ARM® Cordio Stack

ARM-EPM-115877 1.0

HCI API

Confidential

ARM® Cordio Stack HCI API

Reference Manual

Release Information

The following changes have been made to this book:

Document History

Date	Issue	Confidentiality	Change
21 January 2016	**-**	**Confidential**	**First Wicentric release for 1.3.**
1 March 2016	**A**	**Confidential**	**First ARM release for 1.3.**
24 August 2016	**A**	**Confidential**	**AUSPEX # / API Update**

Proprietary Notice

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20348

Confidentiality Status

This document is Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

http://www.arm.com

Table of Contents

1 Preface

This preface introduces the Cordio Stack HCI API Reference Manual.

1.1 About this book

This document describes the API for the host controller interface (HCI) layer of ARM's Bluetooth LE protocol stack and lists the API functions and their parameters.

1.1.1 Intended audience

This book is written for experienced software engineers who might or might not have experience with ARM products. Such engineers typically have experience of writing Bluetooth applications but might have limited experience of the Cordio software stack.

It is also assumed that the readers have access to all necessary tools.

1.1.2 Using this book

This book is organized into the following chapters:

- **Introduction**
  Read this for an overview of the HCI API.
- **Initialization, Registration, and Reset**
  Read this for an overview of the functions that initialize the HCI system.
- **Optimization Interface**
  Read this for an overview of the optimized interface for the some simple HCI commands.
- **Command Interface**
  Read this for an overview of the functions that map directly to HCI commands.
- **Event Interface**
  Read this for an overview of the data structures passed from the HCI to the stack.
- **ACL Data Intervace**
  Read this for an overview of the functions that send and receive data.
- **Event Callback Intervace**
  Read this for an overview of the function that sends events to the stack.
- **Scenarios**
  Read this for an overview of how APIs are used in different scenarios.
- **Revisions**
  Read this chapter for descriptions of the changes between document versions.

1.1.3 Terms and abbreviations

For a list of ARM terms, see the ARM glossary.

Terms specific to the Cordio software are listed below:

Term	Description
ACL	Asynchronous Connectionless data packet

AD	Advertising Data
AE	Advertising Extensions
ARQ	Automatic Repeat reQuest
ATT	Attribute Protocol, also attribute protocol software subsystem
ATTC	Attribute Protocol Client software subsystem
ATTS	Attribute Protocol Server software subsystem
CCC or CCCD	Client Characteristic Configuration Descriptor
CID	Connection Identifier
CSRK	Connection Signature Resolving Key
DM	Device Manager software subsystem
GAP	Generic Access Profile
GATT	Generic Attribute Profile
HCI	Host Controller Interface
IRK	Identity Resolving Key
JIT	Just In Time
L2C	L2CAP software subsystem
L2CAP	Logical Link Control Adaptation Protocol
LE	(Bluetooth) Low Energy
LL	Link Layer
LLPC	Link Layer Control Protocol
LTK	Long Term Key
MITM	Man In The Middle pairing (authenticated pairing)
OOB	Out Of Band data
SMP	Security Manager Protocol, also security manager protocol software subsystem
SMPI	Security Manager Protocol Initiator software subsystem
SMPR	Security Manager Protocol Responder software subsystem
STK	Short Term Key
WSF	Wireless Software Foundation software service and porting layer.

1.1.4 Conventions

The following table describes the typographical conventions:

Typographical conventions

Style	Purpose
Italic	Introduces special terminology, denotes cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`MONOSPACE`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`MONO`SPACE	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace` *`italic`*	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

1.1.5 Additional reading

This section lists publications by ARM and by third parties.

See *Infocenter* for access to ARM documentation.

Other publications

This section lists relevant documents published by third parties:

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

1.2 Feedback

ARM welcomes feedback on this product and its documentation.

1.2.1 Feedback on content

If you have comments on content then send an e-mail to `errata@arm.com`. Give:

- The title.
- The number, ARM-EPM-115147.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note: ARM tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

2 Introduction

2.1 Overview

This document describes the API for the *Host Controller Interface* (HCI) layer of ARM'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.

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 *Specification of the Bluetooth System*.

