Keyglove API Reference Guide

2014-12-07

keyglove.net
github.com/jrowberg/keyglove

This API Reference Guide describes the complete packet structure of all currently defined commands, responses, and events that are part of the Keyglove API binary protocol.

This protocol is still under active development and has not been officially locked to a specific structure and definition. This means that the class/ID values as well as the parameter names, types, and order for any API method MAY CHANGE AT ANY TIME. Please make very sure that you are always working with the latest available code and reference material.

1 Protocol class (ID = 0)

Protocol events occur when you try to use the protocol in an invalid way, or when you unintentionally send an incomplete command, invalid data, bad parameters, etc. They alert you to the fact that something has gone wrong.

There are no commands within this class; it only has events.

1.1 Events

1.1.1 protocol_error `[ 80 02 00 01 ... ]`

This event occurs when a problem exists with a command you have sent.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x02	length	Fixed-length payload (2)
2	0x00	class	Event class: "protocol"
3	0x01	id	Event ID: "error"
4 - 5	**uint16_t**	**code**	**Error code describing what went wrong with the protocol communication** • ***Enum:*** **protocol_error_code**

1.1.1.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_protocol_error(sender, args):
    print("kg_evt_protocol_error: { code: %04X }" % (args['code']))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_protocol_error += my_kg_evt_protocol_error
```

1.2 Enumerations

1.2.1 protocol_error_code

Describes the nature of a protocol error that has occurred.

Value	Name	Description
1	invalid_command	Command class or ID is unknown
2	packet_timeout	Command packet not completed in time
3	bad_length	Length value not supported, 250 bytes or less
4	parameter_length	Length of supplied parameters does not match with command definition
5	parameter_range	Value of supplied parameter(s) outside of valid range
6	not_implemented	Command known but not implemented in this firmware configuration

2 System class (ID = 1)

System commands and events relate to the core device, describing things like system boot and uptime, and verifying proper communication or resetting to an initial state.

2.1 Commands

2.1.1 system_ping `[ C0 00 01 01 ]`

Test communication with Keyglove device and get current uptime.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x00	length	No payload
2	0x01	class	Command class: "system"
3	0x01	id	Command ID: "ping"

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x04	length	Fixed-length payload (4)
2	0x01	class	Command class: "system"
3	0x01	id	Command ID: "ping"
4 - 7	**uint32_t**	**uptime**	**Number of seconds since last boot/reset**

2.1.1.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_system_ping()
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_system_ping())
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_system_ping(), timeout)
print("kg_rsp_system_ping: { uptime: %08X }" % (response['payload']['uptime']))
```

```
# create separate callback for response
def my_kg_rsp_system_ping(sender, args):
    print("kg_rsp_system_ping: { uptime: %08X }" % (args['uptime']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_system_ping += my_kg_rsp_system_ping
```

2.1.2 system_reset `[ C0 01 01 02 ... ]`

Reset Keyglove device.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x01	length	Fixed-length payload (1)
2	0x01	class	Command class: "system"
3	0x02	id	Command ID: "reset"
4	**uint8_t**	**mode**	**Type of reset to perform** • ***Enum:*** [system_reset_mode]

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x01	class	Command class: "system"
3	0x02	id	Command ID: "reset"
4 - 5	**uint16_t**	**result**	**Result code from 'reset' command**

2.1.2.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_system_reset(mode)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_system_reset(mode))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_system_reset(mode), timeout)
print("kg_rsp_system_reset: { result: %04X }" % (response['payload']['result']))
```

```
# create separate callback for response
def my_kg_rsp_system_reset(sender, args):
    print("kg_rsp_system_reset: { result: %04X }" % (args['result']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_system_reset += my_kg_rsp_system_reset
```

2.1.3 system_get_info `[ C0 00 01 03 ]`

Get firmware build info.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x00	length	No payload
2	0x01	class	Command class: "system"
3	0x03	id	Command ID: "get_info"

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description

0	0xC0	type	Response packet
1	0x0C	length	Fixed-length payload (12)
2	0x01	class	Command class: "system"
3	0x03	id	Command ID: "get_info"
4 - 5	**uint16_t**	**major**	**Firmware major version number**
6 - 7	**uint16_t**	**minor**	**Firmware minor version number**
8 - 9	**uint16_t**	**patch**	**Firmware patch version number**
10 - 11	**uint16_t**	**protocol**	**API protocol version number**
12 - 15	**uint32_t**	**timestamp**	**Build timestamp**

2.1.3.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_system_get_info()
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_system_get_info())
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_system_get_info(), timeout)
print("kg_rsp_system_get_info: { major: %04X, minor: %04X, patch: %04X, protocol: %04X," \
        " timestamp: %08X }" % (response['payload']['major'],' \
        ' response['payload']['minor'], response['payload']['patch'], \
        response['payload']['protocol'], response['payload']['timestamp']))
```

```
# create separate callback for response
def my_kg_rsp_system_get_info(sender, args):
    print("kg_rsp_system_get_info: { major: %04X, minor: %04X, patch: %04X, protocol:" \
            " %04X, timestamp: %08X }" % (args['major'], args['minor'], args['patch'],' \
            ' args['protocol'], args['timestamp']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_system_get_info += my_kg_rsp_system_get_info
```

2.1.4 system_get_capabilities [C0 01 01 04 ...]

Get capabilities designed into this unit.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x01	length	Fixed-length payload (1)
2	0x01	class	Command class: "system"
3	0x04	id	Command ID: "get_capabilities"
4	**uint8_t**	**category**	**Category of capabilities to report (0xFF for all)**

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet

1	0x02	length	Fixed-length payload (2)
2	0x01	class	Command class: "system"
3	0x04	id	Command ID: "get_capabilities"
4 - 5	**uint16_t**	**count**	**Number of capability reports to expect**

2.1.4.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_system_get_capabilities(category)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_system_get_capabilities(category))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_system_get_capabilities(category), \
        timeout)
print("kg_rsp_system_get_capabilities: { count: %04X }" % (response['payload']['count']))
```

```
# create separate callback for response
def my_kg_rsp_system_get_capabilities(sender, args):
    print("kg_rsp_system_get_capabilities: { count: %04X }" % (args['count']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_system_get_capabilities += my_kg_rsp_system_get_capabilities
```

2.1.5 system_get_memory `[ C0 00 01 05 ]`

Get system memory usage.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x00	length	No payload
2	0x01	class	Command class: "system"
3	0x05	id	Command ID: "get_memory"

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x08	length	Fixed-length payload (8)
2	0x01	class	Command class: "system"
3	0x05	id	Command ID: "get_memory"
4 - 7	**uint32_t**	**free_ram**	**Free bytes of RAM**
8 - 11	**uint32_t**	**total_ram**	**Total bytes of RAM**

2.1.5.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_system_get_memory()
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_system_get_memory())
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_system_get_memory(), timeout)
print("kg_rsp_system_get_memory: { free_ram: %08X, total_ram: %08X }" % \
        (response['payload']['free_ram'], response['payload']['total_ram']))
```

```
# create separate callback for response
def my_kg_rsp_system_get_memory(sender, args):
    print("kg_rsp_system_get_memory: { free_ram: %08X, total_ram: %08X }" % \
            (args['free_ram'], args['total_ram']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_system_get_memory += my_kg_rsp_system_get_memory
```

