operation was processed successfully. Other values will cause the local GATT server to send an ATT protocol error response.

Table 1.130. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x04	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event
7	uint8	att_errorcode	Attribute protocol error codeValues0: No errorOther: see link

Table 1.131. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
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 codes

BGScript command

```
call gatt_server_send_user_write_response(connection,characteristic,att_errorcode)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_server_send_user_write_response(uint8 connection, uint16 characteristic, uint8 att_errorcod
e);

/* Callback */
struct gecko_msg_gatt_server_send_user_write_response_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_server_send_user_write_response(
  const struct gecko_msg_gatt_server_send_user_write_response_rsp_t *msg
)
```

1.5.6 cmd_gatt_server_write_attribute_value

This command can be used to write the value of an attribute in the local GATT database. Writing the value of a characteristic of the local GATT database will not trigger notifications or indications to the remote GATT client in case such characteristic has property of indicate or notify and the client has enabled notification or indication. Notifications and indications are sent to the remote GATT client using send_characteristic_notification command.

Table 1.132. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x02	method	Message ID
4-5	uint16	attribute	Attribute handle
6-7	uint16	offset	Value offset
8	uint8array	value	Value

Table 1.133. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x02	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call gatt_server_write_attribute_value(attribute,offset,value_len, value_data)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_gatt_server_write_attribute_value(uint16 attribute, uint16 offset, uint8 value_len, const uint8
*value_data);

/* Callback */
struct gecko_msg_gatt_server_write_attribute_value_rsp_t
{
  uint16 result
}
void gecko_rsp_gatt_server_write_attribute_value(
  const struct gecko_msg_gatt_server_write_attribute_value_rsp_t *msg
)
```

1.5.7 evt_gatt_server_attribute_value

This event indicates that the value of an attribute in the local GATT database has been changed by a remote GATT client. Parameter att_opcode describes which attribute procedure was used to change the value.

Table 1.134. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x07	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x00	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	attribute	Attribute Handle
7	uint8	att_opcode	Attribute opcode which informs the procedure from which attribute the value was received from
8-9	uint16	offset	Value offset
10	uint8array	value	Value

BGScript event

```
event gatt_server_attribute_value(connection,attribute,att_opcode,offset,value_len, value_data)
```

C Functions

```
/* Callback */
struct gecko_msg_gatt_server_attribute_value_evt_t
{
  uint8 connection,
  uint16 attribute,
  uint8 att_opcode,
  uint16 offset,
  uint8 value_len,
  const uint8 *value_data
}
void gecko_rsp_gatt_server_attribute_value(
  const struct gecko_msg_gatt_server_attribute_value_evt_t *msg
)
```

1.5.8 evt_gatt_server_characteristic_status

This event states either that a local Client Characteristic Configuration descriptor has been changed by the remote GATT client, or that a confirmation from the remote GATT client was received upon a successful reception of the indication. Confirmation by the remote GATT client should be received within 30 seconds after an indication has been sent with the send_characteristic_notification command, otherwise GATT transactions are disabled by the stack.

Table 1.135. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x06	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x03	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event
7	uint8	status_flags	Describes whether Client Characteristic Configuration was changed or if confirmation was received.
8-9	uint16	client_config_flags	This entry carries the new value of the Client Characteristic Configuration.

BGScript event

```
event gatt_server_characteristic_status(connection,characteristic,status_flags,client_config_flags)
```

C Functions

```
/* Callback */
struct gecko_msg_gatt_server_characteristic_status_evt_t
{
  uint8 connection,
  uint16 characteristic,
  uint8 status_flags,
  uint16 client_config_flags
}
void gecko_rsp_gatt_server_characteristic_status(
  const struct gecko_msg_gatt_server_characteristic_status_evt_t *msg
)
```

1.5.9 evt_gatt_server_user_read_request

This event indicates that a remote GATT client is attempting to read a value of an attribute from the local GATT database, where the attribute was defined in the GATT XML firmware configuration file to have type="user". Parameter att_opcode informs which attribute procedure was used to read the value. The application needs to respond to this request by using the send_user_read_response command within 30 seconds, otherwise this GATT connection is dropped by remote side.

Table 1.136. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x06	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x01	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event
7	uint8	att_opcode	Attribute opcode which informs the procedure from which attribute the value was received from
8-9	uint16	offset	Value offset

BGScript event

```
event gatt_server_user_read_request(connection,characteristic,att_opcode,offset)
```

C Functions

```
/* Callback */
struct gecko_msg_gatt_server_user_read_request_evt_t
{
  uint8 connection,
  uint16 characteristic,
  uint8 att_opcode,
  uint16 offset
}
void gecko_rsp_gatt_server_user_read_request(
  const struct gecko_msg_gatt_server_user_read_request_evt_t *msg
)
```

1.5.10 evt_gatt_server_user_write_request

This event indicates that a remote GATT client is attempting to write a value of an attribute in to the local GATT database, where the attribute was defined in the GATT XML firmware configuration file to have type="user". Parameter att_opcode informs which attribute procedure was used to write the value. The application needs to respond to this request by using the send_user_write_response command within 30 seconds, otherwise otherwise this GATT connection is dropped by remote side.

Table 1.137. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x07	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x02	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handleThis value is normally received from gatt_characteristic event
7	uint8	att_opcode	Attribute opcode which informs the procedure from which attribute the value was received from
8-9	uint16	offset	Value offset
10	uint8array	value	Value

BGScript event

```
event gatt_server_user_write_request(connection,characteristic,att_opcode,offset,value_len, value_data)
```

C Functions

```
/* Callback */
struct gecko_msg_gatt_server_user_write_request_evt_t
{
  uint8 connection,
  uint16 characteristic,
  uint8 att_opcode,
  uint16 offset,
  uint8 value_len,
  const uint8 *value_data
}
void gecko_rsp_gatt_server_user_write_request(
  const struct gecko_msg_gatt_server_user_write_request_evt_t *msg
)
```

1.5.11 enum_gatt_server_characteristic_status_flag

These values describe whether characteristic client configuration was changed or whether a characteristic confirmation was received.

Table 1.138. Enumerations

Value	Name	Description
1	gatt_server_client_config	Characteristic client configuration has been changed.
2	gatt_server_confirmation	Characteristic confirmation has been received.

1.6 Hardware (hardware)

Commands to access system hardware.

1.6.1 cmd_hardware_configure_gpio

Configure I/O-port mode

Table 1.139. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x0c	class	Message class: Hardware
3	0x01	method	Message ID
4	uint8	port	Port index, where A=0, B=1.
5	uint8	gpio	Index of gpio-pin on the port which this command affects.
6	uint8	mode	Pin mode
7	uint8	output	Pin DOUT state

Table 1.140. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: Hardware
3	0x01	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_hardware_configure_gpio(uint8 port, uint8 gpio, uint8 mode, uint8 output);

/* Callback */
struct gecko_msg_hardware_configure_gpio_rsp_t
{
  uint16 result
}
void gecko_rsp_hardware_configure_gpio(
  const struct gecko_msg_hardware_configure_gpio_rsp_t *msg
)
```

1.6.2 cmd_hardware_read_gpio

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

Table 1.141. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x0c	class	Message class: Hardware
3	0x03	method	Message ID
4	uint8	port	Port index to read from, A=0, B=1.
5-6	uint16	mask	Bitmask of which pins on the port should be read

Table 1.142. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x04	lolen	Minimum payload length
2	0x0c	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 codes
6-7	uint16	data	Port data

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_hardware_read_gpio(uint8 port, uint16 mask);

/* Callback */
struct gecko_msg_hardware_read_gpio_rsp_t
{
  uint16 result,
  uint16 data
}
void gecko_rsp_hardware_read_gpio(
  const struct gecko_msg_hardware_read_gpio_rsp_t *msg
)
```

1.6.3 cmd_hardware_read_i2c

This command can be used for reading the specified I2C interface.

Table 1.143. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x0c	class	Message class: Hardware
3	0x04	method	Message ID
4	uint8	channel	I2C channel to use.
5-6	uint16	slave_address	Slave address to use
7	uint8	length	Amount of data to read

Table 1.144. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x0c	class	Message class: Hardware
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 codes
6	uint8array	data	Data that was read if command was successful

BGScript command

```
call hardware_read_i2c(channel,slave_address,length)(result,data_len, data_data)
```

BGLIB C API

```
/* Function */

void gecko_cmd_hardware_read_i2c(uint8 channel, uint16 slave_address, uint8 length);

/* Callback */
struct gecko_msg_hardware_read_i2c_rsp_t
{
  uint16 result,
  uint8 data_len,
  const uint8 *data_data
}
void gecko_rsp_hardware_read_i2c(
  const struct gecko_msg_hardware_read_i2c_rsp_t *msg
)
```

1.6.4 cmd_hardware_set_soft_timer

Start soft timer

Table 1.145. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x06	lolen	Minimum payload length
2	0x0c	class	Message class: Hardware
3	0x00	method	Message ID
4-7	uint32	time	Timeout to when timer triggers
8	uint8	handle	Timer handle to use, is returned in timeout event
9	uint8	single_shot	0 repeating, 1-trigger only once

Table 1.146. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: Hardware
3	0x00	method	Message ID
4-5	uint16	result	

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_hardware_set_soft_timer(uint32 time, uint8 handle, uint8 single_shot);

/* Callback */
struct gecko_msg_hardware_set_soft_timer_rsp_t
{
  uint16 result
}
void gecko_rsp_hardware_set_soft_timer(
  const struct gecko_msg_hardware_set_soft_timer_rsp_t *msg
)
```

1.6.5 cmd_hardware_stop_i2c

This command can be used to stop I2C transmission.

