# RW HCI Software

Functional Specification

RW-HCI-SW-FS

Version 1.4

2015-02-16

RivieraWaves
http://www.rivierawaves.com/

# Revision History

| Version | Date | Revision Description | Author |
|---|---|---|---|
| 1.0 | Apr. 7th 2014 | Initial release | VLE |
| 1.1 | July, 29th 2014 | Added comments on error cases for HCI TX data (buffer overflow and length error) | VLE |
| 1.2 | August, 1st 2014 | Add support of BT dual mode | VLE |
| 1.3 | Sept. 1st 2014 | Add BT ACL data | VLE |
| 1.4 | Feb. 16st 2015 | Minor update on the destination identifiers table | VLE |
| | | | |
| | | | |
| | | | |
| | | | |

# Table of Content

## List of Figures

## List of Tables

# 1 Overview

## 1.1 Document Overview

This document describes the embedded Software implementation of the Host Controller Interface specified by Bluetooth SIG [1].

Its purpose is to explain how the HCI layer is implemented within RivieraWaves Bluetooth 4.1 IP.

This document requires the intended audience to have knowledge about the Bluetooth protocol stack, especially on the layers directly communicating via HCI layer:

- LM and LC for classic Bluetooth controller
- LL for Bluetooth Low Energy controller
- GAP and L2CAP on Host side

The document does not describe the format of all messages that can be exchanged through the HCI layer. For such information, please refer to BT standard specification [1] part II.E.7.

## 1.2 Context Overview

The HCI layer is part of Bluetooth 4.1 protocol stack shown below:

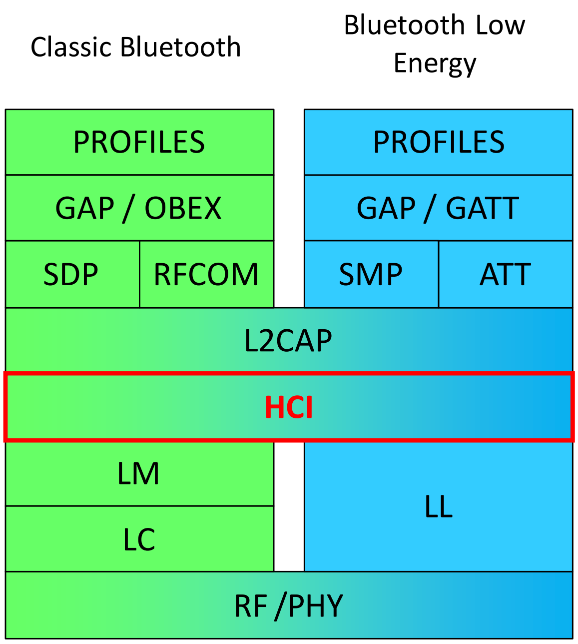


**Figure 1 – Bluetooth dual mode protocol stack**

The role of HCI is to provide a uniform interface method of accessing a Bluetooth Controller's capabilities from the Host.

## 1.3 Modes of operation

In practice, the HCI layer could be included in three kinds of system: a full stack system, a Host or a controller. The full stack system contains both Host and Controller layers. In this case, HCI role is to convey the information from one part to the other by following the rules defined in HCI standard. For a Host or Controller only system, HCI will need to interface with a Transport Layer that manages the reception and transmission of messages over a physical interface, such as USB or UART.

As shown on figure below, the main three configurations are supported by the HCI Software. It also supports one additional configuration where the lower layers of the full stack system can still be controlled by an external Host (for testing purposes).

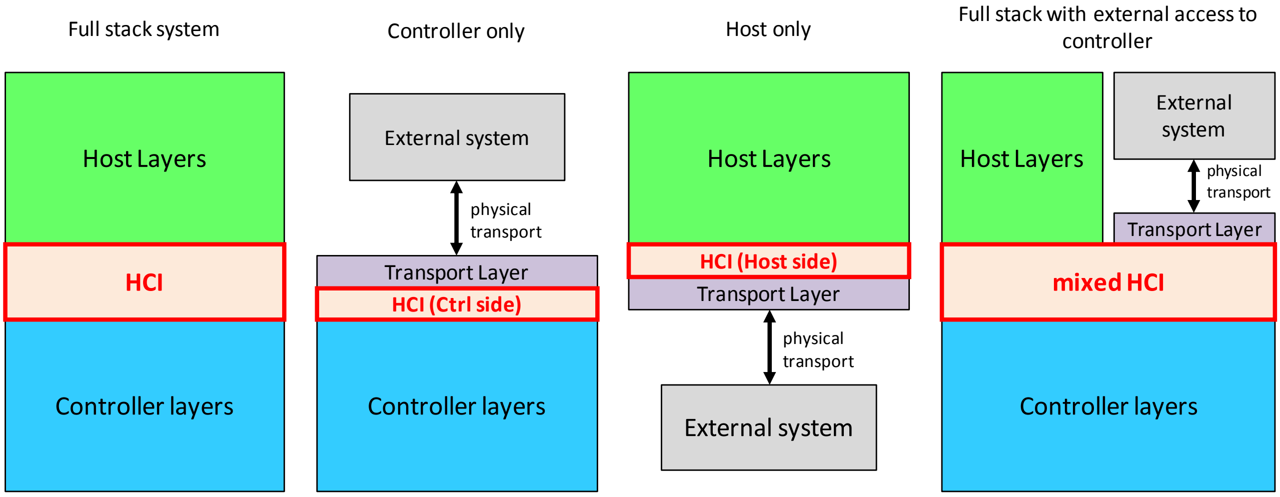


**Figure 2 – HCI working modes**

Even if the HCI Software originally supports the 4 working modes described above, the possible use of these modes is submitted to the possibilities offered by stack partitioning type:

- BLE single-mode IP: any the 4 working modes shown on the above figure
- BT single-mode IP: only the "Controller only" working mode supported
- BT Dual Mode IP: only the "Controller only" working mode supported

## 2 HCI Software architecture

The HCI Software is an interface communication block that could be used for 3 main purposes:

- communication between internal Host and external Controller
- communication between internal Controller and external Host
- communication between internal Controller and internal Host

It is interfaced to other software parts according to following figure:

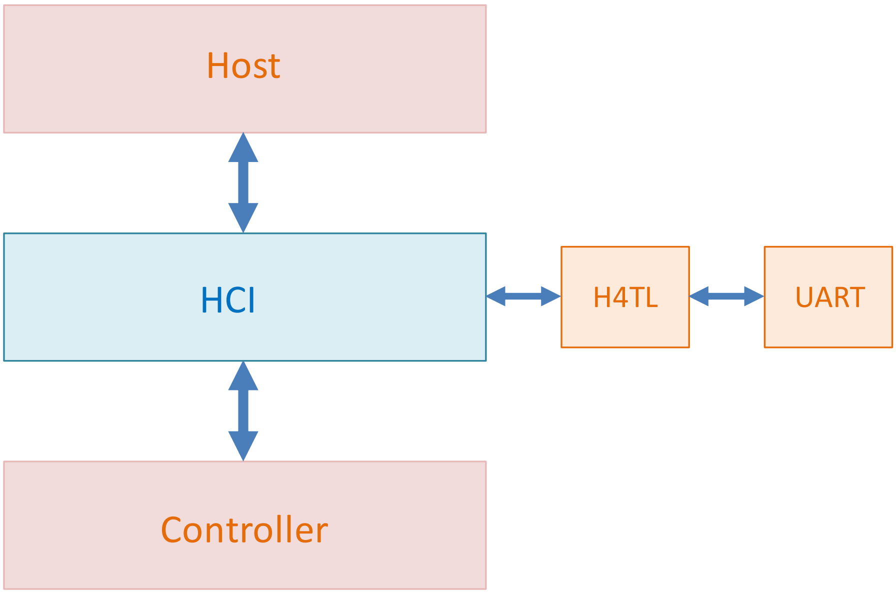


**Figure 3 – HCI Software interfaces**

During reception from external interface, HCI also manages the buffer allocation within the Kernel memory heap, so that, after unpacking, an internal Kernel message is ready for processing by an internal task.

The routing mechanism is when a message is requested to be transferred from somewhere in the stack or external interface. HCI either sends the Kernel message to an internal task of the other stack side (e.g. higher layers if the message comes from lower layers) or transfers the message through external interface.

The figure below gives a more detailed view on the HCI Software architecture:

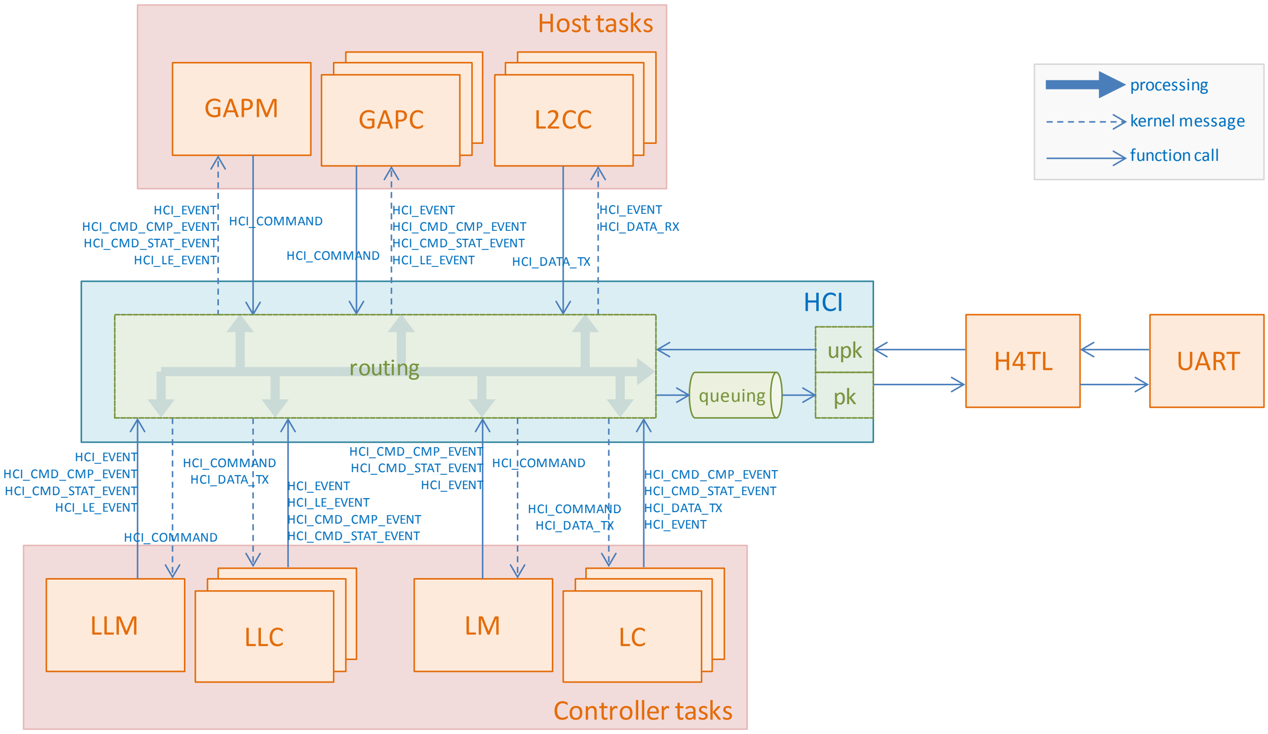


**Figure 4 – HCI Software architecture**

As described on the figure above, the HCI provides two main processing blocks: routing and external interface packet management. When both Host and Controller stack parts are present (Full-Stack mode), the external interface feature is optional and the routing system auto-detects if the lower layers are used by the internal or the external Host.