2.2 HCI topologies

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

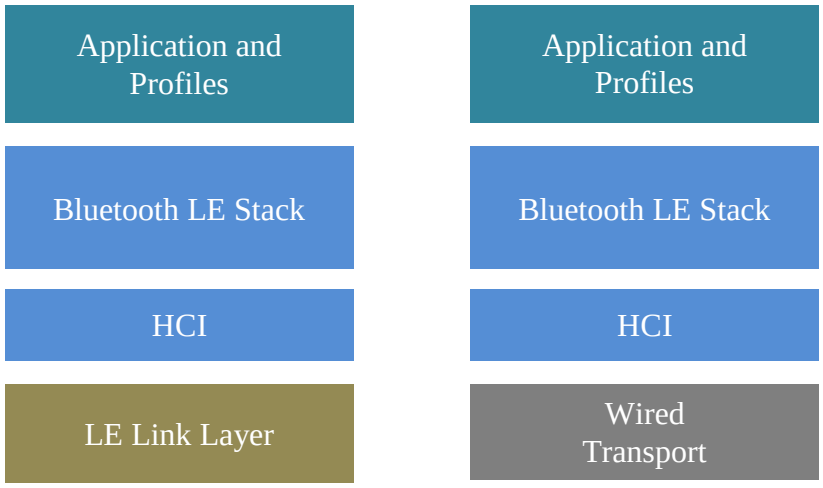

Figure 1. HCI topologies

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.

2.3 Basic data types

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

Table 1: Integer types

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

The following type is used for the Bluetooth device address.

Table 2: Bluetooth address type

Type	Name
`uint8_t`	`bdAddr_t[6]`

3 Initialization, Registration and Reset

3.1 HciHandlerInit()

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.

Syntax:

```
void HciHandlerInit(wsfHandlerId_t handlerId)
```

Where:

- `handlerId`: ID of WSF handler.

3.2 HciEvtRegister()

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

Syntax:

```
void HciEvtRegister(hciEvtCback_t evtCback)
```

Where:

- `evtCback`: Client callback function. See 8.1.

3.3 HciSecRegister()

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

Syntax:

```
void HciSecRegister(hciSecCback_t secCback)
```

Where:

- `secCback`: Client callback function. See 8.2.

3.4 HciAclRegister()

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

Syntax:

```
void HciAclRegister(hciAclCback_t aclCback, hciFlowCback_t flowCback)
```

Where:

- `aclCback`: Client ACL data callback function. See 7.2.
- `flowCback`: Client flow control callback function. See 7.3.

3.5 HciResetSequence()

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.

Syntax:

```
void HciResetSequence(void)
```

A typical reset sequence is as follows:

1. Reset Command
2. Set Event Mask Command
3. LE Set Event Mask Command
4. Set Event Mask Page 2 Command
5. Read BD_ADDR Command
6. LE Read Buffer Size Command
7. Read Buffer Size Command
8. LE Read Supported States Command
9. LE Read White List Size Command
10. LE Read Local Supported Feature Command
11. LE Read Resolving List Size Command
12. LE Read Maximum Data Length Command
13. LE Write Suggested Default Data Length Command
14. LE Random Command

3.6 HciVsInit()

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

Syntax:

```
void HciVsInit(uint8_t param)
```

Where:

- `param`: Vendor-specific parameter.

3.7 HciSetMaxRxAclLen()

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

Syntax:

```
void HciSetMaxRxAclLen(uint16_t len)
```

Where:

- `len`: ACL packet length.

3.8 HciSetAclQueueWatermarks()

Set TX ACL queue high and low watermarks.

Syntax:

```
void HciSetAclQueueWatermarks(uint8_t queueHi, uint8_t queueLo)
```

Where:

- `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.

4 Optimization interface

This is an optimized interface for certain HCI commands that 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.

4.1 HciGetBdAddr()

Return a pointer to the BD address of this device.

Syntax:

```
uint8_t *HciGetBdAddr(void)
```

4.2 HciGetWhiteListSize()

Return the white list size.

Syntax:

```
uint8_t HciGetWhiteListSize(void)
```

4.3 HciGetAdvTxPwr()

Return the advertising transmit power. See the LE Read Advertising Channel TX Power command in the *Specification of the Bluetooth System* document for the format of the value.

Syntax:

```
int8_t HciGetAdvTxPwr(void)
```

4.4 HciGetBufSize()

Return the ACL buffer size supported by the controller.

