Bluetooth Smart Software API Reference Manual

This document contains the full API reference for the Silicon Labs Bluetooth Smart Software, version 2.6.1.

The Blue Gecko family of the Silicon Labs' Bluetooth Smart chipsets deliver a high performance, low energy and easy-to-use Bluetooth Smart solution integrated into a small form factor package.

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.

Table of Contents

1. Data types

Data types used in the documentation are shown in the table below. Unless otherwise noted, all multi-byte fields are in little endian format.

Table 1.1. Data types

Name	Length	Description
errorcode	2 bytes	Result code • **0:** success • **Non-zero:** an error occurred For other values refer to the Error codes
int16	2 bytes	Signed 16-bit integer
bd_addr	6 bytes	Bluetooth address
uint16	2 bytes	Unsigned 16-bit integer
int32	4 bytes	Signed 32-bit integer
uint32	4 bytes	Unsigned 32-bit integer
link_id_t	2 bytes	Link ID
int8	1 byte	Signed 8-bit integer
uint8	1 byte	Unsigned 8-bit integer
uint8array	1 - 256 bytes	Variable length byte array. The first byte defines the length of the data that follows, 0 - 255 bytes.
ser_name	16 bytes	Service name, 16-byte array
dbm	1 byte	Signal strength
connection	1 byte	Connection handle
service	4 bytes	GATT service handle This value is normally received from the gatt_service event.
characteristic	2 bytes	GATT characteristic handle This value is normally received from the gatt_characteristic event.
descriptor	2 bytes	GATT characteristic descriptor handle
uuid	1 byte	Universal Unique Identifier (UUID)
att_errorcode	1 byte	Attribute protocol error code • **0:** No error • **Non-zero:** See Bluetooth specification, Host volume, Attribute Protocol, Error Codes table.
att_opcode	1 byte	Attribute opcode which informs the procedure from which attribute the value was received
uuid_128	16 bytes	128-bit UUID
aes_key_128	16 bytes	128-bit AES Key

2. API Reference

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

BGAPI Payload

The parameters of a BGAPI command, response or event are passed between the application and firmware in a payload. For example, a parameter of uint32 type uses 4 bytes of the payload space. A byte array parameter uses one byte to describe the length of the array and the actual data in array is copied into the remaining free payload space.

Maximum BGAPI Payload Size

The maximum BGAPI payload size is 256 bytes for both NCP and SoC modes. When an application calls a BGAPI command, BGAPI checks the payload length and will return error code 0x018a (command_too_long) if the payload will cause an overflow.

Deprecation Notice

Note that some commands, enumerations and events are marked as deprecated. The usage of those commands is not recommended anymore as they will be removed in the future releases.

2.1 Cooexistence interface (coex)

Coexistence BGAPI class. Coexistence interface is enabled and initialised with gecko_initCoexHAL() function. Interface is configured with HAL configurator.

2.1.1 coex commands

2.1.1.1 cmd_coex_get_counters

This command is used to read coexistence statistic counters from the device. Response contains the list of uint32 type counter values. Counters in the list are in following order: low priority requested, high priority requested, low priority denied, high priority denied, low priority tx aborted, high priority tx aborted.

Table 2.1. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x20	class	Message class: Cooexistence interface
3	0x01	method	Message ID
4	uint8	reset	Reset counter values

Table 2.2. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x20	class	Message class: Cooexistence interface
3	0x01	method	Message ID
4-5	uint16	result	Result code • **0:** success • **Non-zero:** an error occurred For other values refer to the
6	uint8array	counters	Variable length byte array. The first byte defines the length of the data that follows, 0 - 255 bytes.

BGLIB C API

```
/* Function */
struct gecko_msg_coex_get_counters_rsp_t *gecko_cmd_coex_get_counters(uint8 reset);

/* Response id */
gecko_rsp_coex_get_counters_id

/* Response structure */
struct gecko_msg_coex_get_counters_rsp_t
{
  uint16 result;,
  uint8array counters;
};
```

2.1.1.2 cmd_coex_set_options

This command is used to configure coexistence options at runtime.

Table 2.3. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x08	lolen	Minimum payload length
2	0x20	class	Message class: Cooexistence interface
3	0x00	method	Message ID
4-7	uint32	mask	Mask defines which coexistence options are changed.
8-11	uint32	options	Value of options to be changed. This parameter is used together with mask parameter.

Table 2.4. Response

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

BGLIB C API

```
/* Function */
struct gecko_msg_coex_set_options_rsp_t *gecko_cmd_coex_set_options(uint32 mask, uint32 options);

/* Response id */
gecko_rsp_coex_set_options_id

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

2.1.2 coex enumerations

2.1.2.1 enum_coex_option

Coexistence configuration options

Table 2.5. Enumerations

Value	Name	Description
256	coex_option_enable	Enable coexistence feature
1024	coex_option_tx_abort	Abort transmission if grant is denied
2048	coex_option_high_priority	Enable priority signal

2.2 Device Firmware Upgrade (dfu)

These commands and events are related to controlling firmware update over the configured host interface and are available only when the device has been booted into DFU mode. **The DFU process:**

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 when all the data has been uploaded
6. Finalize the DFU firmware update with Reset command.

DFU mode is using UART baudrate from hardware configuration of firmware. Default baudrate 115200 is used if firmware is missing or firmware content does not match with CRC checksum.

2.2.1 dfu commands

2.2.1.1 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 to where the new firmware will be written in.

Table 2.6. 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. Always use the value 0x00000000.

Table 2.7. 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

BGLIB C API

```
/* Function */
struct gecko_msg_dfu_flash_set_address_rsp_t *gecko_cmd_dfu_flash_set_address(uint32 address);

/* Response id */
gecko_rsp_dfu_flash_set_address_id

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

2.2.1.2 cmd_dfu_flash_upload

This command can be used to upload the whole firmware image file into the Bluetooth device. The passed data length must be a multiple of 4 bytes. As the BGAPI command payload size is limited, multiple commands need to be issued one after the other until the whole .bin firmware image file is uploaded to the device. The next address of the flash sector in memory to write to is automatically updated by the bootloader after each individual command.

Table 2.8. 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 which will be written onto the flash.

Table 2.9. 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

BGLIB C API

```
/* Function */
struct gecko_msg_dfu_flash_upload_rsp_t *gecko_cmd_dfu_flash_upload(uint8 data_len, const uint8 *data_data);

/* Response id */
gecko_rsp_dfu_flash_upload_id

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

2.2.1.3 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 DFU Reset must be issued next.

Table 2.10. 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 2.11. 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

BGLIB C API

```
/* Function */
struct gecko_msg_dfu_flash_upload_finish_rsp_t *gecko_cmd_dfu_flash_upload_finish();

/* Response id */
gecko_rsp_dfu_flash_upload_finish_id

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

2.2.1.4 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 2.12. 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 UART DFU mode • **2:** Boot to OTA DFU mode

BGLIB C API

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

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

Table 2.13. Events Generated

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

2.2.2 dfu events

2.2.2.1 evt_dfu_boot

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

Table 2.14. 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

C Functions

```
/* Event id */
gecko_evt_dfu_boot_id

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

2.2.2.2 evt_dfu_boot_failure

This event indicates that there has been error in bootloader, which prevents the device from booting.

Table 2.15. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x02	lolen	Minimum payload length
2	0x00	class	Message class: Device Firmware Upgrade
3	0x01	method	Message ID
4-5	uint16	reason	The reason for boot failure, refer to the Error codes

C Functions

```
/* Event id */
gecko_evt_dfu_boot_failure_id

/* Event structure */
struct gecko_msg_dfu_boot_failure_evt_t
{
  uint16 reason;
};
```

2.3 Endpoint (endpoint)

This class provides a command for closing Bluetooth LE connections.

2.3.1 endpoint commands

2.3.1.1 cmd_endpoint_close

Deprecated. Use new command le_connection_close to close Bluetooth LE connections.

This command can be used to close a Bluetooth LE connection. The parameter is a connection handle which is reported in event le_connection_opened.

Table 2.16. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0b	class	Message class: Endpoint
3	0x02	method	Message ID
4	uint8	endpoint	The connection handle

Table 2.17. 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 connection handle that was closed

BGLIB C API

```
/* Function */
struct gecko_msg_endpoint_close_rsp_t *gecko_cmd_endpoint_close(uint8 endpoint);

/* Response id */
gecko_rsp_endpoint_close_id

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

Table 2.18. Events Generated

Event	Description
le_connection_closed	This event indicates that a connection was closed.

2.4 Persistent Store (flash)

Deprecated. Persistent Store commands can be used to manage the user data in the flash memory of the Bluetooth device. User data stored in PS keys within the flash memory is persistent across reset and power cycling of the device. The maximum user data size associated to a PS key is 56 bytes.

2.4.1 flash commands

2.4.1.1 cmd_flash_ps_dump

Deprecated. 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 2.19. 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 2.20. 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

BGLIB C API

```
/* Function */
struct gecko_msg_flash_ps_dump_rsp_t *gecko_cmd_flash_ps_dump();

/* Response id */
gecko_rsp_flash_ps_dump_id

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

Table 2.21. Events Generated

Event	Description
flash_ps_key	PS Key contents

2.4.1.2 cmd_flash_ps_erase

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

Table 2.22. 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 2.23. 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

BGLIB C API

```
/* Function */
struct gecko_msg_flash_ps_erase_rsp_t *gecko_cmd_flash_ps_erase(uint16 key);

/* Response id */
gecko_rsp_flash_ps_erase_id

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

2.4.1.3 cmd_flash_ps_erase_all

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

Table 2.24. 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 2.25. 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

BGLIB C API

```
/* Function */
struct gecko_msg_flash_ps_erase_all_rsp_t *gecko_cmd_flash_ps_erase_all();

/* Response id */
gecko_rsp_flash_ps_erase_all_id

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

2.4.1.4 cmd_flash_ps_load

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

Table 2.26. 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 2.27. 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.

BGLIB C API

```
/* Function */
struct gecko_msg_flash_ps_load_rsp_t *gecko_cmd_flash_ps_load(uint16 key);

/* Response id */
gecko_rsp_flash_ps_load_id

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

2.4.1.5 cmd_flash_ps_save

Deprecated. This command can be used to store a value into the specified PS key. Allowed PS keys are in range from 0x4000 to 0x407F. At most 56 bytes user data can be stored in one PS key. Error code 0x018a (command_too_long) will be returned if more than 56 bytes data is passed in.

Table 2.28. 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 2.29. 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

BGLIB C API

```
/* Function */
struct gecko_msg_flash_ps_save_rsp_t *gecko_cmd_flash_ps_save(uint16 key, uint8 value_len, const uint8 *value_d
ata);

/* Response id */
gecko_rsp_flash_ps_save_id

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

2.4.2 flash events

2.4.2.1 evt_flash_ps_key

Deprecated. This event indicates that the flash_ps_dump command was given. It returns a single PS key and its value. There can be multiple events if multiple PS keys exist.

Table 2.30. 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.

C Functions

```
/* Event id */
gecko_evt_flash_ps_key_id

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

2.5 Generic Attribute Profile (gatt)

The commands and events in this class can be used to browse and manage attributes in a remote GATT server.

2.5.1 gatt commands

2.5.1.1 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 2.31. 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 handle This value is normally received from the gatt_service event.

Table 2.32. 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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_discover_characteristics_rsp_t *gecko_cmd_gatt_discover_characteristics(uint8 connection
, uint32 service);

/* Response id */
gecko_rsp_gatt_discover_characteristics_id

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

Table 2.33. Events Generated

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

2.5.1.2 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 2.34. 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 handle This value is normally received from the gatt_service event.
9	uint8array	uuid	Characteristic UUID

Table 2.35. 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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_discover_characteristics_by_uuid_rsp_t *gecko_cmd_gatt_discover_characteristics_by_uuid(u
int8 connection, uint32 service, uint8 uuid_len, const uint8 *uuid_data);

/* Response id */
gecko_rsp_gatt_discover_characteristics_by_uuid_id

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

Table 2.36. Events Generated

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

2.5.1.3 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 2.37. 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 handle This value is normally received from the gatt_characteristic event.

Table 2.38. 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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_discover_descriptors_rsp_t *gecko_cmd_gatt_discover_descriptors(uint8 connection, uint16
characteristic);

/* Response id */
gecko_rsp_gatt_discover_descriptors_id

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

Table 2.39. Events Generated

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

2.5.1.4 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 successfully completed or failed with error.

Table 2.40. 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 2.41. 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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_discover_primary_services_rsp_t *gecko_cmd_gatt_discover_primary_services(uint8 connectio
n);

/* Response id */
gecko_rsp_gatt_discover_primary_services_id

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

Table 2.42. Events Generated

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

2.5.1.5 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 2.43. 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	Service UUID

Table 2.44. 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

BGLIB C API

```
/* Function */
struct  gecko_msg_gatt_discover_primary_services_by_uuid_rsp_t  *gecko_cmd_gatt_discover_primary_services_by_uui
d(uint8 connection, uint8 uuid_len, const uint8 *uuid_data);

/* Response id */
gecko_rsp_gatt_discover_primary_services_by_uuid_id

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

Table 2.45. Events Generated

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

2.5.1.6 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 successfully or that an error response has been received.

Table 2.46. 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 8-bit integer

Table 2.47. 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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_execute_characteristic_value_write_rsp_t *gecko_cmd_gatt_execute_characteristic_value_wri
te(uint8 connection, uint8 flags);

/* Response id */
gecko_rsp_gatt_execute_characteristic_value_write_id

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

Table 2.48. Events Generated

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

2.5.1.7 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 2.49. 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 handle This value is normally received from the gatt_service event.

Table 2.50. 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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_find_included_services_rsp_t *gecko_cmd_gatt_find_included_services(uint8 connection, uin
t32 service);

/* Response id */
gecko_rsp_gatt_find_included_services_id

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

Table 2.51. Events Generated

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

2.5.1.8 cmd_gatt_prepare_characteristic_value_reliable_write

This command can be used to add a characteristic value to the write queue of a remote GATT server and verify if the value was correctly received by the server. Received gatt_procedure_completed event indicates that this GATT procedure has succesfully completed or failed with error. Specifically, error code 0x0194 (data_corrupted) will be returned if the value received from the GATT server's response failed to pass the reliable write verification. At most ATT_MTU - 5 amount of data can be sent once. Writes are executed or cancelled with the execute_characteristic_value_write command. Whether the writes succeeded or not are indicated in the response of the execute_characteristic_value_write command.

Table 2.52. 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	0x13	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handle This value is normally received from the gatt_characteristic event.
7-8	uint16	offset	Offset of the characteristic value
9	uint8array	value	Value to write into the specified characteristic of the remote GATT database

Table 2.53. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x04	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x13	method	Message ID
4-5	uint16	result	Result code • **0:** success • **Non-zero:** an error occurred For other values refer to the Error codes
6-7	uint16	sent_len	The length of data sent to the remote GATT server

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_prepare_characteristic_value_reliable_write_rsp_t *gecko_cmd_gatt_prepare_characteristic_
value_reliable_write(uint8 connection, uint16 characteristic, uint16 offset, uint8 value_len, const uint8 *valu
e_data);

/* Response id */
gecko_rsp_gatt_prepare_characteristic_value_reliable_write_id

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