The first main challenge of HCI Software is to route the messages to/from internal task and to/from external interface. Several types of messages are used to carry the information. These messages could carry some basic control information for the BT/BLE operations and then will be conveyed to the main management tasks (LLM/GAPM). But the messages could also carry link dedicated information and then need to be conveyed to the link specific tasks (LLC/GAPC/L2CC).

Another challenge is to deal with the external interface, when the HCI messages are exchanged with an external Host or Controller. In that configuration, HCI has the responsibility to translate the message between HCI data format (HCI packets) and the format supported by the internal system (OS messages). HCI can convert control messages parameters (commands, events) between the HCI packet formats and the internal messages format, taking care about processor endianness, memory word alignment. This step will be referred as packing-unpacking in the rest of this document.

The huge number of control messages implicates that HCI Software defines descriptors tables. Purpose is to refer to its descriptor when processing each message. The data packets need a specific buffer allocation policy managed by the IP Software.

More details are given in the following chapters.

# 3 HCI control messages descriptors

## 3.1 Commands descriptors

As per BT standard specification [1] part II.E.5.4.1, each command is assigned a 2-bytes Opcode used to uniquely identify different types of commands. The Opcode parameter is divided into two fields, called the OpCode Group Field (OGF) and OpCode Command Field (OCF). The OGF occupies the upper 6 bits of the Opcode, while the OCF occupies the remaining 10 bits. The OGF of 0x3F is reserved for vendor-specific debug commands.

Each HCI command supported by RW IP is associated to a command descriptor. The command descriptor is a structure that contains complete information in order to:

- route the message or its response within internal stack tasks
- manipulate its parameters or return parameters when dealing with an external interface (only if external interface is supported)

| | Routing | | Parameters Packing/Unpacking | | | | |
|---|---|---|---|---|---|---|---|
| 2 bytes | 1 byte | | 1 byte | | | 4 bytes | 4 bytes |
| | 7:4 | 3:0 | 7:2 | 1 | 0 | | |
| Opcode | HL ID | LL ID | RFU | RetPar | Par | PARAM FORMAT | RET PAR FORMAT |

**Figure 5 – HCI commands descriptor format**

Command descriptor fields description:

| Field Name | Sub-field Name | Size | Description |
|---|---|---|---|
| Opcode | | | Command opcode |
| Destination ID | LL ID | 4 bits | Identifier of the task that receives the command |
| | HL ID | 4 bits | Identifier of the task that receives the response event (Command Complete or Command Status) |
| Special Pack Settings | Special Return Params packing | 1 bit | Flag indicating that the return parameters are packed/unpacked via a special function |
| | Special Params packing | 1 bit | Flag indicating that the parameters are packed/unpacked via a special function |
| | RFU | 6 bits | Reserved for Future Use |
| Parameters Format | | 4 bytes | String representing the parameters format (NULL if no parameter), used by the generic parameters unpacker.<br>In case of special parameters unpacking, this field points to the dedicated unpacker function. |
| Return Parameters Format | | 4 bytes | String representing the return parameters format (NULL if no parameter), used by the generic parameters packer.<br>In case of special return parameters packing, this field points to the dedicated packer function. |

**Table 1 – Command descriptor fields definition**

**Note**: It is important to notice that for standard commands, all fields used for parameters packing or unpacking rely directly to the BT standard specification.

**Note 2**: The fields for parameters packing or unpacking are present only if external interface is supported. In a Full-stack system that does not support external interface, only the routing fields are present in the command descriptors.

The command group (OGF) allows classifying the descriptors in separate tables.

For example, here are some examples of HCI commands descriptors within the Link Control commands group:

| OGF 1 | Opcode | HL ID | LL ID | RFU | SpU | SpP | PARAM FORMAT | RET PAR FORMAT |
|---|---|---|---|---|---|---|---|---|
| 0 | 0x0401 | N/A | LM | | N | N | "3BBB" | NULL |
| 1 | 0x0402 | N/A | LM | | N | N | NULL | "B" |
| 2 | 0x0403 | N/A | LM | | N | N | "HH3BBB" | "B" |
| … | … | | … | | … | … | … | … |
| N | 0x043E | N/A | LM | | N | N | "6BLL5B5BHHLL5B5B | NULL |

| Command | OCF | Command Parameters | Return Parameters |
|---|---|---|---|
| HCI_Inquiry | 0x0001 | LAP,<br>Inquiry_Length,<br>Num_Responses | |
| HCI_Inquiry_Cancel | 0x0002 | | Status |
| HCI_Periodic_<br>Inquiry_Mode | 0x0003 | Max_Period_Length,<br>Min_Period_Length,<br>LAP,<br>Inquiry_Length,<br>Num_Responses | Status |
| HCI_Enhanced_<br>Accept_<br>Synchronous_<br>Connection_<br>Request | 0x003E | BD_ADDR,<br>Transmit_Bandwidth,<br>Receive_Bandwidth,<br>Transmit_Coding_Format,<br>Receive_Coding_Format,<br>Transmit_Codec_Frame_Size,<br>Receive_Codec_Frame_Size,<br>Input_Bandwidth,<br>Output_Bandwidth,<br>Input_Coding_Format,<br>Output_Coding_Format,<br>Input_Coded_Data_Size,<br>Output_Coded_Data_Size,<br>Input_PCM_Data_Format,<br>Output_PCM_Data_Format,<br>Input_PCM_Sample_Payload_MSB_Position,<br>Output_PCM_Sample_Payload_MSB_Position, | |

**Figure 6 – Link Control group descriptors table**

Another example of HCI commands descriptors within the Controller and Baseband commands group:

| OGF 3 | OCF | LL ID | HL ID | RFU | SpU | SpP | PARAM FORMAT | RET PAR FORMAT |
|---|---|---|---|---|---|---|---|---|
| 0 | 0x0C01 | GAPM | LM/LLM | | N | N | "8B" | "B" |
| 1 | 0x0C02 | N/A | LM | | N | N | NULL | "B" |
| 2 | 0x0C03 | GAPM | LM/LLM | | Y | N | pointer to function | "B" |
| … | … | | … | | … | … | … | … |
| N | 0x0C6D | N/A | LM | | N | N | "BB" | "B" |

| Command | OCF | Command Parameters | Return Parameters |
|---|---|---|---|
| HCI_Set_Event_Mask | 0x0001 | Event_Mask | Status |
| HCI_Reset | 0x0003 | | Status |
| HCI_Set_Event_Filter | 0x0005 | Filter_Type,<br>Filter_Condition_Type,<br>Condition | Status |
| HCI_Write_LE_Host_Support | 0x006D | LE_Supported_Host,<br>Simultaneous_LE_Host | Status |

**Figure 7 – Controller and Baseband group descriptors table**

At any time, the HCI Software can get a descriptor associated to a command by a unique common table referencing all the groups present in the HCI Software:

| | OGF | Pointer to cmd desc tables |
|---|---|---|
| 0 | Link Control | desc_link_ctrl |
| 1 | Link Policy | desc_link_pol |
| 2 | Controller & Baseband | desc_ctrl_bb |
| 3 | Informational Parameters | desc_info_par |
| 4 | Status Parameters | desc_stat_par |
| 5 | Testing | desc_testing |
| 6 | LE Controller | desc_le_ctrl |
| … | … | … |
| N | Vendor Specific | desc_vendor_spec |

| OGF 1 | Opcode | HL ID | LL ID | RFU | SpU | SpP | PARAM FORMAT | RET PAR FORMAT |
|---|---|---|---|---|---|---|---|---|
| 0 | 0x0401 | N/A | LM | | N | N | "3BBB" | NULL |
| 1 | 0x0402 | N/A | LM | | N | N | NULL | "B" |
| 2 | 0x0403 | N/A | LM | | N | N | "HH3BBB" | "B" |
| … | … | | … | | … | … | … | … |
| N | 0x043E | N/A | LM | | N | N | "6BLL5B5BHHLL5B5B | NULL |

| OGF x | OCF | LL ID | HL ID | RFU | SpU | SpP | PARAM FORMAT | RET PAR FORMAT |
|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | |
| 1 | | | | | | | | |
| … | … | | … | | … | … | … | … |
| N | | | | | | | | |

| OGF 3 | OCF | LL ID | HL ID | RFU | SpU | SpP | PARAM FORMAT | RET PAR FORMAT |
|---|---|---|---|---|---|---|---|---|
| 0 | 0x0C01 | GAPM | LM/LLM | | N | N | "8B" | "B" |
| 1 | 0x0C02 | N/A | LM | | N | N | NULL | "B" |
| 2 | 0x0C03 | GAPM | LM/LLM | | Y | N | pointer to function | "B" |
| … | … | | … | | … | … | … | … |
| N | 0x0C6D | N/A | LM | | N | N | "BB" | "B" |

| OGF x | OCF | LL ID | HL ID | RFU | SpU | SpP | PARAM FORMAT | RET PAR FORMAT |
|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | |
| 1 | | | | | | | | |
| … | … | | … | | … | … | … | … |
| N | | | | | | | | |

**Figure 8 – Top level table pointing to group descriptors tables**

## 3.2 Events descriptors

As per BT standard specification [1] part II.E.5.4.4, each command is assigned a 1-byte event code used to uniquely identify any HCI event.

Each HCI event supported by RW IP is associated to an event descriptor. The event descriptor is a structure that contains complete information that could be used for:

- route the message within internal stack tasks
- manipulate its parameters when dealing with an external interface

| | Routing | Parameters packing | |
|---|---|---|---|
| 1 byte | 1 byte | 1 byte | 4 bytes |
| CODE | HL ID | SpP | PARAM FORMAT (ptr) |

**Figure 9 – HCI event descriptor format**

Event descriptor fields description:

| Name | Size | Description |
|---|---|---|
| Code | 1 byte | Event code or event subcode |
| HL ID | 1 byte | Identifier of the task that receives the event |
| Special Parameters Packing | 1 byte | Flag indicating that the parameters are packed/unpacked via a special function |
| Parameters Format | 4 bytes | String representing the parameters format (NULL if no parameter), used by the generic parameters packer |

**Table 2 – Event descriptor fields definition**

| | CODE | HL ID | SpP | PARAM FORMAT (ptr) |
|---|---|---|---|---|
| 0 | 0x01 | N/A | N | "B" |
| 1 | 0x02 | N/A | N | "B6BBBB3BH" |
| 2 | 0x03 | N/A | N | "BH6BBB" |
| 3 | 0x04 | N/A | N | "6B3BB" |
| … | … | … | … | … |
| N | 0x57 | N/A | N | "H" |

| Event | Event Code | Event Parameters |
|---|---|---|
| Inquiry Complete | 0x01 | Status |
| Inquiry Result | 0x02 | Num_Responses, BD_ADDR[i], Page_Scan_Repetition_Mode[i], Reserved[i], Reserved[i], Class_of_Device[i] Clock_Offset[i] |
| Connection Complete | 0x03 | Status, Connection_Handle, BD_ADDR, Link_Type, Encryption_Enabled |
| Connection Request | 0x04 | BD_ADDR, Class_of_Device, Link_Type |
| Authenticated Payload Timeout Expired | 0x57 | Connection_Handle |