Table 1.147. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0c	class	Message class: Hardware
3	0x06	method	Message ID
4	uint8	channel	I2C channel to use.

Table 1.148. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0c	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 codes

BGScript command

```
call hardware_stop_i2c(channel)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_hardware_stop_i2c(uint8 channel);

/* Callback */
struct gecko_msg_hardware_stop_i2c_rsp_t
{
  uint16 result
}
void gecko_rsp_hardware_stop_i2c(
  const struct gecko_msg_hardware_stop_i2c_rsp_t *msg
)
```

1.6.6 cmd_hardware_write_gpio

This command can be used to set the logic states of pins of the specified I/O-port using a bitmask.

Table 1.149. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x0c	class	Message class: Hardware
3	0x02	method	Message ID
4	uint8	port	Port index, where A=0, B=1.
5-6	uint16	mask	Bitmask of which pins on the port this command affects
7-8	uint16	data	Bitmask of which pins to set high and which pins to set low

Table 1.150. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0c	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 codes

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_hardware_write_gpio(uint8 port, uint16 mask, uint16 data);

/* Callback */
struct gecko_msg_hardware_write_gpio_rsp_t
{
  uint16 result
}
void gecko_rsp_hardware_write_gpio(
  const struct gecko_msg_hardware_write_gpio_rsp_t *msg
)
```

1.6.7 cmd_hardware_write_i2c

This command can be used to write data into I2C interface.

Table 1.151. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x0c	class	Message class: Hardware
3	0x05	method	Message ID
4	uint8	channel	I2C channel to use.
5-6	uint16	slave_address	Slave address to use
7	uint8array	data	Data to write

Table 1.152. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0c	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 codes

BGScript command

```
call hardware_write_i2c(channel,slave_address,data_len, data_data)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_hardware_write_i2c(uint8 channel, uint16 slave_address, uint8 data_len, const uint8 *data_data);

/* Callback */
struct gecko_msg_hardware_write_i2c_rsp_t
{
  uint16 result
}
void gecko_rsp_hardware_write_i2c(
  const struct gecko_msg_hardware_write_i2c_rsp_t *msg
)
```

1.6.8 evt_hardware_soft_timer

Timer has triggered

Table 1.153. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0x0c	class	Message class: Hardware
3	0x00	method	Message ID
4	uint8	handle	Timer Handle

BGScript event

```
event hardware_soft_timer(handle)
```

C Functions

```
/* Callback */
struct gecko_msg_hardware_soft_timer_evt_t
{
  uint8 handle
}
void gecko_rsp_hardware_soft_timer(
  const struct gecko_msg_hardware_soft_timer_evt_t *msg
)
```

1.6.9 enum_hardware_gpio_mode

GPIO mode configuration

Table 1.154. Enumerations

Value	Name	Description
0	hardware_gpio_mode_disabled	Input disabled. Pullup if DOUT is set.
1	hardware_gpio_mode_input	Input enabled. Filter if DOUT is set
2	hardware_gpio_mode_input_pull	Input enabled. DOUT determines pull direction
3	hardware_gpio_mode_input_pull_filter	Input enabled with filter. DOUT determines pull direction
4	hardware_gpio_mode_push_pull	Push-pull output

1.7 Connection management for low energy (le_connection)

Connection management commands and events for Bluetooth LE.

1.7.1 cmd_le_connection_set_parameters

This command can be used to request a change in the BLE connection parameters of the currently active link.

Table 1.155. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x09	lolen	Minimum payload length
2	0x08	class	Message class: Connection management for low energy
3	0x00	method	Message ID
4	uint8	connection	Connection Handle
5-6	uint16	min_interval	Minimum connection intervalMinimum value for the connection event interval. This must be set be less than or equal to max_intervalTime = Value x 1.25 msRange: 0x0006 to 0x0c80Time Range: 7.5 ms to 4 s
7-8	uint16	max_interval	Maximum connection intervalMaximum value for the connection event interval. This must be set greater than or equal to min_intervalTime = Value x 1.25 msRange: 0x0006 to 0x0c80Time Range: 7.5 ms to 4 s
9-10	uint16	latency	Slave LatencyThis parameter defines how many connection intervals the slave can skip if it has no data to sendRange: 0x0000 to 0x01f4Use 0x0000 as default
11-12	uint16	timeout	Supervision timeoutThe supervision timeout defines for how long the connection is maintained despite the devices being unable to communicate at the currently configured connection intervals.Range: 0x000a to 0x0c80Time = Value x 10 msRange: 0x000a to 0x0c80Time Range: 100 ms to 32 sThe minimum value must be at least maximum interval * (latency + 1)It is recommended that the supervision timeout value is set with a value which allows communication atttempts for at least a couple of connection intervals.

Table 1.156. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x08	class	Message class: Connection management for low energy
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 codes

BGScript command

```
call le_connection_set_parameters(connection,min_interval,max_interval,latency,timeout)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_le_connection_set_parameters(uint8 connection, uint16 min_interval, uint16 max_interval, uint16
latency, uint16 timeout);

/* Callback */
struct gecko_msg_le_connection_set_parameters_rsp_t
{
  uint16 result
}
void gecko_rsp_le_connection_set_parameters(
  const struct gecko_msg_le_connection_set_parameters_rsp_t *msg
)
```

1.7.2 evt_le_connection_closed

This event indicates that a connection was closed.

Table 1.157. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x08	class	Message class: Connection management for low energy
3	0x01	method	Message ID
4-5	uint16	reason	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes
6	uint8	connection	Handle of the closed connection

BGScript event

```
event le_connection_closed(reason,connection)
```

C Functions

```
/* Callback */
struct gecko_msg_le_connection_closed_evt_t
{
  uint16 reason,
  uint8 connection
}
void gecko_rsp_le_connection_closed(
  const struct gecko_msg_le_connection_closed_evt_t *msg
)
```

1.7.3 evt_le_connection_opened

This event indicates that a new connection was opened, whether the devices are already bonded and what is the role of the Bluetooth module (Slave or Master).

Table 1.158. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x0a	lolen	Minimum payload length
2	0x08	class	Message class: Connection management for low energy
3	0x00	method	Message ID
4-9	bd_addr	address	Remote device address
10	uint8	address_type	Remote device address type
11	uint8	master	Module role in connectionValues0 : Slave 1: Master
12	uint8	connection	Handle for new connection
13	uint8	bonding	Bonding handleValues0xff: No bondingOther: Bonding handle

BGScript event

```
event le_connection_opened(address,address_type,master,connection,bonding)
```

C Functions

```
/* Callback */
struct gecko_msg_le_connection_opened_evt_t
{
  bd_addr address,
  uint8 address_type,
  uint8 master,
  uint8 connection,
  uint8 bonding
}
void gecko_rsp_le_connection_opened(
  const struct gecko_msg_le_connection_opened_evt_t *msg
)
```

1.7.4 evt_le_connection_parameters

This event is triggered whenever the connection parameters are changed and at any time a connection is established.

Table 1.159. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x08	lolen	Minimum payload length
2	0x08	class	Message class: Connection management for low energy
3	0x02	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	interval	Connection interval
7-8	uint16	latency	Slave latency
9-10	uint16	timeout	Supervision timeout
11	uint8	security_mode	Connection security mode

BGScript event

```
event le_connection_parameters(connection,interval,latency,timeout,security_mode)
```

C Functions

```
/* Callback */
struct gecko_msg_le_connection_parameters_evt_t
{
  uint8 connection,
  uint16 interval,
  uint16 latency,
  uint16 timeout,
  uint8 security_mode
}
void gecko_rsp_le_connection_parameters(
  const struct gecko_msg_le_connection_parameters_evt_t *msg
)
```

1.7.5 enum_le_connection_security

These values indicate the Bluetooth LE Security Mode.

Table 1.160. Enumerations

Value	Name	Description
0	le_connection_mode1_level1	No security
1	le_connection_mode1_level2	Unauthenticated pairing with encryption
2	le_connection_mode1_level3	Authenticated pairing with encryption

1.8 Generic Access Profile, Low Energy (le_gap)

The commands and events in this section are related to Generic Access Profile (GAP) in the Bluetooth Low Energy (LE).

1.8.1 cmd_le_gap_discover

This command can be used to start Bluetooth LE discovery procedure.

Table 1.161. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
3	0x02	method	Message ID
4	uint8	mode	LE Discovery mode. For values see link

Table 1.162. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
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 codes

BGScript command