2.1.6 system_get_battery_status `[ C0 00 01 06 ]`

Get battery status (presence, charge status, charge level)

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x00	length	No payload
2	0x01	class	Command class: "system"
3	0x06	id	Command ID: "get_battery_status"

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x01	class	Command class: "system"
3	0x06	id	Command ID: "get_battery_status"
4	**uint8_t**	**status**	**Battery status**
5	**uint8_t**	**level**	**Charge level (0-100)**

2.1.6.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_system_get_battery_status()
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_system_get_battery_status())
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_system_get_battery_status(), \
        timeout)
print("kg_rsp_system_get_battery_status: { status: %02X, level: %02X }" % \
        (response['payload']['status'], response['payload']['level']))
```

```
# create separate callback for response
def my_kg_rsp_system_get_battery_status(sender, args):
    print("kg_rsp_system_get_battery_status: { status: %02X, level: %02X }" % \
            (args['status'], args['level']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_system_get_battery_status += my_kg_rsp_system_get_battery_status
```

2.1.7 system_set_timer `[ C0 04 01 07 ... ]`

Set a timer interval to trigger future behavior.

OUTGOING COMMAND PACKET STRUCTURE			
Byte	**Type**	**Name**	**Description**
0	0xC0	type	Command packet
1	0x04	length	Fixed-length payload (4)
2	0x01	class	Command class: "system"
3	0x07	id	Command ID: "set_timer"
4	**uint8_t**	**handle**	**Timer handle (0-7)**
5 - 6	**uint16_t**	**interval**	**Interval (10ms units)**
7	**uint8_t**	**oneshot**	**Repeating (0) or one-shot (1)**

INCOMING RESPONSE PACKET STRUCTURE			
Byte	**Type**	**Name**	**Description**
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x01	class	Command class: "system"
3	0x07	id	Command ID: "set_timer"
4 - 5	**uint16_t**	**result**	**Result code from 'set_timer' command**

2.1.7.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_system_set_timer(handle, interval, oneshot)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_system_set_timer(handle, interval, oneshot))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_system_set_timer(handle, interval, \
        oneshot), timeout)
print("kg_rsp_system_set_timer: { result: %04X }" % (response['payload']['result']))
```

```
# create separate callback for response
def my_kg_rsp_system_set_timer(sender, args):
    print("kg_rsp_system_set_timer: { result: %04X }" % (args['result']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_system_set_timer += my_kg_rsp_system_set_timer
```

2.2 Events

2.2.1 system_boot [80 0C 01 01 ...]

Indicates that Keyglove has started the boot process.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x0C	length	Fixed-length payload (12)
2	0x01	class	Event class: "system"
3	0x01	id	Event ID: "boot"
4 - 5	**uint16_t**	**major**	**Firmware major version number**
6 - 7	**uint16_t**	**minor**	**Firmware minor version number**
8 - 9	**uint16_t**	**patch**	**Firmware patch version number**
10 - 11	**uint16_t**	**protocol**	**API protocol version number**
12 - 15	**uint32_t**	**timestamp**	**Build timestamp**

2.2.1.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_system_boot(sender, args):
    print("kg_evt_system_boot: { major: %04X, minor: %04X, patch: %04X, protocol: %04X," \
            " timestamp: %08X }" % (args['major'], args['minor'], args['patch'],' \
            ' args['protocol'], args['timestamp']))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_system_boot += my_kg_evt_system_boot
```

2.2.2 system_ready [80 00 01 02]

Indicates that Keyglove has completed the boot process and is ready for use.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x00	length	No payload
2	0x01	class	Event class: "system"
3	0x02	id	Event ID: "ready"

2.2.2.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_system_ready(sender, args):
    print("kg_evt_system_ready: {  }")

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_system_ready += my_kg_evt_system_ready
```

2.2.3 system_error [80 02 01 03 ...]

Indicates that Keyglove has encountered an error (RAM, hardware, etc.) that will result in unintended behavior.

INCOMING EVENT PACKET STRUCTURE			
Byte	**Type**	**Name**	**Description**
0	0x80	type	Event packet
1	0x02	length	Fixed-length payload (2)
2	0x01	class	Event class: "system"
3	0x03	id	Event ID: "error"
4 - 5	**uint16_t**	**code**	**Error code describing what went wrong with the system** • ***Enum:*** **system_error_code**

2.2.3.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_system_error(sender, args):
    print("kg_evt_system_error: { code: %04X }" % (args['code']))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_system_error += my_kg_evt_system_error
```

2.2.4 system_capability `[ 80 03+ 01 04 ... ]`

Provides a record describing specific capabilities designed into this unit.

INCOMING EVENT PACKET STRUCTURE			
Byte	**Type**	**Name**	**Description**
0	0x80	type	Event packet
1	0x03+	length	Variable-length payload (3+)
2	0x01	class	Event class: "system"
3	0x04	id	Event ID: "capability"
4 - 5	**uint16_t**	**category**	**Category that this capability report is included in**
6	**uint8_t[]**	**record**	**Capability record(s) in type-length-value format**

2.2.4.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_system_capability(sender, args):
    print("kg_evt_system_capability: { category: %04X, record: %s }" % (args['category'], \
            ' '.join(['%02X' % b for b in args['record']])))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_system_capability += my_kg_evt_system_capability
```

2.2.5 system_battery_status `[ 80 02 01 05 ... ]`

Indicates that battery status has changed

INCOMING EVENT PACKET STRUCTURE			
Byte	**Type**	**Name**	**Description**
0	0x80	type	Event packet

1	0x02	length	Fixed-length payload (2)
2	0x01	class	Event class: "system"
3	0x05	id	Event ID: "battery_status"
4	**uint8_t**	**status**	**Battery status**
5	**uint8_t**	**level**	**Charge level (0-100)**

2.2.5.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_system_battery_status(sender, args):
    print("kg_evt_system_battery_status: { status: %02X, level: %02X }" % (args['status'], \
            args['level']))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_system_battery_status += my_kg_evt_system_battery_status
```

2.2.6 system_timer_tick [80 06 01 06 ...]

Indicates that a previously scheduled software timer has elapsed.

INCOMING EVENT PACKET STRUCTURE			
Byte	**Type**	**Name**	**Description**
0	0x80	type	Event packet
1	0x06	length	Fixed-length payload (6)
2	0x01	class	Event class: "system"
3	0x06	id	Event ID: "timer_tick"
4	**uint8_t**	**handle**	**Timer handle which triggered this event**
5 - 8	**uint32_t**	**seconds**	**Seconds elapsed since boot**
9	**uint8_t**	**subticks**	**10ms subticks above whole second**

2.2.6.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_system_timer_tick(sender, args):
    print("kg_evt_system_timer_tick: { handle: %02X, seconds: %08X, subticks: %02X }" % \
            (args['handle'], args['seconds'], args['subticks']))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_system_timer_tick += my_kg_evt_system_timer_tick
```

2.3 Enumerations

2.3.1 system_error_code

Describes the nature of a system error that has occurred.

Value	Name	Description
1	out_of_memory	Could not allocate required memory

2.3.1 system_reset_mode

Describes the type of reset to perform.

Value	Name	Description
1	normal	Reset Keyglove hardware and all peripherals (Bluetooth, sensors, etc.)
2	kgonly	Reset Keyglove hardware only, no peripherals

3 Touch class (ID = 2)

Touch commands and events control and report the behavior of the touch detection interface.