**Figure 10 – Legacy events descriptors table**

BT standard defines a special set of events behind the event code “0x3E – LE Meta Event” (see BT standard specification [1] part II.E.7.7.65). All these sub-events are assigned with a subcode. The HCI Software assigns one event descriptor to each of these sub-events called LE events. As a result, a second table is present with all LE events described and indexed by their LE event subcode.

| | CODE | HL ID | SpP | PARAM FORMAT (ptr) |
|---|---|---|---|---|
| 0 | 0x01 | GAPM | N | "BBHBB6BHHHB" |
| 1 | 0x02 | GAPM | Y | pointer to function |
| 2 | 0x03 | GAPC | N | "BBHHHH" |
| … | … | … | … | … |
| N | 0x06 | GAPC | N | "BHHHHH" |

| Event | Event Code | Event parameters |
|---|---|---|
| LE Connection Complete | 0x3E | Subevent_Code, Status, Connection_Handle, Role, Peer_Address_Type, Peer_Address, Conn_Interval, Conn_Latency, Supervision_Timeout, Master_Clock_Accuracy |

| | | |
|---|---|---|
| LE Advertising Report | 0x3E | Subevent_Code, Num_Reports, Event_Type[i], Address_Type[i], Address[i], Length[i], Data[i], RSSI[i] |

| | | |
|---|---|---|
| LE Connection Update Complete | 0x3E | Subevent_Code, Status, Connection_Handle, Conn_Interval, Conn_Latency, Supervision_Timeout |

| | | |
|---|---|---|
| LE Remote Connection Parameter Request | 0x3E | Subevent_Code, Connection_Handle, Interval_Min, Interval_Max, Latency, Timeout |

**Figure 11 – LE events descriptors table**

<u>**Note**</u>: The fields for parameters packing or unpacking are present only if external interface is supported. In a Full-stack system that does not support external interface, only the routing fields are present in the event descriptors.

# 4 Internal messages definition

A Kernel message is a basic exchange element used by RW IP Software tasks to communicate to each other. The information of each HCI message is processed internally using a Kernel message.

However, the Kernel message carrying an HCI message is not sent directly between 2 internal tasks. The HCI Software can then reuse some of the fields normally reserved for Kernel use in order to organize an efficient routing and manipulation of the HCI messages

The following sections describe how HCI Software and the user Software blocks use the Kernel message to transfer HCI messages.

## 4.1 Command

All HCI commands are internally carried through a unique Kernel message filled with following data:

| MSG ID | DEST ID | SRC ID | MSG LENGTH | N + padding |
|---|---|---|---|---|
| CMD | Con Idx | Opcode | Param Length | PARAMS unpk |

**Figure 12 – Kernel message for carrying HCI commands**

Kernel message content:

| KE message field | Values |
|---|---|
| Message ID | HCI Command Message ID |
| Destination Task | Connection Index (only for connection oriented commands) |
| Source Task | Opcode |
| Parameters Length | Unpacked parameters length (0 for parameter-less commands) |
| Parameters | Unpacked parameters |

**Table 3 – HCI command Kernel message values**

Thanks to the information contained in, each task receiving such message can retrieve the information of the HCI command.

**Note:** each lower layer task possibly receiving HCI commands must implement one HCI command message handler as unique entry point. It is responsible to process and free the Kernel message. It is also responsible for replying each HCI command it receives as defined in BT standard specification [1] part II.E.7.

## 4.2 Events

The controller stack may send an event to Host at any moment. It sends a Kernel message that can be one of four types:

- command status event: in response to a procedure start
- command complete event: in response to a completed action
- LE event: message from BLE LL to Host
- Legacy event: message from BT/BLE LL to Host

### 4.2.1 Legacy event

The default container for HCI legacy events is a Kernel message filled with following data:

| MSG ID | DEST ID | SRC ID | MSG LENGTH | N + padding |
|---|---|---|---|---|
| EVT | Con Idx | Event Code | Param Length | PARAMS unpk |

**Figure 13 – Kernel message for carrying HCI events**

Kernel message content:

| KE message field | Values |
|---|---|
| Message ID | HCI Event Message ID |
| Destination Task | Connection Index (only for connection oriented events) |
| Source Task | Event Code |

| | |
|---|---|
| Parameters Length | Unpacked parameters length (0 for parameter-less events) |
| Parameters | Unpacked parameters |

**Table 4 – HCI event Kernel message values**

**Note:** each higher layer task possibly receiving HCI events must implement one HCI event message handler as unique entry point. It is responsible to process and free the Kernel message.

**Note 2:** all supported HCI events except HCI Command Complete, HCI Command Status, and HCI LE Meta events are carried through the default HCI event Kernel message. See following sections for the particular cases.

### 4.2.2 LE event

All HCI Low Energy Meta events are internally carried through a unique Kernel message filled with following data:

| MSG ID | DEST ID | SRC ID | MSG LENGTH | 1 | N + padding - 1 |
|---|---|---|---|---|---|
| LE EVT | Con Idx | - | Param Length | SUB | PARAMS unpk |

**Figure 14 – Kernel message for carrying HCI LE events**

Kernel message content:

| KE message field | Values |
|---|---|
| Message ID | HCI LE Event Message ID |
| Destination Task | Connection Index (only for connection oriented events) |
| Source Task | Not filled |
| Parameters Length | Unpacked parameters length (1 for parameter-less LE events) |
| Parameters | Unpacked parameters |

**Table 5 – HCI LE event Kernel message values**

**Note:** each higher layer task possibly receiving HCI LE events must implement one HCI LE event message handler as unique entry point. It is responsible to process and free the Kernel message.

### 4.2.3 Command Complete event

The HCI Command Complete event is internally carried through a Kernel message filled with following data:

| MSG ID | DEST ID | SRC ID | MSG LENGTH | N + padding |
|---|---|---|---|---|
| CC EVT | Con Idx | Opcode | Param Length | RET PARAMS unpk |

**Figure 15 – Kernel message for carrying HCI Command Complete events**

Kernel message content:

| KE message field | Values |
|---|---|
| Message ID | HCI CC Event Message ID |
| Destination Task | Connection Index (only for connection oriented events) |
| Source Task | Original Command Opcode |
| Parameters Length | Unpacked parameters length |
| Parameters | Unpacked parameters |

**Table 6 – HCI Command Complete event Kernel message values**

**Note:** each higher layer task possibly receiving HCI CC events must implement one HCI CC event message handler as unique entry point. It is responsible to process and free the Kernel message.

### 4.2.4 Command Status event

The HCI Command Status event is internally carried through a Kernel message filled with following data:

| MSG ID | DEST ID | SRC ID | MSG LENGTH | 1 |
|---|---|---|---|---|
| CS EVT | Con Idx | Opcode | 1 | STAT |

**Figure 16 – Kernel message for carrying HCI Command Status events**

Kernel message content:

| KE message field | Values |
|---|---|
| Message ID | HCI CS Event Message ID |
| Destination Task | Connection Index (only for connection oriented events) |
| Source Task | Original Command Opcode |
| Parameters Length | 1 (Length of the parameter Status, BT standard specification [1] part II.E.7.7.15) |
| Parameters | Status of the command processing |

**Table 7 – HCI Command Status event Kernel message values**

**Note:** each higher layer task possibly receiving HCI CS events must implement one HCI CS event message handler as unique entry point. It is responsible to process and free the Kernel message.

## 4.3 LE ACL RX Data

The information related to HCI LE ACL RX Data (received from the peer device on BLE link) is carried through a unique message filled with following data:

| MSG ID | DEST ID | SRC ID | MSG LENGTH | 2 | 1 | 1 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|
| ACL DATA RX | Con Idx | - | LEN | CONHDL | F | Res | LEN | HDL |

**Figure 17 – Kernel message for carrying HCI LE ACL RX Data information**

Kernel message content:

| KE message field | Values |
|---|---|
| Message ID | HCI ACL RX Data Message ID |
| Destination Task | Connection Index |
| Source Task | Not filled |
| Parameters Length | Length of the parameters |
| Parameters | |
| | Connection handle |
| | Packet boundary and packet broadcast flags |
| | Reserved |
| | Data Length |
| | Handle of the RX buffer containing the data |

**Table 8 – HCI LE ACL RX data Kernel message values**

**Note:** the Kernel message carries some information related ACL data packet, not the data itself. The data itself is stored in specific buffers managed by a dedicated Software block. For more details, see RW-BLE SW specification [2] part 6.

## 4.4 LE ACL TX Data

The information related to HCI LE ACL TX Data (sent to the peer device on BLE link) is carried through a unique message filled with following data:

| MSG ID | DEST ID | SRC ID | MSG LENGTH | 2 | 1 | 1 | 2 | 4 |
|---|---|---|---|---|---|---|---|---|
| ACL DATA TX | Con Idx | - | LEN | CONHDL | F | Res | LEN | TX descriptor |

**Figure 18 – Kernel message for carrying HCI LE ACL TX Data information**

Kernel message content:

| KE message field | Values |
|---|---|
| Message ID | HCI LE ACL TX Data Message ID |
| Destination Task | Connection Index |
| Source Task | Not filled |

| | |
|---|---|
| Parameters Length | Length of the parameters |
| Parameters | |
| | Connection handle |
| | Packet boundary and packet broadcast flags |
| | Reserved |
| | Data Length |
| | TX descriptor of the data to send |

**Table 9 – HCI LE ACL TX Data Kernel message values**

**Note:** the Kernel message carries some information related ACL data packet, not the data itself. The data itself is stored in specific buffers managed by a dedicated Software block. For more details, see RW-BLE SW specification [2] part 6.

## 4.5 BT ACL Data

Data exchanged over a BT link is managed at HCI thanks to the Kernel message presented below:

| MSG ID | DEST ID | SRC ID | MSG LENGTH | 4 |
|---|---|---|---|---|
| ACL DATA TX | Con Idx | - | LEN | ACL BUF ELT PTR |

**Figure 19 – Kernel message for carrying HCI BT ACL Data information**

Kernel message content:

| KE message field | Values |
|---|---|
| Message ID | HCI CS Event Message ID |
| Destination Task | Connection Index (only for connection oriented events) |
| Source Task | Not filled |
| Parameters Length | Length of the parameter |
| Parameters | |
| | Pointer to the buffer element |

**Table 10 – HCI BT ACL Data Kernel message values**

The buffer element is used to describe the data all over its way through the SW stack. It contains the description of the data present in the buffer, i.e. length, packet boundary flag, and data broadcast flag.

# 5 Internal messages routing

For each HCI message transferred, the HCI Software decides whether to route the message internally (Software task) or externally (through transport layer).

The features related to communication with external system (Host or Controller) such reception state machine, packet TX queuing, packet packing or unpacking, are described at 6 and 7 of this document. This chapter focuses on finding the internal destination of HCI messages within the internal Host or Controller.