```
call le_gap_discover(mode)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_le_gap_discover(uint8 mode);

/* Callback */
struct gecko_msg_le_gap_discover_rsp_t
{
  uint16 result
}
void gecko_rsp_le_gap_discover(
  const struct gecko_msg_le_gap_discover_rsp_t *msg
)
```

Table 1.163. Events generated

Event	Description
le_gap_scan_response	Discovered device scan response

1.8.2 cmd_le_gap_end_procedure

This command can be used to end a current GAP procedure.

Table 1.164. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
3	0x03	method	Message ID

Table 1.165. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
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 codes

BGScript command

```
call le_gap_end_procedure()(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_le_gap_end_procedure();

/* Callback */
struct gecko_msg_le_gap_end_procedure_rsp_t
{
  uint16 result
}
void gecko_rsp_le_gap_end_procedure(
  const struct gecko_msg_le_gap_end_procedure_rsp_t *msg
)
```

1.8.3 cmd_le_gap_open

This command can be used to start the GAP discovery procedure to scan for advertising devices i.e. to perform a device discovery. Scanning parameters can be configured with the le_gap_set_scan_parameters command before issuing this command. To cancel on an ongoing discovery process use the le_gap_end_procedure command.

Table 1.166. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x07	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
3	0x00	method	Message ID
4-9	bd_addr	address	Address of the device to connect to
10	uint8	address_type	Address type of the device to connect

Table 1.167. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
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 codes
6	uint8	connection	Handle that will be assigned to the connection once the connection will be established. This handle is valid only if the result code of this response is 0 (zero).

BGScript command

```
call le_gap_open(address,address_type)(result,connection)
```

BGLIB C API

```
/* Function */

void gecko_cmd_le_gap_open(bd_addr address, uint8 address_type);

/* Callback */
struct gecko_msg_le_gap_open_rsp_t
{
  uint16 result,
  uint8 connection
}
void gecko_rsp_le_gap_open(
  const struct gecko_msg_le_gap_open_rsp_t *msg
)
```

Table 1.168. Events generated

Event	Description
le_connection_opened	This event is triggered after the connection has been opened, and indicates whether the devices are already bonded and what is the role of the Bluetooth module (Slave or Master).

1.8.4 cmd_le_gap_set_adv_data

This command can be used to set the data in advertisement packets or in the scan response packets. This data is used when advertising in user data mode. It is recommended to set both the advertisement data and scan response data at the same time.

Table 1.169. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
3	0x07	method	Message ID
4	uint8	scan_rsp	This value selects if the data is intended for advertisement packets or scan response packetsValues0 : Advertisement packets1 : Scan response packets
5	uint8array	adv_data	Data to be setMaximum length is 30 bytes

Table 1.170. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
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 codes

BGScript command

```
call le_gap_set_adv_data(scan_rsp,adv_data_len, adv_data_data)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_le_gap_set_adv_data(uint8 scan_rsp, uint8 adv_data_len, const uint8 *adv_data_data);

/* Callback */
struct gecko_msg_le_gap_set_adv_data_rsp_t
{
  uint16 result
}
void gecko_rsp_le_gap_set_adv_data(
  const struct gecko_msg_le_gap_set_adv_data_rsp_t *msg
)
```

1.8.5 cmd_le_gap_set_adv_parameters

This command can be used to set Bluetooth LE advertisement parameters.

Table 1.171. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
3	0x04	method	Message ID
4-5	uint16	interval_min	Minimum connection intervalValue multiplied in units of 0.625 msRange: 0x0020 to 0x4000Time range: 20 ms to 10.24 s
6-7	uint16	interval_max	Maximum connection intervalValue multiplied in units of 0.625 msRange: 0x0020 to 0x4000Time range: 20 ms to 10.24 sNote: interval_max must be at least equal to or bigger than interval_min
8	uint8	channel_map	Advertisement channel map which determines which of the three channels will be used for advertising. This value is given as a bit-mask.Values1 : Advertise on CH372 : Advertise on CH383 : Advertise on CH37 and CH384 : Advertise on CH395 : Advertise on CH37 and CH396 : Advertise on CH38 and CH397 : Advertise on all channelsRecommended value : 7 (0x7)

Table 1.172. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
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 codes

BGScript command

```
call le_gap_set_adv_parameters(interval_min,interval_max,channel_map)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_le_gap_set_adv_parameters(uint16 interval_min, uint16 interval_max, uint8 channel_map);

/* Callback */
struct gecko_msg_le_gap_set_adv_parameters_rsp_t
{
  uint16 result
}
void gecko_rsp_le_gap_set_adv_parameters(
  const struct gecko_msg_le_gap_set_adv_parameters_rsp_t *msg
)
```

1.8.6 cmd_le_gap_set_conn_parameters

This command can be used to set the default Bluetooth LE connection parameters. The configured values are valid for all subsequent connections that will be established. For changing the parameters of an already established connection use the command le_connection_set_parameters.

Table 1.173. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x08	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
3	0x05	method	Message ID
4-5	uint16	min_interval	Minimum connection intervalMinimum value for the connection event interval. This shall be less than or equal to max_intervalTime = Value x 1.25 msRange: 0x0006 to 0x0c80Time Range: 7.5 ms to 4 s
6-7	uint16	max_interval	Maximum connection intervalMaximum value for the connection event interval. This must be set greater than or equal to min_intervalTime = Value x 1.25 msRange: 0x0006 to 0x0c80Time Range: 7.5 ms to 4 s
8-9	uint16	latency	Slave LatencyThis parameter defines how many connection intervals the slave can skip if it has no data to sendRange: 0x0000 to 0x01F4Use 0x0000 as default.
10-11	uint16	timeout	Supervision timeout (in units of 10 ms)The supervision timeout defines for how long the connection is maintained despite the devices being unable to communicate at the currently configured connection intervals.Range: 0x000A to 0x0C80Time = Value x 10 msRange: 0x000A to 0x0C80Time Range: 100 ms to 32 sThe minimum value must be at least maximum interval * (latency + 1)It is recommended that the supervision timeout value is set with a value which allows communication attempts for at least a couple of connection intervals.

Table 1.174. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
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 codes

BGScript command

```
call le_gap_set_conn_parameters(min_interval,max_interval,latency,timeout)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_le_gap_set_conn_parameters(uint16 min_interval, uint16 max_interval, uint16 latency, uint16 time
out);

/* Callback */
struct gecko_msg_le_gap_set_conn_parameters_rsp_t
{
  uint16 result
}
void gecko_rsp_le_gap_set_conn_parameters(
  const struct gecko_msg_le_gap_set_conn_parameters_rsp_t *msg
)
```

1.8.7 cmd_le_gap_set_mode

This command can be used to configure the current Bluetooth LE GAP Connectable and Discoverable modes. It can be used to enable advertisements and/or allow incoming connections. To exit from this mode (to stop advertising) use the command le_gap_end_procedure.

Table 1.175. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
3	0x01	method	Message ID
4	uint8	discover	Discoverable mode
5	uint8	connect	Connectable mode

Table 1.176. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
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 codes

BGScript command

```
call le_gap_set_mode(discover,connect)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_le_gap_set_mode(uint8 discover, uint8 connect);

/* Callback */
struct gecko_msg_le_gap_set_mode_rsp_t
{
  uint16 result
}
void gecko_rsp_le_gap_set_mode(
  const struct gecko_msg_le_gap_set_mode_rsp_t *msg
)
```

1.8.8 cmd_le_gap_set_scan_parameters

This command can be used to set Bluetooth LE scan parameters.

Table 1.177. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
3	0x06	method	Message ID
4-5	uint16	scan_interval	Scanner intervalThis is defined as the time interval from when the module started its last LE scan until it begins the subsequent LE scan, i.e. how often to scanTime = Value x 0.625 msRange: 0x0004 to 0x4000Time Range: 2.5 ms to 10.24 sDefault: 0x0010 (10 ms)
6-7	uint16	scan_window	Scan windowThe duration of the LE scan. scan_window shall be less than or equal to scan_intervalTime = Value x 0.625 msRange: 0x0004 to 0x4000Time Range: 2.5 ms to 10.24 sDefault: 0x0010 (10 ms)
8	uint8	active	Scan type indicated by a flagValues0: Passive scanning1: Active scanningDefault value: 0In passive scanning mode the module only listens for advertising packets and will not transmit any packetIn active scanning mode the module will send out a scan request packet upon receiving advertising packet from a remote device and then it will listen for the scan response packet from remote device

Table 1.178. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
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 codes

BGScript command

```
call le_gap_set_scan_parameters(scan_interval,scan_window,active)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_le_gap_set_scan_parameters(uint16 scan_interval, uint16 scan_window, uint8 active);