3.1 Commands

3.1.1 touch_get_mode `[ C0 00 02 01 ]`

Get the current touch mode.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x00	length	No payload
2	0x02	class	Command class: "touch"
3	0x01	id	Command ID: "get_mode"

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x01	length	Fixed-length payload (1)
2	0x02	class	Command class: "touch"
3	0x01	id	Command ID: "get_mode"
4	**uint8_t**	**mode**	**Current touch mode setting**

3.1.1.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_touch_get_mode()
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_touch_get_mode())
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_touch_get_mode(), timeout)
print("kg_rsp_touch_get_mode: { mode: %02X }" % (response['payload']['mode']))
```

```
# create separate callback for response
def my_kg_rsp_touch_get_mode(sender, args):
    print("kg_rsp_touch_get_mode: { mode: %02X }" % (args['mode']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_touch_get_mode += my_kg_rsp_touch_get_mode
```

3.1.2 touch_set_mode `[ C0 01 02 02 ... ]`

Set a new touch mode.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description

0	0xC0	type	Command packet
1	0x01	length	Fixed-length payload (1)
2	0x02	class	Command class: "touch"
3	0x02	id	Command ID: "set_mode"
4	**uint8_t**	**mode**	**New touch mode to set**

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x02	class	Command class: "touch"
3	0x02	id	Command ID: "set_mode"
4 - 5	**uint16_t**	**result**	**Result code from 'set_mode' command**

3.1.2.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_touch_set_mode(mode)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_touch_set_mode(mode))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_touch_set_mode(mode), timeout)
print("kg_rsp_touch_set_mode: { result: %04X }" % (response['payload']['result']))
```

```
# create separate callback for response
def my_kg_rsp_touch_set_mode(sender, args):
    print("kg_rsp_touch_set_mode: { result: %04X }" % (args['result']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_touch_set_mode += my_kg_rsp_touch_set_mode
```

3.2 Events

3.2.1 touch_mode [80 01 02 01 ...]

Indicates that the touch mode has changed.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x01	length	Fixed-length payload (1)
2	0x02	class	Event class: "touch"
3	0x01	id	Event ID: "mode"
4	**uint8_t**	**mode**	**New touch mode**

3.2.1.1 Example Usage (Python)

```python
# create callback for event
def my_kg_evt_touch_mode(sender, args):
    print("kg_evt_touch_mode: { mode: %02X }" % (args['mode']))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_touch_mode += my_kg_evt_touch_mode
```

3.2.2 touch_status `[ 80 01+ 02 02 ... ]`

Indicates that the touch sensor status has changed.

INCOMING EVENT PACKET STRUCTURE			
Byte	**Type**	**Name**	**Description**
0	0x80	type	Event packet
1	0x01+	length	Variable-length payload (1+)
2	0x02	class	Event class: "touch"
3	0x02	id	Event ID: "status"
4	**uint8_t[]**	**status**	**New touch status**

3.2.2.1 Example Usage (Python)

```python
# create callback for event
def my_kg_evt_touch_status(sender, args):
    print("kg_evt_touch_status: { status: %s }" % (' '.join(['%02X' % b for b in \
            args['status']])))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_touch_status += my_kg_evt_touch_status
```

4 Feedback class (ID = 3)

Feedback commands and events control and report on the various types of feedback subsystems, such as a simple LED or more complex devices such as RGB LEDs or piezo buzzers.

4.1 Commands

4.1.1 feedback_get_blink_mode `[ C0 00 03 01 ]`

Get current blink feedback mode.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x00	length	No payload
2	0x03	class	Command class: "feedback"
3	0x01	id	Command ID: "get_blink_mode"

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x01	length	Fixed-length payload (1)
2	0x03	class	Command class: "feedback"
3	0x01	id	Command ID: "get_blink_mode"
4	**uint8_t**	**mode**	**Current blink feedback mode**

4.1.1.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_feedback_get_blink_mode()
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_feedback_get_blink_mode())
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_feedback_get_blink_mode(), timeout)
print("kg_rsp_feedback_get_blink_mode: { mode: %02X }" % (response['payload']['mode']))
```

```
# create separate callback for response
def my_kg_rsp_feedback_get_blink_mode(sender, args):
    print("kg_rsp_feedback_get_blink_mode: { mode: %02X }" % (args['mode']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_feedback_get_blink_mode += my_kg_rsp_feedback_get_blink_mode
```

4.1.2 feedback_set_blink_mode `[ C0 01 03 02 ... ]`

Set new blink feedback mode.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x01	length	Fixed-length payload (1)
2	0x03	class	Command class: "feedback"
3	0x02	id	Command ID: "set_blink_mode"
4	**uint8_t**	**mode**	**New blink feedback mode to set**

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x03	class	Command class: "feedback"
3	0x02	id	Command ID: "set_blink_mode"
4 - 5	**uint16_t**	**result**	**Result code from command**

4.1.2.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_feedback_set_blink_mode(mode)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_feedback_set_blink_mode(mode))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_feedback_set_blink_mode(mode), \
        timeout)
print("kg_rsp_feedback_set_blink_mode: { result: %04X }" % (response['payload']['result']))
```

```
# create separate callback for response
def my_kg_rsp_feedback_set_blink_mode(sender, args):
    print("kg_rsp_feedback_set_blink_mode: { result: %04X }" % (args['result']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_feedback_set_blink_mode += my_kg_rsp_feedback_set_blink_mode
```

4.1.3 feedback_get_piezo_mode `[ C0 01 03 03 ... ]`

Get current feedback mode for a piezo buzzer.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x01	length	Fixed-length payload (1)
2	0x03	class	Command class: "feedback"
3	0x03	id	Command ID: "get_piezo_mode"
4	**uint8_t**	**index**	**Index of piezo device for which to get the current mode**

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x04	length	Fixed-length payload (4)
2	0x03	class	Command class: "feedback"
3	0x03	id	Command ID: "get_piezo_mode"
4	**uint8_t**	**mode**	**Current feedback mode for specified piezo device**
5	**uint8_t**	**duration**	**Duration to maintain tone**
6 - 7	**uint16_t**	**frequency**	**Frequency of tone to generate**

4.1.3.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_feedback_get_piezo_mode(index)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_feedback_get_piezo_mode(index))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_feedback_get_piezo_mode(index), \
        timeout)
print("kg_rsp_feedback_get_piezo_mode: { mode: %02X, duration: %02X, frequency: %04X }" % \
        (response['payload']['mode'], response['payload']['duration'], \
        response['payload']['frequency']))
```