Syntax:

```
uint16_t HciGetBufSize(void)
```

4.5 HciGetNumBufs()

Return the number of ACL buffers supported by the controller.

Syntax:

```
uint8_t HciGetNumBufs(void)
```

4.6 HciGetSupStates()

Return the states supported by the controller. See the LE Read Supported States command in the *Specification of the Bluetooth System* document for the format of the value.

Syntax:

```
uint8_t *HciGetSupStates(void)
```

4.7 HciGetLeSupFeat()

Return the LE features supported by the controller. See the LE Read Local Supported Features command in the *Specification of the Bluetooth System* document for the format of the value.

Syntax:

```
uint8_t HciGetLeSupFeat(void)
```

4.8 HciGetMaxRxAclLen()

Get the maximum reassembled RX ACL packet length.

Syntax:

```
uint16_t HciGetMaxRxAclLen(void)
```

4.9 HciGetResolvingListSize()

Return the resolving list size supported by the controller.

Syntax:

```
uint8_t HciGetResolvingListSize(void)
```

4.10 HciLlPrivacySupported()

Return whether or not the LL Privacy feature is supported by the controller.

Syntax:

```
bool_t HciLlPrivacySupported(void)
```

4.11 HciGetMaxAdvDataLen()

Get the maximum advertisement (or scan response) data length supported by the controller.

Syntax:

```
uint16_t HciGetMaxAdvDataLen(void)
```

4.12 HciGetNumSupAdvSets()

Get the maximum number of advertising sets supported by the Controller.

Syntax:

```
uint8_t HciGetNumSupAdvSets(void)
```

4.13 HciLeAdvExtSupported()

Return whether or not the LE Advertising Extensions feature is supported by the controller.

Syntax:

```
bool_t HciLeAdvExtSupported(void)
```

5 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 *Specification of the Bluetooth System* document.

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, for example `HciGetBdAddr()`, and might not need to implement the corresponding functions in the command interface (for example HciReadBdAddrCmd()).

5.1 Data structures

5.1.1 hciConnSpec_t

This data structure is used in functions `HciLeCreateConnCmd()` and `HciLeConnUpdateCmd()`. See the *Specification of the Bluetooth System* document for a description of the parameters of this structure.

Table 3: hciConnSpec_t structure

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

5.2 Functions

The command interface functions are shown in the table below. See the *Specification of the Bluetooth System* document for a description of the parameters and operation of these functions. Functions shown as *not implemented* are not used by ARM's Bluetooth LE stack.

5.2.1 HCI Commands

Table 4: HCI commands

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)
Remote Connection Parameter Request Reply	void HciLeRemoteConnParamReqReply(uint16_t handle, uint16_t intervalMin, uint16_t intervalMax, uint16_t latency, uint16_t timeout, uint16_t minCeLen, uint16_t maxCeLen)
Remote Connection Parameter Request Negative Reply	void HciLeRemoteConnParamReqNegReply(uint16_t handle, uint8_t reason)
Set Data Length	void HciLeSetDataLen(uint16_t handle, uint16_t txOctets, uint16_t txTime)
Read Suggested Default Data Length	void HciLeReadDefDataLen(void)
Write Suggested Default Data Length	void HciLeWriteDefDataLen(uint16_t suggestedMaxTxOctets, uint16_t suggestedMaxTxTime)
Read Local P-256 Public Key	void HciLeReadLocalP256PubKey(void)
Generate DH Key	void HciLeGenerateDHKey(uint8_t *pPubKeyX, uint8_t *pPubKeyY)
Read Maximum Data Length	void HciLeReadMaxDataLen(void)
LE Encrypt	void HciLeEncryptCmd(uint8_t *pKey, uint8_t *pData)
LE Long Term Key	void HciLeLtkReqNegReplCmd(uint16_t handle)

Requested Negative Reply	
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)
Set Event Mask Page 2	void HciSetEventMaskPage2Cmd(uint8_t *pEventMask)
Add Device to Resolving List	void HciLeAddDeviceToResolvingListCmd(uint8_t peerAddrType, const uint8_t *pPeerIdentityAddr, const uint8_t *pPeerIrk, const uint8_t *pLocalIrk)
Remove Device from Resolving List	void HciLeRemoveDeviceFromResolvingList(uint8_t peerAddrType, const uint8_t *pPeerIdentityAddr)
Clear Resolving List	void HciLeClearResolvingList(void)
Read Resolving List Size	void HciLeReadResolvingListSize(void)