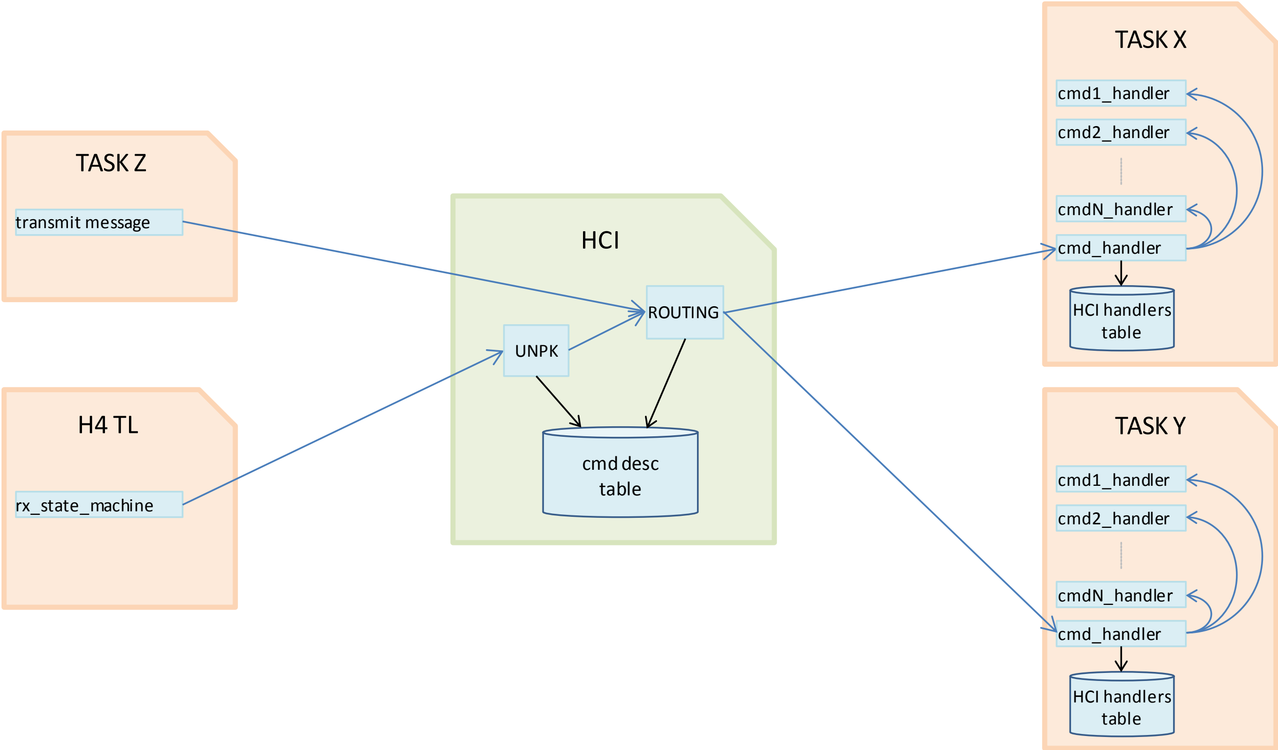


**Figure 20 – Messages transferring through HCI**

For each message transiting through HCI (command, event, RX data, TX data), the HCI Software needs to find the destination task within lower or higher layers, this what the following sections explain.

## 5.1 From external Host to internal Controller

HCI retrieves the command opcode from the HCI packet, which is used for retrieving its associated command descriptor. The descriptor contains the internal identifier that allows HCI associating a destination task to the message.

The possible identifiers for a lower layers destination are listed in the table below:

| Identifier | Description | Destination task (for each configuration) | | |
|---|---|---|---|---|
| | | BLE only | BT only | BT Dual Mode |
| MNG | The message is intended for the main manager task (e.g. HCI_Reset_Cmd) | LLM | LM | LM |
| BLE_MNG | The message is intended for the BLE manager task (e.g. HCI_LE_Create_Connection_Cmd) | LLM | N/A | LLM |
| BT_MNG | The message is intended for the BT manager task (e.g. HCI_Create_Connection_Cmd) | N/A | LM | LM |
| CTRL | The message is intended for one of the controller tasks (e.g. HCI_Disconnect_Cmd / HCI_ACL_Data packet). | LLC | LC | LC or LLC |
| BLE_CTRL | The message is intended for the BT controller task (e.g. HCI_LE_Connection_Update_Cmd) | LLC | N/A | LLC |

| | | | | |
|---|---|---|---|---|
| BT_CTRL_CONHDL | The message is intended for the BT controller task. The associated BT link can be pointed thanks to the connection handle given as command parameter (e.g. HCI_Sniff_Mode_Cmd) | N/A | LC | LC |
| BT_CTRL_BD_ADDR | The message is intended for the BT controller task. The associated BT link can be pointed thanks to the remote BD address given as command parameter (e.g. HCI_Swith_Role_Cmd) | N/A | LC | LC |
| DBG | The message is intended to the debug task (HCI_DBG_Read_Memory_Cmd) | DBG | DBG | DBG |

**Table 11 – Lower layers destination types**

For control messages that are not dedicated to a specific Bluetooth connection, the messages are sent to the main LL manager task (LM/LLM), which is a single instantiated task. Both BT and BLE implement a manager task and are able to handle the messages specific to their own protocol. The messages related to a common management of the device (e.g. HCI_Reset_Cmd, HCI_Read_Local_Version_Information_Cmd) are sent to BT manager task in priority, but may be sent to BLE controller task in BLE Stand-alone configuration.

When a message is specific to a BT or BLE connection (ACL data or link-specific control messages), HCI needs to find the associated instance of the BT or BLE Controller task. BT standard defines a way for Host and Controller to identify which the message applies to. The mechanism is mainly based on a per-connection value named ***"connection handle"***, which is allocated by the controller at link establishment, and freed at link disconnection. Link-specific messages generally include the connection handle as part of their parameters.

The connection handle is indicated by Controller to Host when the connection has been established thanks to following events:

- HCI Connection Complete Event (classic BT asynchronous connection)
- HCI LE Connection Complete Event (BLE asynchronous connection)

A connection is considered closed by HCI when following event is transferred:

- HCI Disconnection Complete event

This section assumes that the connection handle is chosen by the internal Controller as per the rules described at 5.4, so that it is possible to derive a connection handle to a link identifier.

For Classic Bluetooth links, some link-oriented messages do not carry a connection handle in their parameters. Those commands exist only for Bluetooth protocol, where the connection is submitted to Host acceptance and several features may be negotiated before the connection is considered established (and connection handle allocated):

- Connection Acceptance – HCI_Connection_Request event, HCI_Accept_Connection_Request and HCI_Reject_Connection_Request commands
- Master slave role switch – HCI_Switch_Role command, HCI_Role_Change event
- Authentication – HCI_Link_Key_Request event, HCI_Link_Key_Request_Reply command, and many others
- Pairing – HCI_Pin_Code Request event, HCI_Pin_Code_Request_Reply command, and many others

In addition, the messages related to a synchronous connection, which is setup on top of ACL connections, use BD address or the synchronous connection handle. Also, an external Host may use a wrong connection handle, and HCI needs to filter such message for robustness purpose.

Then, to be able to route all link-oriented messages to the right BT or BLE controller task instance, HCI maintains internal data organized as shown below:

| | idx | State (1 byte) | BD address (6 bytes) |
|---|---|---|---|
| BT links | 0 | … | … |
| | 1 | … | … |
| | … | … | … |
| | N-1 | … | … |

| | idx | State |
|---|---|---|
| BLE links | 0 | … |
| | 1 | … |
| | … | … |
| | M-1 | … |

**Table 12 – Table for link identification (messages received from external Host)**

The purpose of associating a status to each link is to filter the potential wrong connection handles received from the Host. A message is transferred to a BT or BLE controller task instance if and only if the connection handle is in the possible range and the associated link exists.

The filling of these tables is made from the Controller tasks themselves at link establishment or disconnection, as shown on the following figures:

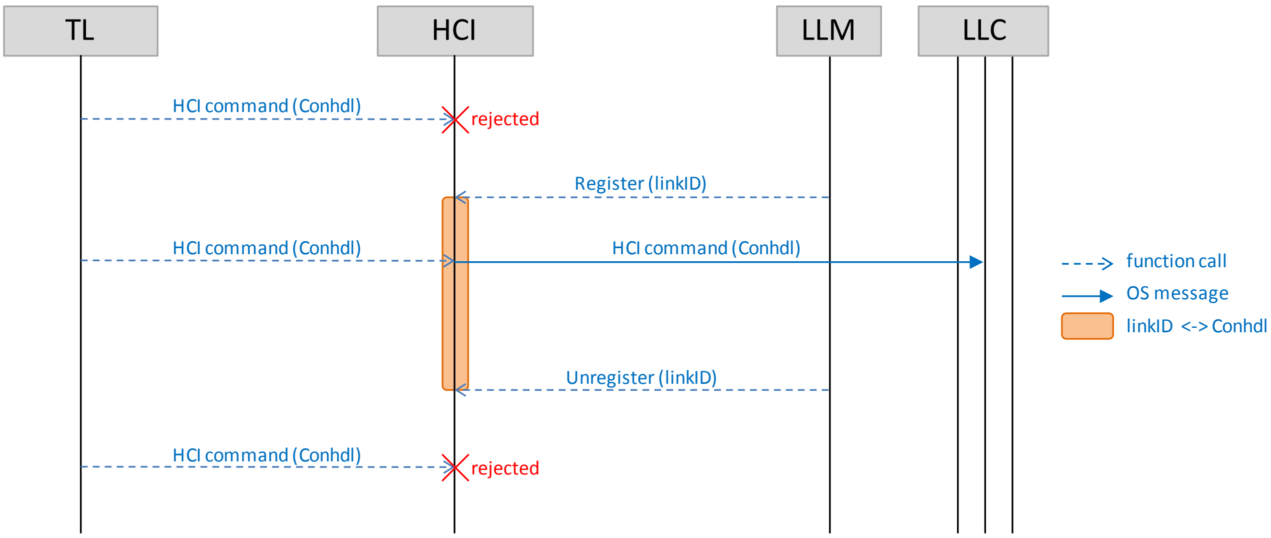


**Figure 21 – BLE connection-oriented message routing**

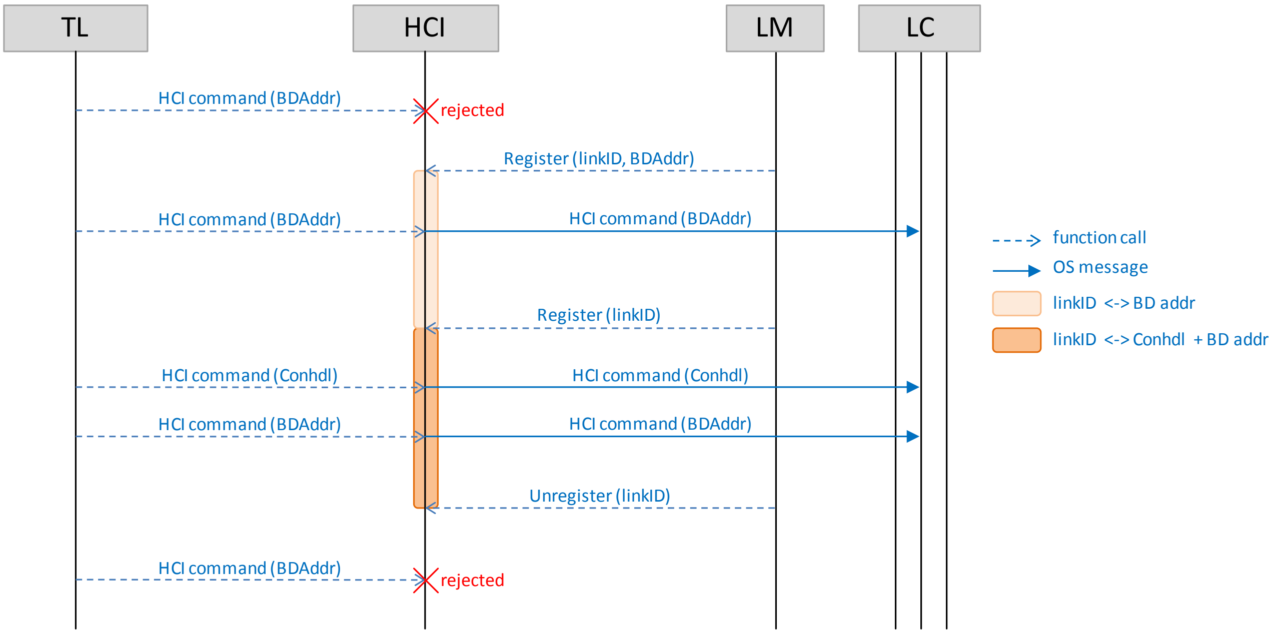


**Figure 22 – BT connection-oriented message routing**

When a link-oriented command is transferred through HCI, HCI checks if there is an active link that could match based “State” flag and the connection handle or BD address. If no link identifier matches, a Command Complete event or Command status is sent back to Host with error code “Unknown Link Identifier”. If matching, the destination task instance is built from the associated link identifier.