```
# create separate callback for response
def my_kg_rsp_feedback_get_piezo_mode(sender, args):
    print("kg_rsp_feedback_get_piezo_mode: { mode: %02X, duration: %02X, frequency: %04X" \
            " }" % (args['mode'], args['duration'], args['frequency']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_feedback_get_piezo_mode += my_kg_rsp_feedback_get_piezo_mode
```

4.1.4 feedback_set_piezo_mode `[ C0 05 03 04 ... ]`

Set a new piezo feedback mode for a piezo buzzer.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x05	length	Fixed-length payload (5)
2	0x03	class	Command class: "feedback"
3	0x04	id	Command ID: "set_piezo_mode"
4	**uint8_t**	**index**	**Index of piezo device for which to set a new mode**
5	**uint8_t**	**mode**	**New feedback mode to set for specified piezo device**
6	**uint8_t**	**duration**	**Duration to maintain tone**
7 - 8	**uint16_t**	**frequency**	**Frequency of tone to generate**

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x03	class	Command class: "feedback"
3	0x04	id	Command ID: "set_piezo_mode"
4 - 5	**uint16_t**	**result**	**Result code from command**

4.1.4.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_feedback_set_piezo_mode(index, mode, duration, frequency)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_feedback_set_piezo_mode(index, mode, duration, \
        frequency))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_feedback_set_piezo_mode(index, \
        mode, duration, frequency), timeout)
print("kg_rsp_feedback_set_piezo_mode: { result: %04X }" % (response['payload']['result']))
```

```
# create separate callback for response
def my_kg_rsp_feedback_set_piezo_mode(sender, args):
    print("kg_rsp_feedback_set_piezo_mode: { result: %04X }" % (args['result']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_feedback_set_piezo_mode += my_kg_rsp_feedback_set_piezo_mode
```

4.1.5 feedback_get_vibrate_mode `[ C0 01 03 05 ... ]`

Get current feedback mode for a vibration motor.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x01	length	Fixed-length payload (1)
2	0x03	class	Command class: "feedback"
3	0x05	id	Command ID: "get_vibrate_mode"
4	**uint8_t**	**index**	**Index of vibration device for which to get the current mode**

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x03	class	Command class: "feedback"
3	0x05	id	Command ID: "get_vibrate_mode"
4	**uint8_t**	**mode**	**Current feedback mode for specified vibration device**
5	**uint8_t**	**duration**	**Duration to maintain vibration**

4.1.5.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_feedback_get_vibrate_mode(index)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_feedback_get_vibrate_mode(index))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_feedback_get_vibrate_mode(index), \
        timeout)
print("kg_rsp_feedback_get_vibrate_mode: { mode: %02X, duration: %02X }" % \
        (response['payload']['mode'], response['payload']['duration']))
```

```
# create separate callback for response
def my_kg_rsp_feedback_get_vibrate_mode(sender, args):
    print("kg_rsp_feedback_get_vibrate_mode: { mode: %02X, duration: %02X }" % \
            (args['mode'], args['duration']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_feedback_get_vibrate_mode += my_kg_rsp_feedback_get_vibrate_mode
```

4.1.6 feedback_set_vibrate_mode [C0 03 03 06 ...]

Set a new vibration motor feedback mode.

OUTGOING COMMAND PACKET STRUCTURE			
Byte	**Type**	**Name**	**Description**
0	0xC0	type	Command packet
1	0x03	length	Fixed-length payload (3)
2	0x03	class	Command class: "feedback"
3	0x06	id	Command ID: "set_vibrate_mode"
4	**uint8_t**	**index**	**Index of vibration device for which to set a new mode**
5	**uint8_t**	**mode**	**New feedback mode to set for specified vibration device**
6	**uint8_t**	**duration**	**Duration to maintain vibration**

INCOMING RESPONSE PACKET STRUCTURE			
Byte	**Type**	**Name**	**Description**
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x03	class	Command class: "feedback"
3	0x06	id	Command ID: "set_vibrate_mode"
4 - 5	**uint16_t**	**result**	**Result code from command**

4.1.6.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_feedback_set_vibrate_mode(index, mode, duration)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_feedback_set_vibrate_mode(index, mode, duration))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_feedback_set_vibrate_mode(index, \
        mode, duration), timeout)
print("kg_rsp_feedback_set_vibrate_mode: { result: %04X }" % \
        (response['payload']['result']))
```

```
# create separate callback for response
def my_kg_rsp_feedback_set_vibrate_mode(sender, args):
    print("kg_rsp_feedback_set_vibrate_mode: { result: %04X }" % (args['result']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_feedback_set_vibrate_mode += my_kg_rsp_feedback_set_vibrate_mode
```

4.1.7 feedback_get_rgb_mode `[ C0 01 03 07 ... ]`

Get current feedback mode for an RGB LED.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x01	length	Fixed-length payload (1)
2	0x03	class	Command class: "feedback"
3	0x07	id	Command ID: "get_rgb_mode"
4	**uint8_t**	**index**	**Index of RGB device for which to get the current mode**

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x03	length	Fixed-length payload (3)
2	0x03	class	Command class: "feedback"
3	0x07	id	Command ID: "get_rgb_mode"
4	**uint8_t**	**mode_red**	**Current feedback mode for specified RGB device red LED**
5	**uint8_t**	**mode_green**	**Current feedback mode for specified RGB device green LED**
6	**uint8_t**	**mode_blue**	**Current feedback mode for specified RGB device blue LED**

4.1.7.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_feedback_get_rgb_mode(index)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_feedback_get_rgb_mode(index))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_feedback_get_rgb_mode(index), \
        timeout)
```

```
print("kg_rsp_feedback_get_rgb_mode: { mode_red: %02X, mode_green: %02X, mode_blue: %02X" \
        " }" % (response['payload']['mode_red'], response['payload']['mode_green'], \
        response['payload']['mode_blue']))
```

```
# create separate callback for response
def my_kg_rsp_feedback_get_rgb_mode(sender, args):
    print("kg_rsp_feedback_get_rgb_mode: { mode_red: %02X, mode_green: %02X, mode_blue:" \
            " %02X }" % (args['mode_red'], args['mode_green'], args['mode_blue']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_feedback_get_rgb_mode += my_kg_rsp_feedback_get_rgb_mode
```

4.1.8 feedback_set_rgb_mode [C0 04 03 08 ...]

Set a new RGB LED feedback mode.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x04	length	Fixed-length payload (4)
2	0x03	class	Command class: "feedback"
3	0x08	id	Command ID: "set_rgb_mode"
4	**uint8_t**	**index**	**Index of RGB device for which to set a new mode**
5	**uint8_t**	**mode_red**	**New feedback mode to set for specified RGB device red LED**
6	**uint8_t**	**mode_green**	**New feedback mode to set for specified RGB device green LED**
7	**uint8_t**	**mode_blue**	**New feedback mode to set for specified RGB device blue LED**

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x03	class	Command class: "feedback"
3	0x08	id	Command ID: "set_rgb_mode"
4 - 5	**uint16_t**	**result**	**Result code from command**

4.1.8.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_feedback_set_rgb_mode(index, mode_red, mode_green, mode_blue)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_feedback_set_rgb_mode(index, mode_red, \
        mode_green, mode_blue))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_feedback_set_rgb_mode(index, \
        mode_red, mode_green, mode_blue), timeout)
print("kg_rsp_feedback_set_rgb_mode: { result: %04X }" % (response['payload']['result']))
```

```
# create separate callback for response
def my_kg_rsp_feedback_set_rgb_mode(sender, args):
    print("kg_rsp_feedback_set_rgb_mode: { result: %04X }" % (args['result']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_feedback_set_rgb_mode += my_kg_rsp_feedback_set_rgb_mode
```

4.2 Events

4.2.1 feedback_blink_mode `[ 80 01 03 01 ... ]`

Indicates that the blink feedback mode has changed.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x01	length	Fixed-length payload (1)
2	0x03	class	Event class: "feedback"
3	0x01	id	Event ID: "blink_mode"
4	**uint8_t**	**mode**	**New blink feedback mode**