```
  uint16 sent_len;
};
```

Table 2.54. Events Generated

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

2.5.1.9 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. This command can be used in cases where very long attributes need to be written, or a set of values needs to be written atomically. At most ATT_MTU - 5 amount of data can be sent once. Writes are executed or cancelled with the execute_characteristic_value_write command. Whether the writes succeeded or not are indicated in the response of the execute_characteristic_value_write command.

In all cases where the amount of data to transfer fits into the BGAPI payload the command gatt_write_characteristic_value is recommended for writing long values since it transparently performs the prepare_write and execute_write commands.

Table 2.55. 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 handle This value is normally received from the gatt_characteristic event.
7-8	uint16	offset	Offset of the characteristic value
9	uint8array	value	Value to write into the specified characteristic of the remote GATT database

Table 2.56. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x04	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
6-7	uint16	sent_len	The length of data sent to the remote GATT server

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_prepare_characteristic_value_write_rsp_t *gecko_cmd_gatt_prepare_characteristic_value_wri
te(uint8 connection, uint16 characteristic, uint16 offset, uint8 value_len, const uint8 *value_data);

/* Response id */
gecko_rsp_gatt_prepare_characteristic_value_write_id

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

```
  uint16 sent_len;
};
```

Table 2.57. Events Generated

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

2.5.1.10 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 characteristic value fits in one ATT PDU. Otherwise more than one gatt_characteristic_value events are generated because the firmware will automatically use the "read long" GATT procedure. A received gatt_procedure_completed event indicates that all data has been read successfully or that an error response has been received.

Table 2.58. 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 handle This value is normally received from the gatt_characteristic event.

Table 2.59. 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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_read_characteristic_value_rsp_t *gecko_cmd_gatt_read_characteristic_value(uint8 connectio
n, uint16 characteristic);

/* Response id */
gecko_rsp_gatt_read_characteristic_value_id

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

Table 2.60. Events Generated

Event	Description
gatt_characteristic_value	This event contains the data belonging to a characteristic sent by the GATT Server.
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

2.5.1.11 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 characteristic value fits in one ATT PDU. Otherwise more than one gatt_characteristic_value events are generated because the firmware will automatically use the "read long" GATT procedure. A received gatt_procedure_completed event indicates that all data has been read successfully or that an error response has been received.

Table 2.61. 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 handle This value is normally received from the gatt_service event.
9	uint8array	uuid	Characteristic UUID

Table 2.62. 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

BGLIB C API

```
/* Function */
struct  gecko_msg_gatt_read_characteristic_value_by_uuid_rsp_t  *gecko_cmd_gatt_read_characteristic_value_by_uui
d(uint8 connection, uint32 service, uint8 uuid_len, const uint8 *uuid_data);

/* Response id */
gecko_rsp_gatt_read_characteristic_value_by_uuid_id

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

Table 2.63. Events Generated

Event	Description
gatt_characteristic_value	This event contains the data belonging to a characteristic sent by the GATT Server.

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

2.5.1.12 cmd_gatt_read_characteristic_value_from_offset

This command can be used to read a partial characteristic value with specified offset and maximum length from a remote GATT database. It is equivalent to gatt_read_characteristic_value if both the offset and maximum length parameters are 0. A single gatt_characteristic_value event is generated if the value to read fits in one ATT PDU. Otherwise more than one gatt_characteristic_value events are generated because the firmware will automatically use the "read long" GATT procedure. A received gatt_procedure_completed event indicates that all data has been read successfully or that an error response has been received.

Table 2.64. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x07	lolen	Minimum payload length
2	0x09	class	Message class: Generic Attribute Profile
3	0x12	method	Message ID
4	uint8	connection	Connection handle
5-6	uint16	characteristic	GATT characteristic handle This value is normally received from the gatt_characteristic event.
7-8	uint16	offset	Offset of the characteristic value
9-10	uint16	maxlen	Maximum bytes to read. If this parameter is 0 all characteristic value starting at given offset will be read.

Table 2.65. 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	0x12	method	Message ID
4-5	uint16	result	Result code • **0:** success • **Non-zero:** an error occurred For other values refer to the Error codes

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_read_characteristic_value_from_offset_rsp_t *gecko_cmd_gatt_read_characteristic_value_fro
m_offset(uint8 connection, uint16 characteristic, uint16 offset, uint16 maxlen);

/* Response id */
gecko_rsp_gatt_read_characteristic_value_from_offset_id

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

Table 2.66. Events Generated

Event	Description
gatt_characteristic_value	This event contains the data belonging to a characteristic sent by the GATT Server.
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

2.5.1.13 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 descriptor value fits in one ATT PDU. Otherwise more than one gatt_descriptor_value events are generated because the firmware will automatically use the "read long" GATT procedure. A received gatt_procedure_completed event indicates that all data has been read successfully or that an error response has been received.

Table 2.67. 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 2.68. 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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_read_descriptor_value_rsp_t *gecko_cmd_gatt_read_descriptor_value(uint8 connection, uint1
6 descriptor);

/* Response id */
gecko_rsp_gatt_read_descriptor_value_id

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

Table 2.69. Events Generated

Event	Description
gatt_descriptor_value	Descriptor value received from the remote GATT server.
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

2.5.1.14 cmd_gatt_read_multiple_characteristic_values

This command can be used to 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. A received gatt_procedure_completed event indicates that either all data has been read successfully or that an error response has been received.

Table 2.70. 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 2.71. 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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_read_multiple_characteristic_values_rsp_t *gecko_cmd_gatt_read_multiple_characteristic_va
lues(uint8 connection, uint8 characteristic_list_len, const uint8 *characteristic_list_data);

/* Response id */
gecko_rsp_gatt_read_multiple_characteristic_values_id

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

Table 2.72. Events Generated

Event	Description
gatt_characteristic_value	This event contains the data belonging to a characteristic sent by the GATT Server.
gatt_procedure_completed	Procedure has been successfully completed or failed with error.

2.5.1.15 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 (0x1d) 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 2.73. 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 2.74. 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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_send_characteristic_confirmation_rsp_t *gecko_cmd_gatt_send_characteristic_confirmation(u
int8 connection);

/* Response id */
gecko_rsp_gatt_send_characteristic_confirmation_id

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

2.5.1.16 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 2.75. 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 handle This value is normally received from the gatt_characteristic event.
7	uint8	flags	Characteristic client configuration flags

Table 2.76. 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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_set_characteristic_notification_rsp_t *gecko_cmd_gatt_set_characteristic_notification(uin
t8 connection, uint16 characteristic, uint8 flags);

/* Response id */
gecko_rsp_gatt_set_characteristic_notification_id

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

Table 2.77. Events Generated

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

Event	Description
gatt_characteristic_value	If an indication or notification has been enabled for a characteristic, this event is triggered whenever an indication or notification is sent by the remote GATT server. The triggering conditions on the GATT server side are defined by an upper level, for example by a profile; **so it is possible that no values are ever received, or that it may take time, depending on how the server is configured.**

2.5.1.17 cmd_gatt_set_max_mtu

This command can be used to set the maximum size of ATT Message Transfer Units (MTU). If the given value is too large according to the maximum BGAPI payload size, the system will select the maximal possible value as the maximum ATT_MTU. If maximum ATT_MTU is larger than 23, MTU is exchanged automatically after Bluetooth LE connection has been established.

Table 2.78. 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 size of Message Transfer Units (MTU) allowed • Range: 23 to 250 Default: 247

Table 2.79. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x04	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
6-7	uint16	max_mtu	The maximum ATT_MTU selected by the system if this command succeeded

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_set_max_mtu_rsp_t *gecko_cmd_gatt_set_max_mtu(uint16 max_mtu);

/* Response id */
gecko_rsp_gatt_set_max_mtu_id

/* Response structure */
struct gecko_msg_gatt_set_max_mtu_rsp_t
{
  uint16 result;,
  uint16 max_mtu;
};
```

2.5.1.18 cmd_gatt_write_characteristic_value

This command can be used to write the value of a characteristic in a remote GATT database. If the given value does not fit in one ATT PDU, "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 2.80. 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 handle This value is normally received from the gatt_characteristic event.
7	uint8array	value	Characteristic value

Table 2.81. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_write_characteristic_value_rsp_t *gecko_cmd_gatt_write_characteristic_value(uint8 connect
ion, uint16 characteristic, uint8 value_len, const uint8 *value_data);

/* Response id */
gecko_rsp_gatt_write_characteristic_value_id

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

Table 2.82. Events Generated

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

2.5.1.19 cmd_gatt_write_characteristic_value_without_response

This command can be used to write the value of a characteristic in a remote GATT server. 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 will be reported to the local GATT client. Note that this command cannot be used to write long values. At most ATT_MTU - 3 amount of data can be sent once.

Table 2.83. 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 handle This value is normally received from the gatt_characteristic event.
7	uint8array	value	Characteristic value

Table 2.84. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x04	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
6-7	uint16	sent_len	The length of data sent to the remote GATT server

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_write_characteristic_value_without_response_rsp_t *gecko_cmd_gatt_write_characteristic_va
lue_without_response(uint8 connection, uint16 characteristic, uint8 value_len, const uint8 *value_data);

/* Response id */
gecko_rsp_gatt_write_characteristic_value_without_response_id

/* Response structure */
struct gecko_msg_gatt_write_characteristic_value_without_response_rsp_t
{
  uint16 result;,
  uint16 sent_len;
};
```

2.5.1.20 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 given value does not fit in one ATT PDU, "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 2.85. 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 2.86. 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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_write_descriptor_value_rsp_t *gecko_cmd_gatt_write_descriptor_value(uint8 connection, uin
t16 descriptor, uint8 value_len, const uint8 *value_data);

/* Response id */
gecko_rsp_gatt_write_descriptor_value_id

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

Table 2.87. Events Generated

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

2.5.2 gatt events

2.5.2.1 evt_gatt_characteristic

This event indicates that a GATT characteristic in the remote GATT database was discovered. This event is generated after issuing either the gatt_discover_characteristics or command.

Table 2.88. 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 handle
7	uint8	properties	Characteristic properties
8	uint8array	uuid	Characteristic UUID

C Functions

```
/* Event id */
gecko_evt_gatt_characteristic_id

/* Event structure */
struct gecko_msg_gatt_characteristic_evt_t
{
  uint8 connection;,
  uint16 characteristic;,
  uint8 properties;,
  uint8array uuid;
};
```

2.5.2.2 evt_gatt_characteristic_value

This event indicates that the value of a characteristic in the remote GATT server was received. This event is triggered as a result of several commands: gatt_read_characteristic_value, , , gatt_read_multiple_characteristic_values; and when the remote GATT server sends indications or notifications after enabling notifications with gatt_set_characteristic_notification. The parameter att_opcode reveals which type of GATT transaction triggered this event. In particular, if the att_opcode type is handle_value_indication (0x1d), the application needs to confirm the indication with gatt_send_characteristic_confirmation.

Table 2.89. 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 handle This value is normally received from the 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

C Functions

```
/* Event id */
gecko_evt_gatt_characteristic_value_id

/* Event structure */
struct gecko_msg_gatt_characteristic_value_evt_t
{
  uint8 connection;,
  uint16 characteristic;,
  uint8 att_opcode;,
  uint16 offset;,
  uint8array value;
};
```

2.5.2.3 evt_gatt_descriptor

This event indicates that a GATT characteristic descriptor in the remote GATT database was discovered. This event is generated after issuing the gatt_discover_descriptors command.

Table 2.90. 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	Descriptor UUID

C Functions

```
/* Event id */
gecko_evt_gatt_descriptor_id

/* Event structure */
struct gecko_msg_gatt_descriptor_evt_t
{
  uint8 connection;,
  uint16 descriptor;,
  uint8array uuid;
};
```

2.5.2.4 evt_gatt_descriptor_value

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

Table 2.91. 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

C Functions

```
/* Event id */
gecko_evt_gatt_descriptor_value_id

/* Event structure */
struct gecko_msg_gatt_descriptor_value_evt_t
{
  uint8 connection;,
  uint16 descriptor;,
  uint16 offset;,
  uint8array value;
};
```

2.5.2.5 evt_gatt_mtu_exchanged

This event indicates that an ATT_MTU exchange procedure has been completed. Parameter mtu describes new MTU size. MTU size 23 is used before this event is received.

Table 2.92. 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 ATT_MTU

C Functions

```
/* Event id */
gecko_evt_gatt_mtu_exchanged_id

/* Event structure */
struct gecko_msg_gatt_mtu_exchanged_evt_t
{
  uint8 connection;,
  uint16 mtu;
};
```

2.5.2.6 evt_gatt_procedure_completed

This event indicates that the current GATT procedure has been completed successfully or that it has failed with an error. All GATT commands excluding gatt_write_characteristic_value_without_response and gatt_send_characteristic_confirmation will trigger this event, so the application must wait for this event before issuing another GATT command (excluding the two aforementioned exceptions).

Table 2.93. 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

C Functions

```
/* Event id */
gecko_evt_gatt_procedure_completed_id

/* Event structure */
struct gecko_msg_gatt_procedure_completed_evt_t
{
  uint8 connection;,
  uint16 result;
};
```

2.5.2.7 evt_gatt_service

This event indicates that a GATT service in the remote GATT database was discovered. This event is generated after issuing either the gatt_discover_primary_services or command.

Table 2.94. 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 handle
9	uint8array	uuid	Service UUID

C Functions

```
/* Event id */
gecko_evt_gatt_service_id

/* Event structure */
struct gecko_msg_gatt_service_evt_t
{
  uint8 connection;,
  uint32 service;,
  uint8array uuid;
};
```

2.5.3 gatt enumerations

2.5.3.1 enum_gatt_att_opcode

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

Table 2.95. 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	Execute write request
25	gatt_execute_write_response	Execute write response
27	gatt_handle_value_notification	Notification
29	gatt_handle_value_indication	Indication

2.5.3.2 enum_gatt_client_config_flag

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

Table 2.96. Enumerations

Value	Name	Description
0	gatt_disable	Disable notifications and indications
1	gatt_notification	Notification
2	gatt_indication	Indication

2.5.3.3 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 2.97. Enumerations

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

2.6 Generic Attribute Profile Server (gatt_server)

These commands and events are used by the local GATT server to manage the local GATT database.

2.6.1 gatt_server commands

2.6.1.1 cmd_gatt_server_find_attribute

This command can be used to find attributes of certain type from a local GATT database. Type is usually given as 16-bit or 128-bit UUID.

Table 2.98. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x06	method	Message ID
4-5	uint16	start	Search start index
6	uint8array	type	Variable length byte array. The first byte defines the length of the data that follows, 0 - 255 bytes.