/* Callback */
struct gecko_msg_le_gap_set_scan_parameters_rsp_t
{
  uint16 result
```

```
}
void gecko_rsp_le_gap_set_scan_parameters(
  const struct gecko_msg_le_gap_set_scan_parameters_rsp_t *msg
)
```

1.8.9 evt_le_gap_scan_response

This event reports any advertisement or scan response packet that is received by the module's radio while in scanning mode.

Table 1.179. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x0b	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
3	0x00	method	Message ID
4	int8	rssi	Received signal strength indicator (RSSI)Range: -127 to +20Units: dBm
5	uint8	packet_type	Advertisement packet type. Values: • **0x00:** Connectable undirected advertising • **0x02:** Scannable undirected advertising • **0x03:** Non connectable undirected advertising • **0x04:** Scan Response Note: Scan response (0x04) is only received if module is in active scan mode.
6-11	bd_addr	address	Bluetooth address of the remote device
12	uint8	address_type	Advertiser address type. Values: • **0:** Public address • **1:** Random address
13	uint8	bonding	Bonding handle if the remote advertising module has previously bonded with the local module. Values: • **0xff:** No bonding • **Other:** Bonding handle
14	uint8array	data	Advertisement or scan response data

BGScript event

```
event le_gap_scan_response(rssi,packet_type,address,address_type,bonding,data_len, data_data)
```

C Functions

```
/* Callback */
struct gecko_msg_le_gap_scan_response_evt_t
{
  int8 rssi,
  uint8 packet_type,
  bd_addr address,
  uint8 address_type,
  uint8 bonding,
  uint8 data_len,
  const uint8 *data_data
}
void gecko_rsp_le_gap_scan_response(
  const struct gecko_msg_le_gap_scan_response_evt_t *msg
)
```

1.8.10 enum_le_gap_address_type

Bluetooth Address types used by stack

Table 1.180. Enumerations

Value	Name	Description
0	le_gap_address_type_public	LE public address
1	le_gap_address_type_random	LE random address
2	le_gap_address_type_public_identity	LE public identity address resolved by stack
3	le_gap_address_type_random_identity	LE random identity address resolved by stack
16	le_gap_address_type_bredr	Classic Bluetooth address

1.8.11 enum_le_gap_connectable_mode

These values indicate which connectable mode the module is to be set in.

Table 1.181. Enumerations

Value	Name	Description
0	le_gap_non_connectable	Not connectable
1	le_gap_directed_connectable	Direct ConnectableDO NOT USE
2	le_gap_undirected_connectable	Undirected connectable
3	le_gap_scannable_non_connectable	Not connectable but responds to scan_req-packets

1.8.12 enum_le_gap_discoverable_mode

These values indicate which discoverable mode the module is to be set in.

Table 1.182. Enumerations

Value	Name	Description
0	le_gap_non_discoverable	Not discoverable
1	le_gap_limited_discoverable	Discoverable using both limited discoverable mode
2	le_gap_general_discoverable	Discoverable using general discoverable mode
3	le_gap_broadcast	Limited or general discoverable bits not in flags ad type-Device is not discoverable in limited or generic discoverable procedure
4	le_gap_user_data	Send advertisement and/or scan response data defined by user using le_gap_set_adv_data command

1.8.13 enum_le_gap_discover_mode

These values indicate which Bluetooth LE discovery mode to use when scanning for advertising slaves.

Table 1.183. Enumerations

Value	Name	Description
0	le_gap_discover_limited	Discover only limited discoverable devices
1	le_gap_discover_generic	Discover limited and generic discoverable devices
2	le_gap_discover_observation	Discover all devices

1.9 Security Manager (sm)

The commands in this section are used to manage Bluetooth security, including commands for starting and stopping encryption and commands for management of all bonding operations.

1.9.1 cmd_sm_configure

This command can be used to configure authentication methods and I/O capabilities of the system.

Table 1.184. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x01	method	Message ID
4	uint8	mitm_required	Require MITM flagValues0 : Allow bonding without MITM protection1 : Require MITM protection
5	uint8	io_capabilities	I/O Capabilities.See link

Table 1.185. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x00	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x01	method	Message ID

BGScript command

```
call sm_configure(mitm_required,io_capabilities)()
```

BGLIB C API

```
/* Function */

void gecko_cmd_sm_configure(uint8 mitm_required, uint8 io_capabilities);

/* Callback */
struct gecko_msg_sm_configure_rsp_t
{
}
void gecko_rsp_sm_configure(
  const struct gecko_msg_sm_configure_rsp_t *msg
)
```

1.9.2 cmd_sm_delete_bonding

This command can be used to delete specified bonding information from persistent store.

Table 1.186. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x06	method	Message ID
4	uint8	bonding	Bonding handle

Table 1.187. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
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 codes

BGScript command

```
call sm_delete_bonding(bonding)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_sm_delete_bonding(uint8 bonding);

/* Callback */
struct gecko_msg_sm_delete_bonding_rsp_t
{
  uint16 result
}
void gecko_rsp_sm_delete_bonding(
  const struct gecko_msg_sm_delete_bonding_rsp_t *msg
)
```

1.9.3 cmd_sm_delete_bondings

This command can be used to delete all bonding information from persistent store.

Table 1.188. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x07	method	Message ID

Table 1.189. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
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 codes

BGScript command

```
call sm_delete_bondings()(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_sm_delete_bondings();

/* Callback */
struct gecko_msg_sm_delete_bondings_rsp_t
{
  uint16 result
}
void gecko_rsp_sm_delete_bondings(
  const struct gecko_msg_sm_delete_bondings_rsp_t *msg
)
```

1.9.4 cmd_sm_enter_passkey

This command can be used to enter a passkey after receiving a passkey request event.

Table 1.190. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x05	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x08	method	Message ID
4	uint8	connection	Connection handle
5-8	uint32	passkey	Passkey

Table 1.191. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
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 codes

BGScript command

```
call sm_enter_passkey(connection,passkey)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_sm_enter_passkey(uint8 connection, uint32 passkey);

/* Callback */
struct gecko_msg_sm_enter_passkey_rsp_t
{
  uint16 result
}
void gecko_rsp_sm_enter_passkey(
  const struct gecko_msg_sm_enter_passkey_rsp_t *msg
)
```

1.9.5 cmd_sm_increase_security

This command can be used to enhance the security of a connection to current security requirements.

Table 1.192. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x04	method	Message ID
4	uint8	connection	Connection handle

Table 1.193. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
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 codes

BGScript command

```
call sm_increase_security(connection)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_sm_increase_security(uint8 connection);

/* Callback */
struct gecko_msg_sm_increase_security_rsp_t
{
  uint16 result
}
void gecko_rsp_sm_increase_security(
  const struct gecko_msg_sm_increase_security_rsp_t *msg
)
```

1.9.6 cmd_sm_list_all_bondings

This command can be used to list all bondings stored in the bonding database. Bondings are reported by using the sm_list_bonding_event for each bonding and the report is ended with sm_list_all_bonding_complete event. Recommended to be used only for debugging purposes.

Table 1.194. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x0b	method	Message ID

Table 1.195. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x0b	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call sm_list_all_bondings()(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_sm_list_all_bondings();

/* Callback */
struct gecko_msg_sm_list_all_bondings_rsp_t
{
  uint16 result
}
void gecko_rsp_sm_list_all_bondings(
  const struct gecko_msg_sm_list_all_bondings_rsp_t *msg
)
```

1.9.7 cmd_sm_passkey_confirm

This command can be used for confirming the reported confirm value.

Table 1.196. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x09	method	Message ID
4	uint8	connection	Connection handle
5	uint8	confirm	Accept confirm valueValues0: Reject1: Accept confirm value

Table 1.197. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x09	method	Message ID
4-5	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call sm_passkey_confirm(connection,confirm)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_sm_passkey_confirm(uint8 connection, uint8 confirm);

/* Callback */
struct gecko_msg_sm_passkey_confirm_rsp_t
{
  uint16 result
}
void gecko_rsp_sm_passkey_confirm(
  const struct gecko_msg_sm_passkey_confirm_rsp_t *msg
)
```

1.9.8 cmd_sm_read_bonding

This command can be used to read the encryption key for a specific bonding. Used in debugging for reading encryption keys which can be used e.g. for protocol sniffing.

Table 1.198. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x05	method	Message ID
4	uint8	bonding	Bonding index of bonding data

Table 1.199. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x0a	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
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 codes
6-11	bd_addr	address	Bluetooth address of the remote device
12	uint8	address_type	Bluetooth address of the remote device this bonding entry refers to
13	uint8array	bonding_key	Encryption key stored for this bonding entryMaximum 16 bytes

BGScript command

```
call sm_read_bonding(bonding)(result,address,address_type,bonding_key_len, bonding_key_data)
```

BGLIB C API

```
/* Function */

void gecko_cmd_sm_read_bonding(uint8 bonding);