4.2.1.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_feedback_blink_mode(sender, args):
    print("kg_evt_feedback_blink_mode: { mode: %02X }" % (args['mode']))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_feedback_blink_mode += my_kg_evt_feedback_blink_mode
```

4.2.2 feedback_piezo_mode `[ 80 05 03 02 ... ]`

Indicates that a piezo buzzer feedback mode has changed.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x05	length	Fixed-length payload (5)
2	0x03	class	Event class: "feedback"
3	0x02	id	Event ID: "piezo_mode"
4	**uint8_t**	**index**	**Piezo feedback device index**
5	**uint8_t**	**mode**	**New piezo feedback mode for indicated piezo device**
6	**uint8_t**	**duration**	**Duration to maintain tone**
7 - 8	**uint16_t**	**frequency**	**Frequency of tone to generate**

4.2.2.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_feedback_piezo_mode(sender, args):
    print("kg_evt_feedback_piezo_mode: { index: %02X, mode: %02X, duration: %02X," \
            " frequency: %04X }" % (args['index'], args['mode'], args['duration'],' \
            ' args['frequency']))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_feedback_piezo_mode += my_kg_evt_feedback_piezo_mode
```

4.2.3 feedback_vibrate_mode `[ 80 03 03 03 ... ]`

Indicates that a vibration feedback mode has changed.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x03	length	Fixed-length payload (3)
2	0x03	class	Event class: "feedback"
3	0x03	id	Event ID: "vibrate_mode"
4	**uint8_t**	**index**	**Vibration feedback device index**
5	**uint8_t**	**mode**	**New vibration feedback mode for indicated vibration device**
6	**uint8_t**	**duration**	**Duration to maintain vibration**

4.2.3.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_feedback_vibrate_mode(sender, args):
    print("kg_evt_feedback_vibrate_mode: { index: %02X, mode: %02X, duration: %02X }" % \
            (args['index'], args['mode'], args['duration']))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_feedback_vibrate_mode += my_kg_evt_feedback_vibrate_mode
```

4.2.4 feedback_rgb_mode `[ 80 04 03 04 ... ]`

Indicates that an RGB LED feedback mode has changed.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x04	length	Fixed-length payload (4)
2	0x03	class	Event class: "feedback"
3	0x04	id	Event ID: "rgb_mode"
4	**uint8_t**	**index**	**RGB feedback device index**
5	**uint8_t**	**mode_red**	**New feedback mode for indicated RGB device red LED**
6	**uint8_t**	**mode_green**	**New feedback mode for indicated RGB device green LED**
7	**uint8_t**	**mode_blue**	**New feedback mode for indicated RGB device blue LED**

4.2.4.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_feedback_rgb_mode(sender, args):
```

```python
    print("kg_evt_feedback_rgb_mode: { index: %02X, mode_red: %02X, mode_green: %02X," \
            " mode_blue: %02X }" % (args['index'], args['mode_red'], args['mode_green'],' \
            ' args['mode_blue']))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_feedback_rgb_mode += my_kg_evt_feedback_rgb_mode
```

5 Motion class (ID = 4)

Motion commands and events allow the control and detection of various motion sensors in the design.

5.1 Commands

5.1.1 motion_get_mode [C0 01 04 01 ...]

Get current mode for specified motion sensor.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x01	length	Fixed-length payload (1)
2	0x04	class	Command class: "motion"
3	0x01	id	Command ID: "get_mode"
4	**uint8_t**	**index**	**Index of motion sensor for which to get the current mode**

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x01	length	Fixed-length payload (1)
2	0x04	class	Command class: "motion"
3	0x01	id	Command ID: "get_mode"
4	**uint8_t**	**mode**	**Current motion sensor mode**

5.1.1.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_motion_get_mode(index)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_motion_get_mode(index))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_motion_get_mode(index), timeout)
print("kg_rsp_motion_get_mode: { mode: %02X }" % (response['payload']['mode']))
```

```
# create separate callback for response
def my_kg_rsp_motion_get_mode(sender, args):
    print("kg_rsp_motion_get_mode: { mode: %02X }" % (args['mode']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_motion_get_mode += my_kg_rsp_motion_get_mode
```

5.1.2 motion_set_mode [C0 02 04 02 ...]

Set new mode for specified motion sensor.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x02	length	Fixed-length payload (2)
2	0x04	class	Command class: "motion"
3	0x02	id	Command ID: "set_mode"
4	**uint8_t**	**index**	**Index of motion sensor for which to get the current mode**
5	**uint8_t**	**mode**	**New motion sensor mode to set**

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x04	class	Command class: "motion"
3	0x02	id	Command ID: "set_mode"
4 - 5	**uint16_t**	**result**	**Result code from command**

5.1.2.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_motion_set_mode(index, mode)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_motion_set_mode(index, mode))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_motion_set_mode(index, mode), \
        timeout)
print("kg_rsp_motion_set_mode: { result: %04X }" % (response['payload']['result']))
```

```
# create separate callback for response
def my_kg_rsp_motion_set_mode(sender, args):
    print("kg_rsp_motion_set_mode: { result: %04X }" % (args['result']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_motion_set_mode += my_kg_rsp_motion_set_mode
```

5.2 Events

5.2.1 motion_mode `[ 80 02 04 01 ... ]`

Indicates that a motion sensor's mode has changed.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x02	length	Fixed-length payload (2)
2	0x04	class	Event class: "motion"

3	0x01	id	Event ID: "mode"
4	**uint8_t**	**index**	**Affected motion sensor**
5	**uint8_t**	**mode**	**New motion sensor mode**

5.2.1.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_motion_mode(sender, args):
    print("kg_evt_motion_mode: { index: %02X, mode: %02X }" % (args['index'], args['mode']))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_motion_mode += my_kg_evt_motion_mode
```

5.2.2 motion_data [80 03+ 04 02 ...]

Indicates that a motion sensor's measurement data has been updated.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x03+	length	Variable-length payload (3+)
2	0x04	class	Event class: "motion"
3	0x02	id	Event ID: "data"
4	**uint8_t**	**index**	**Relevant motion sensor**
5	**uint8_t**	**flags**	**Flags indicating which measurement data is represented**
6	**uint8_t[]**	**data**	**New measurement data**

5.2.2.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_motion_data(sender, args):
    print("kg_evt_motion_data: { index: %02X, flags: %02X, data: %s }" % (args['index'], \
            args['flags'], ' '.join(['%02X' % b for b in args['data']])))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_motion_data += my_kg_evt_motion_data
```

5.2.3 motion_state [80 02 04 03 ...]

Motion state change detected, such as 'still' or 'moving'.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x02	length	Fixed-length payload (2)
2	0x04	class	Event class: "motion"
3	0x03	id	Event ID: "state"
4	**uint8_t**	**index**	**Relevant motion sensor**
5	**uint8_t**	**state**	**Type of motion state detected**

5.2.3.1 Example Usage (Python)

```python
# create callback for event
def my_kg_evt_motion_state(sender, args):
    print("kg_evt_motion_state: { index: %02X, state: %02X }" % (args['index'], \
            args['state']))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_motion_state += my_kg_evt_motion_state
```

6 Bluetooth class (ID = 8)

Bluetooth commands and events control and report on the wireless functionality.