Table 2.99. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x04	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
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
6-7	uint16	attribute	Attribute handle

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_server_find_attribute_rsp_t *gecko_cmd_gatt_server_find_attribute(uint16 start, uint8 type_len, const uint8 *type_data);

/* Response id */
gecko_rsp_gatt_server_find_attribute_id

/* Response structure */
struct gecko_msg_gatt_server_find_attribute_rsp_t
{
  uint16 result;,
  uint16 attribute;
};
```

2.6.1.2 cmd_gatt_server_read_attribute_type

This command can be used to read the type of an attribute from a local GATT database. The type is a UUID, usually 16 or 128 bits long.

Table 2.100. 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 2.101. 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. The first byte defines the length of the data that follows, 0 - 255 bytes.

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_server_read_attribute_type_rsp_t *gecko_cmd_gatt_server_read_attribute_type(uint16 attrib
ute);

/* Response id */
gecko_rsp_gatt_server_read_attribute_type_id

/* Response structure */
struct gecko_msg_gatt_server_read_attribute_type_rsp_t
{
  uint16 result;,
  uint8array type;
};
```

2.6.1.3 cmd_gatt_server_read_attribute_value

This command can be used to read the value of an attribute from a local GATT database. Only (maximum BGAPI payload size - 3) amount of data can be read once. The application can continue reading with increased offset value if it receives (maximum BGAPI payload size - 3) amount of data.

Table 2.102. 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 2.103. 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. The first byte defines the length of the data that follows, 0 - 255 bytes.

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_server_read_attribute_value_rsp_t *gecko_cmd_gatt_server_read_attribute_value(uint16 attr
ibute, uint16 offset);

/* Response id */
gecko_rsp_gatt_server_read_attribute_value_id

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

2.6.1.4 cmd_gatt_server_send_characteristic_notification

This command can be used to send notifications or indications to a remote GATT client. At most ATT_MTU - 3 amount of data can be sent once. 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 indication cannot be sent before a confirmation from the GATT client is first received. The confirmation is indicated by gatt_server_characteristic_status event.

Table 2.104. 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. Values: • **0xff:** 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 2.105. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x04	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
6-7	uint16	sent_len	The length of data sent out if only one connected device is the receiver; otherwise unused value

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_server_send_characteristic_notification_rsp_t *gecko_cmd_gatt_server_send_characteristic_
notification(uint8 connection, uint16 characteristic, uint8 value_len, const uint8 *value_data);

/* Response id */
gecko_rsp_gatt_server_send_characteristic_notification_id

/* Response structure */
struct gecko_msg_gatt_server_send_characteristic_notification_rsp_t
{
  uint16 result;,
  uint16 sent_len;
};
```

2.6.1.5 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 second, 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 attr_errorcode values will cause the local GATT server to send an attribute protocol error response instead of the actual data. At most ATT_MTU - 1 amount of data can be sent once. Client will continue reading by sending new read request with increased offset value if it receives ATT_MTU - 1 amount of data.

Table 2.106. 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 handle This value is normally received from the gatt_characteristic event.
7	uint8	att_errorcode	Attribute protocol error code • **0:** No error • **Non-zero:** See Bluetooth specification, Host volume, Attribute Protocol, Error Codes table.
8	uint8array	value	Characteristic value to send to the GATT client. Ignored if att_errorcode is not 0.

Table 2.107. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x04	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
6-7	uint16	sent_len	The length of data sent to the remote GATT client

BGLIB C API

```
/* Function */
struct  gecko_msg_gatt_server_send_user_read_response_rsp_t  *gecko_cmd_gatt_server_send_user_read_response(uint
8 connection, uint16 characteristic, uint8 att_errorcode, uint8 value_len, const uint8 *value_data);

/* Response id */
gecko_rsp_gatt_server_send_user_read_response_id

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

```
  uint16 sent_len;
};
```

2.6.1.6 cmd_gatt_server_send_user_write_response

This command must be used to send a response to a gatt_server_user_write_request event when parameter att_opcode in the event is Write Request (see att_opcode). 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 2.108. 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 handle This value is normally received from the gatt_characteristic event.
7	uint8	att_errorcode	Attribute protocol error code • **0:** No error • **Non-zero:** See Bluetooth specification, Host volume, Attribute Protocol, Error Codes table.

Table 2.109. 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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_server_send_user_write_response_rsp_t *gecko_cmd_gatt_server_send_user_write_response(uin
t8 connection, uint16 characteristic, uint8 att_errorcode);

/* Response id */
gecko_rsp_gatt_server_send_user_write_response_id

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

2.6.1.7 cmd_gatt_server_set_capabilities

This command can be used to set which capabilities should be enabled in the local GATT database. A service is visible to remote GATT clients if at least one of its capabilities has been enabled. The same applies to a characteristic and its attributes. Capability identifiers and their corresponding bit flag values can be found in the auto-generated database header file. See UG118 for how to declare capabilities in GATT database.

Changing the capabilities of a database effectively causes a database change (attributes being added or removed) from a remote GATT client point of view. If the database has a Generic Attribute service and Service Changed characteristic, the stack will monitor local database change status and manage service changed indications for a GATT client that has enabled the indication configuration of the Service Changed characteristic.

Table 2.110. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x08	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x08	method	Message ID
4-7	uint32	caps	Bit flags of capabilities to enable.
8-11	uint32	reserved	Value 0 should be used on this reserved field. None-zero values are reserved for future, do not use now.

Table 2.111. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_server_set_capabilities_rsp_t *gecko_cmd_gatt_server_set_capabilities(uint32 caps, uint3
2 reserved);

/* Response id */
gecko_rsp_gatt_server_set_capabilities_id

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

2.6.1.8 cmd_gatt_server_set_database

Deprecated. Use GATT capability feature for dynamic configuration of services and characteristics in the local GATT database. See gatt_server_set_capabilities command for the details.

This command can be used to set the local GATT database. The database should not be changed while this device is connected as peripheral since it may cause GATT attributes and data synchronization problems. If the database is changed during advertising mode, advertisement packets will not be updated until the advertising is restarted.

Table 2.112. 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	0x07	method	Message ID
4-7	uint32	ptr	The pointer to the GATT database

Table 2.113. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_server_set_database_rsp_t *gecko_cmd_gatt_server_set_database(uint32 ptr);

/* Response id */
gecko_rsp_gatt_server_set_database_id

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

2.6.1.9 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 gatt_server_send_characteristic_notification command.

Table 2.114. 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 2.115. 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

BGLIB C API

```
/* Function */
struct gecko_msg_gatt_server_write_attribute_value_rsp_t *gecko_cmd_gatt_server_write_attribute_value(uint16 at
tribute, uint16 offset, uint8 value_len, const uint8 *value_data);

/* Response id */
gecko_rsp_gatt_server_write_attribute_value_id

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

2.6.2 gatt_server events

2.6.2.1 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 GATT procedure was used to change the value.

Table 2.116. 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	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
8-9	uint16	offset	Value offset
10	uint8array	value	Value

C Functions

```
/* Event id */
gecko_evt_gatt_server_attribute_value_id

/* Event structure */
struct gecko_msg_gatt_server_attribute_value_evt_t
{
  uint8 connection;,
  uint16 attribute;,
  uint8 att_opcode;,
  uint16 offset;,
  uint8array value;
};
```

2.6.2.2 evt_gatt_server_characteristic_status

This event indicates 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 gatt_server_send_characteristic_notification command, otherwise further GATT transactions over this connection are disabled by the stack.

Table 2.117. 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 handle This value is normally received from the 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 field carries the new value of the Client Characteristic Configuration. If the status_flags is 0x2 (confirmation received), the value of this field can be ignored.

C Functions

```
/* Event id */
gecko_evt_gatt_server_characteristic_status_id

/* Event structure */
struct gecko_msg_gatt_server_characteristic_status_evt_t
{
  uint8 connection;,
  uint16 characteristic;,
  uint8 status_flags;,
  uint16 client_config_flags;
};
```

2.6.2.3 evt_gatt_server_execute_write_completed

Execute write completed event indicates that the execute write command from a remote GATT client has completed with the given result.

Table 2.118. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x03	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	result	Execute write result

C Functions

```
/* Event id */
gecko_evt_gatt_server_execute_write_completed_id

/* Event structure */
struct gecko_msg_gatt_server_execute_write_completed_evt_t
{
  uint8 connection;,
  uint16 result;
};
```

2.6.2.4 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 GATT procedure was used to read the value. The application needs to respond to this request by using the gatt_server_send_user_read_response command within 30 seconds, otherwise this GATT connection is dropped by remote side.

Table 2.119. 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 handle This value is normally received from the gatt_characteristic event.
7	uint8	att_opcode	Attribute opcode which informs the procedure from which attribute the value was received
8-9	uint16	offset	Value offset

C Functions

```
/* Event id */
gecko_evt_gatt_server_user_read_request_id

/* Event structure */
struct gecko_msg_gatt_server_user_read_request_evt_t
{
  uint8 connection;,
  uint16 characteristic;,
  uint8 att_opcode;,
  uint16 offset;
};
```

2.6.2.5 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. If the att_opcode is Write Request (see att_opcode), the application needs to respond to this request by using the gatt_server_send_user_write_response command within 30 seconds, otherwise this GATT connection is dropped by the remote side. If the value of att_opcode is Execute Write Request, it indicates that this is a queued prepare write request received earlier and now the GATT server is processing the execute write. The event gatt_server_execute_write_completed will be emitted after all queued requests have been processed.

Table 2.120. 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 handle This value is normally received from the gatt_characteristic event.
7	uint8	att_opcode	Attribute opcode which informs the procedure from which attribute the value was received
8-9	uint16	offset	Value offset
10	uint8array	value	Value

C Functions

```
/* Event id */
gecko_evt_gatt_server_user_write_request_id

/* Event structure */
struct gecko_msg_gatt_server_user_write_request_evt_t
{
  uint8 connection;,
  uint16 characteristic;,
  uint8 att_opcode;,
  uint16 offset;,
  uint8array value;
};
```

2.6.3 gatt_server enumerations

2.6.3.1 enum_gatt_server_characteristic_status_flag

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

Table 2.121. Enumerations

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

2.7 Hardware (hardware)

The commands and events in this class can be used to access and configure the system hardware and peripherals.

2.7.1 hardware commands

2.7.1.1 cmd_hardware_enable_dcdc

Deprecated. This command can be used to enable or disable DC/DC.

Table 2.122. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x0c	class	Message class: Hardware
3	0x0d	method	Message ID
4	uint8	enable	Set DC/DC as enabled or disabled. • **0:** disabled • **1:** enabled

Table 2.123. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x0c	class	Message class: Hardware
3	0x0d	method	Message ID
4-5	uint16	result	Result code • **0:** success • **Non-zero:** an error occurred For other values refer to the Error codes

BGLIB C API

```
/* Function */
struct gecko_msg_hardware_enable_dcdc_rsp_t *gecko_cmd_hardware_enable_dcdc(uint8 enable);

/* Response id */
gecko_rsp_hardware_enable_dcdc_id

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

2.7.1.2 cmd_hardware_get_time

Deprecated. Get elapsed time since last reset of RTCC

Table 2.124. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x00	lolen	Minimum payload length
2	0x0c	class	Message class: Hardware
3	0x0b	method	Message ID

Table 2.125. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x06	lolen	Minimum payload length
2	0x0c	class	Message class: Hardware
3	0x0b	method	Message ID
4-7	uint32	seconds	seconds since last reset
8-9	uint16	ticks	Subsecond ticks of hardware clock, range 0-32767

BGLIB C API

```
/* Function */
struct gecko_msg_hardware_get_time_rsp_t *gecko_cmd_hardware_get_time();

/* Response id */
gecko_rsp_hardware_get_time_id

/* Response structure */
struct gecko_msg_hardware_get_time_rsp_t
{
  uint32 seconds;,
  uint16 ticks;
};
```

2.7.1.3 cmd_hardware_set_lazy_soft_timer

This command can be used to start a software timer with some slack. Slack parameter allows stack to optimize wake ups and save power. Timer event is triggered between time and time + slack. See also description of hardware_set_soft_timer command.

Table 2.126. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x0a	lolen	Minimum payload length
2	0x0c	class	Message class: Hardware
3	0x0c	method	Message ID
4-7	uint32	time	Interval between how often to send events, in hardware clock ticks (1 second is equal to 32768 ticks). The smallest interval value supported is 328 which is around 10 milliseconds, any parameters between 0 and 328 will be rounded up to 328. The maximum value is 2147483647, which corresponds to about 18.2 hours. If time is 0, removes the scheduled timer with the same handle.
8-11	uint32	slack	Slack time in hardware clock ticks
12	uint8	handle	Timer handle to use, is returned in timeout event
13	uint8	single_shot	Timer mode. Values: • **0:** false (timer is repeating) • **1:** true (timer runs only once)

Table 2.127. Response

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

BGLIB C API

```
/* Function */
struct gecko_msg_hardware_set_lazy_soft_timer_rsp_t *gecko_cmd_hardware_set_lazy_soft_timer(uint32 time, uint3
2 slack, uint8 handle, uint8 single_shot);

/* Response id */
gecko_rsp_hardware_set_lazy_soft_timer_id

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

Table 2.128. Events Generated

Event	Description
hardware_soft_timer	Sent after specified interval

2.7.1.4 cmd_hardware_set_soft_timer

This command can be used to start a software timer. Multiple concurrent timers can be running simultaneously. There are 256 unique timer IDs available. The maximum number of concurrent timers is configurable at device initialization. Up to 16 concurrent timers can be configured. The default configuration is 4. As the RAM for storing timer data is pre-allocated at initialization, an application should not configure the amount more than it needs for minimizing RAM usage.

Table 2.129. 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	Interval between how often to send events, in hardware clock ticks (1 second is equal to 32768 ticks). The smallest interval value supported is 328 which is around 10 milliseconds, any parameters between 0 and 328 will be rounded up to 328. The maximum value is 2147483647, which corresponds to about 18.2 hours. If time is 0, removes the scheduled timer with the same handle.
8	uint8	handle	Timer handle to use, is returned in timeout event
9	uint8	single_shot	Timer mode. Values: • **0:** false (timer is repeating) • **1:** true (timer runs only once)

Table 2.130. 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	Result code • **0:** success • **Non-zero:** an error occurred For other values refer to the Error codes

BGLIB C API

```
/* Function */
struct gecko_msg_hardware_set_soft_timer_rsp_t *gecko_cmd_hardware_set_soft_timer(uint32 time, uint8 handle, ui
nt8 single_shot);

/* Response id */
gecko_rsp_hardware_set_soft_timer_id

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

Table 2.131. Events Generated

Event	Description
hardware_soft_timer	Sent after specified interval

2.7.2 hardware events

2.7.2.1 evt_hardware_soft_timer

This event indicates that the soft timer has lapsed.

