wicentric

HCI API

Document 2009-0006
Revision 1.2
September 25, 2015

Table of Contents

1 Introduction

1.1 Overview

This document describes the API for the host controller interface (HCI) layer of Wicentric's Bluetooth LE protocol stack. This API is used by the Bluetooth LE stack to communicate with a Bluetooth LE controller or link layer. Traditionally, HCI is a message passing interface consisting of command and event messages defined by the Bluetooth specification. In the Wicentric stack the HCI API is optimized as a thin interface layer for single chip systems that can also be configured to run in a traditional system with a separate controller and wired HCI transport.

The Bluetooth specification defines the HCI messages and parameters. Rather than repeat that information here this document describes how the details of the API implementation differ from the Bluetooth specification [1].

1.2 HCI Topologies

The different HCI topologies for a single chip system and a traditional stack with HCI are shown below in Figure 1.

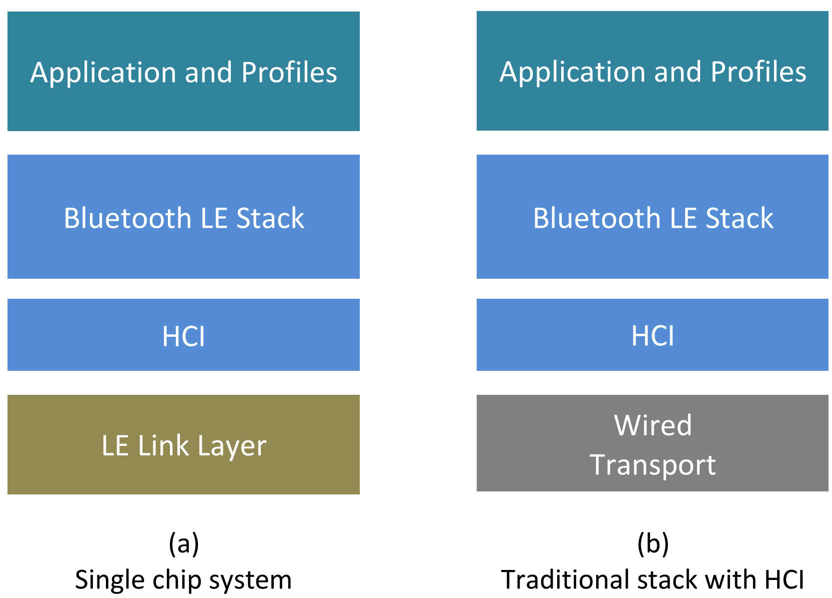

Figure 1.

In a single chip system the HCI layer mainly serves two purposes. First, it implements a message passing interface between the stack and the link layer. Second, it translates the HCI API used by the stack (as defined in this document) to the API used by the link layer.

In a traditional stack with HCI the HCI layer serves several purposes. It implements a message passing interface between the stack and the wired transport. It builds and parses byte-oriented messages for transmission on the wired transport. It also implements a transport-specific driver to send and receive data on the wired transport.

There are also differences in the data flow. In a traditional stack the HCI layer also implements handling of transmit path data flow and processing of HCI Number of Completed Packets events. In a single chip system the HCI layer adapts the data path interface required by the stack to the link layer's data path.

1.3 Basic Data Types

The Bluetooth HCI specification defines parameters in terms of octets. These map to integer data types as shown below:

Octets	Stack Data Type
1 Octet	int8_t or uint8_t
2 Octets	uint16_t
3-4 Octets	uint32_t
> 4 Octets	uint8_t array or other data structure

The following type is used for the Bluetooth device address.

Type	Name
uint8_t	bdAddr_t[6]

2 Initialization, Registration and Reset

2.1 void HciHandlerInit(wsfHandlerId_t handlerId)

This function initializes the HCI subsystem. It is typically called once upon system initialization. It must be called before any other function in the HCI API is called.

2.2 void HciEvtRegister(hciEvtCback_t evtCback)

This function is called by a client to register for HCI events. This function is called by DM.

- **evtCback**: Client callback function. See 7.1.

2.3 void HciSecRegister(hciSecCback_t secCback)

This function is called by a client to register for certain HCI security events. This function is called by WSF.

- **secCback**: Client callback function. See 7.2.

2.4 void HciAclRegister(hciAclCback_t aclCback, hciFlowCback_t flowCback)

This function is called by a client to register for ACL data. This function is called by L2C.