/* Callback */
struct gecko_msg_sm_read_bonding_rsp_t
{
  uint16 result,
  bd_addr address,
  uint8 address_type,
  uint8 bonding_key_len,
  const uint8 *bonding_key_data
}
void gecko_rsp_sm_read_bonding(
  const struct gecko_msg_sm_read_bonding_rsp_t *msg
)
```

1.9.9 cmd_sm_read_bonding_configuration

This command can be used to read the maximum number of allowed bonding entries and to reveal the currently set bonding policy.

Table 1.200. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x03	method	Message ID

Table 1.201. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x04	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x03	method	Message ID
4	uint8	max_bonding_count	Maximum allowed bonding countRange: 1 to 32
5	uint8	policy_flags	Bonding policyValues0: If database is full bonding fails1: New bonding will overwrite the oldest existing bondNOT IMPLEMENTED YET
6-7	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call sm_read_bonding_configuration()(max_bonding_count,policy_flags,result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_sm_read_bonding_configuration();

/* Callback */
struct gecko_msg_sm_read_bonding_configuration_rsp_t
{
  uint8 max_bonding_count,
  uint8 policy_flags,
  uint16 result
}
void gecko_rsp_sm_read_bonding_configuration(
  const struct gecko_msg_sm_read_bonding_configuration_rsp_t *msg
)
```

1.9.10 cmd_sm_set_bondable_mode

This command can be used to set the device into bondable mode.

Table 1.202. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x00	method	Message ID
4	uint8	bondable	Bondable modeValues0: New bondings not accepted1: Bondings allowed

Table 1.203. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
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 codes

BGScript command

```
call sm_set_bondable_mode(bondable)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_sm_set_bondable_mode(uint8 bondable);

/* Callback */
struct gecko_msg_sm_set_bondable_mode_rsp_t
{
  uint16 result
}
void gecko_rsp_sm_set_bondable_mode(
  const struct gecko_msg_sm_set_bondable_mode_rsp_t *msg
)
```

1.9.11 cmd_sm_set_oob_data

This command can be used to set the oob data (out-of-band encryption data) for a device. The oob data is commonly referred to as the PIN code exchanged over an alternate path like NFC.

Table 1.204. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x0a	method	Message ID
4	uint8array	oob	OOB dataThis value can be either empty meaning that current oob data is cleared or 16 bytes carrying the oob data to setValue-sEmpty : Clear oob data16 bytes : Set oob data to value

Table 1.205. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x00	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x0a	method	Message ID

BGScript command

```
call sm_set_oob_data(oob_len, oob_data)()
```

BGLIB C API

```
/* Function */

void gecko_cmd_sm_set_oob_data(uint8 oob_len, const uint8 *oob_data);

/* Callback */
struct gecko_msg_sm_set_oob_data_rsp_t
{
}
void gecko_rsp_sm_set_oob_data(
  const struct gecko_msg_sm_set_oob_data_rsp_t *msg
)
```

1.9.12 cmd_sm_store_bonding_configuration

Set maximum allowed bonding count.

Table 1.206. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x02	method	Message ID
4	uint8	max_bonding_count	Maximum allowed bonding countRange: 1 to 32
5	uint8	policy_flags	Bonding policyValues0: If database is full, new bonding attempts will fail1: New bonding will overwrite the oldest existing bonding-NOT IMPLEMENTED YET

Table 1.207. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
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 codes

BGScript command

```
call sm_store_bonding_configuration(max_bonding_count,policy_flags)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_sm_store_bonding_configuration(uint8 max_bonding_count, uint8 policy_flags);

/* Callback */
struct gecko_msg_sm_store_bonding_configuration_rsp_t
{
  uint16 result
}
void gecko_rsp_sm_store_bonding_configuration(
  const struct gecko_msg_sm_store_bonding_configuration_rsp_t *msg
)
```

1.9.13 evt_sm_bonded

This event is triggered after the bonding procedure has been succesfully completed.

Table 1.208. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x02	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x03	method	Message ID
4	uint8	connection	Connection handle
5	uint8	bonding	Bonding handleValues0xff: Procedure completed but bonding was not createdOther: Procedure completed, bonding handle

BGScript event

```
event sm_bonded(connection,bonding)
```

C Functions

```
/* Callback */
struct gecko_msg_sm_bonded_evt_t
{
  uint8 connection,
  uint8 bonding
}
void gecko_rsp_sm_bonded(
  const struct gecko_msg_sm_bonded_evt_t *msg
)
```

1.9.14 evt_sm_bonding_failed

This event is triggered if the bonding procedure has failed.

Table 1.209. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x03	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x04	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	reason	Describes error that occurred during bonding

BGScript event

```
event sm_bonding_failed(connection,reason)
```

C Functions

```
/* Callback */
struct gecko_msg_sm_bonding_failed_evt_t
{
  uint8 connection,
  uint16 reason
}
void gecko_rsp_sm_bonding_failed(
  const struct gecko_msg_sm_bonding_failed_evt_t *msg
)
```

1.9.15 evt_sm_confirm_passkey

This event indicates a request to display the passkey to the user and for the user to confirm the displayed passkey. Use the command sm_passkey_confirm to accept the displayed passkey.

Table 1.210. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x05	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x02	method	Message ID
4	uint8	connection	Connection handle
5-8	uint32	passkey	PasskeyRange: 0 to 999999NOTE! NOTE! When displaying the passkey to the user you must insert prefix zeros in order to obtain a 6 digit number if the passkey reported by this event is less than 6 digits.Example:Passkey value is 42Number to display to user is 000042

BGScript event

```
event sm_confirm_passkey(connection,passkey)
```

C Functions

```
/* Callback */
struct gecko_msg_sm_confirm_passkey_evt_t
{
  uint8 connection,
  uint32 passkey
}
void gecko_rsp_sm_confirm_passkey(
  const struct gecko_msg_sm_confirm_passkey_evt_t *msg
)
```

1.9.16 evt_sm_list_all_bondings_complete

This event is triggered by the sm_list_all_bondings and follows sm_list_bonding_entry events.

Table 1.211. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x00	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x06	method	Message ID

BGScript event

```
event sm_list_all_bondings_complete()
```

C Functions

```
/* Callback */
struct gecko_msg_sm_list_all_bondings_complete_evt_t
{
}
void gecko_rsp_sm_list_all_bondings_complete(
  const struct gecko_msg_sm_list_all_bondings_complete_evt_t *msg
)
```

1.9.17 evt_sm_list_bonding_entry

This event is triggered by the command sm_list_all_bondings if bondings exist in the local database.

Table 1.212. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x08	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x05	method	Message ID
4	uint8	bonding	Bonding index of bonding data
5-10	bd_addr	address	Bluetooth address of the remote device
11	uint8	address_type	Address type

BGScript event

```
event sm_list_bonding_entry(bonding,address,address_type)
```

C Functions

```
/* Callback */
struct gecko_msg_sm_list_bonding_entry_evt_t
{
  uint8 bonding,
  bd_addr address,
  uint8 address_type
}
void gecko_rsp_sm_list_bonding_entry(
  const struct gecko_msg_sm_list_bonding_entry_evt_t *msg
)
```

1.9.18 evt_sm_passkey_display

This event indicates a request to display the passkey to the user.

Table 1.213. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x05	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x00	method	Message ID
4	uint8	connection	Connection handle
5-8	uint32	passkey	PasskeyRange: 0 to 999999NOTE! When displaying the passkey to the user insert prefix zeros in order to obtain a 6 digit number-Example:Passkey value is 42Number to display to user is 000042

BGScript event

```
event sm_passkey_display(connection,passkey)
```

C Functions

```
/* Callback */
struct gecko_msg_sm_passkey_display_evt_t
{
  uint8 connection,
  uint32 passkey
}
void gecko_rsp_sm_passkey_display(
  const struct gecko_msg_sm_passkey_display_evt_t *msg
)
```

1.9.19 evt_sm_passkey_request

This event indicates a request for the user to enter the passkey displayed on the other remote device. Use the command sm_enter_passkey to input the passkey value.

Table 1.214. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x01	method	Message ID
4	uint8	connection	Connection handle

BGScript event

```
event sm_passkey_request(connection)
```

C Functions

```
/* Callback */
struct gecko_msg_sm_passkey_request_evt_t
{
  uint8 connection
}
void gecko_rsp_sm_passkey_request(
  const struct gecko_msg_sm_passkey_request_evt_t *msg
)
```

1.9.20 enum_sm_bonding_key

These values define the bonding information of the bonded device stored in persistent store.