Table 2.132. 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

C Functions

```
/* Event id */
gecko_evt_hardware_soft_timer_id

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

2.8 Connection management for low energy (le_connection)

The commands and events in this class are related to managing connection establishment, parameter setting, and disconnection procedures.

2.8.1 le_connection commands

2.8.1.1 cmd_le_connection_close

This command can be used to close a Bluetooth LE connection or cancel an ongoing connection establishment process. The parameter is a connection handle which is reported in le_connection_opened event or le_gap_open response.

Table 2.133. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x08	class	Message class: Connection management for low energy
3	0x04	method	Message ID
4	uint8	connection	Handle of the connection to be closed

Table 2.134. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_le_connection_close_rsp_t *gecko_cmd_le_connection_close(uint8 connection);

/* Response id */
gecko_rsp_le_connection_close_id

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

Table 2.135. Events Generated

Event	Description
le_connection_closed	This event indicates that a connection was closed.

2.8.1.2 cmd_le_connection_disable_slave_latency

This command temporarily enables or disables slave latency. Used only when Bluetooth device is in slave role.

Table 2.136. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	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	uint8	disable	0 enable, 1 disable slave latency

Table 2.137. 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	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

BGLIB C API

```
/* Function */
struct  gecko_msg_le_connection_disable_slave_latency_rsp_t  *gecko_cmd_le_connection_disable_slave_latency(uint
8 connection, uint8 disable);

/* Response id */
gecko_rsp_le_connection_disable_slave_latency_id

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

2.8.1.3 cmd_le_connection_get_rssi

This command can be used to get the latest RSSI value of a Bluetooth LE connection.

Table 2.138. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x08	class	Message class: Connection management for low energy
3	0x01	method	Message ID
4	uint8	connection	Connection handle

Table 2.139. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_le_connection_get_rssi_rsp_t *gecko_cmd_le_connection_get_rssi(uint8 connection);

/* Response id */
gecko_rsp_le_connection_get_rssi_id

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

2.8.1.4 cmd_le_connection_set_parameters

This command can be used to request a change in the connection parameters of a Bluetooth LE connection.

Table 2.140. 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 value for the connection event interval. This must be set be less than or equal to max_interval. • Time = Value x 1.25 ms • Range: 0x0006 to 0x0c80 • Time Range: 7.5 ms to 4 s
7-8	uint16	max_interval	Maximum value for the connection event interval. This must be set greater than or equal to min_interval. • Time = Value x 1.25 ms • Range: 0x0006 to 0x0c80 • Time Range: 7.5 ms to 4 s
9-10	uint16	latency	Slave latency. This parameter defines how many connection intervals the slave can skip if it has no data to send • Range: 0x0000 to 0x01f4 Use 0x0000 for default value
11-12	uint16	timeout	Supervision timeout. 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 0x0c80 • Time = Value x 10 ms • Time Range: 100 ms to 32 s • The value in milliseconds must be larger than (1 + latency) * max_interval * 2, where max_interval is given in milliseconds It is recommended that the supervision timeout is set at a value which allows communication attempts over at least a few connection intervals.

Table 2.141. 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

BGLIB C API

```
/* Function */
struct gecko_msg_le_connection_set_parameters_rsp_t *gecko_cmd_le_connection_set_parameters(uint8 connection, u
int16 min_interval, uint16 max_interval, uint16 latency, uint16 timeout);

/* Response id */
gecko_rsp_le_connection_set_parameters_id

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

Table 2.142. Events Generated

Event	Description
le_connection_parameters	This event is triggered after new connection parameters has been applied on the connection.

2.8.1.5 cmd_le_connection_set_phy

This command can be used to set preferred PHYs for connection. Preferred PHYs are connection specific. Event le_connection_phy_status is received when PHY update procedure has been completed. Other than preferred PHY can also be set if remote device does not accept any of the preferred PHYs.

NOTE: Minimum connection interval for 125 kbit and 500 kbit phys is currently 32 (40 ms). Coded PHY is not used if connection interval is under that limit.

NOTE: 2 Mbit and Coded PHYs are not supported by all devices.

Table 2.143. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x08	class	Message class: Connection management for low energy
3	0x03	method	Message ID
4	uint8	connection	
5	uint8	phy	Preferred PHYs for connection. This parameter is bitfield and multiple PHYs can be preferred by setting multiple bits. • **0x01:** 1 Mbit PHY • **0x02:** 2 Mbit PHY • **0x04:** 125 kbit Coded PHY (S=8) • **0x08:** 500 kbit Coded PHY (S=2)

Table 2.144. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_le_connection_set_phy_rsp_t *gecko_cmd_le_connection_set_phy(uint8 connection, uint8 phy);

/* Response id */
gecko_rsp_le_connection_set_phy_id

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

Table 2.145. Events Generated

Event	Description
le_connection_phy_status	This event indicates that PHY update procedure has been completed.

2.8.2 le_connection events

2.8.2.1 evt_le_connection_bt5_opened

Event is only sent if bt5 command was used to enter advertising mode. This event indicates that a new connection was opened, whether the devices are already bonded, and what is the role of the Bluetooth device (Slave or Master). An open connection can be closed with the le_connection_close command by giving the connection handle ID obtained from this event.

Table 2.146. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x0b	lolen	Minimum payload length
2	0x08	class	Message class: Connection management for low energy
3	0x05	method	Message ID
4-9	bd_addr	address	Remote device address
10	uint8	address_type	Remote device address type
11	uint8	master	Device role in connection. Values: • **0:** Slave • **1:** Master
12	uint8	connection	Handle for new connection
13	uint8	bonding	Bonding handle. Values: • **0xff:** No bonding • **Other:** Bonding handle
14	uint8	advertiser	Advertisement set this connection was opened to

C Functions

```
/* Event id */
gecko_evt_le_connection_bt5_opened_id

/* Event structure */
struct gecko_msg_le_connection_bt5_opened_evt_t
{
  bd_addr address;,
  uint8 address_type;,
  uint8 master;,
  uint8 connection;,
  uint8 bonding;,
  uint8 advertiser;
};
```

2.8.2.2 evt_le_connection_closed

This event indicates that a connection was closed.

Table 2.147. 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

C Functions

```
/* Event id */
gecko_evt_le_connection_closed_id

/* Event structure */
struct gecko_msg_le_connection_closed_evt_t
{
  uint16 reason;,
  uint8 connection;
};
```

2.8.2.3 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 device (Slave or Master). An open connection can be closed with the le_connection_close command by giving the connection handle ID obtained from this event.

Table 2.148. 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	Device role in connection. Values: • **0:** Slave • **1:** Master
12	uint8	connection	Handle for new connection
13	uint8	bonding	Bonding handle. Values: • **0xff:** No bonding • **Other:** Bonding handle

C Functions

```
/* Event id */
gecko_evt_le_connection_opened_id

/* Event structure */
struct gecko_msg_le_connection_opened_evt_t
{
  bd_addr address;,
  uint8 address_type;,
  uint8 master;,
  uint8 connection;,
  uint8 bonding;
};
```

2.8.2.4 evt_le_connection_parameters

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

Table 2.149. 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	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
12-13	uint16	txsize	Maximum Data Channel PDU Payload size the controller can send in an air packet

C Functions

```
/* Event id */
gecko_evt_le_connection_parameters_id

/* Event structure */
struct gecko_msg_le_connection_parameters_evt_t
{
  uint8 connection;,
  uint16 interval;,
  uint16 latency;,
  uint16 timeout;,
  uint8 security_mode;,
  uint16 txsize;
};
```

2.8.2.5 evt_le_connection_phy_status

This event indicates that PHY update procedure has been completed.

Table 2.150. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x02	lolen	Minimum payload length
2	0x08	class	Message class: Connection management for low energy
3	0x04	method	Message ID
4	uint8	connection	
5	uint8	phy	Current active PHY. See values from le_connection_set_phy command.

C Functions

```
/* Event id */
gecko_evt_le_connection_phy_status_id

/* Event structure */
struct gecko_msg_le_connection_phy_status_evt_t
{
  uint8 connection;,
  uint8 phy;
};
```

2.8.2.6 evt_le_connection_rssi

This event is triggered when an le_connection_get_rssi command has completed.

Table 2.151. 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	0x03	method	Message ID
4	uint8	connection	Connection handle
5	uint8	status	Command complete status
6	int8	rssi	RSSI of the Bluetooth LE connection Range: -127 to +20. Units: dBm.

C Functions

```
/* Event id */
gecko_evt_le_connection_rssi_id

/* Event structure */
struct gecko_msg_le_connection_rssi_evt_t
{
  uint8 connection;,
  uint8 status;,
  int8 rssi;
};
```

2.8.3 le_connection enumerations

2.8.3.1 enum_le_connection_security

These values indicate the Bluetooth LE Security Mode.

Table 2.152. 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
3	le_connection_mode1_level4	Authenticated LE Secure Connections pairing with encryption using a 128-bit strength encryption key

2.9 Generic Access Profile, Low Energy (le_gap)

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

2.9.1 le_gap commands

2.9.1.1 cmd_le_gap_bt5_set_adv_data

Same as le_gap_set_adv_data except this command supports Bluetooth 5 advertisement sets

Table 2.153. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x03	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
3	0x0c	method	Message ID
4	uint8	handle	Advertisement set handle index, number of sets available is defined in stack configuration
5	uint8	scan_rsp	This value selects if the data is intended for advertisement packets, scan response packets or advertisement packet in OTA. Values: • **0:** Advertisement packets • **1:** Scan response packets • **2:** OTA Advertisement packets • **4:** OTA scan response packets
6	uint8array	adv_data	Data to be set. Maximum length is 31 bytes

Table 2.154. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_le_gap_bt5_set_adv_data_rsp_t *gecko_cmd_le_gap_bt5_set_adv_data(uint8 handle, uint8 scan_rsp
, uint8 adv_data_len, const uint8 *adv_data_data);

/* Response id */
gecko_rsp_le_gap_bt5_set_adv_data_id

/* Response structure */
struct gecko_msg_le_gap_bt5_set_adv_data_rsp_t
{
  uint16 result;
};
```

2.9.1.2 cmd_le_gap_bt5_set_adv_parameters

This command can be used to set Bluetooth 5 LE advertisement parameters of an advertisement set that is specified by the given handle. These parameters are applied to the specified advertisement set only. Other advertisement sets are not affected.

This command is the same as le_gap_set_adv_parameters except it supports Bluetooth 5 extended features.

Table 2.155. 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	0x0b	method	Message ID
4	uint8	handle	Advertisement set handle index, number of sets available is defined in stack configuration
5-6	uint16	interval_min	Minimum advertisement interval. Value in units of 0.625 ms • Range: 0x0020 to 0x4000 (connectable advertising) • Time range: 20 ms to 10.24 s (connectable advertising) • Range: 0x00a0 to 0x4000 (non-connectable advertising) • Time range: 100 ms to 10.24 s (non-connectable advertising) Default value: 100 ms
7-8	uint16	interval_max	Maximum advertisement interval. Value in units of 0.625 ms • Range: 0x0020 to 0x4000 • Time range: 20 ms to 10.24 s • Note: interval_max must be at least equal to or bigger than interval_min Default value: 200 ms
9	uint8	channel_map	Advertisement channel map which determines which of the three channels will be used for advertising. This value is given as a bitmask. Values: • **1:** Advertise on CH37 • **2:** Advertise on CH38 • **3:** Advertise on CH37 and CH38 • **4:** Advertise on CH39 • **5:** Advertise on CH37 and CH39 • **6:** Advertise on CH38 and CH39 • **7:** Advertise on all channels Recommended value: 7 Default value: 7
10	uint8	report_scan	If nonzero report scan requests as events Default value: 0

Table 2.156. 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

Byte	Type	Name	Description
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

BGLIB C API

```
/* Function */
struct gecko_msg_le_gap_bt5_set_adv_parameters_rsp_t *gecko_cmd_le_gap_bt5_set_adv_parameters(uint8 handle, uin
t16 interval_min, uint16 interval_max, uint8 channel_map, uint8 report_scan);

/* Response id */
gecko_rsp_le_gap_bt5_set_adv_parameters_id

/* Response structure */
struct gecko_msg_le_gap_bt5_set_adv_parameters_rsp_t
{
  uint16 result;
};
```

2.9.1.3 cmd_le_gap_bt5_set_mode

This command is used to set one advertisement sets mode, this works the same way as le_gap_set_mode except has extra parameters to support Bluetooth 5 advertisement set and features

Table 2.157. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x06	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
3	0x0a	method	Message ID
4	uint8	handle	Advertisement set handle index, number of sets available is defined in stack configuration
5	uint8	discover	Discoverable mode
6	uint8	connect	Connectable mode
7-8	uint16	maxevents	Maximum number of events to send before stopping advertiser, if 0 do not stop advertiser
9	uint8	address_type	Address type to use for packets

Table 2.158. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_le_gap_bt5_set_mode_rsp_t *gecko_cmd_le_gap_bt5_set_mode(uint8 handle, uint8 discover, uint8 c
onnect, uint16 maxevents, uint8 address_type);

/* Response id */
gecko_rsp_le_gap_bt5_set_mode_id

/* Response structure */
struct gecko_msg_le_gap_bt5_set_mode_rsp_t
{
  uint16 result;
};
```

2.9.1.4 cmd_le_gap_discover

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

Table 2.159. 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	Bluetooth LE Discovery Mode. For values see link

Table 2.160. 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

BGLIB C API

```
/* Function */
struct gecko_msg_le_gap_discover_rsp_t *gecko_cmd_le_gap_discover(uint8 mode);

/* Response id */
gecko_rsp_le_gap_discover_id

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

Table 2.161. Events Generated

Event	Description
le_gap_scan_response	Every time an advertisement packet is received, this event is triggered. The packets are not filtered in any way, so multiple events will be received for every advertising device in range.

2.9.1.5 cmd_le_gap_end_procedure

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

Table 2.162. 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 2.163. 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

BGLIB C API

```
/* Function */
struct gecko_msg_le_gap_end_procedure_rsp_t *gecko_cmd_le_gap_end_procedure();

/* Response id */
gecko_rsp_le_gap_end_procedure_id

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

2.9.1.6 cmd_le_gap_open

This command can be used to connect an advertising device. Scanning parameters can be configured with the le_gap_set_scan_parameters command before issuing this command. The Bluetooth stack will enter a state where it continuously scans for the connectable advertisement packets from the remote device which matches the Bluetooth address given as a parameter. Upon receiving the advertisement packet, the module will send a connection request packet to the target device to initiate a Bluetooth connection. To cancel an ongoing connection process use the le_connection_close command with the handle received in the le_gap_open response.

A connection is opened in no-security mode. If the GATT client needs to read or write the attributes on GATT server requiring encryption or authentication, it must first encrypt the connection using an appropriate authentication method.

This command fails with "Connection Limit Exceeded" error if the number of connections attempted to be opened exceeds the max_connections value configured.

Later calls of this command have to wait for the ongoing command to complete. A received event le_connection_opened indicates connection opened successfully and a received event le_connection_closed indicates connection failures have occurred.

Table 2.164. 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 to

Table 2.165. 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).

BGLIB C API

```
/* Function */
struct gecko_msg_le_gap_open_rsp_t *gecko_cmd_le_gap_open(bd_addr address, uint8 address_type);

/* Response id */
gecko_rsp_le_gap_open_id

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

```
  uint8 connection;
};
```

Table 2.166. 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 device (Slave or Master).
le_connection_parameters	This event indicates the connection parameters and security mode of the connection.