- **aclCback**: Client ACL data callback function. See 6.2.
- **flowCback**: Client flow control callback function. See 6.3.

2.5 void HciResetSequence(void)

This function initiates an HCI reset sequence, which sends an HCI Reset command followed by several other HCI commands. This HCI command sequence is configurable for each platform. When the reset sequence is complete, a Reset Sequence Complete event is sent via the event callback.

A typical reset sequence is as follows:

1. Reset Command
2. Set Event Mask Command
3. LE Set Event Mask Command
4. Read BD_ADDR Command
5. LE Read Buffer Size Command
6. Read Buffer Size Command
7. LE Read Supported States Command
8. LE Read White List Size Command
9. LE Read Advertising Channel TX Power Command

2.6 void HciVsInit(uint8_t param)

Vendor-specific controller initialization function. This function is optionally used if the HCI controller or link-layer requires custom initialization procedures.

- **param**: Vendor-specific parameter.

2.7 void HciSetMaxRxAclLen(uint16_t len)

Set the maximum reassembled RX ACL packet length. Minimum value is 27.

- **len**: ACL packet length.

2.8 void HciSetAclQueueWatermarks(uint8_t queueHi, uint8_t queueLo)

Set TX ACL queue high and low watermarks.

- **queueHi**: Disable flow on a connection when this many ACL buffers are queued.
- **queueLo**: Enable flow on a connection when this many ACL buffers are queued.

3 Optimization Interface

This is an optimized interface for certain HCI commands which simply read a value. The stack uses these functions rather than their corresponding functions in the command interface.

These functions can only be called after the reset sequence has been completed.

3.1 uint8_t *HciGetBdAddr(void)

Return a pointer to the BD address of this device.

3.2 uint8_t HciGetWhiteListSize(void)

Return the white list size.

3.3 int8_t HciGetAdvTxPwr(void)

Return the advertising transmit power. See the LE Read Advertising Channel TX Power command in [1] for the format of the value.

3.4 uint16_t HciGetBufSize(void)

Return the ACL buffer size supported by the controller.

3.5 uint8_t HciGetNumBufs(void)

Return the number of ACL buffers supported by the controller.

3.6 uint8_t *HciGetSupStates(void)

Return the states supported by the controller. See the LE Read Supported States command in [1] for the format of the value.

3.7 uint8_t HciGetLeSupFeat(void)

Return the LE features supported by the controller. See the LE Read Local Supported Features command in [1] for the format of the value.

4 Command Interface

This interface contains functions that map directly to HCI commands. The operation of the HCI commands and their parameters are not described in this document—instead see the Bluetooth specification [1].

The HCI implementation for a particular platform does not need to implement all functions in the command interface. For example, a single chip system that implements the functions in the optimization interface (e.g. HciGetBdAddr()) and may not need to implement the corresponding functions in the command interface (e.g. HciReadBdAddrCmd()).

4.1 Data Structures

4.1.1 hciConnSpec_t

This data structure is used in functions HciLeCreateConnCmd() and HciLeConnUpdateCmd(). See [1] for a description of the parameters of this structure.

Type	Name
uint16_t	connIntervalMin
uint16_t	connIntervalMax
uint16_t	connLatency
uint16_t	supTimeout
uint16_t	minCeLen

uint16_t	maxCeLen

4.2 Functions

The command interface functions are shown in the table below. See [1] for a description of the parameters and operation of these functions. Functions shown as "not implemented" are not used by Wicentric's Bluetooth LE stack.