Table 1.215. Enumerations

Value	Name	Description
1	sm_bonding_key_ltk	LTK saved in master
2	sm_bonding_key_addr_public	Public Address
4	sm_bonding_key_addr_static	Static Address
8	sm_bonding_key_irk	Identity resolving key for resolvable private addresses
16	sm_bonding_key_edivrand	EDIV+RAND received from slave
32	sm_bonding_key_csrk	Connection signature resolving key
64	sm_bonding_key_masterid	EDIV+RAND sent to master

1.9.21 enum_sm_io_capability

These values define the security management related I/O capabilities supported by the module

Table 1.216. Enumerations

Value	Name	Description
0	sm_io_capability_displayonly	Display Only
1	sm_io_capability_displayyesno	Display with Yes/No-buttons
2	sm_io_capability_keyboardonly	Keyboard Only
3	sm_io_capability_noinputnooutput	No Input and No Output
4	sm_io_capability_keyboarddisplay	Display with Keyboard

1.10 System (system)

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

1.10.1 cmd_system_get_bt_address

This command can be used to read the local Bluetooth address used by the module.

Table 1.217. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x03	method	Message ID

Table 1.218. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x06	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x03	method	Message ID
4-9	bd_addr	address	Bluetooth local address in little endian format

BGScript command

```
call system_get_bt_address()(address)
```

BGLIB C API

```
/* Function */

void gecko_cmd_system_get_bt_address();

/* Callback */
struct gecko_msg_system_get_bt_address_rsp_t
{
  bd_addr address
}
void gecko_rsp_system_get_bt_address(
  const struct gecko_msg_system_get_bt_address_rsp_t *msg
)
```

1.10.2 cmd_system_get_class_of_device

This command can be used to read the device's Class Of Device (COD) information.

Table 1.219. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x04	method	Message ID

Table 1.220. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x06	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x04	method	Message ID
4-7	uint32	cod	Class of DeviceFollows conventions listed in the Bluetooth specification published by Bluetooth SIGExample:0x001f00Uncategorized device
8-9	uint16	result	Result code • **0 : success** • **Non-zero : an error occurred** For other values refer to the Error codes

BGScript command

```
call system_get_class_of_device()(cod,result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_system_get_class_of_device();

/* Callback */
struct gecko_msg_system_get_class_of_device_rsp_t
{
  uint32 cod,
  uint16 result
}
void gecko_rsp_system_get_class_of_device(
  const struct gecko_msg_system_get_class_of_device_rsp_t *msg
)
```

1.10.3 cmd_system_get_local_name

This command can be used to read the friendly name of the local BR/EDR device. Note that for Bluetooth LE the device name is stored in the GAP Service of the GATT database.

Table 1.221. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x08	method	Message ID

Table 1.222. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x01	class	Message class: System
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 codes
6	uint8array	name	Local devices Bluetooth BR/EDR friendly name

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_system_get_local_name();

/* Callback */
struct gecko_msg_system_get_local_name_rsp_t
{
  uint16 result,
  uint8 name_len,
  const uint8 *name_data
}
void gecko_rsp_system_get_local_name(
  const struct gecko_msg_system_get_local_name_rsp_t *msg
)
```

1.10.4 cmd_system_hello

This command does not trigger any event but the response to the command is used to verify that communication between the host and the module is working.

Table 1.223. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x00	method	Message ID

Table 1.224. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x01	class	Message class: System
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 codes

BGScript command

```
call system_hello()(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_system_hello();

/* Callback */
struct gecko_msg_system_hello_rsp_t
{
  uint16 result
}
void gecko_rsp_system_hello(
  const struct gecko_msg_system_hello_rsp_t *msg
)
```

1.10.5 cmd_system_reset

This command can be used to reset the system. It does not have a response, but it triggers one of the boot events (normal reset or boot to DFU mode) depending on the selected BOOT mode.

Table 1.225. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x01	method	Message ID
4	uint8	dfu	Boot mode: • **0:** Normal reset • **1:** Boot to DFU mode

BGScript command

```
call system_reset(dfu)
```

BGLIB C API

```
/* Function */

void gecko_cmd_system_reset(uint8 dfu);

/* Command does not have callback */
```

Table 1.226. Events generated

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

1.10.6 cmd_system_reset_factory_settings

This command can be used to clear all customer settings including all pairing information. Also the PS Store (persistent store) is cleared. Note that Bluetooth address is conserved.

Table 1.227. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x06	method	Message ID

Table 1.228. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x01	class	Message class: System
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 codes

BGScript command

```
call system_reset_factory_settings()(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_system_reset_factory_settings();

/* Callback */
struct gecko_msg_system_reset_factory_settings_rsp_t
{
  uint16 result
}
void gecko_rsp_system_reset_factory_settings(
  const struct gecko_msg_system_reset_factory_settings_rsp_t *msg
)
```

1.10.7 cmd_system_set_class_of_device

This command is used to set the Class Of Device (COD) setting..

Table 1.229. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x05	method	Message ID
4-7	uint32	cod	Class of DeviceFollows conventions listed in the Bluetooth specification published by Bluetooth SIGExample:0x001f00Uncategorized device

Table 1.230. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x01	class	Message class: System
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 codes

BGScript command

```
call system_set_class_of_device(cod)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_system_set_class_of_device(uint32 cod);

/* Callback */
struct gecko_msg_system_set_class_of_device_rsp_t
{
  uint16 result
}
void gecko_rsp_system_set_class_of_device(
  const struct gecko_msg_system_set_class_of_device_rsp_t *msg
)
```

1.10.8 cmd_system_set_local_name

This command can be used to set the local devices Bluetooth BR/EDR friendly name. For BLE the device name should be stored in the GAP Service local name parameter of the GATT database.

Table 1.231. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x07	method	Message ID
4	uint8array	name	Local devices Bluetooth BR/EDR friendly name.Maximum name length is 30 bytes

Table 1.232. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x01	class	Message class: System
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 codes

BGScript command

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

BGLIB C API

```
/* Function */

void gecko_cmd_system_set_local_name(uint8 name_len, const uint8 *name_data);

/* Callback */
struct gecko_msg_system_set_local_name_rsp_t
{
  uint16 result
}
void gecko_rsp_system_set_local_name(
  const struct gecko_msg_system_set_local_name_rsp_t *msg
)
```

1.10.9 cmd_system_set_max_power_mode

This command can be used to set the most aggressive power power saving state allowed for the system.

Table 1.233. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x02	method	Message ID
4	uint8	power_mode	Maximum power mode. Values: • **1:** CPU allowed to enter idle mode • **2:** CPU allowed to enter idle and sleep modes The default value is 2. See BGM111 data sheet for details concerning power modes.

Table 1.234. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x01	class	Message class: System
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 codes

BGScript command

```
call system_set_max_power_mode(power_mode)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_system_set_max_power_mode(uint8 power_mode);

/* Callback */
struct gecko_msg_system_set_max_power_mode_rsp_t
{
  uint16 result
}
void gecko_rsp_system_set_max_power_mode(
  const struct gecko_msg_system_set_max_power_mode_rsp_t *msg
)
```

1.10.10 evt_system_boot

This event indicates the device has started and is ready to receive any command except Bluetooth-related commands. This event carries among others the firmware build number and other SW and HW identification codes.

Table 1.235. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x0c	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 number
10-11	uint16	build	Build number
12-13	uint16	bootloader	Bootloader version
14-15	uint16	hw	Hardware type

BGScript event

```
event system_boot(major,minor,patch,build,bootloader,hw)
```

C Functions

```
/* Callback */
struct gecko_msg_system_boot_evt_t
{
  uint16 major,
  uint16 minor,
  uint16 patch,
  uint16 build,
  uint16 bootloader,
  uint16 hw
}
void gecko_rsp_system_boot(
  const struct gecko_msg_system_boot_evt_t *msg
)
```

1.10.11 evt_system_initialized

This event indicates that Bluetooth radio is initialized and is ready to be used.

Table 1.236. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x06	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x01	method	Message ID
4-9	bd_addr	address	Bluetooth public address in little endian format

BGScript event

```
event system_initialized(address)
```

C Functions

```
/* Callback */
struct gecko_msg_system_initialized_evt_t
{
  bd_addr address
}
void gecko_rsp_system_initialized(
  const struct gecko_msg_system_initialized_evt_t *msg
)
```

1.11 testing commands (test)

1.11.1 cmd_test_device_under_test_mode

This command can be used to enable the DUT mode (Device Under Test).

Table 1.237. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
3	0x05	method	Message ID

Table 1.238. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
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 codes

BGScript command

```
call test_device_under_test_mode()(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_test_device_under_test_mode();

/* Callback */
struct gecko_msg_test_device_under_test_mode_rsp_t
{
  uint16 result
}
void gecko_rsp_test_device_under_test_mode(
  const struct gecko_msg_test_device_under_test_mode_rsp_t *msg
)
```

1.11.2 cmd_test_dtm_end

Direct Test Mode, Request to end test

Table 1.239. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
3	0x02	method	Message ID

Table 1.240. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
3	0x02	method	Message ID
4-5	uint16	result	Command result

BGScript command

```
call test_dtm_end()(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_test_dtm_end();