2.9.1.7 cmd_le_gap_set_adv_data

This command can be used together with le_gap_set_mode to advertise user defined data. First use this command to set the data in advertisement packets and/or in the scan response packets, and then use command le_gap_set_mode to configure this device to be discoverable in user_data mode.

Note that the user defined data may be overwritten by the system when this device is later configured to other discoverable mode than user_data. It is recommended to set both the advertisement data and scan response data at the same time.

If advertisement mode is currently active, then new advertisement data will be used immediately.

Table 2.167. 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, scan response packets or advertisement packet in OTA. Values: • **0:** Advertisement packets • **1:** Scan response packets • **2:** OTA Advertisement packets • **4:** OTA scan response packets
5	uint8array	adv_data	Data to be set. Maximum length is 31 bytes

Table 2.168. 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

BGLIB C API

```
/* Function */
struct gecko_msg_le_gap_set_adv_data_rsp_t *gecko_cmd_le_gap_set_adv_data(uint8 scan_rsp, uint8 adv_data_len, c
onst uint8 *adv_data_data);

/* Response id */
gecko_rsp_le_gap_set_adv_data_id

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

2.9.1.8 cmd_le_gap_set_adv_parameters

This command can be used together with le_gap_set_mode to set Bluetooth LE advertisement parameters. First, use this command to set advertisement parameters, and then use command le_gap_set_mode to start or restart advertisement. If parameters can't be used with currently active mode an error will be returned.

Table 2.169. 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 advertisement interval. Value in units of 0.625 ms • Range: 0x0020 to 0x4000 (connectable advertising) • Time range: 20 ms to 10.24 s (connectable advertising) • Range: 0x00a0 to 0x4000 (non-connectable advertising) • Time range: 100 ms to 10.24 s (non-connectable advertising) Default value: 100 ms
6-7	uint16	interval_max	Maximum advertisement interval. Value in units of 0.625 ms • Range: 0x0020 to 0x4000 • Time range: 20 ms to 10.24 s • Note: interval_max must be at least equal to or bigger than interval_min Default value: 200 ms
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 bitmask. Values: • **1:** Advertise on CH37 • **2:** Advertise on CH38 • **3:** Advertise on CH37 and CH38 • **4:** Advertise on CH39 • **5:** Advertise on CH37 and CH39 • **6:** Advertise on CH38 and CH39 • **7:** Advertise on all channels Recommended value: 7 Default value: 7

Table 2.170. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_le_gap_set_adv_parameters_rsp_t *gecko_cmd_le_gap_set_adv_parameters(uint16 interval_min, uint
16 interval_max, uint8 channel_map);

/* Response id */
gecko_rsp_le_gap_set_adv_parameters_id

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

2.9.1.9 cmd_le_gap_set_adv_timeout

This command can be used together with le_gap_set_mode to set the number of advertisements before stopping. Passing value 0 to this command will remove the timeout setting.

Table 2.171. 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	0x08	method	Message ID
4	uint8	intervals	Number of advertisement before stopping advertiser, if 0 do not stop advertiser

Table 2.172. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_le_gap_set_adv_timeout_rsp_t *gecko_cmd_le_gap_set_adv_timeout(uint8 intervals);

/* Response id */
gecko_rsp_le_gap_set_adv_timeout_id

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

2.9.1.10 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 2.173. 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 value for the connection event interval. This must be set be less than or equal to max_interval. • Time = Value x 1.25 ms • Range: 0x0006 to 0x0c80 • Time Range: 7.5 ms to 4 s Default value: 125 ms
6-7	uint16	max_interval	Maximum value for the connection event interval. This must be set greater than or equal to min_interval. • Time = Value x 1.25 ms • Range: 0x0006 to 0x0c80 • Time Range: 7.5 ms to 4 s Default value: 250 ms
8-9	uint16	latency	Slave latency. This parameter defines how many connection intervals the slave can skip if it has no data to send • Range: 0x0000 to 0x01f4 Default value: 0
10-11	uint16	timeout	Supervision timeout. 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 0x0c80 • Time = Value x 10 ms • Time Range: 100 ms to 32 s • The value in milliseconds must be larger than (1 + latency) * max_interval * 2, where max_interval is given in milliseconds It is recommended that the supervision timeout is set at a value which allows communication attempts over at least a few connection intervals. Default value: 1000 ms

Table 2.174. 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

Byte	Type	Name	Description
4-5	uint16	result	Result code • **0:** success • **Non-zero:** an error occurred For other values refer to the Error codes

BGLIB C API

```
/* Function */
struct gecko_msg_le_gap_set_conn_parameters_rsp_t *gecko_cmd_le_gap_set_conn_parameters(uint16 min_interval, ui
nt16 max_interval, uint16 latency, uint16 timeout);

/* Response id */
gecko_rsp_le_gap_set_conn_parameters_id

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

2.9.1.11 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 and/or disallow incoming connections), issue this command with the Not Discoverable / Not Connectable parameter values. Command will take effect immediately. If currently set parameters can't be used with new mode then an error will be returned.

If advertisement will be enabled in user_data mode, le_gap_set_adv_data should be used to set advertisement and scan response data before issuing this command. When the advertisement is enabled in other modes than user_data, the advertisement and scan response data is generated by the stack using the following procedure:

1. Add a Flags field to advertisement data.
2. Add a TX power level field to advertisement data if TX power service exists in the local GATT database.
3. Add a Slave Connection Interval Range field to advertisement data if the GAP peripheral preferred connection parameters characteristic exists in the local GATT database.
4. Add a list of 16-bit Service UUIDs to advertisement data if there are one or more 16-bit service UUIDs to advertise. The list is complete if all advertised 16-bit UUIDs are in advertisement data; otherwise the list is incomplete.
5. Add a list of 128-bit service UUIDs to advertisement data if there are one or more 128-bit service UUIDs to advertise and there is still free space for this field. The list is complete if all advertised 128-bit UUIDs are in advertisement data; otherwise the list is incomplete. Note that an advertisement data packet can contain at most one 128-bit service UUID.
6. Try to add the full local name to advertisement data if device is not in privacy mode. In case the full local name does not fit into the remaining free space, the advertised name is a shortened version by cutting off the end if the free space has at least 6 bytes; Otherwise, the local name is added to scan response data.

This command fails with "Connection Limit Exceeded" error if the number of connections has reached the max_connections value configured.

Table 2.175. 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 2.176. 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

BGLIB C API

```
/* Function */
struct gecko_msg_le_gap_set_mode_rsp_t *gecko_cmd_le_gap_set_mode(uint8 discover, uint8 connect);
```

```
/* Response id */
gecko_rsp_le_gap_set_mode_id

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

2.9.1.12 cmd_le_gap_set_privacy_mode

This command can be used to enable or disable privacy feature on all GAP roles. The new privacy mode will take effect for advertising on the next advertising enabling, for scanning on the next scan enabling, and for initiating on the next open connection command. When privacy is enabled and the device is advertising or scanning, the stack will maintain a periodic timer with the specified time interval as timeout value. At each timeout the stack will generate a new private resolvable address and use it in advertising data packets and scanning requests.

By default, privacy feature is disabled.

Table 2.177. 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	0x0d	method	Message ID
4	uint8	privacy	Values: • **0:** Disable privacy • **1:** Enable privacy
5	uint8	interval	The minimum time interval between private address change. This parameter is ignored if this command is issued for disabling privacy mode. Values: • **0:** Use default interval, 15 minutes • **others:** The time interval in minutes

Table 2.178. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_le_gap_set_privacy_mode_rsp_t *gecko_cmd_le_gap_set_privacy_mode(uint8 privacy, uint8 interval
);

/* Response id */
gecko_rsp_le_gap_set_privacy_mode_id

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

2.9.1.13 cmd_le_gap_set_scan_parameters

This command can be used to set Bluetooth LE scan parameters. If the device is currently scanning for advertising devices new parameters will take effect when the scanning is restarted.

Table 2.179. 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 interval. This is defined as the time interval from when the device started its last LE scan until it begins the subsequent LE scan, that is how often to scan • Time = Value x 0.625 ms • Range: 0x0004 to 0x4000 • Time Range: 2.5 ms to 10.24 s Default value: 10 ms There is a delay when switching channels at the end of each scanning interval of about 0.5ms. This is included in the scanning interval time itself.
6-7	uint16	scan_window	Scan window. The duration of the LE scan. scan_window shall be less than or equal to scan_interval • Time = Value x 0.625 ms • Range: 0x0004 to 0x4000 • Time Range: 2.5 ms to 10.24 s Default value: 10 ms Note that packet reception is aborted if it has been started before scan window ends.
8	uint8	active	Scan type indicated by a flag. Values: • **0:** Passive scanning • **1:** Active scanning • In passive scanning mode the device only listens to advertising packets and will not transmit any packet • In active scanning mode the device will send out a scan request packet upon receiving advertising packet from a remote device and then it will listen to the scan response packet from remote device Default value: 0

Table 2.180. 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

BGLIB C API

```
/* Function */
struct gecko_msg_le_gap_set_scan_parameters_rsp_t *gecko_cmd_le_gap_set_scan_parameters(uint16 scan_interval, u
int16 scan_window, uint8 active);

/* Response id */
gecko_rsp_le_gap_set_scan_parameters_id

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

2.9.2 le_gap events

2.9.2.1 evt_le_gap_adv_timeout

This event indicates that number of advertisement set by le_gap_set_adv_timeout has been done and advertisement is stopped.

Table 2.181. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x00	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
3	0x01	method	Message ID

C Functions

```
/* Event id */
gecko_evt_le_gap_adv_timeout_id

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

2.9.2.2 evt_le_gap_bt5_adv_timeout

This event indicates that the advertiser has completed the configured number of advertising events in the advertisement set and advertisement is stopped. The maximum number of advertising events can be configured by the maxevents parameter in command le_gap_bt5_set_mode.

Table 2.182. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
3	0x03	method	Message ID
4	uint8	handle	The advertisement set handle

C Functions

```
/* Event id */
gecko_evt_le_gap_bt5_adv_timeout_id

/* Event structure */
struct gecko_msg_le_gap_bt5_adv_timeout_evt_t
{
  uint8 handle;
};
```

2.9.2.3 evt_le_gap_scan_request

This event reports any scan request received in advertisement mode if scan request reporting is enabled

Table 2.183. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x09	lolen	Minimum payload length
2	0x03	class	Message class: Generic Access Profile, Low Energy
3	0x02	method	Message ID
4	uint8	handle	Advertisement set handle where scan request was received
5-10	bd_addr	address	Bluetooth address of the scanning device
11	uint8	address_type	Scanner address type. Values: • **0:** Public address • **1:** Random address
12	uint8	bonding	Bonding handle if the remote scanning device has previously bonded with the local device. Values: • **0xff:** No bonding • **Other:** Bonding handle

C Functions

```
/* Event id */
gecko_evt_le_gap_scan_request_id

/* Event structure */
struct gecko_msg_le_gap_scan_request_evt_t
{
  uint8 handle;,
  bd_addr address;,
  uint8 address_type;,
  uint8 bonding;
};
```

2.9.2.4 evt_le_gap_scan_response

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

Table 2.184. 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 +20 • Units: dBm
5	uint8	packet_type	Advertisement packet type • 0x00: Connectable undirected advertising • 0x02: Scannable undirected advertising • 0x03: Non connectable undirected advertising • 0x04: Scan Response Note: Scan response (0x04) is only received if the device 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 device has previously bonded with the local device. Values: • **0xff:** No bonding • **Other:** Bonding handle
14	uint8array	data	Advertisement or scan response data

C Functions

```
/* Event id */
gecko_evt_le_gap_scan_response_id

/* Event structure */
struct gecko_msg_le_gap_scan_response_evt_t
{
  int8 rssi;,
  uint8 packet_type;,
  bd_addr address;,
  uint8 address_type;,
  uint8 bonding;,
  uint8array data;
};
```

2.9.3 le_gap enumerations

2.9.3.1 enum_le_gap_address_type

These values define the Bluetooth Address types used by the stack.

Table 2.185. 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

2.9.3.2 enum_le_gap_adv_address_type

Address type to use for advertisements

Table 2.186. Enumerations

Value	Name	Description
0	le_gap_identity_address	Use public or static device address, or identity address if privacy mode is enabled
1	le_gap_non_resolvable	Use non resolvable address type, advertisement mode must also be non-connectable

2.9.3.3 enum_le_gap_connectable_mode

These values define the available Connectable Modes.

Table 2.187. Enumerations

Value	Name	Description
0	le_gap_non_connectable	Not connectable
1	le_gap_directed_connectable	Directed Connectable (RESERVED, DO NOT USE)
2	le_gap_undirected_connectable	Undirected connectable
3	le_gap_scannable_non_connectable	Not connectable but responds to scan_req-packets

2.9.3.4 enum_le_gap_discover_mode

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

Table 2.188. 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

2.9.3.5 enum_le_gap_discoverable_mode

These values define the available Discoverable Modes, which dictate how the device is visible to other devices.

Table 2.189. Enumerations

Value	Name	Description
0	le_gap_non_discoverable	Not discoverable
1	le_gap_limited_discoverable	Discoverable using both limited and general discovery procedures
2	le_gap_general_discoverable	Discoverable using general discovery procedure
3	le_gap_broadcast	Device is not discoverable in either limited or generic discovery procedure, but may be discovered by using the Observation procedure
4	le_gap_user_data	Send advertisement and/or scan response data defined by the user using le_gap_set_adv_data. The limited/ general discoverable flags are defined by the user.

2.10 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.

The following procedure can be used to bond with a remote device:

- Use command sm_configure to configure security requirements and I/O capabilities of this device.
- Use command sm_set_bondable_mode to set this device into bondable mode.
- Use command le_gap_open to open a connection to the remote device.
- After the connection is open, use command sm_increase_security to encrypt the connection. This will also start the bonding process.

If MITM is required, the application needs to display or ask user to enter a passkey during the process. See events sm_passkey_display and sm_passkey_request for more information. The following procedure can be used to respond the bonding initiated by a remote device:

- Use command sm_configure to configure security requirements and I/O capabilities of this device.
- Use command sm_set_bondable_mode to set this device into bondable mode.
- Use le_gap_set_mode to set this device into advertising and connectable mode.
- Open a connection to this device from the remote device.
- After the connection is open, start the bonding process on the remote device.

If MITM is required, the application needs to display or ask user to enter a passkey during the process. See events sm_passkey_display and sm_passkey_request for more information.

2.10.1 sm commands

2.10.1.1 cmd_sm_bonding_confirm

This command can be used for accepting or rejecting bonding request.

Table 2.190. 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	0x0e	method	Message ID
4	uint8	connection	Connection handle
5	uint8	confirm	Accept bonding request. Values: • **0:** Reject • **1:** Accept bonding request

Table 2.191. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_sm_bonding_confirm_rsp_t *gecko_cmd_sm_bonding_confirm(uint8 connection, uint8 confirm);

/* Response id */
gecko_rsp_sm_bonding_confirm_id

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

2.10.1.2 cmd_sm_configure

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

Table 2.192. 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	flags	Security requirement bitmask.Bit 0: • **0:** Allow bonding without MITM protection • **1:** Bonding requires MITM protection Bit 1: • **0:** Allow encryption without bonding • **1:** Encryption requires bonding. Note that this setting will also enable bonding. Bit 2: • **0:** Allow bonding with legacy pairing • **1:** Secure connections only Bit 3: • **0:** Bonding request does not need to be confirmed • **1:** Bonding requests need to be confirmed. Received bonding requests are notified with sm_confirm_bonding events. Bit 4 to 7: ReservedDefault value: 0x00
5	uint8	io_capabilities	I/O Capabilities. See link

Table 2.193. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_sm_configure_rsp_t *gecko_cmd_sm_configure(uint8 flags, uint8 io_capabilities);

/* Response id */
gecko_rsp_sm_configure_id

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

2.10.1.3 cmd_sm_delete_bonding

This command can be used to delete specified bonding information from Persistent Store.

Table 2.194. 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 2.195. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_sm_delete_bonding_rsp_t *gecko_cmd_sm_delete_bonding(uint8 bonding);

/* Response id */
gecko_rsp_sm_delete_bonding_id

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

2.10.1.4 cmd_sm_delete_bondings

This command can be used to delete all bonding information from Persistent Store.

Table 2.196. 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 2.197. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_sm_delete_bondings_rsp_t *gecko_cmd_sm_delete_bondings();

/* Response id */
gecko_rsp_sm_delete_bondings_id

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

2.10.1.5 cmd_sm_enter_passkey

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

Table 2.198. 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	int32	passkey	Passkey. Valid range: 0-999999. Set -1 to cancel pairing.

Table 2.199. 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

BGLIB C API

