ARM® Cordio Stack

ARM-EPM-115879 1.0

L2CAP API

Confidential

ARM® Cordio Stack L2CAP API

Reference Manual

Release Information

The following changes have been made to this book:

Document History

Date	Issue	Confidentiality	Change
25 September 2015	-	Confidential	First Wicentric release for 1.3 as 2009-0007.
1 March 2016	A	Confidential	First ARM release for 1.3.
24 August 2016	A	Confidential	AUSPEX # / API Update

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of ARM. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information: (i) for the purposes of determining whether implementations infringe any third party patents; (ii) for developing technology or products which avoid any of ARM's intellectual property; or (iii) as a reference for modifying existing patents or patent applications or creating any continuation, continuation in part, or extension of existing patents or patent applications; or (iv) for generating data for publication or disclosure to third parties, which compares the performance or functionality of the ARM technology described in this document with any other products created by you or a third party, without obtaining ARM's prior written consent.

IMPORTANT. Your use of this document is governed by a Software License Agreement ("Agreement") that must be accepted in order to download or otherwise receive a copy of this document. You may not use or copy this document for any purpose other than as described in the Agreement. If you do not agree to all of the terms of the Agreement do not use this document and delete all copies in your possession or control; if you do not have a copy of the Agreement, you must contact ARM, Ltd. prior to any use, copying or further distribution of this document.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, ARM makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to ARM's customers is not intended to create or refer to any partnership relationship with any other company. ARM may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any signed written agreement covering this document with ARM, then the signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited or its affiliates in the EU and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow ARM's trademark usage guidelines at http://www.arm.com/about/trademark-usage-guidelines.php

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

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`

Contents

1 Preface

This preface introduces the Cordio Stack L2CAP API.

1.1 About this book

This document describes the Cordio Stack L2CAP and describes how to use the software.

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 L2 API.
- **System Context**
  Read this for a description of the L2C subsystem in the Bluetooth LE stack.
- **System Architecture**
  Read this for a description of the modules and functions in the L2C subsystem.
- **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
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.
`MONOSPACE`	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-115148.
- 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

This document describes the API and software design of the L2CAP subsystem, L2C.

3 System Context

Figure 1 shows the context of the L2C subsystem in the Bluetooth LE stack.

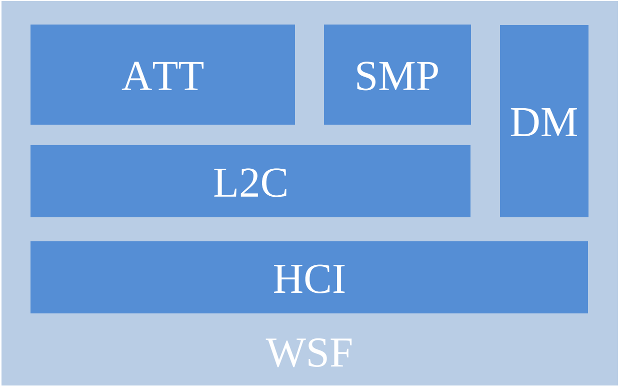

Figure 1: Bluetooth LE stack software system.

L2C interfaces to HCI to send and receive ACL packets. The ATT and SMP protocol layers interface to L2C to send and receive L2CAP packets. L2C also interfaces to DM to perform the L2CAP connection update procedure.

4 Subsystem Architecture

Figure 2 shows the different modules that make up the L2C subsystem.