## 5.2 From external Controller to internal Host

HCI retrieves the event code from the HCI packet, which is used for retrieving its associated event descriptor. In case of HCI_Command_Complete_Evt or HCI_Command_Status_Evt, HCI retrieves the opcode of the original command the event is replying to, then get the associated command descriptor. The descriptor contains the internal identifier that allows HCI associating a destination task to the message.

The possible identifiers for a higher layers destination are listed in the table below:

| Identifier | Description | Destination task |
|---|---|---|
| MNG | The message is intended for the main manager task (e.g. HCI_Reset_Cmd_Cmp_Evt) | GAPM |
| CTRL | The message is intended for one of the controller task (e.g. HCI_Disconnect_Cmp_Evt). | GAPC |
| DATA | The message is intended for one of the data task (e.g. HCI_Number_Of_Completed_packets event / HCI_ACL_Data RX packet). | L2CC |

**Table 13 – Lower layers destination types**

When a message is specific to a BLE connection (ACL data or link-specific control messages), HCI needs to find the associated instance of the BLE Controller task. To do that, HCI maintains a connection handle table:

| | | 2 bytes |
|---|---|---|
| | idx | Con Hdl |
| BLE links | 0 | … |
| | 1 | … |
| | … | … |
| | M-1 | … |

**Table 14 – Table for link identification (messages received from external Controller)**

The purpose of storing the connection handle is to be able to find the destination task instance for the link-specific messages, whatever the connection handle allocation rules followed by the external Controller.

The filling of this tables is made from the Host tasks themselves at link establishment or disconnection, as shown on the following figures:

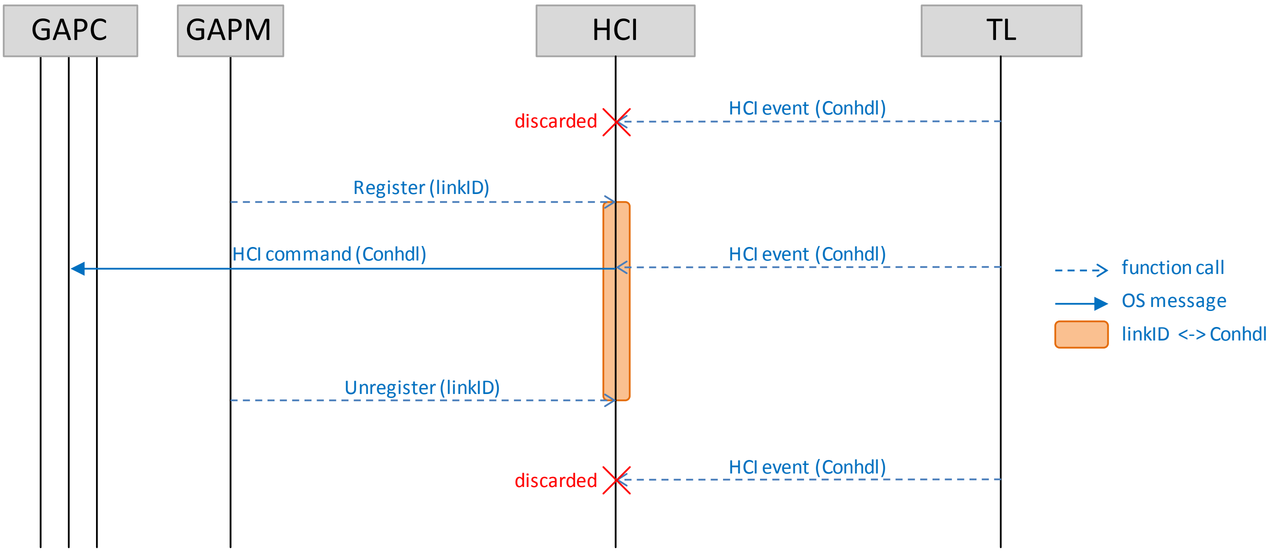


**Figure 23 – BLE connection-oriented message routing**

When a link-oriented event is transferred through HCI, HCI checks if there is an active link that could match based on the “State” flag and the connection handle or BD address. If no link identifier matches, the event is discarded. If matching, the destination task instance is built from the associated link identifier.

## 5.3 Between internal Host and Controller

Communication between internal Host and Controller implies that the device is in full stack configuration, and then in BLE single mode, as the full stack mode is supported in BLE single mode only.

In both directions, HCI retrieves the command opcode or event code from the Kernel message, and derives it to a higher layers or lower layers destination type. The manager task just depends on the direction (LLM task in Controller, or GAPM task in Host).

As explained at chapter 1.3, the full stack configuration involves an internal controller only, where the connection handle allocation rules is considered known (see 5.4). Then, the connection handle can be directly associated to a link identifier without the need of any association table, and it is assumed that the internal Host or Controller never tries to transmit a message with a wrong connection handle. Therefore, when composing the controller task destination (LLC task in Controller, or GAPC task in Host), the instance selection is the link ID derived from the connection handle.

## 5.4 Proprietary rules for connection handle allocation:

The RW BT/BLE controller IP allocates internally a link identifier in the range [0 : N-1], where N is the number of BT links supported, or a link identifier in the range [0 : M-1], where M is the number of BLE links supported. There are proprietary rules to create a connection handle from the link ID:

- BT ACL conhdl = 0x80 & BT ACL link ID
- BT SCO conhdl = (SCO link ID << 8) & 0x80 & BT ACL link ID
- BLE conhdl = BLE link ID

Examples:

- ✓ 0x85 refers to BT ACL link number 5
- ✓ 0x02 refers to BLE link number 2
- ✓ 0x281 refers to BT SCO link number 2 (associated to BT ACL link number 1)

These rules are given as information, they are not standard but specific to RW Controller IP. In order to be compatible to 3rd party systems, RW HCI Software stores any connection handle in the link identification table as described above.

# 6 Communication with external Host

## 6.1 HCI commands

As described in chapter 5, the HCI Software handles the message routing of any message received by Transport Layer to a destination block within the controller layers. Additionally, it also handles command parameters unpacking depending on the receiving system structure padding and endianness policies. It finally handles the command flow control as described in BT specification [1] part II.E.4.

When receiving an HCI command from an external system, the transport may proceed in one or several steps. After a complete packet has been received, it delegates the packet management to HCI layer. For example, to receive a command over UART, TL gets a packet in two steps for commands with no parameter or three steps for commands with parameters:

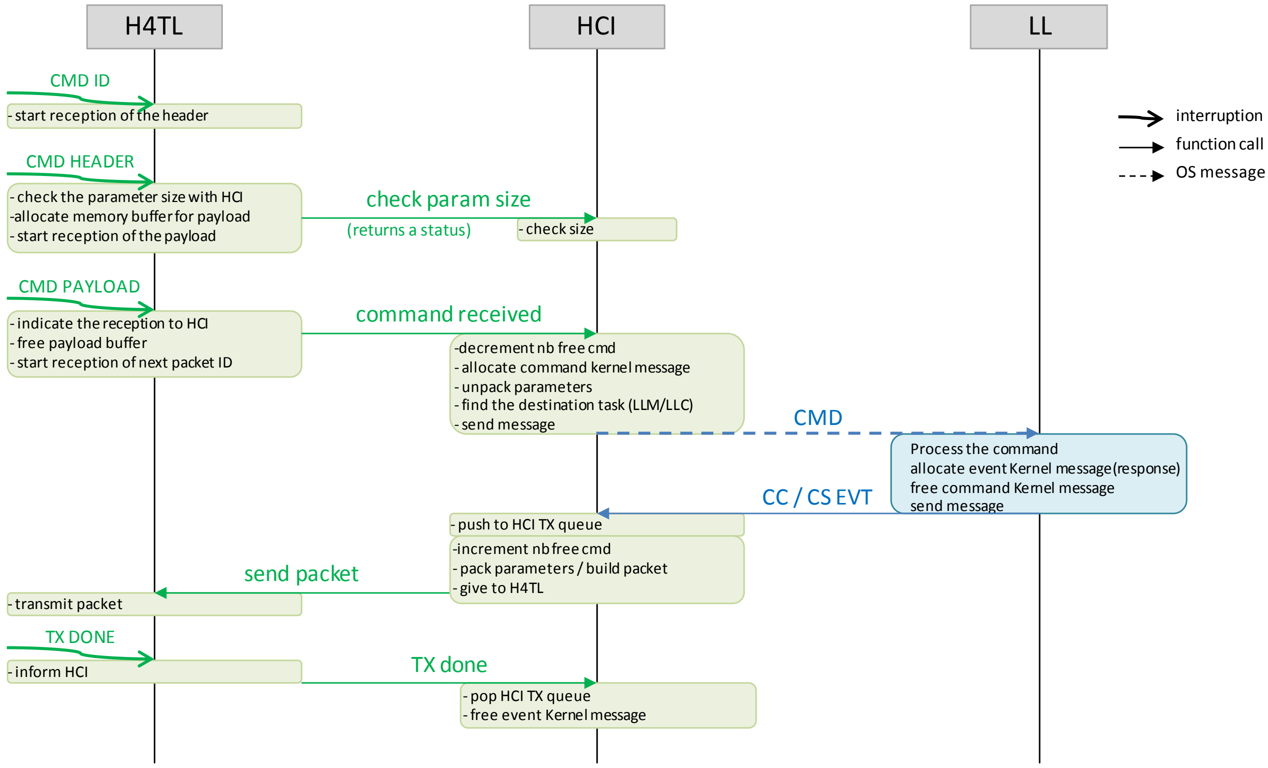


**Figure 24 – HCI command reception flow over UART (command with parameters)**

As shown on figure above, the UART Transport Layer (H4TL), which generally works under interrupt, calls HCI Software at header and payload reception. A packet is considered fully received at header reception for parameter-less commands, otherwise it is considered received once the payload is received. For each command which has parameters and is checked as 'valid' by HCI, the transport Layer must allocate a memory with the appropriate space for receiving the payload.

The processing made by HCI at packet reception is based on the HCI command opcode, which is associated to a command descriptor, as described in section 3.1.

For each known packet, HCI builds a Kernel message and send it to the right task within BT/BLE Controller stack.