```
/* Function */
struct gecko_msg_sm_enter_passkey_rsp_t *gecko_cmd_sm_enter_passkey(uint8 connection, int32 passkey);

/* Response id */
gecko_rsp_sm_enter_passkey_id

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

2.10.1.6 cmd_sm_increase_security

This command can be used to enhance the security of a connection to current security requirements. On an unencrypted connection, this will encrypt the connection and will also perform bonding if requested by both devices. On an encrypted connection, this will cause the connection re-encrypted.

Table 2.200. 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 2.201. 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

BGLIB C API

```
/* Function */
struct gecko_msg_sm_increase_security_rsp_t *gecko_cmd_sm_increase_security(uint8 connection);

/* Response id */
gecko_rsp_sm_increase_security_id

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

Table 2.202. Events Generated

Event	Description
le_connection_parameters	This event is triggered after increasing security has been completed successfully, and indicates the latest security mode of the connection.
sm_bonded	This event is triggered if pairing or bonding was performed in this operation and the result is success.
sm_bonding_failed	This event is triggered if pairing or bonding was performed in this operation and the result is failure.

2.10.1.7 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_entry event for each bonding and the report is ended with sm_list_all_bondings_complete event. Recommended to be used only for debugging purposes, because reading from the Persistent Store is relatively slow.

Table 2.203. 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 2.204. 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

BGLIB C API

```
/* Function */
struct gecko_msg_sm_list_all_bondings_rsp_t *gecko_cmd_sm_list_all_bondings();

/* Response id */
gecko_rsp_sm_list_all_bondings_id

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

Table 2.205. Events Generated

Event	Description
sm_list_bonding_entry	This event is triggered by the command sm_list_all_bondings if bondings exist in the local database.
sm_list_all_bondings_complete	This event is triggered by the sm_list_all_bondings and follows sm_list_bonding_entry events.

2.10.1.8 cmd_sm_passkey_confirm

This command can be used for accepting or rejecting reported confirm value.

Table 2.206. 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	0x09	method	Message ID
4	uint8	connection	Connection handle
5	uint8	confirm	Accept confirm value. Values: • **0:** Reject • **1:** Accept confirm value

Table 2.207. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_sm_passkey_confirm_rsp_t *gecko_cmd_sm_passkey_confirm(uint8 connection, uint8 confirm);

/* Response id */
gecko_rsp_sm_passkey_confirm_id

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

2.10.1.9 cmd_sm_set_bondable_mode

This command can be used to set whether the device accepts new bondings or not.

Table 2.208. 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 mode. Values: • **0:** New bondings not accepted • **1:** Bondings allowed

Table 2.209. 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

BGLIB C API

```
/* Function */
struct gecko_msg_sm_set_bondable_mode_rsp_t *gecko_cmd_sm_set_bondable_mode(uint8 bondable);

/* Response id */
gecko_rsp_sm_set_bondable_mode_id

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

2.10.1.10 cmd_sm_set_debug_mode

This command can be used to set Security Manager in debug mode. In this mode the secure connections bonding uses debug keys, so that the encrypted packet can be opened by Bluetooth protocol analyzer. To disable the debug mode, you need to restart the device.

Table 2.210. 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	0x0f	method	Message ID

Table 2.211. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_sm_set_debug_mode_rsp_t *gecko_cmd_sm_set_debug_mode();

/* Response id */
gecko_rsp_sm_set_debug_mode_id

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

2.10.1.11 cmd_sm_set_oob_data

This command can be used to set the OOB data (out-of-band encryption data) for legacy pairing for a device. The OOB data may be, for example, a PIN code exchanged over an alternate path like NFC. The device will not allow any other kind of bonding if OOB data is set. The OOB data cannot be set simultaneously with secure connections OOB data.

Table 2.212. 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_data	OOB data. To set OOB data, send a 16-byte array. To clear OOB data, send a zero-length array.

Table 2.213. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_sm_set_oob_data_rsp_t *gecko_cmd_sm_set_oob_data(uint8 oob_data_len, const uint8 *oob_data_dat
a);

/* Response id */
gecko_rsp_sm_set_oob_data_id

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

2.10.1.12 cmd_sm_set_passkey

This command can be used to enter a fixed passkey which will be used in the sm_passkey_display event.

Table 2.214. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
3	0x10	method	Message ID
4-7	int32	passkey	Passkey. Valid range: 0-999999. Set -1 to disable and start using random passkeys.

Table 2.215. 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	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

BGLIB C API

```
/* Function */
struct gecko_msg_sm_set_passkey_rsp_t *gecko_cmd_sm_set_passkey(int32 passkey);

/* Response id */
gecko_rsp_sm_set_passkey_id

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

2.10.1.13 cmd_sm_set_sc_remote_oob_data

This command can be used to set OOB data and confirm values (out-of-band encryption) received from the remote device for secure connections pairing. OOB data must be enabled with sm_use_sc_oob before setting the remote device OOB data.

Table 2.216. 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	0x12	method	Message ID
4	uint8array	oob_data	Remote device OOB data and confirm values. To set OOB data, send a 32-byte array. First 16-bytes is the OOB data and last 16-bytes the confirm value. To clear OOB data, send a zero-length array.

Table 2.217. 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	0x12	method	Message ID
4-5	uint16	result	Result code • **0:** success • **Non-zero:** an error occurred For other values refer to the Error codes

BGLIB C API

```
/* Function */
struct gecko_msg_sm_set_sc_remote_oob_data_rsp_t *gecko_cmd_sm_set_sc_remote_oob_data(uint8 oob_data_len, cons
t uint8 *oob_data_data);

/* Response id */
gecko_rsp_sm_set_sc_remote_oob_data_id

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

2.10.1.14 cmd_sm_store_bonding_configuration

This command can be used to set maximum allowed bonding count and bonding policy. The default value is maximum number of bondings supported.

Table 2.218. 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 count. Range: 1 to 14
5	uint8	policy_flags	Bonding policy. Values: • **0:** If database is full, new bonding attempts will fail • **1:** New bonding will overwrite the oldest existing bonding • **2:** New bonding will overwrite longest time ago used existing bonding

Table 2.219. 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

BGLIB C API

```
/* Function */
struct gecko_msg_sm_store_bonding_configuration_rsp_t *gecko_cmd_sm_store_bonding_configuration(uint8 max_bondi
ng_count, uint8 policy_flags);

/* Response id */
gecko_rsp_sm_store_bonding_configuration_id

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

2.10.1.15 cmd_sm_use_sc_oob

This command can be used to enable the use of OOB data (out-of-band encryption data) for a device for secure connections pairing. The enabling will genarate new OOB data and confirm values which can be sent to the remote device. After enabling the secure connections OOB data, the remote devices OOB data can be set with sm_set_sc_remote_oob_data. Calling this function will erase any set remote device OOB data and confirm values. The device will not allow any other kind of bonding if OOB data is set. The secure connections OOB data cannot be enabled simultaneously with legacy pairing OOB data.

Table 2.220. 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	0x11	method	Message ID
4	uint8	enable	Enable OOB with secure connections pairing. Values: • **0:** disable • **1:** enable

Table 2.221. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x0f	class	Message class: Security Manager
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
6	uint8array	oob_data	OOB data. 32-byte array. First 16-bytes contain randomly generated OOB data and last 16-bytes confirm value.

BGLIB C API

```
/* Function */
struct gecko_msg_sm_use_sc_oob_rsp_t *gecko_cmd_sm_use_sc_oob(uint8 enable);

/* Response id */
gecko_rsp_sm_use_sc_oob_id

/* Response structure */
struct gecko_msg_sm_use_sc_oob_rsp_t
{
  uint16 result;,
  uint8array oob_data;
};
```

2.10.2 sm events

2.10.2.1 evt_sm_bonded

This event is triggered after the pairing or bonding procedure has been successfully completed.

Table 2.222. 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 handle. Values: • **0xff:** Pairing completed without bonding - the pairing key will be discarded after disconnection. • **Other:** Procedure completed, pairing key stored with given bonding handle

C Functions

```
/* Event id */
gecko_evt_sm_bonded_id

/* Event structure */
struct gecko_msg_sm_bonded_evt_t
{
  uint8 connection;,
  uint8 bonding;
};
```

2.10.2.2 evt_sm_bonding_failed

This event is triggered if the pairing or bonding procedure has failed.

Table 2.223. 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

C Functions

```
/* Event id */
gecko_evt_sm_bonding_failed_id

/* Event structure */
struct gecko_msg_sm_bonding_failed_evt_t
{
  uint8 connection;,
  uint16 reason;
};
```

2.10.2.3 evt_sm_confirm_bonding

This event indicates a request to display that new bonding request has been received to the user and for the user to confirm the request. Use the command sm_bonding_confirm to accept or reject the bonding request.

Table 2.224. 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	0x09	method	Message ID
4	uint8	connection	Connection handle
5	int8	bonding_handle	Bonding handle for the request. Range: -1 to 31. • NOTE! When the bonding handle is anything else than -1 there is already existing bonding for this connection. Overwriting existing bonding is potential security risk.

C Functions

```
/* Event id */
gecko_evt_sm_confirm_bonding_id

/* Event structure */
struct gecko_msg_sm_confirm_bonding_evt_t
{
  uint8 connection;,
  int8 bonding_handle;
};
```

2.10.2.4 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 or reject the displayed passkey.

Table 2.225. 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	Passkey. Range: 0 to 999999. • NOTE! When displaying the passkey to the user, prefix the number with zeros in order to obtain a 6 digit number • Example: Passkey value is 42 • Number to display to user is 000042

C Functions

```
/* Event id */
gecko_evt_sm_confirm_passkey_id

/* Event structure */
struct gecko_msg_sm_confirm_passkey_evt_t
{
  uint8 connection;,
  uint32 passkey;
};
```

2.10.2.5 evt_sm_list_all_bondings_complete

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

Table 2.226. 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

C Functions

```
/* Event id */
gecko_evt_sm_list_all_bondings_complete_id

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

2.10.2.6 evt_sm_list_bonding_entry

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

Table 2.227. 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

C Functions

```
/* Event id */
gecko_evt_sm_list_bonding_entry_id

/* Event structure */
struct gecko_msg_sm_list_bonding_entry_evt_t
{
  uint8 bonding;,
  bd_addr address;,
  uint8 address_type;
};
```

2.10.2.7 evt_sm_passkey_display

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

Table 2.228. 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	Passkey. Range: 0 to 999999. • NOTE! When displaying the passkey to the user, prefix the number with zeros in order to obtain a 6 digit number • Example: Passkey value is 42 • Number to display to user is 000042

C Functions

```
/* Event id */
gecko_evt_sm_passkey_display_id

/* Event structure */
struct gecko_msg_sm_passkey_display_evt_t
{
  uint8 connection;,
  uint32 passkey;
};
```

2.10.2.8 evt_sm_passkey_request

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

Table 2.229. 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

C Functions

```
/* Event id */
gecko_evt_sm_passkey_request_id

/* Event structure */
struct gecko_msg_sm_passkey_request_evt_t
{
  uint8 connection;
};
```

2.10.3 sm enumerations

2.10.3.1 enum_sm_bonding_key

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

Table 2.230. 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

2.10.3.2 enum_sm_io_capability

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

Table 2.231. 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

2.11 System (system)

The commands and events in this class can be used to access and query the local device.

2.11.1 system commands

2.11.1.1 cmd_system_get_bt_address

This command can be used to read the Bluetooth public address used by the device.

Table 2.232. 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 2.233. 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 public address in little endian format

BGLIB C API

```
/* Function */
struct gecko_msg_system_get_bt_address_rsp_t *gecko_cmd_system_get_bt_address();

/* Response id */
gecko_rsp_system_get_bt_address_id

/* Response structure */
struct gecko_msg_system_get_bt_address_rsp_t
{
  bd_addr address;
};
```

2.11.1.2 cmd_system_get_random_data

This command can be used to get random data up to 16 bytes.

Table 2.234. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x0b	method	Message ID
4	uint8	length	Length of random data. Maximum length is 16 bytes.

Table 2.235. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0x01	class	Message class: System
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
6	uint8array	data	Random data

BGLIB C API

```
/* Function */
struct gecko_msg_system_get_random_data_rsp_t *gecko_cmd_system_get_random_data(uint8 length);

/* Response id */
gecko_rsp_system_get_random_data_id

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

2.11.1.3 cmd_system_halt

This command forces radio to idle state and allows device to sleep. Advertising, scanning, connections and software timers are halted by this commands. Halted operations are resumed by calling this command with parameter 0. Connections stay alive if system is resumed before connection supervision timeout.

NOTE: Software timer is also halted. Hardware interrupts are the only way to wake up from energy mode 2 when system is halted.