HCI Command	Function
Disconnect	void HciDisconnectCmd(uint16_t handle, uint8_t reason)
Host Buffer Size	[not implemented]
Host Number Of Completed Packets	[not implemented]
LE Add Device To White List	void HciLeAddDevWhiteListCmd(uint8_t addrType, uint8_t *pAddr)
LE Clear White List	void HciLeClearWhiteListCmd(void)
LE Connection Update	void HciLeConnUpdateCmd(uint16_t handle, hciConnSpec_t *pConnSpec)
LE Create Connection	void HciLeCreateConnCmd(uint16_t scanInterval, uint16_t scanWindow, uint8_t filterPolicy, uint8_t peerAddrType, uint8_t *pPeerAddr, uint8_t ownAddrType, hciConnSpec_t *pConnSpec)
LE Create Connection Cancel	void HciLeCreateConnCancelCmd(void)
LE Encrypt	void HciLeEncryptCmd(uint8_t *pKey, uint8_t *pData)
LE Long Term Key Requested Negative Reply	void HciLeLtkReqNegReplCmd(uint16_t handle)
LE Long Term Key Requested Reply	void HciLeLtkReqReplCmd(uint16_t handle, uint8_t *pKey)
LE Rand	void HciLeRandCmd(void)
LE Read Advertising Channel TX Power	void HciLeReadAdvTXPowerCmd(void)
LE Read Buffer Size	void HciLeReadBufSizeCmd(void)
LE Read Channel Map	void HciLeReadChanMapCmd(uint16_t handle)
LE Read Local Supported Features	void HciLeReadLocalSupFeatCmd(void)
LE Read Remote Used Features	void HciLeReadRemoteFeatCmd(uint16_t handle)
LE Read Supported States	void HciLeReadSupStatesCmd(void)
LE Read White List Size	void HciLeReadWhiteListSizeCmd(void)
LE Receiver Test	[not implemented]
LE Remove Device From White List	void HciLeRemoveDevWhiteListCmd(uint8_t addrType, uint8_t *pAddr)
LE Set Advertise Enable	void HciLeSetAdvEnableCmd(uint8_t enable)
LE Set Advertising Data	void HciLeSetAdvDataCmd(uint8_t len, uint8_t *pData)
LE Set Advertising Parameters	void HciLeSetAdvParamCmd(uint16_t advIntervalMin, uint16_t advIntervalMax, uin8_t advType, uint8_t ownAddrType, uint8_t directAddrType, uint8_t *pDirectAddr, uint8_t advChanMap, uint8_t advFiltPolicy)
LE Set Event Mask	void HciLeSetEventMaskCmd(uint8_t *pLeEventMask)
LE Set Host Channel Classification	void HciLeSetHostChanClassCmd(uint8_t *pChanMap)

LE Set Random Address	void HciLeSetRandAddrCmd(uint8 *pAddr)
LE Set Scan Enable	void HciLeSetScanEnableCmd(uint8_t enable, uint8_t filterDup)
LE Set Scan Parameters	void HciLeSetScanParamCmd(uint8_t scanType, uint16_t scanInterval, uint16_t scanWindow, uint8_t ownAddrType, uint8_t scanFiltPolicy)
LE Set Scan Response Data	void HciLeSetScanRespDataCmd(uint8_t len, uint8_t *pData)
LE Start Encryption	void HciLeStartEncryptionCmd(uint16_t handle, uint8_t *pRand, uint16_t diversifier, uint8_t *pKey)
LE Test End	[not implemented]
LE Transmitter Test	[not implemented]
Read BD_ADDR	void HciReadBdAddrCmd(void)
Read Buffer Size	void HciReadBufSizeCmd(void)
Read Local Supported Features	void HciReadLocalSupFeatCmd(void)
Read Local Version Information	void HciReadLocalVerInfoCmd(void)
Read Remote Version Information	void HciReadRemoteVerInfoCmd(uint16_t handle)
Read RSSI	void HciReadRssiCmd(uint16_t handle)
Read Transmit Power Level	void HciReadTxPwrLvlCmd(uint16_t handle, uint8_t type)
Reset	void HciResetCmd(void)
Set Controller To Host Flow Control	[not implemented]
Set Event Mask	void HciSetEventMaskCmd(uint8_t *pEventMask)
Vendor Specific	void HciVendorSpecificCmd(uint16_t opcode, uint8_t len, uint8_t *pData)

5 Event Interface

The event interface defines event data structures which are passed from HCI to the stack. HCI events and their parameters defined in [1] are mapped to internal event values and data structures which can be processed efficiently by the stack.

5.1 Event Values

The following internal event values are used in the HCI event and security callbacks.