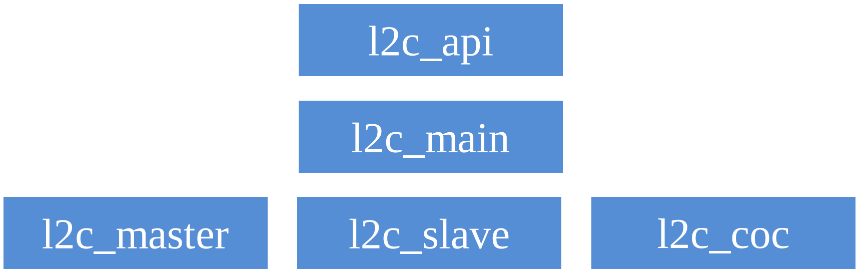

Figure 2: Subsystem architecture

Module `l2c_api` contains the API. Module l2c_main contains the main API function implementation, main event handler, and functions for processing packets. Module l2c_master contains API functions and other functions used only when operating as an LE master. Module l2c_slave contains API functions and other functions used only when operating as an LE slave. Module lcc_coc contains functions for L2CAP Connection Oriented Channels.

4.1 l2c_api

4.1.1 Constants and data structures

Table 1: Connection identifiers

Name	Value	Description
`L2C_CID_ATT`	0x0004	CID for attribute protocol.
`L2C_CID_LE_SIGNALING`	0x0005	CID for LE signaling.
`L2C_CID_SMP`	0x0006	CID for security manager protocol.

Table 2: Connection parameter result

Name	Value	Description
`L2C_CONN_PARAM_ACCEPTED`	0x0000	Connection parameters accepted.
`L2C_CONN_PARAM_REJECTED`	0x0001	Connection parameters rejected.

Table 3: Control callback events

Name	Value	Description
`L2C_CTRL_FLOW_ENABLE_IND`	`0x00`	Data flow enabled. The client may call `L2cDataReq().`
`L2C_CTRL_FLOW_DISABLE_IND`	`0x01`	Data flow disabled. The client should not call `L2cDataReq()` until it receives a `L2C_CTRL_FLOW_ENABLE_IND`.

4.1.2 Function calls

4.1.2.1 L2cInit()

This function is called to initialize L2C. This function is generally called once during system initialization before any other non-initialization L2C API functions are called. .

Syntax:

```
void L2cInit (void)
```

4.1.2.2 L2cMasterInit()

This function is called to initialize L2C for operation as a Bluetooth LE master. This function is generally called once during system initialization before any other non-initialization L2C API functions are called.

Syntax:

```
void L2cMasterInit(void)
```

4.1.2.3 void L2cSlaveInit(void)

This function is called to initialize L2C for operation as a Bluetooth LE slave. This function is generally called once during system initialization before any other non-initialization L2C API functions are called.

Syntax:

```
void L2cSlaveInit(void)
```

4.1.2.4 L2cRegister()

This function is called by the L2C client, such as ATT or SMP, to register for the given CID. This allows the client to send and receive data using that CID.

Syntax:

```
void L2cRegister(uint16_t cid, l2cDataCback_t dataCback, l2cCtrlCback_t
        ctrlCback)
```

Where:

- dataCback: Callback function for L2CAP data received for this CID. This cannot be set to NULL.
- ctrlCback: Callback function for control events for this CID. This cannot be set to NULL.

This function stores the callback parameters in l2cMain.

4.1.2.5 L2cDataReq()

This function sends an L2CAP data packet on the given CID.

Syntax:

```
void L2cDataReq(uint16_t cid, uint16_t handle, uint16_t len, uint8_t
        *pL2cPacket)
```

Where:

- cid: The channel identifier.
- handle: The connection handle. The client receives this handle from DM when the connection is established.
- len: The length of the payload data in pPacket.
- pL2cPacket: A buffer containing the packet. This is a WSF buffer allocated by the client.

The buffer pointed to by pL2cPacket must be a WSF buffer allocated by the client.

This function first checks if there is an active connection associated with the handle. If not, the packet is discarded and the buffer containing the packet is deallocated. Then it builds an L2CAP data packet, setting both the L2CAP and HCI headers. Then it calls function HciSendAclData() to send the packet to HCI.

4.1.2.6 L2cDmConnUpdateReq()

This function is called by DM to send an L2CAP connection update request.

Syntax:

```
void L2cDmConnUpdateReq(uint16_t handle, hciConnSpec_t *pConnSpec)
```

Where:

- handle: The connection handle.