Table 2.236. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x0c	method	Message ID
4	uint8	halt	Values: • **1:** halt • **0:** resume

Table 2.237. Response

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

BGLIB C API

```
/* Function */
struct gecko_msg_system_halt_rsp_t *gecko_cmd_system_halt(uint8 halt);

/* Response id */
gecko_rsp_system_halt_id

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

2.11.1.4 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 device is working.

Table 2.238. 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 2.239. 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

BGLIB C API

```
/* Function */
struct gecko_msg_system_hello_rsp_t *gecko_cmd_system_hello();

/* Response id */
gecko_rsp_system_hello_id

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

2.11.1.5 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 2.240. 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 UART DFU mode • **2:** Boot to OTA DFU mode

BGLIB C API

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

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

Table 2.241. Events Generated

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

2.11.1.6 cmd_system_set_bt_address

This command can be used to set the Bluetooth public address used by the device. A valid address set with this command overrides the default Bluetooth public address programmed at production, and it will be effective in the next system reboot. The stack treats 00:00:00:00:00:00 and ff:ff:ff:ff:ff:ff as invalid addresses. Thus passing one of them into this command will cause the stack to use the default address in the next system reboot.

Table 2.242. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x06	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x04	method	Message ID
4-9	bd_addr	address	Bluetooth public address in little endian format

Table 2.243. Response

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

BGLIB C API

```
/* Function */
struct gecko_msg_system_set_bt_address_rsp_t *gecko_cmd_system_set_bt_address(bd_addr address);

/* Response id */
gecko_rsp_system_set_bt_address_id

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

2.11.1.7 cmd_system_set_device_name

This command can be used to set the device name. Currently it is possible to set the name which will be used during the OTA update. The name will be stored in persistent storage. If the OTA device name is also set in gecko configuration, the name stored in persistent storage is overwritten with the name in gecko configuration during device boot.

Table 2.244. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x0d	method	Message ID
4	uint8	type	Device name to set. Values: • **0:** OTA device name
5	uint8array	name	Device name

Table 2.245. Response

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

BGLIB C API

```
/* Function */
struct gecko_msg_system_set_device_name_rsp_t *gecko_cmd_system_set_device_name(uint8 type, uint8 name_len, con
st uint8 *name_data);

/* Response id */
gecko_rsp_system_set_device_name_id

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

2.11.1.8 cmd_system_set_tx_power

This command can be used to set the TX power. It returns the power that was set. If the GATT server contains a Tx Power service, the Tx Power Level attribute of the service will be updated accordingly.

NOTE: This command should not be used while advertising, scanning or during connection.

Table 2.246. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x0a	method	Message ID
4-5	int16	power	TX power in 0.1dBm steps, for example the value of 10 is 1dBm and 55 is 5.5dBm

Table 2.247. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x02	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x0a	method	Message ID
4-5	int16	set_power	Returns the power value currently in use after setting

BGLIB C API

```
/* Function */
struct gecko_msg_system_set_tx_power_rsp_t *gecko_cmd_system_set_tx_power(int16 power);

/* Response id */
gecko_rsp_system_set_tx_power_id

/* Response structure */
struct gecko_msg_system_set_tx_power_rsp_t
{
  int16 set_power;
};
```

2.11.2 system events

2.11.2.1 evt_system_awake

This event indicates that the device has woken up from sleep mode.

NOTE: Stack does not generate this event by itself as sleep and wakeup are managed in applications. If this event is needed, the application should call function gecko_send_system_awake() to signal stack to send this event.

Table 2.248. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x00	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x04	method	Message ID

C Functions

```
/* Event id */
gecko_evt_system_awake_id

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

2.11.2.2 evt_system_boot

This event indicates the device has started and the radio is ready. This even carries the firmware build number and other SW and HW identification codes.

Table 2.249. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x0e	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x00	method	Message ID
4-5	uint16	major	Major release version
6-7	uint16	minor	Minor release version
8-9	uint16	patch	Patch release number
10-11	uint16	build	Build number
12-15	uint32	bootloader	Bootloader version
16-17	uint16	hw	Hardware type

C Functions

```
/* Event id */
gecko_evt_system_boot_id

/* Event structure */
struct gecko_msg_system_boot_evt_t
{
  uint16 major;,
  uint16 minor;,
  uint16 patch;,
  uint16 build;,
  uint32 bootloader;,
  uint16 hw;
};
```

2.11.2.3 evt_system_external_signal

This event indicates external signals have been received. External signals are generated from native application.

Table 2.250. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x04	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x03	method	Message ID
4-7	uint32	extsignals	Bitmask of external signals received since last event.

C Functions

```
/* Event id */
gecko_evt_system_external_signal_id

/* Event structure */
struct gecko_msg_system_external_signal_evt_t
{
  uint32 extsignals;
};
```

2.11.2.4 evt_system_hardware_error

This event indicates that hardware related errors occurred.

Table 2.251. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x02	lolen	Minimum payload length
2	0x01	class	Message class: System
3	0x05	method	Message ID
4-5	uint16	status	Result code • **0:** success • **Non-zero:** an error occurred For other values refer to the Error codes

C Functions

```
/* Event id */
gecko_evt_system_hardware_error_id

/* Event structure */
struct gecko_msg_system_hardware_error_evt_t
{
  uint16 status;
};
```

2.12 testing commands (test)

2.12.1 test commands

2.12.1.1 cmd_test_dtm_end

This command can be used to end a transmitter or a receiver test. When the command is processed by the radio and the test has ended, a test_dtm_completed event is triggered.

Table 2.252. 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 2.253. 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

BGLIB C API

```
/* Function */
struct gecko_msg_test_dtm_end_rsp_t *gecko_cmd_test_dtm_end();

/* Response id */
gecko_rsp_test_dtm_end_id

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

Table 2.254. Events Generated

Event	Description
test_dtm_completed	This event is received when the command is processed.

2.12.1.2 cmd_test_dtm_rx

This command can be used to start a receiver test. The test is meant to be used against a separate Bluetooth tester device. When the command is processed by the radio, a test_dtm_completed event is triggered. This event indicates if the test started successfully.

Parameter **phy** specifies which PHY is used to receive the packets. All devices support at least the 1M PHY.

The test may be stopped using the test_dtm_end command. This will trigger another test_dtm_completed event, which carries the number of packets received during the test.

Table 2.255. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x02	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
3	0x01	method	Message ID
4	uint8	channel	Bluetooth channel **Range:** 0-39 Channel is (F - 2402) / 2, where F is frequency in MHz
5	uint8	phy	PHY to use

Table 2.256. 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

BGLIB C API

```
/* Function */
struct gecko_msg_test_dtm_rx_rsp_t *gecko_cmd_test_dtm_rx(uint8 channel, uint8 phy);

/* Response id */
gecko_rsp_test_dtm_rx_id

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

Table 2.257. Events Generated

Event	Description
test_dtm_completed	This event is received when the command is processed.

2.12.1.3 cmd_test_dtm_tx

This command can be used to start a transmitter test. The test is meant to be used against a separate Bluetooth tester device. When the command is processed by the radio, a test_dtm_completed event is triggered. This event indicates if the test started successfully.

In the transmitter test, the device sends packets continuously with a fixed interval. The type and length of each packet is set by **packet_type** and **length** parameters. Parameter **phy** specifies which PHY is used to transmit the packets. All devices support at least the 1M PHY. There is also a special packet type, **test_pkt_carrier**, which can be used to transmit continuous unmodulated carrier. The **length** field is ignored in this mode.

The test may be stopped using the test_dtm_end command.

Table 2.258. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
3	0x00	method	Message ID
4	uint8	packet_type	Packet type to transmit
5	uint8	length	Packet length in bytes **Range:** 0-255
6	uint8	channel	Bluetooth channel **Range:** 0-39 Channel is (F - 2402) / 2, where F is frequency in MHz
7	uint8	phy	PHY to use

Table 2.259. 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

BGLIB C API

```
/* Function */
struct gecko_msg_test_dtm_tx_rsp_t *gecko_cmd_test_dtm_tx(uint8 packet_type, uint8 length, uint8 channel, uint
8 phy);

/* Response id */
gecko_rsp_test_dtm_tx_id

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

Table 2.260. Events Generated

Event	Description
test_dtm_completed	This event is received when the command is processed.

2.12.2 test events

2.12.2.1 evt_test_dtm_completed

This event indicates that the radio has processed a test start or end command. The **result** parameter indicates the success of the command.

After the receiver or transmitter test is stopped, the **number_of_packets** parameter in this event indicates the number of received or transmitted packets.

Table 2.261. 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 Only valid for test_dtm_end command.

C Functions

```
/* Event id */
gecko_evt_test_dtm_completed_id

/* Event structure */
struct gecko_msg_test_dtm_completed_evt_t
{
  uint16 result;,
  uint16 number_of_packets;
};
```

2.12.3 test enumerations

2.12.3.1 enum_test_packet_type

Test packet types

Number 3 corresponding to unmodulated carrier packet type is deprecated. Valid number is 254.

Table 2.262. 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
3	test_pkt_carrier_deprecated	Unmodulated carrier - deprecated
253	test_pkt_pn9	PN9
254	test_pkt_carrier	Unmodulated carrier

2.12.3.2 enum_test_phy

Test PHY types

Table 2.263. Enumerations

Value	Name	Description
1	test_phy_1m	1M PHY
2	test_phy_2m	2M PHY
3	test_phy_125k	125k Coded PHY
4	test_phy_500k	500k Coded PHY

2.13 User messaging (user)

This class provides one command and one event which can be utilized by a NCP host and target to implement a communication mechanism with own proprietary protocol. An application has the full responsibility on deciding whether and how the command and event are used. The stack does not produce or consume any messages belonging to this class.

2.13.1 user commands

2.13.1.1 cmd_user_message_to_target

This command can be used by an NCP host to send a message to the target application on device.

Table 2.264. Command

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x01	lolen	Minimum payload length
2	0xff	class	Message class: User messaging
3	0x00	method	Message ID
4	uint8array	data	The message

Table 2.265. Response

Byte	Type	Name	Description
0	0x20	hilen	Message type: Response
1	0x03	lolen	Minimum payload length
2	0xff	class	Message class: User messaging
3	0x00	method	Message ID
4-5	uint16	result	Result code • **0:** success • **Non-zero:** an error occurred For other values refer to the
6	uint8array	data	The response message

BGLIB C API

```
/* Function */
struct gecko_msg_user_message_to_target_rsp_t *gecko_cmd_user_message_to_target(uint8 data_len, const uint8 *da
ta_data);

/* Response id */
gecko_rsp_user_message_to_target_id

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

2.13.2 user events

2.13.2.1 evt_user_message_to_host

This event can be used by the target application on device to send a message to NCP host.

Table 2.266. Event

Byte	Type	Name	Description
0	0xa0	hilen	Message type: Event
1	0x01	lolen	Minimum payload length
2	0xff	class	Message class: User messaging
3	0x00	method	Message ID
4	uint8array	data	The message

C Functions