Read Peer Resolvable Address	void HciLeReadPeerResolvableAddr(uint8_t addrType, const uint8_t *pIdentityAddr)
Read Local Resolvable Address	void HciLeReadLocalResolvableAddr(uint8_t addrType, const uint8_t *pIdentityAddr)
Enable or Disable Address Resolution	void HciLeSetAddrResolutionEnable(uint8_t enable)
Set Resolvable Private Address Timeout	void HciLeSetResolvablePrivateAddrTimeout(uint16_t rpaTimeout)
Read Authenticated Payload Timeout	void HciReadAuthPayloadTimeout(uint16_t handle)
Write Authenticated Payload Timeout	void HciWriteAuthPayloadTimeout(uint16_t handle, uint16_t timeout)
HCI LE set advertising set random device address command	void HciLeSetAdvSetRandAddrCmd(uint8_t advHandle, const uint8_t *pAddr);
HCI LE set extended advertising parameters command	void HciLeSetExtAdvParamCmd(uint8_t advHandle, hciExtAdvParam_t *pExtAdvParam);
HCI LE set extended advertising data command	void HciLeSetExtAdvDataCmd(uint8_t advHandle, uint8_t op, uint8_t fragPref, uint8_t len, const uint8_t *pData);
HCI LE set extended scan response data command	void HciLeSetExtScanRespDataCmd(uint8_t advHandle, uint8_t op, uint8_t fragPref, uint8_t len, const uint8_t *pData);
HCI LE set extended advertising enable command	void HciLeSetExtAdvEnableCmd(uint8_t enable, uint8_t numSets, hciExtAdvEnableParam_t *pEnableParam);
HCI LE read maximum advertising data length command	void HciLeReadMaxAdvDataLen(void);
HCI LE read number of supported advertising sets command	void HciLeReadNumSupAdvSets(void);
HCI LE remove advertising set command	void HciLeRemoveAdvSet(uint8_t advHandle);
HCI LE clear advertising	void HciLeClearAdvSets(void);

sets command	
HCI LE read transmit power command	void HciLeReadTxPower(void);
HCI LE read RF path compensation command	void HciLeReadRfPathComp(void);
HCI LE write RF path compensation command	void HciLeWriteRfPathComp(int16_t txPathComp, int16_t rxPathComp);
HCI LE set extended scanning parameters command	void HciLeSetExtScanParamCmd(uint8_t scanPhys, hciExtScanParam_t *pScanParam);
HCI LE extended scan enable command	void HciLeExtScanEnableCmd(uint8_t enable, uint8_t filterDup, uint16_t duration, uint16_t period);
HCI LE extended create connection command	void HciLeExtCreateConnCmd(uint8_t initPhys, hciExtCreateConnParam_t *pConnParam, uint8_t peerAddrType, uint8_t *pPeerAddr, hciConnSpec_t *pConnSpec);
Vendor Specific	void HciVendorSpecificCmd(uint16_t opcode, uint8_t len, uint8_t *pData)

5.2.2 HCI AE Commands

Table 5: HCI AE Commands

AE Command	Function
HCI LE set advertising random address command	void HciLeSetAdvSetRandAddrCmd(uint8_t advHandle, const uint8_t *pAddr)
HCI LE set extended advertising parameters command	void HciLeSetExtAdvParamCmd(uint8_t advHandle, hciExtAdvParam_t *pExtAdvParam)
HCI LE set extended advertising data command	void HciLeSetExtAdvDataCmd(uint8_t advHandle, uint8_t op, uint8_t fragPref, uint8_t len, const uint8_t *pData)
HCI LE set extended scan response data command	void HciLeSetExtScanRespDataCmd(uint8_t advHandle, uint8_t op, uint8_t fragPref, uint8_t len, const uint8_t *pData)
HCI LE set extended advertising enable command	void HciLeSetExtAdvEnableCmd(uint8_t enable, uint8_t numSets, hciExtAdvEnableParam_t *pEnableParam)
HCI LE read maximum advertising data length command	void HciLeReadMaxAdvDataLen(void)