- pConnSpec: Pointer to the connection specification structure. This structure is defined in the *HCI API Reference Manual*. The following elements in the structure must be set:
    - connIntervalMin
    - connIntervalMax
    - connLatency
    - supTimeout

This function starts the signaling request timeout timer, builds an L2CAP connection update request packet and then calls L2cDataReq() to send the packet.

4.1.2.7 L2cDmConnUpdateRsp()

This function is called by DM to send an L2CAP connection update response.

Syntax:

```
void L2cDmConnUpdateRsp(uint8 identifier, uint16_t handle, uint16_t result)
```

Where:

- identifier: Identifier value previously passed from L2C to DM.
- handle: The connection handle.
- result: Connection update response result. See 0.

This function builds an L2CAP connection update response packet and then calls L2cDataReq() to send the packet.

4.1.2.8 L2cSlaveHandler()

This function is the WSF event handler for L2C when operating as a slave. This function is only called from the WSF OS implementation.

Syntax:

```
L2cSlaveHandler(wsfEventMask_t event, wsfMsgHdr_t *pMsg)
```

Where:

- event: Event mask.
- pMsg: Pointer to message.

The implementation of this function handles the L2CAP signaling request timeout timer.

4.1.2.9 L2cSlaveHandlerInit(wsfHandlerId_t handlerId)

This is the event handler initialization function for L2C when operating as a slave. This function is generally called once during system initialization.

Syntax:

```
L2cSlaveHandlerInit(wsfHandlerId_t handlerId)
```

Where:

- `handlerId`: ID for this event handler.

This function stores the hander ID and performs other initialization procedures.

4.1.2.10 L2cCocInit()

This function initializes the L2Cap Connection Oriented Channels. This function is generally called once during initialization.

Syntax:

```
L2cCocInit(void)
```

4.1.2.11 L2cCocRegister()

This function is used to register an instance of a connection oriented channel. The instance can be a channel acceptor, initiator, or both. If registering as channel as acceptor, then the PSM is specified. After registering a connection, the connections can be established by the client using this registration instance.

Syntax:

```
l2cCocRegId_t L2cCocRegister(l2cCocCback_t cback, l2cCocReg_t *pReg)
```

Where:

- `cback`: Callback for the connection oriented channel.
- `pReg`: Registration parameters.

This function returns an identifier for the channel.

4.1.2.12 L2cCocDeregister()

This function deregisters and deallocates a connection oriented channel registered instance. This function should only be called if there are no active channels using the registration instance.

Syntax:

```
L2cCocDeregister(l2cCocRegId_t regId)
```

Where:

- `regId`: The identifier for the channel (returned by L2cCocRegister).

4.1.2.13 L2cCocConnectReq()

This function initiates a connection to the given peer PSM using the connection oriented channel subsystem.

Syntax:

```
uint16_t L2cCocConnectReq(dmConnId_t connId, l2cCocRegId_t regId, uint16_t
         psm)
```

Where:

- `connId`**:** The DM connection ID.
- `regId`**:** The identifier for the channel (returned by L2cCocRegister).
- `psm`**:** The peers PSM.

This function returns the local CID or L2C_COC_CID_NONE if there was a failure.

4.1.2.14 L2cCocDisconnectReq()

This function disconnects the channel to the peer for the given CID.

Syntax:

```
L2cCocDisconnectReq(uint16_t cid)
```

Where:

- `cid`**:** The channel CID (returned by L2cCocConnectReq).

4.1.2.15 L2cCocDataReq()

This function sends an L2CAP data packet on the given connection oriented channel with the given CID.

Syntax:

```
L2cCocDataReq(uint16_t cid, uint16_t len, uint8_t *pPayload)
```

Where:

- `cid`**:** The channel CID (returned by L2cCocConnectReq).
- `len`**:** The length of the pPayload in bytes.
- `pPayload`**:** The packet to send.

4.1.3 Callback functions

4.1.3.1 (*l2cDataCback_t)()

This callback function sends a received L2CAP packet to the client.