```
/* Event id */
gecko_evt_user_message_to_host_id

/* Event structure */
struct gecko_msg_user_message_to_host_evt_t
{
  uint8array data;
};
```

2.14 Error codes

This chapter describes all BGAPI error codes.

- **Errors related to hardware**

Code	Name	Description
0x0501	ps_store_full	Flash reserved for PS store is full
0x0502	ps_key_not_found	PS key not found
0x0503	i2c_ack_missing	Acknowledge for i2c was not received.
0x0504	i2c_timeout	I2C read or write timed out.

- **Errors related to BGAPI protocol**

Code	Name	Description
0x0101	invalid_conn_handle	Invalid GATT connection handle.
0x0102	waiting_response	Waiting response from GATT server to previous procedure.
0x0103	gatt_connection_timeout	GATT connection is closed due procedure timeout.
0x0180	invalid_param	Command contained invalid parameter
0x0181	wrong_state	Device is in wrong state to receive command
0x0182	out_of_memory	Device has run out of memory
0x0183	not_implemented	Feature is not implemented
0x0184	invalid_command	Command was not recognized
0x0185	timeout	Command or Procedure failed due to timeout

Code	Name	Description
0x0186	not_connected	Connection handle passed is to command is not a valid handle
0x0187	flow	Command would cause either underflow or overflow error
0x0188	user_attribute	User attribute was accessed through API which is not supported
0x0189	invalid_license_key	No valid license key found
0x018a	command_too_long	Command maximum length exceeded
0x018b	out_of_bonds	Bonding procedure can't be started because device has no space left for bond.
0x018c	unspecified	Unspecified error
0x018d	hardware	Hardware failure
0x018e	buffers_full	Command not accepted, because internal buffers are full
0x018f	disconnected	Command or Procedure failed due to disconnection
0x0190	too_many_requests	Too many Simultaneous Requests
0x0191	not_supported	Feature is not supported in this firmware build
0x0192	no_bonding	The bonding does not exist.
0x0193	crypto	Error using crypto functions
0x0194	data_corrupted	Data was corrupted.

- **Errors from Security Manager Protocol**

Code	Name	Description
0x0301	passkey_entry_failed	The user input of passkey failed, for example, the user cancelled the operation
0x0302	oob_not_available	Out of Band data is not available for authentication
0x0303	authentication_requirements	The pairing procedure cannot be performed as authentication requirements cannot be met due to IO capabilities of one or both devices
0x0304	confirm_value_failed	The confirm value does not match the calculated compare value
0x0305	pairing_not_supported	Pairing is not supported by the device
0x0306	encryption_key_size	The resultant encryption key size is insufficient for the security requirements of this device
0x0307	command_not_supported	The SMP command received is not supported on this device
0x0308	unspecified_reason	Pairing failed due to an unspecified reason
0x0309	repeated_attempts	Pairing or authentication procedure is disallowed because too little time has elapsed since last pairing request or security request
0x030a	invalid_parameters	The Invalid Parameters error code indicates: the command length is invalid or a parameter is outside of the specified range.

Code	Name	Description
0x030b	dhkey_check_failed	Indicates to the remote device that the DHKey Check value received doesn't match the one calculated by the local device.
0x030c	numeric_comparison_failed	Indicates that the confirm values in the numeric comparison protocol do not match.
0x030d	bredr_pairing_in_progress	Indicates that the pairing over the LE transport failed due to a Pairing Request sent over the BR/EDR transport in process.
0x030e	cross_transport_key_derivation_generation_not_allowed	Indicates that the BR/EDR Link Key generated on the BR/EDR transport cannot be used to derive and distribute keys for the LE transport.

- **Bluetooth errors**

Code	Name	Description
0x0202	unknown_connection_identifier	A command was sent from the Host that should identify a connection, but that connection does not exist.
0x0204	page_timeout	The Page Timeout error code indicates that a page timed out because of the Page Timeout configuration parameter.
0x0205	authentication_failure	Pairing or authentication failed due to incorrect results in the pairing or authentication procedure. This could be due to an incorrect PIN or Link Key
0x0206	pin_or_key_missing	Pairing failed because of missing PIN, or authentication failed because of missing Key
0x0207	memory_capacity_exceeded	Controller is out of memory.
0x0208	connection_timeout	Link supervision timeout has expired.
0x0209	connection_limit_exceeded	Controller is at limit of connections it can support.
0x020a	synchronous_connectiontion_limit_exceeded	The Synchronous Connection Limit to a Device Exceeded error code indicates that the Controller has reached the limit to the number of synchronous connections that can be achieved to a device.
0x020b	acl_connection_already_exists	The ACL Connection Already Exists error code indicates that an attempt to create a new ACL Connection to a device when there is already a connection to this device.
0x020c	command_disallowed	Command requested cannot be executed because the Controller is in a state where it cannot process this command at this time.
0x020d	connection_rejected_due_to_limited_resources	The Connection Rejected Due To Limited Resources error code indicates that an incoming connection was rejected due to limited resources.
0x020e	connection_rejected_due_to_security_reasons	The Connection Rejected Due To Security Reasons error code indicates that a connection was rejected due to security requirements not being fulfilled, like authentication or pairing.
0x020f	connection_rejected_due_to_unacceptable_bd_addr	The Connection was rejected because this device does not accept the BD_ADDR. This may be because the device will only accept connections from specific BD_ADDRs.

Code	Name	Description
0x0210	connection_accept_timeout_exceeded	The Connection Accept Timeout has been exceeded for this connection attempt.
0x0211	unsupported_feature_or_parameter_value	A feature or parameter value in the HCI command is not supported.
0x0212	invalid_command_parameters	Command contained invalid parameters.
0x0213	remote_user_terminated	User on the remote device terminated the connection.
0x0214	remote_device_terminated_connection_due_to_low_resources	The remote device terminated the connection because of low resources
0x0215	remote_powering_off	Remote Device Terminated Connection due to Power Off
0x0216	connection_terminated_by_local_host	Local device terminated the connection.
0x0217	repeated_attempts	The Controller is disallowing an authentication or pairing procedure because too little time has elapsed since the last authentication or pairing attempt failed.
0x0218	pairing_not_allowed	The device does not allow pairing. This can be for example, when a device only allows pairing during a certain time window after some user input allows pairing
0x0219	unknown_lmp_pdu	The Controller has received an unknown LMP OpCode.
0x021a	unsupported_remote_feature	The remote device does not support the feature associated with the issued command or LMP PDU.
0x021b	sco_offset_rejected	The offset requested in the LMP_SCO_link_req PDU has been rejected.
0x021c	sco_interval_rejected	The interval requested in the LMP_SCO_link_req PDU has been rejected.
0x021d	sco_air_mode_rejected	The air mode requested in the LMP_SCO_link_req PDU has been rejected.
0x021e	invalid_lmp_parameters	Some LMP PDU / LL Control PDU parameters were invalid.
0x021f	unspecified_error	No other error code specified is appropriate to use.
0x0220	unsupported_lmp_parameter_value	An LMP PDU or an LL Control PDU contains at least one parameter value that is not supported by the Controller at this time.
0x0221	role_change_not_allowed	Controller will not allow a role change at this time.
0x0222	ll_response_timeout	Connection terminated due to link-layer procedure timeout.
0x0223	lmp_error_transaction_collision	LMP transaction has collided with the same transaction that is already in progress.
0x0224	lmp_pdu_not_allowed	Controller sent an LMP PDU with an OpCode that was not allowed.
0x0225	encryption_mode_not_acceptable	The requested encryption mode is not acceptable at this time.
0x0226	link_key_cannot_be_changed	Link key cannot be changed because a fixed unit key is being used.
0x0227	requested_qos_not_supported	The requested Quality of Service is not supported.

Code	Name	Description
0x0228	instant_passed	LMP PDU or LL PDU that includes an instant cannot be performed because the instant when this would have occurred has passed.
0x0229	pairing_with_unit_key_not_supported	It was not possible to pair as a unit key was requested and it is not supported.
0x022a	different_transaction_collision	LMP transaction was started that collides with an ongoing transaction.
0x022c	qos_unacceptable_parameter	The specified quality of service parameters could not be accepted at this time, but other parameters may be acceptable.
0x022d	qos_rejected	The specified quality of service parameters cannot be accepted and QoS negotiation should be terminated.
0x022e	channel_assesment_not_supported	The Controller cannot perform channel assessment because it is not supported.
0x022f	insufficient_security	The HCI command or LMP PDU sent is only possible on an encrypted link.
0x0230	parameter_out_of_mandatory_range	A parameter value requested is outside the mandatory range of parameters for the given HCI command or LMP PDU.
0x0232	role_switch_pending	Role Switch is pending. This can be used when an HCI command or LMP PDU cannot be accepted because of a pending role switch. This can also be used to notify a peer device about a pending role switch.
0x0234	reserved_slot_violation	The current Synchronous negotiation was terminated with the negotiation state set to Reserved Slot Violation.
0x0235	role_switch_failed	role switch was attempted but it failed and the original piconet structure is restored. The switch may have failed because the TDD switch or piconet switch failed.
0x0236	extended_inquiry_response_too_large	The extended inquiry response, with the requested requirements for FEC, is too large to fit in any of the packet types supported by the Controller.
0x0237	simple_pairing_not_supported_by_host	The IO capabilities request or response was rejected because the sending Host does not support Secure Simple Pairing even though the receiving Link Manager does.
0x0238	host_busy_pairing	The Host is busy with another pairing operation and unable to support the requested pairing. The receiving device should retry pairing again later.
0x0239	connection_rejected_due_to_no_suitable_channel_found	The Controller could not calculate an appropriate value for the Channel selection operation.
0x023a	controller_busy	Operation was rejected because the controller is busy and unable to process the request.
0x023b	unacceptable_connection_interval	Remote device terminated the connection because of an unacceptable connection interval.
0x023c	directed_advertising_timeout	Directed advertising completed without a connection being created.
0x023d	connection_terminated_due_to_mic_failure	Connection was terminated because the Message Integrity Check (MIC) failed on a received packet.

Code	Name	Description
0x023e	connection_failed_to_be_established	LL initiated a connection but the connection has failed to be established. Controller did not receive any packets from remote end.
0x023f	mac_connection_failed	The MAC of the 802.11 AMP was requested to connect to a peer, but the connection failed.
0x0240	coarse_clock_adjustment_rejected_but_will_try_to_adjust_using_clock_dragging	The master, at this time, is unable to make a coarse adjustment to the piconet clock, using the supplied parameters. Instead the master will attempt to move the clock using clock dragging.

- **Application errors**

Code	Name	Description
0x0a01	file_open_failed	File open failed.
0x0a02	xml_parse_failed	XML parsing failed.
0x0a03	device_connection_failed	Device connection failed.
0x0a04	device_comunication_failed	Device communication failed.
0x0a05	authentication_failed	Device authentication failed.
0x0a06	incorrect_gatt_database	Device has incorrect GATT database.
0x0a07	disconnected_due_to_procedure_collision	Device disconnected due to procedure collision.
0x0a08	disconnected_due_to_secure_session_failed	Device disconnected due to failure to establish or reestablish a secure session.
0x0a09	encryption_decryption_error	Encrypion/decryption operation failed.
0x0a0a	maximum_retries	Maximum allowed retries exceeded.
0x0a0b	data_parse_failed	Data parsing failed.
0x0a0c	pairing_removed	Pairing established by the application layer protocol has been removed.

- **Errors from Attribute Protocol**

Code	Name	Description
0x0401	invalid_handle	The attribute handle given was not valid on this server
0x0402	read_not_permitted	The attribute cannot be read
0x0403	write_not_permitted	The attribute cannot be written
0x0404	invalid_pdu	The attribute PDU was invalid
0x0405	insufficient_authentication	The attribute requires authentication before it can be read or written.
0x0406	request_not_supported	Attribute Server does not support the request received from the client.
0x0407	invalid_offset	Offset specified was past the end of the attribute
0x0408	insufficient_authorization	The attribute requires authorization before it can be read or written.
0x0409	prepare_queue_full	Too many prepare writes have been queueud

Code	Name	Description
0x040a	att_not_found	No attribute found within the given attribute handle range.
0x040b	att_not_long	The attribute cannot be read or written using the Read Blob Request
0x040c	insufficient_enc_key_size	The Encryption Key Size used for encrypting this link is insufficient.
0x040d	invalid_att_length	The attribute value length is invalid for the operation
0x040e	unlikely_error	The attribute request that was requested has encountered an error that was unlikely, and therefore could not be completed as requested.
0x040f	insufficient_encryption	The attribute requires encryption before it can be read or written.
0x0410	unsupported_group_type	The attribute type is not a supported grouping attribute as defined by a higher layer specification.
0x0411	insufficient_resources	Insufficient Resources to complete the request
0x0480	application	When this is returned in a BGAPI response, the application tried to read or write the value of a user attribute from the GATT database.

- **Bluetooth Mesh errors**

Code	Name	Description
0x0c01	already_exists	Returned when trying to add a key or some other unique resource with an ID which already exists
0x0c02	does_not_exist	Returned when trying to manipulate a key or some other resource with an ID which does not exist
0x0c03	limit_reached	Returned when an operation cannot be executed because a pre-configured limit for keys, key bindings, elements, models, virtual addresses, provisioned devices, or provisioning sessions is reached
0x0c04	invalid_address	Returned when trying to use a reserved address or add a "pre-provisioned" device using an address already used by some other device
0x0c05	malformed_data	In a BGAPI response, the user supplied malformed data; in a BGAPI event, the remote end responded with malformed or unrecognized data

- **Filesystem errors**

Code	Name	Description
0x0901	file_not_found	File not found

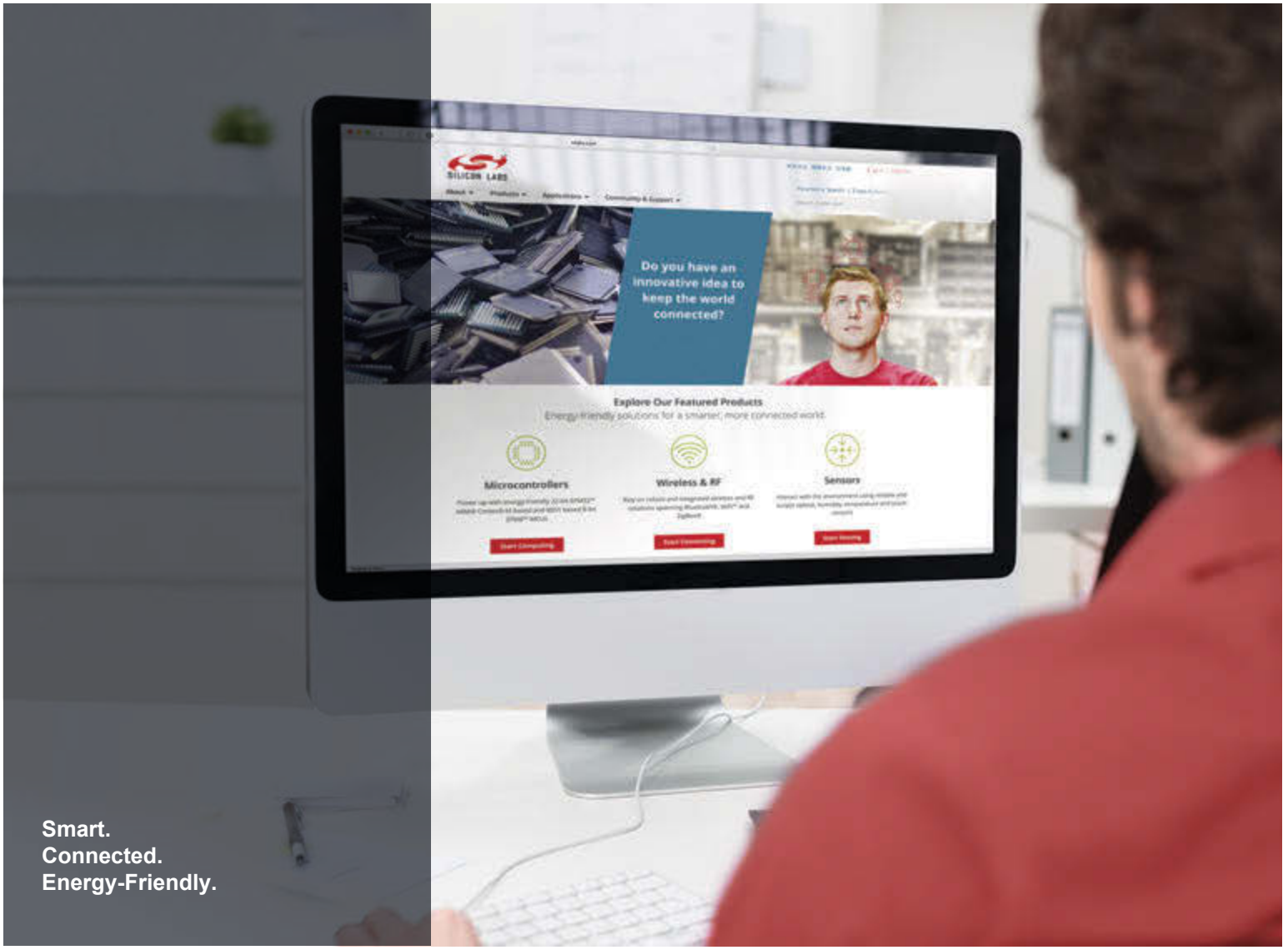
- **Security errors**

Code	Name	Description
0x0b01	image_signature_verification_failed	Device firmware signature verification failed.
0x0b02	file_signature_verification_failed	File signature verification failed.
0x0b03	image_checksum_error	Device firmware checksum is not valid.

3. Document Revision History

Table 3.1. Document Revision History

Revision Number	Effective Date	Change Description
1.0	April 1st 2015	Initial version.
1.1	December 23rd 2015	Updated for firmware version 0.9.2.
1.2	January 15th 2016	Corrected typography and formatting issues.
1.3	February 12th 2016	Updated for firmware version 1.0.0.
1.4	March 24th 2016	Updated for firmware version 1.0.2.
1.5	June 6th 2016	Updated for firmware version 1.0.4.
1.6	June 15th 2016	Revised description for timestamp parameter in evt_hardware_interrupt.
1.7	September 2nd 2016	Updated for firmware version 2.0.0.
1.8	October 13th 2016	Corrected default ATT MTU value.
1.9	December 2nd 2016	Updated for firmware version 2.1.0.
2.0	March 10th 2017	Updated for firmware version 2.3.0.
2.1	July 9th 2017	Updated for firmware version 2.4.0.
2.2	July 21st, 2017	Updated for firmware version 2.4.1.
2.3	August 16th, 2017	Updated for firmware version 2.4.2.
2.4	September 29th, 2017	Updated for firmware version 2.6.0 beta.
2.5	October 27th, 2017	Updated for firmware version 2.6.0.



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