HCI LE read number of supported advertising sets command	`void HciLeReadNumSupAdvSets(void)`
HCI LE remove advertising set command	`void HciLeRemoveAdvSet(uint8_t advHandle)`
HCI LE clear advertising sets command	`void HciLeClearAdvSets(void)`
HCI LE read transmit power command	`void HciLeReadTxPower(void)`
HCI LE read RF path compensation command	`void HciLeReadRfPathComp(void)`
HCI LE write RF path compensation command	`void HciLeWriteRfPathComp(int16_t txPathComp, int16_t rxPathComp)`
HCI LE set extended scanning parameters command	`void HciLeSetExtScanParamCmd(uint8_t scanPhys, hciExtScanParam_t *pScanParam)`
HCI LE extended scan enable command	`void HciLeExtScanEnableCmd(uint8_t enable, uint8_t filterDup, uint16_t duration, uint16_t period)`
HCI LE extended create connection command	`void HciLeExtCreateConnCmd(uint8_t initPhys, hciExtCreateConnParam_t *pConnParam, uint8_t peerAddrType, uint8_t *pPeerAddr, hciConnSpec_t *pConnSpec)`

6 Event interface

The event interface defines event data structures which are passed from HCI to the stack. HCI events and their parameters defined in the *Specification of the Bluetooth System* document are mapped to internal event values and data structures that can be processed efficiently by the stack.

6.1 Event values

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

Table 6: 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_LE_ENHANCED_CONN_CMPL_CBACK_EVT	LE enhanced 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_ADD_DEV_TO_RES_LIST_CMD_CMPL_CBACK_EVT	LE add device to resolving list command complete
HCI_LE_REM_DEV_FROM_RES_LIST_CMD_CMPL_CBACK_EVT	LE remove device from resolving command complete
HCI_LE_CLEAR_RES_LIST_CMD_CMPL_CBACK_EVT	LE clear resolving list command complete
HCI_LE_READ_PEER_RES_ADDR_CMD_CMPL_CBACK_EVT	LE read peer resolving address command complete
HCI_LE_READ_LOCAL_RES_ADDR_CMD_CMPL_CBACK_EVT	LE read local resolving address command complete
HCI_LE_SET_ADDR_RES_ENABLE_CMD_CMPL_CBACK_EVT	LE set address resolving enable command complete
HCI_LE_ENCRYPT_CMD_CMPL_CBACK_EVT	LE encrypt command complete
HCI_LE_RAND_CMD_CMPL_CBACK_EVT	LE rand command complete
HCI_LE_REM_CONN_PARAM_REP_CMD_CMPL_CBACK_EVT	LE remote connection parameter request reply complete
HCI_LE_REM_CONN_PARAM_NEG_REP_CMD_CMPL_CBACK_EVT	LE remote connection parameter request negative reply complete
HCI_LE_READ_DEF_DATA_LEN_CMD_CMPL_CBACK_EVT	LE read suggested default data length command complete
HCI_LE_WRITE_DEF_DATA_LEN_CMD_CMPL_CBACK_EVT	LE write suggested default data length command complete
HCI_LE_SET_DATA_LEN_CMD_CMPL_CBACK_EVT	LE set data length command complete
HCI_LE_READ_MAX_DATA_LEN_CMD_CMPL_CBACK_EVT	LE read maximum data length command complete
HCI_LE_REM_CONN_PARAM_REQ_CBACK_EVT	LE remote connection parameter request
HCI_LE_DATA_LEN_CHANGE_CBACK_EVT	LE data length change
HCI_LE_READ_LOCAL_P256_PUB_KEY_CMPL_CBACK_EVT	LE read local P256 public key
HCI_LE_GENERATE_DHKEY_CMPL_CBACK_EVT	LE generate DHKey complete
HCI_WRITE_AUTH_PAYLOAD_TO_CMD_CMPL_CBACK_EVT	Write authenticated payload timeout command complete
HCI_AUTH_PAYLOAD_TO_EXPIRED_CBACK_EVT	Authenticated payload timeout expired
HCI_LE_READ_PHY_CMD_COMPL_CBACK_EVT	LE read phy command complete
HCI_LE_SET_DEF_PHY_CMD_CMPL_CBACK_EVT	LE set default phy command complete