Syntax:

```
void (*l2cDataCback_t)(uint16_t handle, uint16_t len, uint8_t *pPacket)
```

Where:

- `handle`: The connection handle.
- `len`: The length of the L2CAP payload data in pPacket.
- `pPacket`**:** A buffer containing the packet.

4.1.3.2 (*l2cCtrlCback_t)()

This callback function sends control events to the client. It is currently used only for flow control.

Syntax:

```
void (*l2cCtrlCback_t)(uint8_t event)
```

Where:

- `event`: Control event. See 0

4.1.3.3 (*l2cCocCback_t)()

This callback function sends data and other events to connection oriented channel clients.

Syntax:

```
void (*l2cCocCback_t)(l2cCocEvt_t *pMsg)
```

Where:

- `pMsg`**:** Pointer to the message structure

4.1.3.4 (*l2cCocAuthorCback_t)()

This callback function is used for authorization of connection oriented channels.

Syntax:

```
uint16_t (*l2cCocAuthorCback_t)(dmConnId_t connId, l2cCocRegId_t regId,
  uint16_t psm)
```

Where:

- `connId`: The connection identifier.
- `regId`: The connection oriented channel registration instance identifier.
- `psm`: The psm of the connection.

5 Scenarios

This section describes example scenarios for initialization and connection.

5.1 Initialization

Figure 3 shows the initialization process. In this example, the system supports operation as both a master and a slave so `L2cMasterInit()` and `L2cSlaveInit()` are called. Then function `L2cSlaveHandlerInit()` is called after `L2cSlaveHandler()` is set up in the WSF OS implementation.

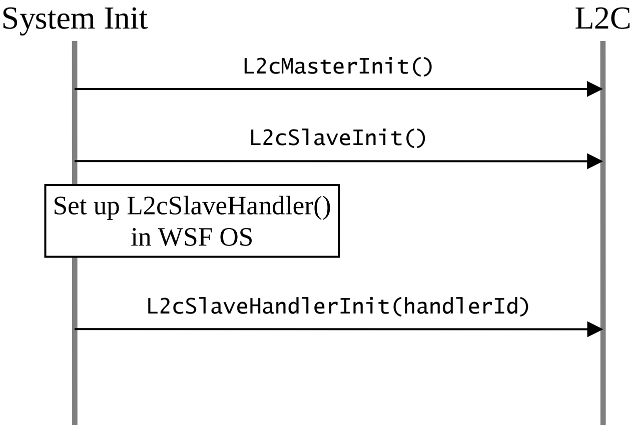
System Init
L2C
L2cMasterInit()
L2cSlaveInit()
Set up L2cSlaveHandler() in WSF OS
L2cSlaveHandlerInit(handlerId)

Figure 3: Initialization

5.2 Data path

Figure 4 shows the operation of the data path with ATT shown as an example L2C client. ATT calls `L2cDataReq()` to send a packet to L2C. Then L2C calls `HciSendAclData()` to send the packet to HCI. In the receive direction, HCI calls `HciAclDataCback()` to send a packet to L2C. L2C calls ATT callback function `attDataCback()` to send the packet to ATT.

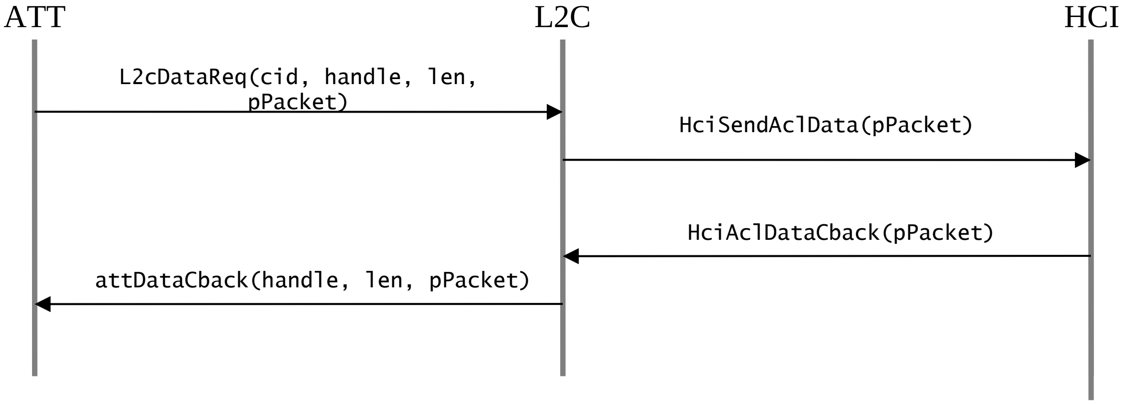
ATT
L2C
HCI
L2cDataReq(cid, handle, len, pPacket)
HciSendAclData(pPacket)
HciAclDataCback(pPacket)
attDataCback(handle, len, pPacket)