/* Callback */
struct gecko_msg_test_dtm_end_rsp_t
{
  uint16 result
}
void gecko_rsp_test_dtm_end(
  const struct gecko_msg_test_dtm_end_rsp_t *msg
)
```

1.11.3 cmd_test_dtm_rx

Direct Test Mode, Start RX test mode

Table 1.241. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
3	0x01	method	Message ID
4	uint8	channel	Bluetooth channel to use in testing: 0-78

Table 1.242. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
3	0x01	method	Message ID
4-5	uint16	result	Command result

BGScript command

```
call test_dtm_rx(channel)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_test_dtm_rx(uint8 channel);

/* Callback */
struct gecko_msg_test_dtm_rx_rsp_t
{
  uint16 result
}
void gecko_rsp_test_dtm_rx(
  const struct gecko_msg_test_dtm_rx_rsp_t *msg
)
```

1.11.4 cmd_test_dtm_tx

Direct test mode, Start TX test

Table 1.243. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
3	0x00	method	Message ID
4	uint8	packet_type	packet type to use for testing
5	uint8	length	Packet length to use in testing: 0-78
6	uint8	channel	Bluetooth channel to use in testing: 0-78

Table 1.244. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
3	0x00	method	Message ID
4-5	uint16	result	Command result

BGScript command

```
call test_dtm_tx(packet_type,length,channel)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_test_dtm_tx(uint8 packet_type, uint8 length, uint8 channel);

/* Callback */
struct gecko_msg_test_dtm_tx_rsp_t
{
  uint16 result
}
void gecko_rsp_test_dtm_tx(
  const struct gecko_msg_test_dtm_tx_rsp_t *msg
)
```

1.11.5 cmd_test_packet_test

This command can be used to start the packet mode testing.

Table 1.245. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x08	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
3	0x07	method	Message ID
4	uint8	mode	0 hopping 1 fixed
5	uint8	tx_freq	Bluetooth channel/frequency to use in testingRange 0 to 76 F = 2402 + 2k, when k from 0 to 39 F = 2403 +2(k-40) when k from 40 to 78
6	uint8	rx_freq	Bluetooth channel/frequency to use in testingRange 0 to 76 F = 2402 + 2k, when k from 0 to 39 F = 2403 +2(k-40) when k from 40 to 78
7	uint8	acl_type	
8-9	uint16	acl_len	
10	uint8	power	
11	uint8	disable_whitening	

Table 1.246. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
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 codes

BGScript command

```
call test_packet_test(mode,tx_freq,rx_freq,acl_type,acl_len,power,disable_whitening)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_test_packet_test(uint8 mode, uint8 tx_freq, uint8 rx_freq, uint8 acl_type, uint16 acl_len, uint
8 power, uint8 disable_whitening);

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

```
void gecko_rsp_test_packet_test(
  const struct gecko_msg_test_packet_test_rsp_t *msg
)
```

1.11.6 cmd_test_rx_test

This command can be used to start the RX test mode with continuous reception.

Table 1.247. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
3	0x06	method	Message ID
4	uint8	channel	Bluetooth channel/frequency to use in testingRange 0 to 76 F = 2402 + 2k, when k from 0 to 39 F = 2403 +2(k-40) when k from 40 to 78

Table 1.248. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
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 codes

BGScript command

```
call test_rx_test(channel)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_test_rx_test(uint8 channel);

/* Callback */
struct gecko_msg_test_rx_test_rsp_t
{
  uint16 result
}
void gecko_rsp_test_rx_test(
  const struct gecko_msg_test_rx_test_rsp_t *msg
)
```

1.11.7 cmd_test_ssp_debug

Turn simple pairing debug key on

Table 1.249. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
3	0x04	method	Message ID
4	uint8	enable	debug enable flag

Table 1.250. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x00	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
3	0x04	method	Message ID

BGScript command

```
call test_ssp_debug(enable)()
```

BGLIB C API

```
/* Function */

void gecko_cmd_test_ssp_debug(uint8 enable);

/* Callback */
struct gecko_msg_test_ssp_debug_rsp_t
{
}
void gecko_rsp_test_ssp_debug(
  const struct gecko_msg_test_ssp_debug_rsp_t *msg
)
```

1.11.8 cmd_test_tx_test

This command can be used to start the TX test mode with continuous transmission.

Table 1.251. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
3	0x03	method	Message ID
4	uint8	modulation	Modulation typeValues0: CW1: 1 Mbit GFSK2: 2 Mbit Pi/4-DQPSK3: 3 Mbit 8-PSK4: 1 Mbit GFSK (BLE)
5	uint8	channel	Bluetooth channel/frequency to use in testingRange 0 to 76 F = 2402 + 2k, when k from 0 to 39 F = 2403 +2(k-40) when k from 40 to 78
6	uint8	power	TX power levelRange: 0 to 150: Max TX power15: Min TX power

Table 1.252. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
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 codes

BGScript command

```
call test_tx_test(modulation,channel,power)(result)
```

BGLIB C API

```
/* Function */

void gecko_cmd_test_tx_test(uint8 modulation, uint8 channel, uint8 power);

/* Callback */
struct gecko_msg_test_tx_test_rsp_t
{
  uint16 result
}
void gecko_rsp_test_tx_test(
  const struct gecko_msg_test_tx_test_rsp_t *msg
)
```

1.11.9 evt_test_dtm_completed

Direct Test Mode, Command completed

Table 1.253. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x04	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
3	0x00	method	Message ID
4-5	uint16	result	Command result
6-7	uint16	number_of_packets	Number of packets received, only valid for test end command

BGScript event

```
event test_dtm_completed(result,number_of_packets)
```

C Functions

```
/* Callback */
struct gecko_msg_test_dtm_completed_evt_t
{
  uint16 result,
  uint16 number_of_packets
}
void gecko_rsp_test_dtm_completed(
  const struct gecko_msg_test_dtm_completed_evt_t *msg
)
```

1.11.10 enum_test_packet_type

Test packet types

Table 1.254. Enumerations

Value	Name	Description
0	test_pkt_prbs9	PRBS9 packet payload
1	test_pkt_11110000	11110000 packet payload
2	test_pkt_10101010	10101010 packet payload

1.12 Error codes

Table 1.255. Errors related to hardware

Code	Name	Description
0x0501	ps_store_full	PS Store is full
0x0502	ps_key_not_found	PS key not found
0x0503	i2c_ack_missing	i2c ack missing
0x0504	i2c_timeout	i2c timeout
0x0505	not_configured	Not configured

Table 1.256. Errors related to BGAPI protocol

Code	Name	Description
0x0101	invalid_conn_handle	Invalid connection handle
0x0102	waiting_response	Waiting response
0x0103	gatt_connection_timeout	GATT connection timeout
0x0180	invalid_param	Invalid Parameter
0x0181	wrong_state	Device in Wrong State
0x0182	out_of_memory	Out Of Memory
0x0183	not_implemented	Feature Not Implemented
0x0184	invalid_command	Command Not Recognized
0x0185	timeout	Timeout
0x0186	not_connected	Not Connected
0x0187	flow	flow
0x0188	user_attribute	User Attribute
0x0189	invalid_license_key	Invalid License Key
0x018a	command_too_long	Command Too Long
0x018b	out_of_bonds	Out of Bonds
0x018c	unspecified	Unspecified error
0x018d	hardware	Hardware failure
0x018e	buffers_full	Internal buffers are full
0x018f	disconnected	Disconnected
0x0190	too_many_requests	Too many requests
0x0191	not_supported	Feature Not Supported

Table 1.257. SDP errors

Code	Name	Description
0x0601	record_not_found	Service Record not found
0x0602	record_already_exist	Service Record already exist

Table 1.258. Errors from Security Manager Protocol

Code	Name	Description
0x0301	passkey_entry_failed	Passkey Entry Failed
0x0302	oob_not_available	OOB Data is not available
0x0303	authentication_requirements	Authentication Requirements
0x0304	confirm_value_failed	Confirm Value Failed
0x0305	pairing_not_supported	Pairing Not Supported
0x0306	encryption_key_size	Encryption Key Size
0x0307	command_not_supported	Command Not Supported
0x0308	unspecified_reason	Unspecified Reason
0x0309	repeated_attempts	Repeated Attempts
0x030a	invalid_parameters	Invalid Parameters

Table 1.259. Bluetooth errors

Code	Name	Description
0x0204	page_timeout	Page Timeout
0x0205	authentication_failure	Authentication Failure
0x0206	pin_or_key_missing	Pin or Key Missing
0x0207	memory_capacity_exceeded	Memory Capacity Exceeded
0x0208	connection_timeout	Connection Timeout
0x0209	connection_limit_exceeded	Connection Limit Exceeded
0x020a	synchronous_connectiontion_limit_exceeded	Synchronous connection limit to a device exceeded
0x020b	acl_connection_already_exists	ACL Connection already exists
0x020c	command_disallowed	Command Disallowed
0x020d	connection_rejected_due_to_limited_resources	Connection rejected due to limited resources
0x020e	connection_rejected_due_to_security_reasons	Connection rejected due to security reasons
0x020f	connection_rejected_due_to_unacceptable_bd_addr	Connection rejected due to unacceptable Bluetooth address
0x0210	connection_accept_timeout_exceeded	Connection accept timeout exceeded
0x0211	unsupported_feature_or_parameter_value	Unsupported feature or parameter value
0x0212	invalid_command_parameters	Invalid Command Parameters
0x0213	remote_user_terminated	Remote User Terminated Connection
0x0214	remote_device_terminated_connection_due_to_low_resources	Remote device terminated connection due to low resources
0x0215	remote_powering_off	Remote Device Terminated Connection due to Power Off
0x0216	connection_terminated_by_local_host	Connection Terminated by Local Host
0x0217	repeated_attempts	Repeated attempts