HCI_LE_PHY_UPDATE_CMPL_CBACK_EVT	LE phy update complete
HCI_LE_EXT_ADV_REPORT_CBACK_EVT	LE extended advertising report
HCI_LE_SCAN_TIMEOUT_CBACK_EVT	LE scan timeout event
HCI_LE_ADV_SET_TERM_CBACK_EVT	LE advertising set terminated event
HCI_LE_SCAN_REQ_RCVD_CBACK_EVT	LE scan request received event

6.2 Event data types

The following data types are used in the event interface. Event parameters are as defined in the *Specification of the Bluetooth System* document . The event header using `wsfMsgHdr_t` is set as follows for all events:

Table 7: wsfMsgHdr_t

Type	Name	Value
uint16_t	param	HCI handle, if applicable.
uint8_t	event	Event value. See section 6.1.
uint8_t	status	HCI event status code, if applicable.

Table 8: 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
// enhanced fields	
bdAddr_t	localRpa
bdAddr_t	peerRpa

Table 9: hciDisconnectCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle
uint8_t	reason

Table 10: hciLeConnUpdateCmplEvt_t

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

Table 11: hciLeCreateConnCancelCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status

Table 12: hciLeAdvReportEvt_t

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

// direct fields	
uint8_t	directAddrType
bdAddr_t	directAddr

Table 13: hciLeExtAdvReportEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint16_t	eventType
uint8_t	addrType
bdAddr_t	addr
uint8_t	**priPhy**
uint8_t	**secPhy**
uint8_t	**advSid**
int8_t	**txPower**
int8_t	Rssi
int16_t	**perAdvInter**
uint8_t	**directAddrType**
bdAddr_t	directAddr
uint8_t	Len
uint8_t *	pData

Table 14: hciLeScanTimeoutEvt_t

Type	Name
wsfMsgHdr_t	hdr

Table 15: hciLeAdvSetTermEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint8_t	advHandle

uint16_t	handle
uint8_t	numComplEvts

Table 16: hciLeScanReqRcvdEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	advHandle
uint8_t	scanAddrType
bdAddr_t	scanAddr

Table 17: hciReadRssiCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle
int8_t	rssi

Table 18: hciLeReadChanMapCmdCmplEvt_t

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

Table 19: hciReadTxPwrLevelCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle
int8_t	pwrLevel

Table 20: hciReadRemoteVerInfoCmplEvt_t

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

Table 21: hciLeReadRemoteFeatCmplEvt_t

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

Table 22: hciLeLtkReqReplCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle

Table 23: hciLeLtkReqNegReplCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle

Table 24: hciEncKeyRefreshCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle

Table 25: hciEncChangeEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle
uint8_t	enabled

Table 26: hciLeLtkReqEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint16_t	handle
uint8_t	randNum[HCI_RAND_LEN]
uint16_t	encDiversifier

Table 27: hciVendorSpecCmdStatusEvt_t

Type	Name
wsfMsgHdr_t	hdr
Uint16_t	opcode

Table 28: hciVendorSpecCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint16_t	opcode
uint8_t	param[1]

Table 29: hciVendorSpecEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	param[1]

Table 30: hciHwErrorEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	code

Table 31: hciLeEncryptCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint8_t	data[HCI_ENCRYPT_DATA_LEN]

Table 32: hciLeRandCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint8_t	randNum[HCI_RAND_LEN]

Table 33: hciLeAddDevToResListCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status

Table 34: hciLeRemDevFromResListCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status

Table 35: hciLeClearResListCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status

Table 36: hciLeReadPeerResAddrCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint8_t	peerRpa[BDA_ADDR_LEN]

Table 37: hciLeReadLocalResAddrCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint8_t	localRpa[BDA_ADDR_LEN]

Table 38: hciLeSetAddrResEnableCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status

Table 39: hciLeRemConnParamRepEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle

Table 40: hciLeRemConnParamNegRepEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle

Table 41: hciLeReadDefDataLenEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	suggestedMaxTxOctets
uint16_t	suggestedMaxTxTime

Table 42: hciLeWriteDefDataLenEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	Status

Table 43: hciLeSetDataLenEvt_t

Type	Name
wsfMsgHdr_t	hdr

uint8_t	status
uint16_t	handle

Table 44: hciLeReadMaxDataLenEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	supportedMaxTxOctets
uint16_t	supportedMaxTxTime
uint16_t	supportedMaxRxOctets
uint16_t	supportedMaxRxTime

Table 45: hciLeRemConnParamReqEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint16_t	handle
uint16_t	intervalMin
uint16_t	intervalMax
uint16_t	latency
uint16_t	timeout

Table 46: hciLeDataLenChangeEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint16_t	handle
uint16_t	maxTxOctets
uint16_t	maxTxTime

uint16_t	maxRxOctets
uint16_t	maxRxTime

Table 47: hciLeP256CmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	Status
uint8_t	key[HCI_P256_KEY_LEN]

Table 48: hciLeGenDhKeyEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint8_t	key[HCI_DH_KEY_LEN]

Table 49: hciWriteAuthPayloadToCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle

Table 50: hciAuthPayloadToExpiredEv_t

Type	Name
wsfMsgHdr_t	hdr
Uint16_t	handle

Table 51: hciLeReadPhyCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle
uint8_t	txPhy
uint8_t	rxPhy

Table 52: hciLeSetDefPhyCmdCmplEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status

Table 53: hciLePhyUpdateEvt_t

Type	Name
wsfMsgHdr_t	hdr
uint8_t	status
uint16_t	handle
uint8_t	txPhy
uint8_t	rxPhy

6.3 hciEvt_t

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

7 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.

7.1 HciSendAclData()

This function sends data from the stack to HCI.

Syntax:

```
void HciSendAclData(uint8_t *pData)
```

Where:

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

The ACL packet is formatted as defined in the *Specification of the Bluetooth System* document:

- The first two bytes of the buffer contain the handle for the ACL connection.
- 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.

7.2 (*hciAclCback_t)()

This callback function sends data from HCI to the stack.

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

Syntax:

```
void (*hciAclCback_t)(uint8_t *pData)
```

Where:

- `pData`: Points to a WSF buffer containing an ACL packet.
    - The first two bytes of the buffer contain the handle for the ACL connection.
    - The next two bytes of the buffer contain the length.

  The ACL packet is formatted as defined the *Specification of the Bluetooth System* document:

7.3 (*hciFlowCback_t)()

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.

Syntax:

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

Where:

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

8 Event callback interface

8.1 (*hciEvtCback_t)()

This callback function sends events from HCI to the stack.

Syntax:

```
void (*hciEvtCback_t)(hciEvt_t *pEvent)
```

Where:

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

8.2 (*hciSecCback_t)()

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.

Syntax:

```
void(*hciSecCback_t)(hciEvt_t *pEvent)
```

Where:

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

9 Scenarios

9.1 Reset

Figure 2 shows the operation of the reset sequence.

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

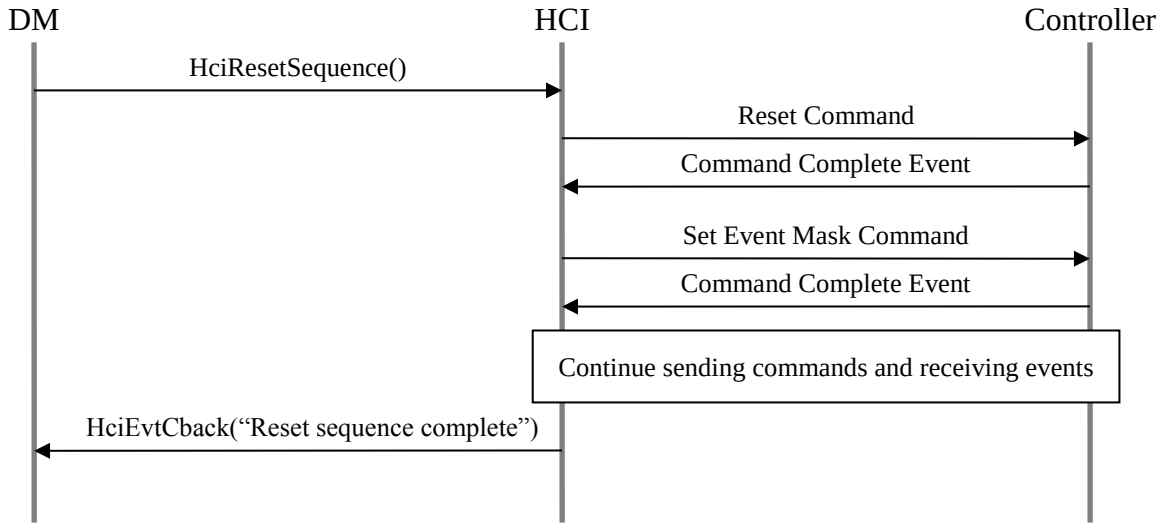

Figure 2. Reset

9.2 HCI command and event

Figure 3 shows an HCI command and event.