Event Name	Description
HCI_RESET_SEQ_CMPL_CBACK_EVT	Reset sequence complete
HCI_LE_CONN_CMPL_CBACK_EVT	LE connection complete
HCI_DISCONNECT_CMPL_CBACK_EVT	LE disconnect complete
HCI_LE_CONN_UPDATE_CMPL_CBACK_EVT	LE connection update complete
HCI_LE_CREATE_CONN_CANCEL_CMD_CMPL_CBACK_EVT	LE create connection cancel command complete
HCI_LE_ADV_REPORT_CBACK_EVT	LE advertising report
HCI_READ_RSSI_CMD_CMPL_CBACK_EVT	Read RSSI command complete
HCI_LE_READ_CHAN_MAP_CMD_CMPL_CBACK_EVT	LE Read channel map command complete
HCI_READ_TX_PWR_LVL_CMD_CMPL_CBACK_EVT	Read transmit power level command complete

HCI_READ_REMOTE_VER_INFO_CMPL_CBACK_EVT	Read remote version information complete
HCI_LE_READ_REMOTE_FEAT_CMPL_CBACK_EVT	LE read remote features complete
HCI_LE_LTK_REQ_REPL_CMD_CMPL_CBACK_EVT	LE LTK request reply command complete
HCI_LE_LTK_REQ_NEG_REPL_CMD_CMPL_CBACK_EVT	LE LTK request negative reply command complete
HCI_ENC_KEY_REFRESH_CMPL_CBACK_EVT	Encryption key refresh complete
HCI_ENC_CHANGE_CBACK_EVT	Encryption change
HCI_LE_LTK_REQ_CBACK_EVT	LE LTK request
HCI_VENDOR_SPEC_CMD_STATUS_CBACK_EVT	Vendor specific command status
HCI_VENDOR_SPEC_CMD_CMPL_CBACK_EVT	Vendor specific command complete
HCI_VENDOR_SPEC_CBACK_EVT	Vendor specific
HCI_HW_ERROR_CBACK_EVT	Hardware error
HCI_LE_ENCRYPT_CMD_CMPL_CBACK_EVT	LE encrypt command complete
HCI_LE_RAND_CMD_CMPL_CBACK_EVT	LE rand command complete

5.2 Event Data Types

The following data types are used in the event interface. Event parameters are as defined [1]. The event header using wsfMsgHdr_t is set as follows for all events:

Type	Name	Value
uint16_t	param	HCI handle, if applicable.
uint8_t	event	Event value. See section 5.1.
uint8_t	status	HCI event status code, if applicable.

5.2.1 hciLeConnCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle
uint8_t	role
uint8_t	addrType
bdAddr_t	peerAddr
uint16_t	connInterval
uint16_t	connLatency
uint16_t	supTimeout
uint8_t	clockAccuracy

5.2.2 hciDisconnectCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle
uint8_t	reason

5.2.3 hciLeConnUpdateCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	subevent
uint8_t	status
uint16_t	handle
uint16_t	connInterval
uint16_t	connLatency
uint16_t	supTimeout

5.2.4 hciLeCreateConnCancelCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status

5.2.5 hciLeAdvReport_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t *	pData
uint8_t	len
int8_t	rssi
uint8_t	eventType
uint8_t	addrType
bdAddr_t	addr

5.2.6 hciReadRssiCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle
int8_t	rssi

5.2.7 hciLeReadChanMapCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle
uint8_t	chanMap[5]

5.2.8 hciReadTxPwrLevelCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle
int8_t	pwrLevel

5.2.9 hciReadRemoteVerInfoCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle
uint8_t	version
uint16_t	mfrName
uint16_t	subversion

5.2.10 hciLeReadRemoteFeatCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle
uint8_t	features[8]

5.2.11 hciLeLtkReqReplCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle

5.2.12 hciLeLtkReqNegReplCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle

5.2.13 hciEncKeyRefreshCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle

5.2.14 hciEncChangeEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle
uint8_t	enabled

5.2.15 hciLeLtkReqEvt_t

Type	Name
wsfMsgHdr_t	hdr

uint16_t	handle
uint8_t	randNum[8]
uint16_t	encDiversifier

5.2.16 hciVendorSpecCmdStatusEvt_t

Type	Name
wsfMsgHdr_t	hdr
Uint16_t	opcode

5.2.17 hciVendorSpecCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint16_t	opcode
uint8_t	param[]

5.2.18 hciVendorSpecEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint16_t	opcode
uint8_t	param[]

5.2.19 hciHwErrorEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	code

5.2.20 hciLeEncryptCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint8_t	encData[16]

5.2.21 hciLeRandCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint8_t	randNum[16]

5.3 hciEvt_t

This data type is a union of all event data types. It is used in the HCI event and security callbacks.

6 ACL Data Interface

The ACL data interface contains the following functions: An API function for sending data to HCI, a callback function for receiving data from HCI, and a callback function for managing flow control.