Figure 4: Data path

5.3 Connection parameter update

Figure 5 shows a connection parameter update procedure with the stack operating as a slave. DM calls `L2cDmConnUpdateReq()` to initiate the process. L2C builds and sends an L2CAP Connection Parameter Update Request. The peer device receives the request and initiates a connection update procedure. When the procedure completes an HCI LE Connection Update Complete Event is sent from HCI to DM. Then the L2CAP Connection Parameter Update Response is received from the peer and L2C calls `DmL2cConnUpdateCnf()`.

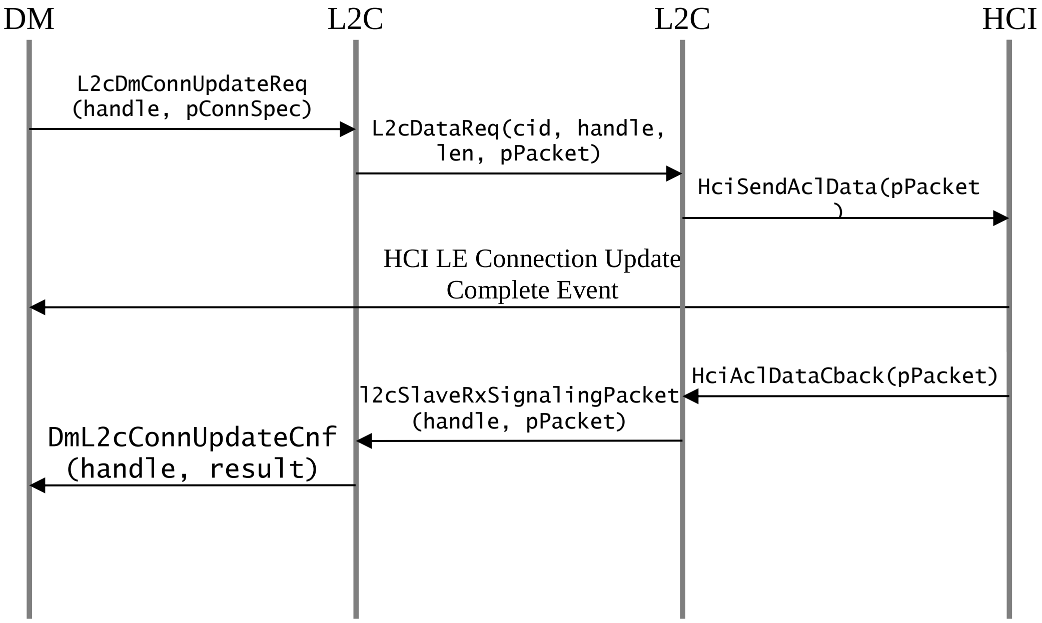
DM
L2C
L2C
HCI
L2cDmConnUpdateReq (handle, pConnSpec)
L2cDataReq(cid, handle, len, pPacket)
HciSendAclData(pPacket)
HCI LE Connection Update Complete Event
HciAclDataCback(pPacket)
l2cSlaveRxSignalingPacket (handle, pPacket)
DmL2cConnUpdateCnf (handle, result)

Figure 5: Connection parameter update