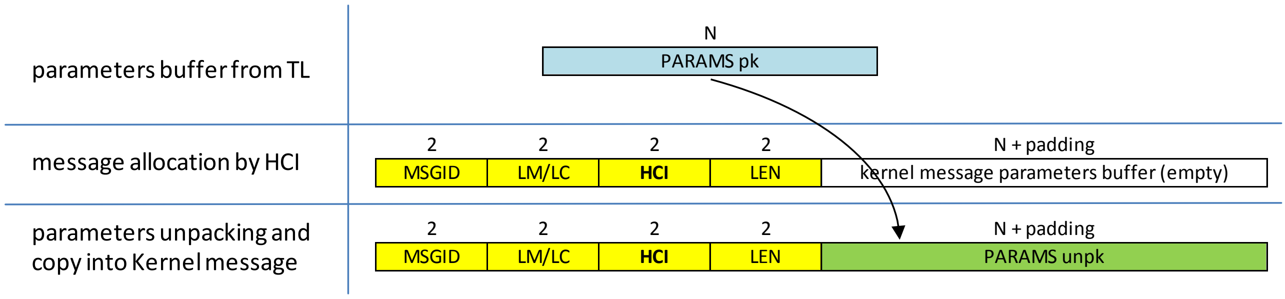


**Figure 25 – Data manipulation during HCI commands reception**

Note: the kernel message parameters size handles the space needed by the parameters upon a C structure basis. This means that for any compiler, the space reserved is the size of the final structure. Some compilers may include padding between structure fields. For that reason, the allocated size is based on the parameters format string available in the descriptor rather than the received parameters size.

As each HCI command shall be replied with a HCI Command Status or Command Complete event. These two events have the particularity that they are responding to an HCI command. Then their transmission through HCI makes increment the current number of HCI command the system can handle. Their special parameters manipulation is explained in following section.

## 6.2 HCI events

In case of external routing, HCI pushes the message in a transmission queue. Once Transport Layer TX channel is available, HCI builds the HCI packet and transmits the buffer to the TL (see 7 for details).

The Kernel message buffer is used by HCI to build the HCI event packet to transmit over Transport Layer. It is not freed right after HCI task processing but only after TL has confirmed the transmission (HCI TX Done), as shown by the figure below:

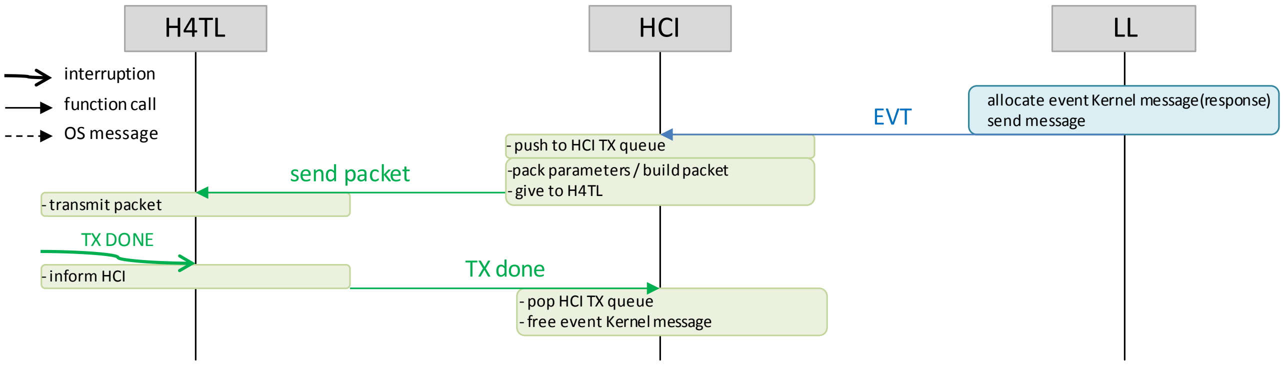


**Figure 26 – HCI event transmission flow over UART**

The events are classified in four different categories; each has a specific packet format and potentially specific parameters manipulation.

### 6.2.1 Legacy events

All legacy events are managed in a common way. The controller task that needs to send an event to Host uses the legacy HCI event message. When receiving this message, HCI software will proceed to the parameters packing and the sending to Transport Layer, as shown on the figure below:

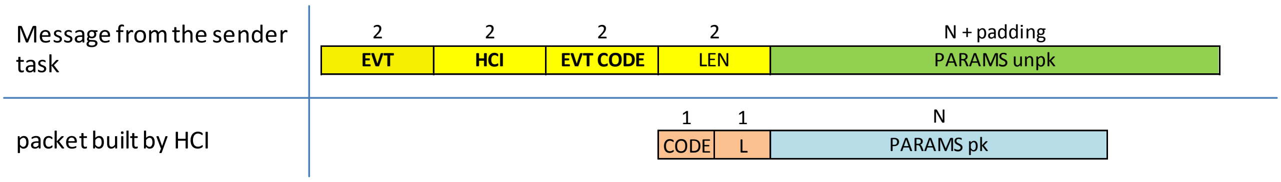


**Figure 27 – HCI events packet building**

The packet building is performed thanks to the legacy event descriptor table that contains descriptors for each supported event.

### 6.2.2 Command Complete events

The command complete (CC) events is managed separately as it presents the particularity to reply to an HCI command. It contains the original command opcode and the number of HCI commands that controller can receive, for HCI flow control. The command complete event has also the particularity to contain the return parameters of the original command.

To send a CC event, a controller task composes a CC event message to the HCI. When receiving this message, HCI performs following actions:

- increment number of free commands HCI can receive (HCI flow control)
- pack return parameters
- fill other fields
- push to HCI TX queue

The data manipulation over the kernel message buffer is shown on following figure:

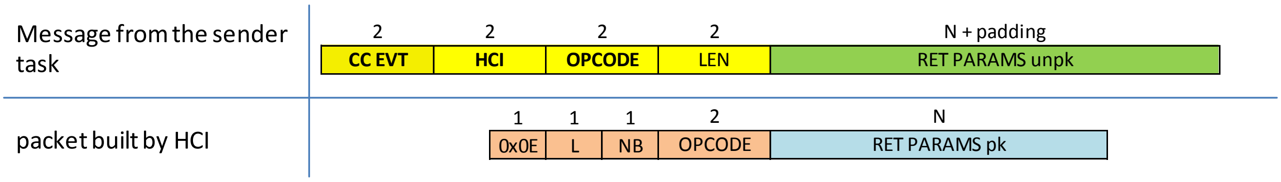


**Figure 28 – HCI CC event packet building**

The packet parameter unpacking is performed thanks to the original command descriptor found in the command descriptors table (see 3.1).

### 6.2.3 Command Status events

The command status (CS) events is managed separately as it presents the particularity to reply to an HCI command. It contains the original command opcode and the number of HCI commands that controller can receive, for HCI flow control.

To send a CS event, a controller task composes a CS event message to the HCI. When receiving this message, HCI performs following actions:

- increment number of free commands HCI can receive (HCI flow control)
- build the packet
- push to HCI TX queue

The data manipulation over the kernel message buffer is shown on following figure:

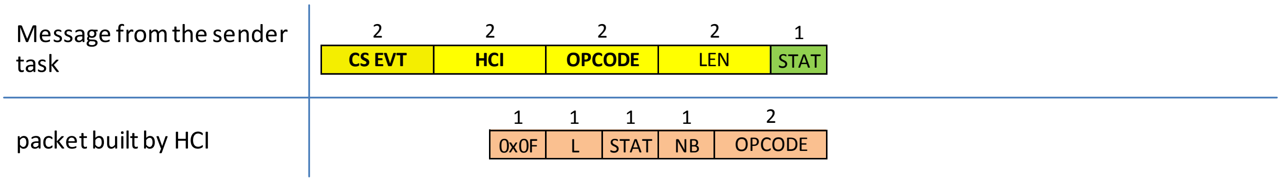


**Figure 29 – HCI CS event packet building**

### 6.2.4 LE events

All LE events are managed in a common way. The controller task that needs to send a LE event to Host uses the LE event message. When receiving this message, HCI performs following actions:

- pack parameters
- build the packet
- push to HCI TX queue

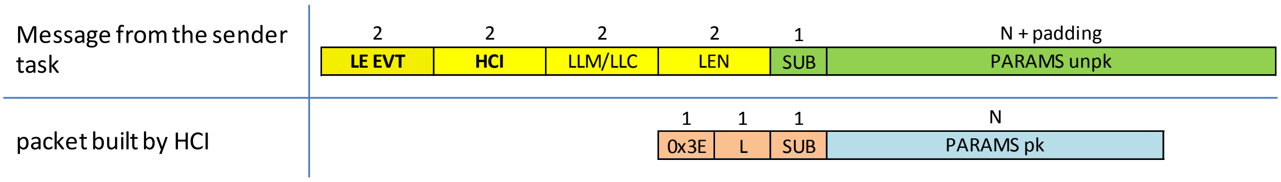


**Figure 30 – HCI LE events packet building**

The packet building is performed thanks to the LE event descriptor table that contains descriptors for each supported LE event.

## 6.3 HCI ACL TX Data

The data given by an external Host to be transmitted over the air triggers this mechanism:

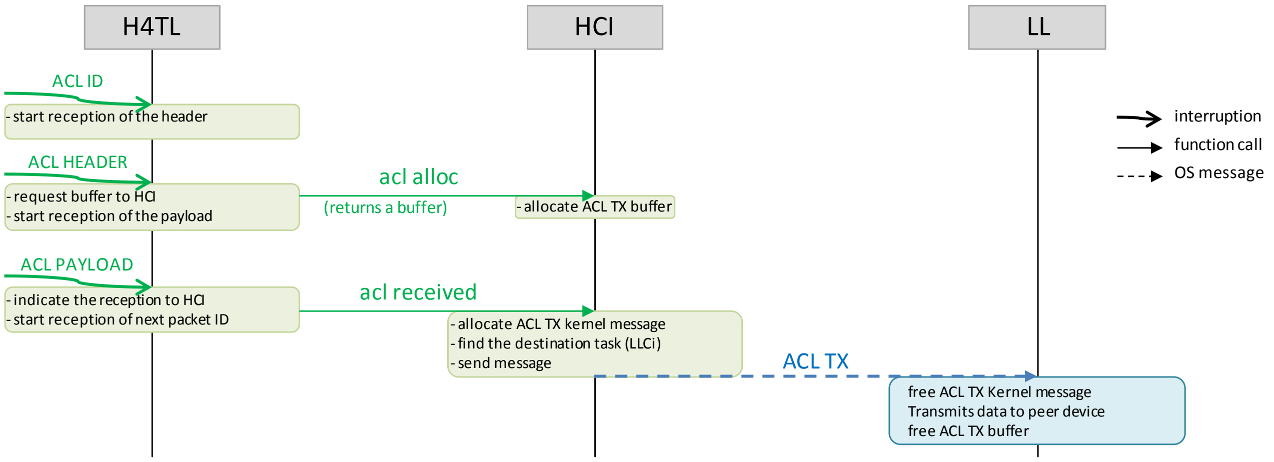


**Figure 31 – Reception of HCI ACL TX Data from external Host**

The above figure shows the behavior of HCI in a normal case, when a correct packet is received from Host and buffers are available. However, two error cases are possible when Transport layer receives the HCI data packet header:

1. Data length error:

If the field received in HCI header is higher than the maximum buffer size, the reception over physical interface is considered erroneous. In this case, HCI returns a NULL pointer, and TL reset its reception path.

2. Buffer overflow:

If there is no more available buffers within the stack, HCI allocates a buffer from the RAM heaps. It frees the buffer once the TL indicates the payload reception.