6.1 Commands

6.1.1 bluetooth_get_mode [C0 00 08 01]

Get current mode for Bluetooth subsystem.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x00	length	No payload
2	0x08	class	Command class: "bluetooth"
3	0x01	id	Command ID: "get_mode"

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x03	length	Fixed-length payload (3)
2	0x08	class	Command class: "bluetooth"
3	0x01	id	Command ID: "get_mode"
4 - 5	**uint16_t**	**result**	**Result code from command**
6	**uint8_t**	**mode**	**Current Bluetooth mode**

6.1.1.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_bluetooth_get_mode()
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_bluetooth_get_mode())
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_bluetooth_get_mode(), timeout)
print("kg_rsp_bluetooth_get_mode: { result: %04X, mode: %02X }" % \
        (response['payload']['result'], response['payload']['mode']))
```

```
# create separate callback for response
def my_kg_rsp_bluetooth_get_mode(sender, args):
    print("kg_rsp_bluetooth_get_mode: { result: %04X, mode: %02X }" % (args['result'], \
            args['mode']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_bluetooth_get_mode += my_kg_rsp_bluetooth_get_mode
```

6.1.2 bluetooth_set_mode [C0 01 08 02 ...]

Set new mode for Bluetooth subsystem.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x01	length	Fixed-length payload (1)
2	0x08	class	Command class: "bluetooth"
3	0x02	id	Command ID: "set_mode"
4	**uint8_t**	**mode**	**New Bluetooth mode to set**

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x08	class	Command class: "bluetooth"
3	0x02	id	Command ID: "set_mode"
4 - 5	**uint16_t**	**result**	**Result code from command**

6.1.2.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_bluetooth_set_mode(mode)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_bluetooth_set_mode(mode))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_bluetooth_set_mode(mode), timeout)
print("kg_rsp_bluetooth_set_mode: { result: %04X }" % (response['payload']['result']))
```

```
# create separate callback for response
def my_kg_rsp_bluetooth_set_mode(sender, args):
    print("kg_rsp_bluetooth_set_mode: { result: %04X }" % (args['result']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_bluetooth_set_mode += my_kg_rsp_bluetooth_set_mode
```

6.1.3 bluetooth_reset `[ C0 00 08 03 ]`

Reset Bluetooth subsystem.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x00	length	No payload
2	0x08	class	Command class: "bluetooth"
3	0x03	id	Command ID: "reset"

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x08	class	Command class: "bluetooth"
3	0x03	id	Command ID: "reset"
4 - 5	**uint16_t**	**result**	**Result code from command**

6.1.3.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_bluetooth_reset()
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_bluetooth_reset())
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_bluetooth_reset(), timeout)
print("kg_rsp_bluetooth_reset: { result: %04X }" % (response['payload']['result']))
```

```
# create separate callback for response
def my_kg_rsp_bluetooth_reset(sender, args):
    print("kg_rsp_bluetooth_reset: { result: %04X }" % (args['result']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_bluetooth_reset += my_kg_rsp_bluetooth_reset
```

6.1.4 bluetooth_get_mac `[ C0 00 08 04 ]`

Get local Bluetooth MAC address.

OUTGOING COMMAND PACKET STRUCTURE			
Byte	**Type**	**Name**	**Description**
0	0xC0	type	Command packet
1	0x00	length	No payload
2	0x08	class	Command class: "bluetooth"
3	0x04	id	Command ID: "get_mac"

INCOMING RESPONSE PACKET STRUCTURE			
Byte	**Type**	**Name**	**Description**
0	0xC0	type	Response packet
1	0x08	length	Fixed-length payload (8)
2	0x08	class	Command class: "bluetooth"
3	0x04	id	Command ID: "get_mac"
4 - 5	**uint16_t**	**result**	**Result code from command**
6 - 11	**macaddr_t**	**address**	**Local six-byte Bluetooth MAC address**

6.1.4.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_bluetooth_get_mac()
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_bluetooth_get_mac())
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_bluetooth_get_mac(), timeout)
print("kg_rsp_bluetooth_get_mac: { result: %04X, address: %s }" % \
        (response['payload']['result'], ' '.join(['%02X' % b for b in' \
        ' response['payload']['address']])))
```

```
# create separate callback for response
def my_kg_rsp_bluetooth_get_mac(sender, args):
    print("kg_rsp_bluetooth_get_mac: { result: %04X, address: %s }" % (args['result'], '' \
            ' '.join(['%02X' % b for b in args['address']])))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_bluetooth_get_mac += my_kg_rsp_bluetooth_get_mac
```

6.1.5 bluetooth_get_pairings [C0 00 08 05]

Get a list of all paired devices. The response will be followed by one 'bluetooth_pairing_status' event for each existing pairing entry.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x00	length	No payload
2	0x08	class	Command class: "bluetooth"
3	0x05	id	Command ID: "get_pairings"

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x03	length	Fixed-length payload (3)
2	0x08	class	Command class: "bluetooth"
3	0x05	id	Command ID: "get_pairings"
4 - 5	**uint16_t**	**result**	**Result code from command**
6	**uint8_t**	**count**	**Number of paired devices**

6.1.5.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_bluetooth_get_pairings()
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_bluetooth_get_pairings())
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_bluetooth_get_pairings(), timeout)
```

```
print("kg_rsp_bluetooth_get_pairings: { result: %04X, count: %02X }" % \
        (response['payload']['result'], response['payload']['count']))
```

```
# create separate callback for response
def my_kg_rsp_bluetooth_get_pairings(sender, args):
    print("kg_rsp_bluetooth_get_pairings: { result: %04X, count: %02X }" % \
            (args['result'], args['count']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_bluetooth_get_pairings += my_kg_rsp_bluetooth_get_pairings
```

6.1.6 bluetooth_discover [C0 01 08 06 ...]

Perform Bluetooth inquiry to locate nearby devices. The response will be followed by one 'bluetooth_inquiry_response' event for each device that is discovered during the inquiry. Once the inquiry is finished, the 'bluetooth_inquiry_complete' event will occur.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x01	length	Fixed-length payload (1)
2	0x08	class	Command class: "bluetooth"
3	0x06	id	Command ID: "discover"
4	**uint8_t**	**duration**	**Number of seconds to run discovery process**

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x08	class	Command class: "bluetooth"
3	0x06	id	Command ID: "discover"
4 - 5	**uint16_t**	**result**	**Result code from command**

6.1.6.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_bluetooth_discover(duration)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_bluetooth_discover(duration))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_bluetooth_discover(duration), \
        timeout)
print("kg_rsp_bluetooth_discover: { result: %04X }" % (response['payload']['result']))
```

```
# create separate callback for response
def my_kg_rsp_bluetooth_discover(sender, args):
    print("kg_rsp_bluetooth_discover: { result: %04X }" % (args['result']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_bluetooth_discover += my_kg_rsp_bluetooth_discover
```

6.1.7 bluetooth_pair [C0 06 08 07 ...]