6.1 void HciSendAclData(uint8_t *pAclData)

This function sends data from the stack to HCI. Parameter pAclData points to a WSF buffer containing an ACL packet. The ACL packet is formatted as defined in [1]: The first two bytes of the buffer contain the handle for the ACL connection and the next two bytes of the buffer contain the length.

The caller of this function is responsible for allocating the WSF buffer pointed to by pAclData. HCI is responsible for deallocating the buffer.

6.2 void (*hciAclCback_t)(uint8_t *pData)

This callback function sends data from HCI to the stack. Parameter pData points to a WSF buffer containing an ACL packet. The ACL packet is formatted as defined in [1]: The first two bytes of the buffer contain the handle for the ACL connection and the next two bytes of the buffer contain the length.

HCI allocates the WSF buffer pointed to by pData. The stack is responsible for deallocating the buffer.

- **pData**: WSF buffer containing an ACL packet.

6.3 void (*hciFlowCback_t)(uint16_t handle, bool_t flowDisabled)

This callback function manages flow control in the TX path between the stack and HCI. If parameter flowDisabled is TRUE then the stack cannot send ACL data to HCI. If flowDisabled is FALSE then data flow can resume.

- **handle**: The connection handle.
- **flowDisabled**: TRUE if data flow is disabled.

7 Event Callback Interface

7.1 void (*hciEvtCback_t)(hciEvt_t *pEvent)

This callback function sends events from HCI to the stack.

- **pEvent**: Pointer to HCI callback event structure.

7.2 void(*hciSecCback_t)(hciEvt_t *pEvent)

This callback function sends certain security events from HCI to the stack. The security events passed in this callback are the LE Rand Command Complete event and the LE Encrypt Command Complete event.

- **pEvent**: Pointer to HCI callback event structure.

8 Scenarios

8.1 Reset

Figure 2 shows the operation of the reset sequence. First, the DM subsystem of the stack calls HciResetSequence() to initiate the reset sequence. HCI begins sending a sequence of HCI commands to the controller, starting with the HCI Reset command. After each command a Command Complete event is received. HCI continues sending commands until it has sent all the commands in its sequence. When it has received a Command Complete event for the last command it calls the event callback and sends a Reset Sequence Complete event.

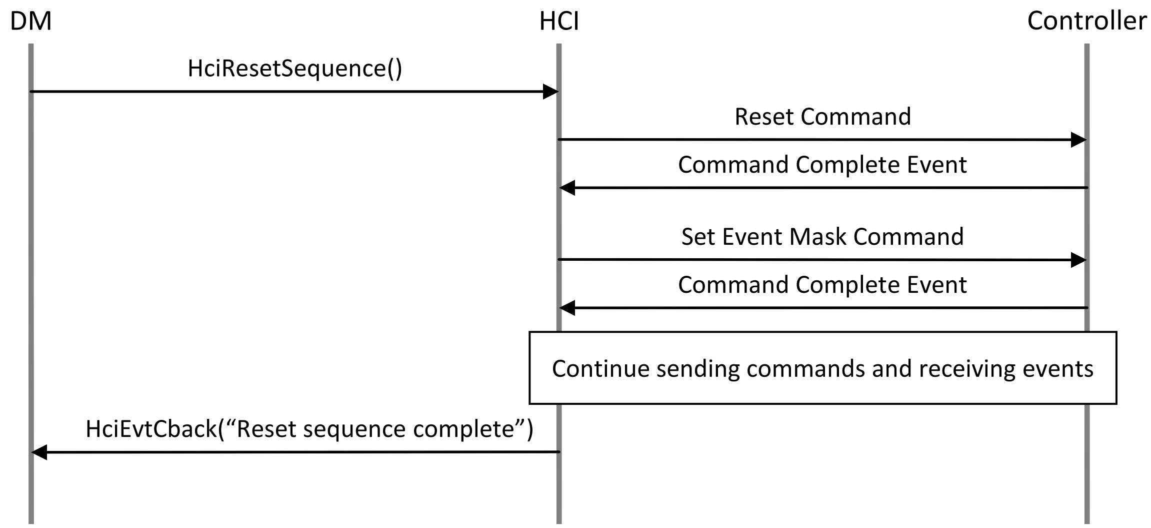

Figure 2.