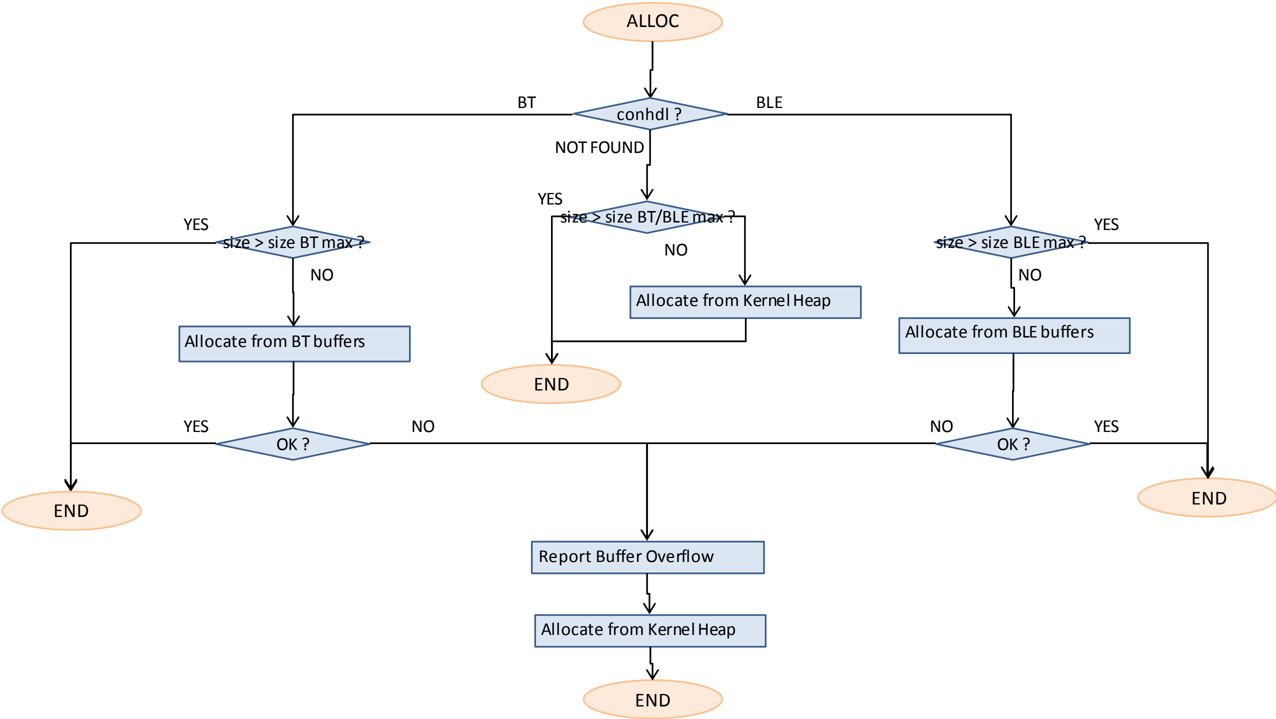


**Figure 32 –HCI ACL TX Data buffer allocation algorithm**

This figure shows the algorithm executed when trying to allocate a buffer for TX data. Possible results are:

- If the payload size is higher than expected, no buffer is allocated
- If connection handle does not match with any active connection, or there is no more BT or BLE buffer, a buffer is allocated from the Heap.
- In normal cases, each BT/BLE respective buffer management system provide a buffer able to receive the packet payload

Then, after reception of the payload through TL, the action taken by HCI follows the result of buffer allocation:

- BT buffer: send message to LC
- BLE buffer: send message to LLC
- Kernel Heap buffer: free the buffer

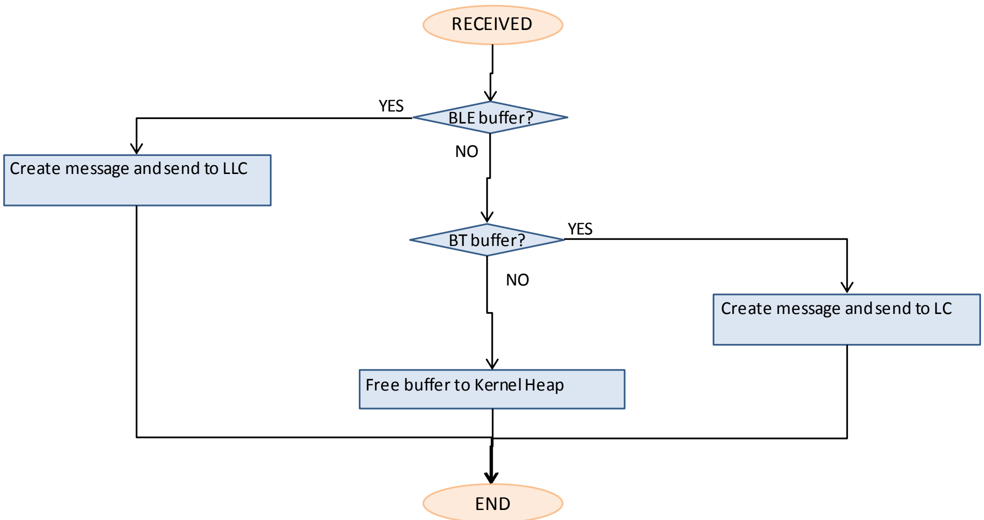


**Figure 33 –HCI ACL TX Data received algorithm**

## 6.4 HCI ACL RX Data

The data received from the air is given to an external Host according to following mechanism:

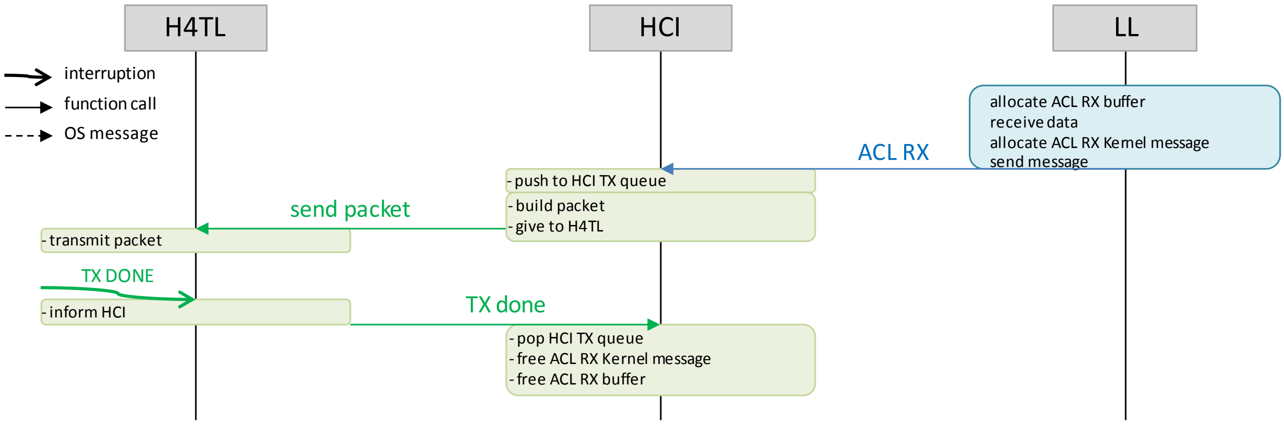


**Figure 34 – Transmission of HCI ACL RX Data to external Host**

# 7 Communication with external Controller

## 7.1 HCI commands

Host sends commands to an external Controller according to following mechanism:

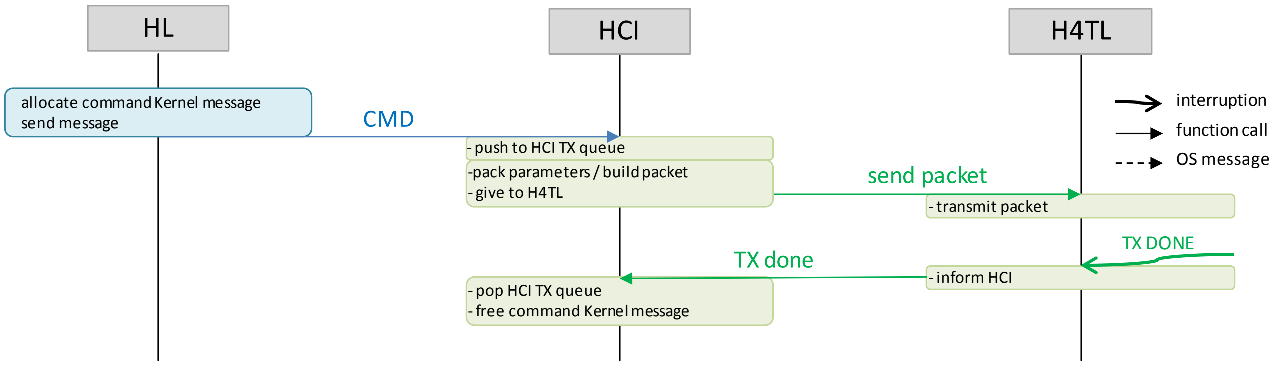


**Figure 35 – Transmission of HCI command to external Controller**

## 7.2 HCI events

Host receives events from an external Controller according to following mechanism:

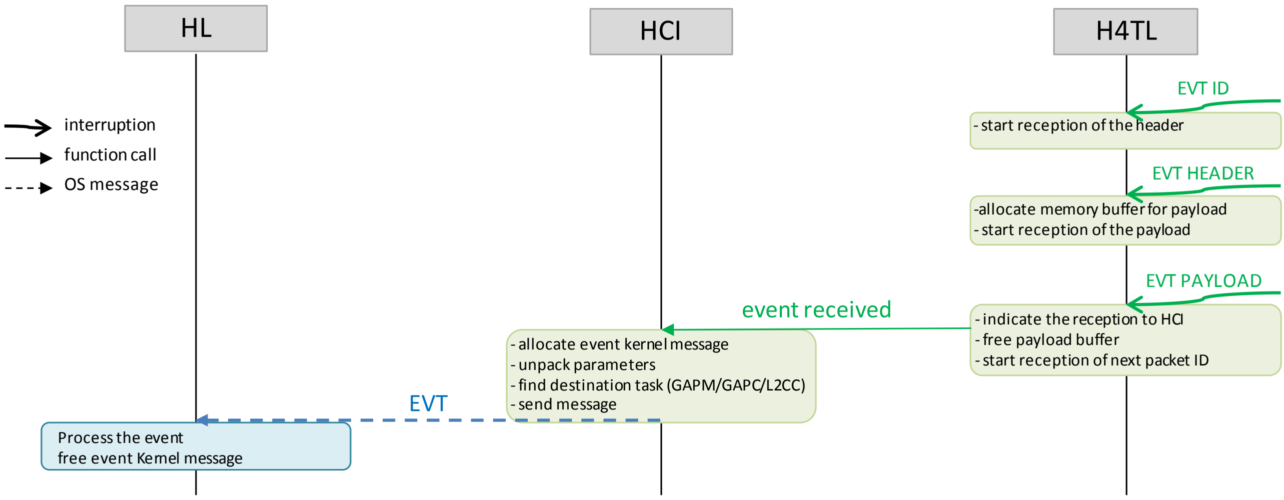


**Figure 36 – Reception of HCI event from external Controller**

## 7.3 HCI ACL RX Data

The data received from the air is received from an external Controller according to following mechanism:

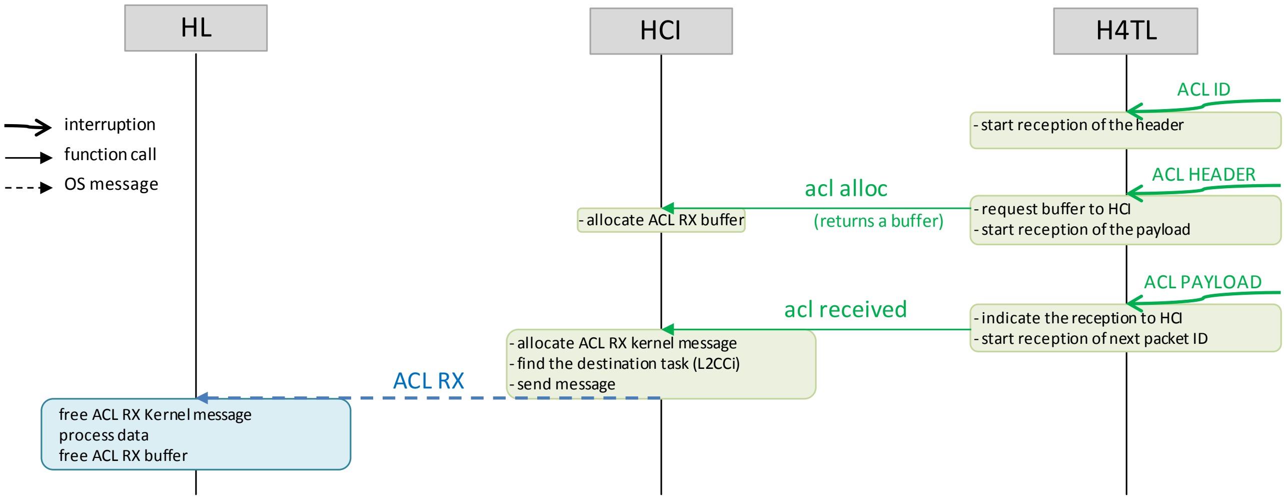


**Figure 37 – Reception of HCI ACL RX Data from external Controller**

## 7.4 HCI ACL TX Data

Host gives data for transmission over the air by an external Controller according to following mechanism:

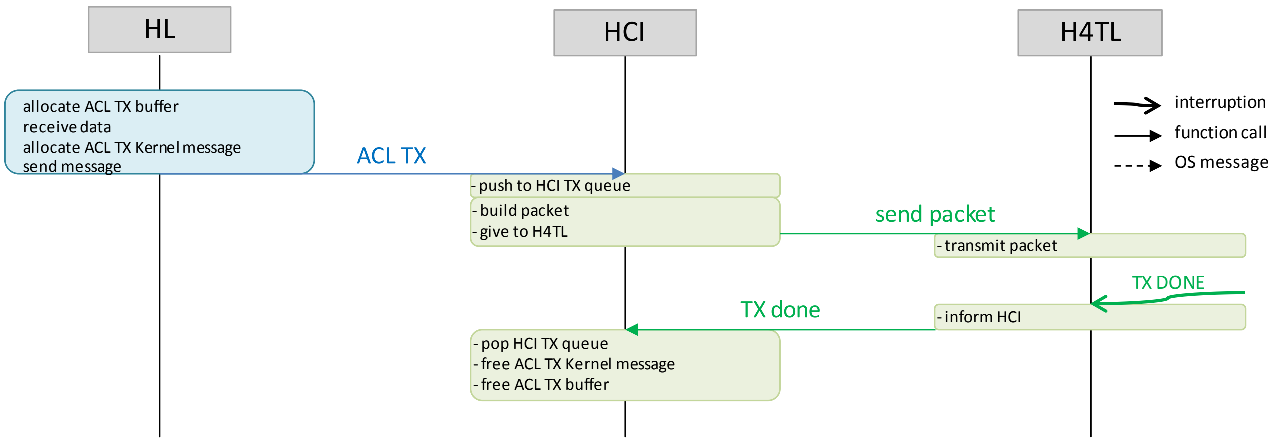


**Figure 38 – Transmission of HCI ACL TX Data to external Controller**

# 8 Generic parameters packing-unpacking

For several reasons, including portability, code size and flexibility, the HCI Software uses as much as possible a common way for packing and unpacking the parameters according to the needs of both sides:

- the HCI interface, which deals with byte streams where the parameters are packed and the bytes are serialized in a specific ordering
- the internal system, with its owns processor and memory constraints (endianess, data alignment, structure padding)

A SW utility package is included within HCI layer. It defines generic packer and unpacker functions explained below.

### 8.1.1 Parameters format definition

Both the packer and unpacker take as input a string representing the parameters format. The string is a concatenation of elements that describes parameters one-by-one.

Following table lists the supported format elements:

| Element | Packed format | Unpacked format |
|---|---|---|
| **B** | 1 byte | 1 x 8-bits variable |
| **H** | 2 bytes | 1 x 16-bits variable |
| **L** | 4 bytes | 1 x 32-bits variable |
| **nB** | n bytes | table of n x 8-bits values *<br>Example: “2B”, “16B”, “128B” |
| **nH** | n x 2 bytes | table of n x 16-bits values *<br>Example: “2H”, “16H”, “124H” |
| **nL** | n x 4 bytes | table of n x 32-bits values * |
| * The table sizes must respect the maximum buffer size | | |

**Table 15 – Format elements definition**

### 8.1.2 Generic packer

The generic packer takes a format string as input. It also takes the parameters buffer that initially contains unpacked data. It is able to work directly within the unpacked parameters buffer.

It parses the input format string up to the end. For each element, it computes the ‘read’ position (where the unpacked data is located), taking care of the current compiler alignment constraint. Then it copies the data to the ‘write’ position by taking care of the processor endianness. The ‘write’ location is incremented by the length of the copied data.

Note: the generic unpacker can also be used to know the size of the packed data. If no buffer is given to the function, the algorithm performs a space computation without any data copy. This can be useful to check a packet consistency when TL has received the header.

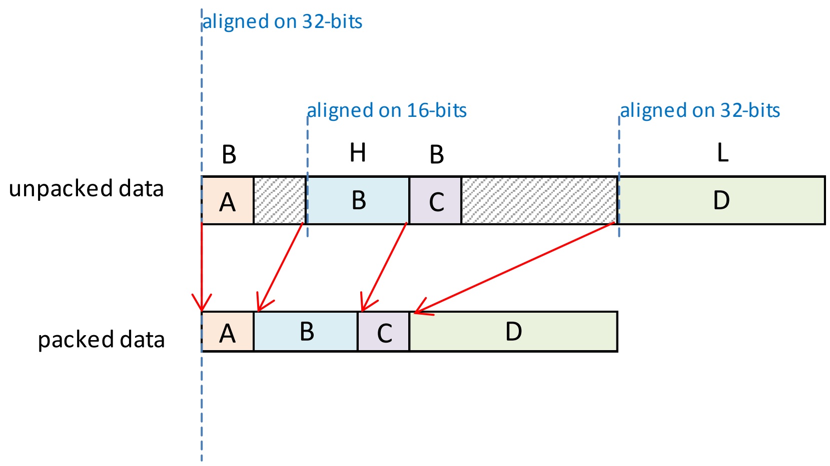


**Figure 39 – Example of data packing for an ARM processor**

### 8.1.3 Generic unpacker

The generic unpacker takes a format string, as input. It also takes the input buffer containing packed data, and the output buffer where to put the unpacked data.

It parses the input format string up to the end. For each element, it computes the 'write' position (where the unpacked data has to be written), taking care of the current compiler alignment constraint. Then it copies the data to the 'write' position by taking care of the processor endianness. The 'read' location is incremented by the length of the copied data.

Note: the generic unpacker can also be used to know the size of the unpacked data. If no buffer is given to the function, the algorithm performs a space computation without any data copy. This can be useful at the buffer allocation time before receiving the data from TL.

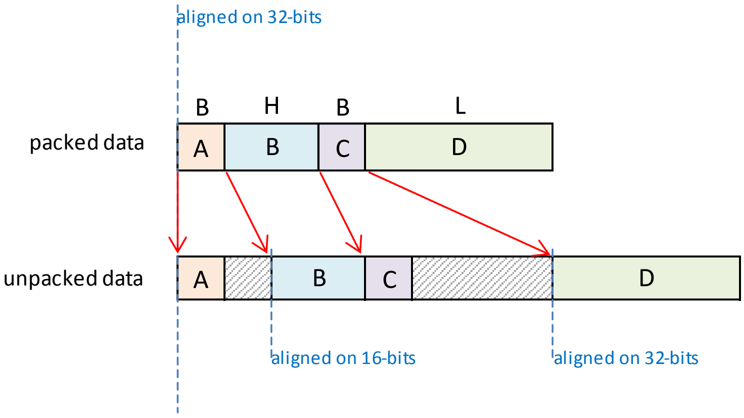


**Figure 40 – Example of data unpacking for an ARM processor**

### 8.1.4 Alignment and data copy primitives

The primitives used for address alignment and data copy are located in a utility package common for all the FW (***common***).

Here is a list of the ones used for HCI packing-unpacking:

- ✓ ***CO_ALIGN2_HI(val)*** → align address to the following 16-bits address
- ✓ ***CO_ALIGN4_HI(val)*** → align address to the following 32-bits address
- ✓ ***co_read16p(ptr)*** → return a 16-bits value read at 'ptr' position
- ✓ ***co_read32p(ptr)*** → return a 32-bits value read at 'ptr' position
- ✓ ***co_write16p(ptr, val)*** → write 'val' as a 16-bits value to 'ptr' position
- ✓ ***co_write32p(ptr, val)*** → write 'val' as a 32-bits value to 'ptr' position

These macros or functions have to be adapted to each compiler/processor it is used for.

# Abbreviations

| Abbreviation | Original Terminology |
|---|---|
| ACL | Asynchronous connection-oriented logical transport |
| API | Application Programming Interface |
| BLE | Bluetooth Low Energy |
| BT | Classic Bluetooth |
| GAP | Generic Access Profile |
| GAPC | Generic Access Profile Controller Software task |
| GAPM | Generic Access Profile Manager Software task |
| H4TL | UART Transport Layer |
| HCI | Host Controller Interface |
| L2CAP | Logical Link Protocol |
| L2CC | Logical Link Protocol Software task |
| LC | Link Controller |
| LL | Link Layer |
| LLC | Link Layer Controller Software task |
| LLM | Link Layer Manager Software task |
| LM | Link Manager |
| RW | RivieraWaves |
| SCO | Synchronous connection-oriented logical transport |
| TL | Transport Layer |
| UART | Universal Asynchronous Receiver Transmitter |
| USB | Universal Serial Bus |

**Table 16 – List of abbreviations**

# References


| | | | | |
|---|---|---|---|---|
| **[1]** | **Title** | Core Bluetooth Specification V4.1 | | |
| | **Reference** | Core_V4.1 | | |
| | **Version** | V4.1 | **Date** | 2013-12-09 |
| | **Source** | Bluetooth SIG | | |

| | | | | |
|---|---|---|---|---|
| **[2]** | **Title** | RW-BLE Link Layer Software | | |
| | **Reference** | RW-BLE-LL-SW-FS | | |
| | **Version** | 7.0.11 | **Date** | 2014-03-26 |
| | **Source** | RivieraWaves | | |

| | | | | |
|---|---|---|---|---|
| **[3]** | **Title** | | | |
| | **Reference** | | | |
| | **Version** | | **Date** | |
| | **Source** | | | |