Initiate pairing request to remote device. The response will be followed by a 'bluetooth_pairing_status' event upon success, or a 'bluetooth_pairing_failed' event if unsuccessful.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x06	length	Fixed-length payload (6)
2	0x08	class	Command class: "bluetooth"
3	0x07	id	Command ID: "pair"
4 - 9	**macaddr_t**	**address**	**Six-byte Bluetooth MAC address of remote device to pair with**

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x08	class	Command class: "bluetooth"
3	0x07	id	Command ID: "pair"
4 - 5	**uint16_t**	**result**	**Result code from command**

6.1.7.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_bluetooth_pair(address)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_bluetooth_pair(address))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_bluetooth_pair(address), timeout)
print("kg_rsp_bluetooth_pair: { result: %04X }" % (response['payload']['result']))
```

```
# create separate callback for response
def my_kg_rsp_bluetooth_pair(sender, args):
    print("kg_rsp_bluetooth_pair: { result: %04X }" % (args['result']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_bluetooth_pair += my_kg_rsp_bluetooth_pair
```

6.1.8 bluetooth_delete_pairing [C0 01 08 08 ...]

Remove a specific pairing entry. Note that this will not actively close any Bluetooth connections to that device, if they are already open.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x01	length	Fixed-length payload (1)
2	0x08	class	Command class: "bluetooth"

3	0x08	id	Command ID: "delete_pairing"
4	**uint8_t**	**pairing**	**Index of pairing to delete**

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x08	class	Command class: "bluetooth"
3	0x08	id	Command ID: "delete_pairing"
4 - 5	**uint16_t**	**result**	**Result code from command**

6.1.8.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_bluetooth_delete_pairing(pairing)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_bluetooth_delete_pairing(pairing))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_bluetooth_delete_pairing(pairing), \
        timeout)
print("kg_rsp_bluetooth_delete_pairing: { result: %04X }" % (response['payload']['result']))
```

```
# create separate callback for response
def my_kg_rsp_bluetooth_delete_pairing(sender, args):
    print("kg_rsp_bluetooth_delete_pairing: { result: %04X }" % (args['result']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_bluetooth_delete_pairing += my_kg_rsp_bluetooth_delete_pairing
```

6.1.9 bluetooth_clear_pairings `[ C0 00 08 09 ]`

Remove all pairing entries. Note that this will not immediately close any Bluetooth connections.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x00	length	No payload
2	0x08	class	Command class: "bluetooth"
3	0x09	id	Command ID: "clear_pairings"

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x08	class	Command class: "bluetooth"
3	0x09	id	Command ID: "clear_pairings"

4 - 5	**uint16_t**	**result**	**Result code from command**

6.1.9.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_bluetooth_clear_pairings()
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_bluetooth_clear_pairings())
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_bluetooth_clear_pairings(), timeout)
print("kg_rsp_bluetooth_clear_pairings: { result: %04X }" % (response['payload']['result']))
```

```
# create separate callback for response
def my_kg_rsp_bluetooth_clear_pairings(sender, args):
    print("kg_rsp_bluetooth_clear_pairings: { result: %04X }" % (args['result']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_bluetooth_clear_pairings += my_kg_rsp_bluetooth_clear_pairings
```

6.1.10 bluetooth_get_connections `[ C0 00 08 0A ]`

Get a list of all open or pending connections. The response will be followed by one 'bluetooth_connection_status' event for each open or pending connection.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x00	length	No payload
2	0x08	class	Command class: "bluetooth"
3	0x0A	id	Command ID: "get_connections"

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x03	length	Fixed-length payload (3)
2	0x08	class	Command class: "bluetooth"
3	0x0A	id	Command ID: "get_connections"
4 - 5	**uint16_t**	**result**	**Result code from command**
6	**uint8_t**	**count**	**Number of open or pending connections**

6.1.10.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_bluetooth_get_connections()
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_bluetooth_get_connections())
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_bluetooth_get_connections(), \
        timeout)
print("kg_rsp_bluetooth_get_connections: { result: %04X, count: %02X }" % \
        (response['payload']['result'], response['payload']['count']))
```

```
# create separate callback for response
def my_kg_rsp_bluetooth_get_connections(sender, args):
    print("kg_rsp_bluetooth_get_connections: { result: %04X, count: %02X }" % \
            (args['result'], args['count']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_bluetooth_get_connections += my_kg_rsp_bluetooth_get_connections
```

6.1.11 bluetooth_connect [C0 02 08 0B ...]

Attempt to open a connection to a specific paired device using a specific profile. This will be followed by a 'bluetooth_connection_status' event once the handle has been allocated.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x02	length	Fixed-length payload (2)
2	0x08	class	Command class: "bluetooth"
3	0x0B	id	Command ID: "connect"
4	**uint8_t**	**pairing**	**Index of pairing to use**
5	**uint8_t**	**profile**	**Profile to use for connection**

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x08	class	Command class: "bluetooth"
3	0x0B	id	Command ID: "connect"
4 - 5	**uint16_t**	**result**	**Result code from command**

6.1.11.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_bluetooth_connect(pairing, profile)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_bluetooth_connect(pairing, profile))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_bluetooth_connect(pairing, \
        profile), timeout)
print("kg_rsp_bluetooth_connect: { result: %04X }" % (response['payload']['result']))
```

```
# create separate callback for response
def my_kg_rsp_bluetooth_connect(sender, args):
    print("kg_rsp_bluetooth_connect: { result: %04X }" % (args['result']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_bluetooth_connect += my_kg_rsp_bluetooth_connect
```

6.1.12 bluetooth_disconnect `[ C0 01 08 0C ... ]`

Close a specific Bluetooth connection.

OUTGOING COMMAND PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Command packet
1	0x01	length	Fixed-length payload (1)
2	0x08	class	Command class: "bluetooth"
3	0x0C	id	Command ID: "disconnect"
4	**uint8_t**	**handle**	**Link ID of connection to close**

INCOMING RESPONSE PACKET STRUCTURE

Byte	Type	Name	Description
0	0xC0	type	Response packet
1	0x02	length	Fixed-length payload (2)
2	0x08	class	Command class: "bluetooth"
3	0x0C	id	Command ID: "disconnect"
4 - 5	**uint16_t**	**result**	**Result code from command**

6.1.12.1 Example Usage (Python)

```
# generate command packet only
packet = kglib.kg_cmd_bluetooth_disconnect(handle)
```

```
# generate and send command
kglib.send_command(rxtx_obj, kglib.kg_cmd_bluetooth_disconnect(handle))
```

```
# send command and wait for captured response
response = kglib.send_and_return(rxtx_obj, kglib.kg_cmd_bluetooth_disconnect(handle), \
        timeout)
print("kg_rsp_bluetooth_disconnect: { result: %04X }" % (response['payload']['result']))
```

```
# create separate callback for response
def my_kg_rsp_bluetooth_disconnect(sender, args):
    print("kg_rsp_bluetooth_disconnect: { result: %04X }" % (args['result']))

# assign separate callback function to appropriate KGLib response handler collection
kglib.kg_rsp_bluetooth_disconnect += my_kg_rsp_bluetooth_disconnect
```

6.2 Events

6.2.1 bluetooth_mode `[ 80 01 08 01 ... ]`

Indicates that the Bluetooth mode has been changed.

INCOMING EVENT PACKET STRUCTURE			
Byte	**Type**	**Name**	**Description**
0	0x80	type	Event packet
1	0x01	length	Fixed-length payload (1)
2	0x08	class	Event class: "bluetooth"
3	0x01	id	Event ID: "mode"
4	**uint8_t**	**mode**	**New Bluetooth connectivity mode**

6.2.1.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_bluetooth_mode(sender, args):
    print("kg_evt_bluetooth_mode: { mode: %02X }" % (args['mode']))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_bluetooth_mode += my_kg_evt_bluetooth_mode
```

6.2.2 bluetooth_ready [80 00 08 02]

Indicates that the Bluetooth subsystem is ready for use.