Code	Name	Description
0x0218	pairing_not_allowed	Pairing not allowed
0x0219	unknown_lmp_pdu	Unknown LMP PDU
0x021a	unsupported_remote_feature	Unsupported remote feature / unsupported LMP feature
0x021b	sco_offset_rejected	SCO offset rejected
0x021c	sco_interval_rejected	SCO interval rejected
0x021d	sco_air_mode_rejected	SCO air mode rejected
0x021e	invalid_lmp_parameters	Invalid LMP parameters / Invalid LL parameters
0x021f	unspecified_error	Unspecified error
0x0220	unsupported_lmp_parameter_value	Unsupported LMP Parameter value / unsupported LL parameter value
0x0221	role_change_not_allowed	Role change not allowed
0x0222	ll_response_timeout	LL Response Timeout
0x0223	lmp_error_transaction_collision	LMP error transaction collision
0x0224	lmp_pdu_not_allowed	LMP PDU not allowed
0x0225	encryption_mode_not_acceptable	Encryption mode not acceptable
0x0226	link_key_cannot_be_changed	Link key cannot be changed
0x0227	requested_qos_not_supported	Requested QoS not supported
0x0228	instant_passed	Instant passed
0x0229	pairing_with_unit_key_not_supported	Pairing with unit key not supported
0x022a	different_transaction_collision	Different transaction collision
0x022c	qos_unacceptable_parameter	QoS unacceptable parameter
0x022d	qos_rejected	QoS rejected
0x022e	channel_assesment_not_supported	Channel assessment not supported
0x022f	insufficient_security	Insufficient security
0x0230	parameter_out_of_mandatory_range	Parameter out of mandatory range
0x0232	role_switch_pending	Role switch pending
0x0234	reserved_slot_violation	Reserved slot violation
0x0235	role_switch_failed	Role switch failed
0x0236	extended_inquiry_response_too_large	Extended inquiry response too large
0x0237	simple_pairing_not_supported_by_host	Simple pairing not supported by host
0x0238	host_busy_pairing	Host bysy-pairing
0x0239	connection_rejected_due_to_no_suitable_channel_found	Connection rejected due to no suitable channel found
0x023a	controller_busy	Controller Busy
0x023b	unacceptable_connection_interval	Unacceptable Connection Interval
0x023c	directed_advertising_timeout	Directed Advertising Timeout
0x023d	connection_terminated_due_to_mic_failure	MIC Failure
0x023e	connection_failed_to_be_established	Connection Failed to be Established

Code	Name	Description
0x023f	mac_connection_failed	MAC connection failed
0x0240	coarse_clock_adjustment_rejected_but_will_try_to_adjust_using_clock_dragging	Coarse clock adjustment rejected but will try to adjust using clock dragging

Table 1.260. Application errors

Code	Name	Description
0x0a01	file_open_failed	File open failed
0x0a02	xml_parse_failed	XML parse failed
0x0a03	device_connection_failed	Device connection failed
0x0a04	device_comunication_failed	Device communication failed

Table 1.261. Errors from Attribute Protocol

Code	Name	Description
0x0401	invalid_handle	Invalid Handle
0x0402	read_not_permitted	Read Not Permitted
0x0403	write_not_permitted	Write Not Permitted
0x0404	invalid_pdu	Invalid PDU
0x0405	insufficient_authentication	Insufficient Authentication
0x0406	request_not_supported	Request Not Supported
0x0407	invalid_offset	Invalid Offset
0x0408	insufficient_authorization	Insufficient Authorization
0x0409	prepare_queue_full	Prepare Queue Full
0x040a	att_not_found	Attribute Not Found
0x040b	att_not_long	Attribute Not Long
0x040c	insufficient_enc_key_size	Insufficient Encryption Key Size
0x040d	invalid_att_length	Invalid Attribute Value Length
0x040e	unlikely_error	Unlikely Error
0x040f	insufficient_encryption	Insufficient Encryption
0x0410	unsupported_group_type	Unsupported Group Type
0x0411	insufficient_resources	Insufficient Resources
0x0480	application	Application Error Codes

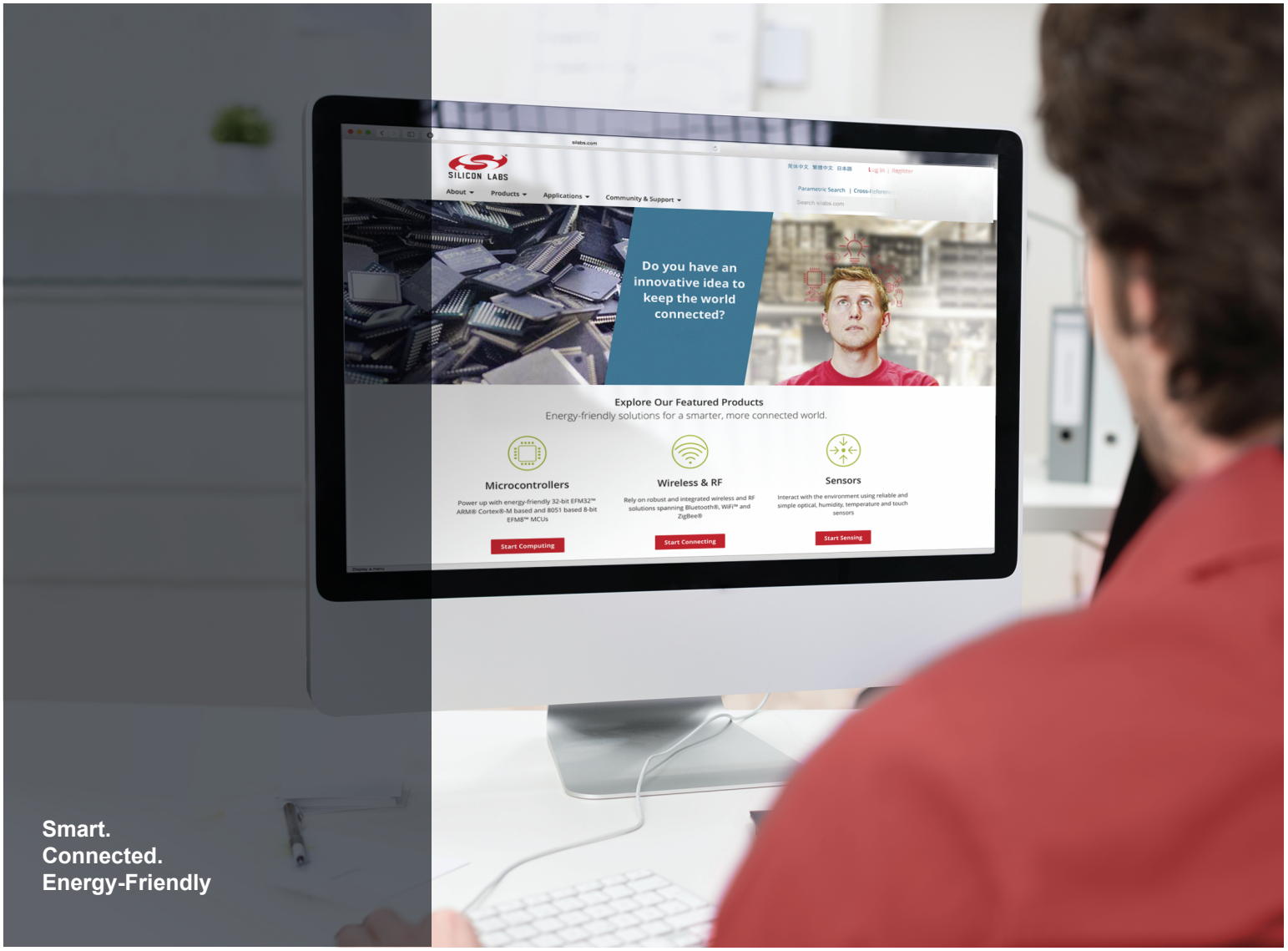
Table 1.262. Filesystem errors

Code	Name	Description
0x0901	file_not_found	File not found

2. Document Revision History

Table 2.1. Document Revision History

Revision Number	Effective Date	Change Description
1.0	01.04.2015	Initial version.



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS®

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

http://www.silabs.com