8.2 HCI Command and Event

Figure 3 shows an HCI command and event. The DM subsystem of the stack calls an HCI function to create a connection. HCI then sends an LE Create Connection command to the controller. The controller responds with a Command Status event. Note that this event is not sent to the stack; it is processed internally by HCI. Then the controller sends an LE Connection Complete event. HCI then calls the event callback and sends an LE Connection Complete event to the stack.

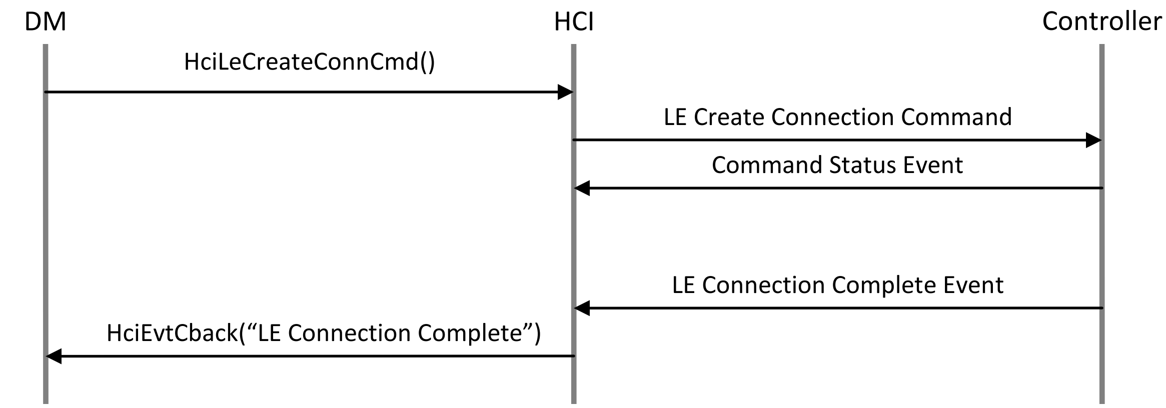

Figure 3.

8.3 ACL Data Transmit and Receive

Figure 4 shows ACL data transmit and receive. The L2CAP layer of the stack calls function HciSendAclData() to send data from the stack to HCI. HCI builds and sends an ACL data packet to the controller. The controller then sends a Number of Completed Packets event to HCI and HCI processes this event internally without passing it to the stack.

For receive data, the controller sends an ACL data packet to HCI, processes the packet and calls the ACL data callback to send the packet to L2CAP.

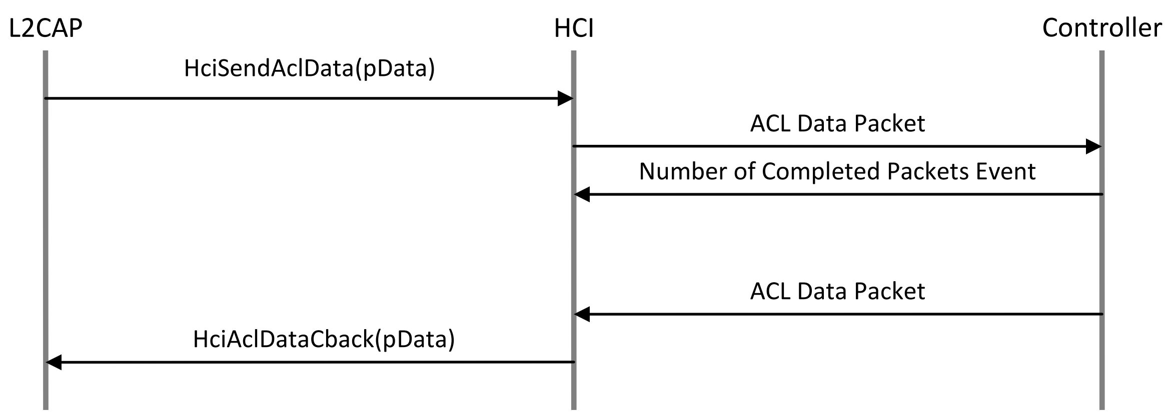

Figure 4.

9 References

1. Bluetooth SIG, "Specification of the Bluetooth System", Version 4.2, December 2, 2015.

10 Definitions

ACL	Asynchronous Connectionless data packet
DM	Device Manager software subsystem
HCI	Host Controller Interface
L2C	L2CAP software subsystem
L2CAP	Logical Link Control Adaptation Protocol
LE	(Bluetooth) Low Energy
WSF	Wicentric Software Foundation software service and porting layer