1. The DM subsystem of the stack calls an HCI function to create a connection.
2. HCI sends an LE Create Connection command to the controller.
3. The controller responds with a Command Status event.
   Note: This event is not sent to the stack; it is processed internally by HCI.
4. The controller sends an LE Connection Complete event.
5. HCI calls the event callback and sends an LE Connection Complete event to the stack.

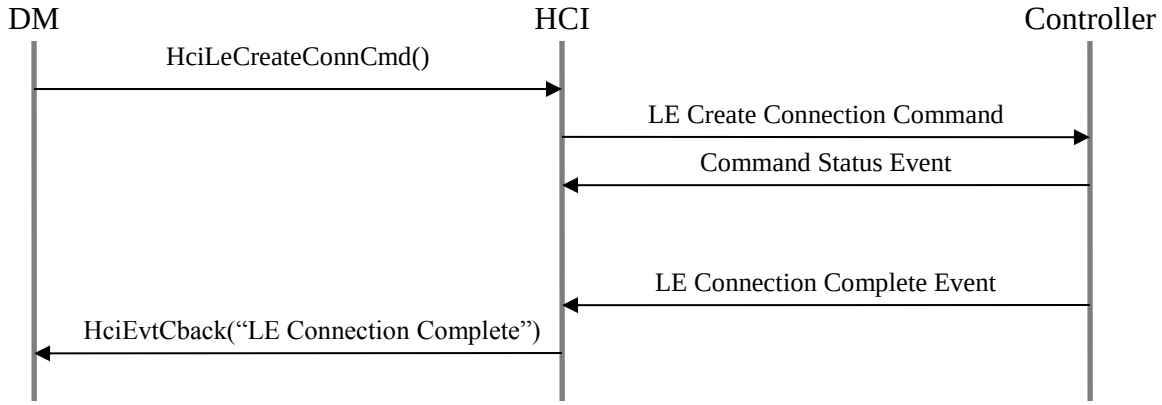

Figure 3. HCI command and resulting event

9.3 ACL data transmit and receive

Figure 4 shows ACL data transmit and receive.

1. The L2CAP layer of the stack calls function `HciSendAclData()` to send data from the stack to HCI.
2. HCI builds and sends an ACL data packet to the controller.
3. 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.

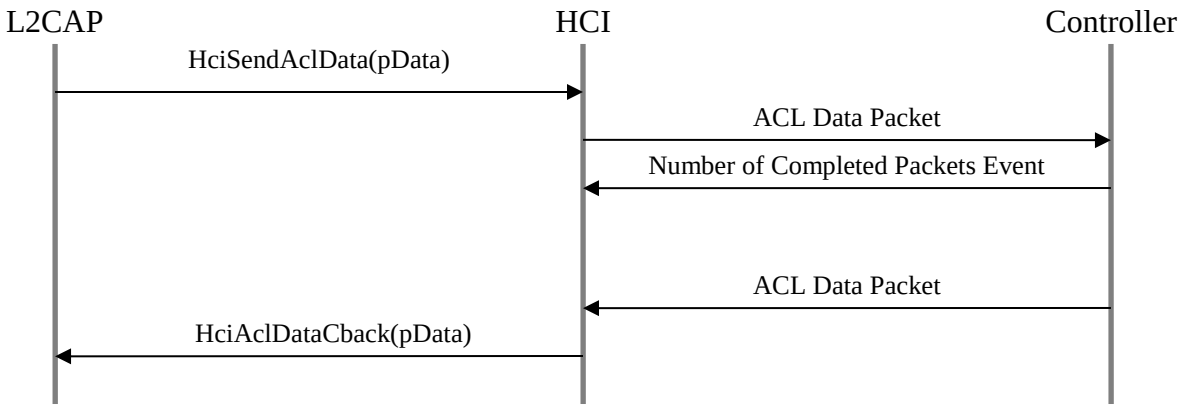

Figure 4. Data transmit and receive