INCOMING EVENT PACKET STRUCTURE			
Byte	**Type**	**Name**	**Description**
0	0x80	type	Event packet
1	0x00	length	No payload
2	0x08	class	Event class: "bluetooth"
3	0x02	id	Event ID: "ready"

6.2.2.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_bluetooth_ready(sender, args):
    print("kg_evt_bluetooth_ready: {  }")

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_bluetooth_ready += my_kg_evt_bluetooth_ready
```

6.2.3 bluetooth_inquiry_response [80 0D+ 08 03 ...]

Indicates that a new device has been paired.

INCOMING EVENT PACKET STRUCTURE			
Byte	**Type**	**Name**	**Description**
0	0x80	type	Event packet
1	0x0D+	length	Variable-length payload (13+)
2	0x08	class	Event class: "bluetooth"
3	0x03	id	Event ID: "inquiry_response"
4 - 9	**macaddr_t**	**address**	**Six-byte Bluetooth MAC address of remote device**
10 - 12	**btcod_t**	**cod**	**Three-byte Bluetooth Class-of-Device value**

13	**int8_t**	**rssi**	**RSSI value from discovered device**
14	**uint8_t**	**status**	**Status within inquiry process**
15	**uint8_t**	**pairing**	**Index of device in pairing list (0xFF if not paired)**
16	**uint8_t[]**	**name**	**Friendly name of remote device (if available)**

6.2.3.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_bluetooth_inquiry_response(sender, args):
    print("kg_evt_bluetooth_inquiry_response: { address: %s, cod: %s, rssi: %02X, status:" \
            " %02X, pairing: %02X, name: %s }" % (' '.join(['%02X' % b for b in' \
            ' args['address']]), ' '.join(['%02X' % b for b in args['cod']]),' \
            ' args['rssi'], args['status'], args['pairing'], ' '.join(['%02X' % b for b in \
            args['name']])))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_bluetooth_inquiry_response += my_kg_evt_bluetooth_inquiry_response
```

6.2.4 bluetooth_inquiry_complete `[ 80 01 08 04 ... ]`

Indicates that an ongoing Bluetooth discovery process has finished.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x01	length	Fixed-length payload (1)
2	0x08	class	Event class: "bluetooth"
3	0x04	id	Event ID: "inquiry_complete"
4	**uint8_t**	**count**	**Total number of devices found during discovery**

6.2.4.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_bluetooth_inquiry_complete(sender, args):
    print("kg_evt_bluetooth_inquiry_complete: { count: %02X }" % (args['count']))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_bluetooth_inquiry_complete += my_kg_evt_bluetooth_inquiry_complete
```

6.2.5 bluetooth_pairing_status `[ 80 0B+ 08 05 ... ]`

Provides a single pairing entry detailed status record.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x0B+	length	Variable-length payload (11+)
2	0x08	class	Event class: "bluetooth"
3	0x05	id	Event ID: "pairing_status"
4	**uint8_t**	**pairing**	**Index of device in pairing list**

5 - 10	**macaddr_t**	**address**	**Six-byte Bluetooth MAC address of remote device**
11	**uint8_t**	**priority**	**Auto-connection priority**
12	**uint8_t**	**profiles_supported**	**Bitmask of supported profiles**
13	**uint8_t**	**profiles_active**	**Bitmask of active profiles**
14	**uint8_t[]**	**handle_list**	**Handles for all active Bluetooth profile connections**

6.2.5.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_bluetooth_pairing_status(sender, args):
    print("kg_evt_bluetooth_pairing_status: { pairing: %02X, address: %s, priority: %02X," \
            " profiles_supported: %02X, profiles_active: %02X, handle_list: %s }" %" \
            " (args['pairing'], ' '.join(['%02X' % b for b in args['address']]),' \
            ' args['priority'], args['profiles_supported'], args['profiles_active'], ' \
            '.join(['%02X' % b for b in args['handle_list']])))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_bluetooth_pairing_status += my_kg_evt_bluetooth_pairing_status
```

6.2.6 bluetooth_pairing_failed [80 06 08 06 ...]

Indicates that a pending pair attempt has failed.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x06	length	Fixed-length payload (6)
2	0x08	class	Event class: "bluetooth"
3	0x06	id	Event ID: "pairing_failed"
4 - 9	**macaddr_t**	**address**	**Six-byte Bluetooth MAC address which failed pairing attempt**

6.2.6.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_bluetooth_pairing_failed(sender, args):
    print("kg_evt_bluetooth_pairing_failed: { address: %s }" % (' '.join(['%02X' % b for b \
            in args['address']])))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_bluetooth_pairing_failed += my_kg_evt_bluetooth_pairing_failed
```

6.2.7 bluetooth_pairings_cleared [80 00 08 07]

Indicates that all pairings have been removed.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x00	length	No payload
2	0x08	class	Event class: "bluetooth"
3	0x07	id	Event ID: "pairings_cleared"

6.2.7.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_bluetooth_pairings_cleared(sender, args):
    print("kg_evt_bluetooth_pairings_cleared: {  }")

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_bluetooth_pairings_cleared += my_kg_evt_bluetooth_pairings_cleared
```

6.2.8 bluetooth_connection_status [80 0A 08 08 ...]

Indicates that a paired device has connected.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x0A	length	Fixed-length payload (10)
2	0x08	class	Event class: "bluetooth"
3	0x08	id	Event ID: "connection_status"
4	**uint8_t**	**handle**	**Connection handle**
5 - 10	**macaddr_t**	**address**	**Six-byte Bluetooth MAC address of remote device**
11	**uint8_t**	**pairing**	**Index of device in pairing list**
12	**uint8_t**	**profile**	**Bluetooth profile used for this connection**
13	**uint8_t**	**status**	**Status**

6.2.8.1 Example Usage (Python)

```
# create callback for event
def my_kg_evt_bluetooth_connection_status(sender, args):
    print("kg_evt_bluetooth_connection_status: { handle: %02X, address: %s, pairing:" \
            " %02X, profile: %02X, status: %02X }" % (args['handle'], ' '.join(['%02X' %' \
            ' b for b in args['address']]), args['pairing'], args['profile'], \
            args['status']))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_bluetooth_connection_status += my_kg_evt_bluetooth_connection_status
```

6.2.9 bluetooth_connection_closed [80 03 08 09 ...]

Indicates that an active connection has been closed.

INCOMING EVENT PACKET STRUCTURE

Byte	Type	Name	Description
0	0x80	type	Event packet
1	0x03	length	Fixed-length payload (3)
2	0x08	class	Event class: "bluetooth"
3	0x09	id	Event ID: "connection_closed"
4	**uint8_t**	**handle**	**Connection handle**
5 - 6	**uint16_t**	**reason**	**Reason for connection closure**

6.2.9.1 Example Usage (Python)

```python
# create callback for event
def my_kg_evt_bluetooth_connection_closed(sender, args):
    print("kg_evt_bluetooth_connection_closed: { handle: %02X, reason: %04X }" % \
            (args['handle'], args['reason']))

# assign callback function to appropriate KGLib event handler collection
kglib.kg_evt_bluetooth_connection_closed += my_kg_evt_bluetooth_connection_closed
```