

Specification of the **Bluetooth®** System



Specification Volume 3

Core System Package [Host volume]

Covered Core Package version: 4.2

Publication date: Dec 02 2014

Bluetooth SIG Proprietary





Revision History

The Revision History is shown in the [\[Vol 0\] Part C](#), Appendix.

Contributors

The persons who contributed to this specification are listed in the [\[Vol 0\] Part C](#), Appendix.

Web Site

This specification can also be found on the official Bluetooth web site:
<https://www.bluetooth.org/en-us/specification/adopted-specifications>

Disclaimer and Copyright Notice

This disclaimer applies to all draft specifications and final specifications adopted by the Bluetooth SIG Board of Directors (both of which are hereinafter referred to herein as a Bluetooth "Specification"). Your use of this Specification in any way is subject to your compliance with all conditions of such use, and your acceptance of all disclaimers and limitations as to such use, contained in this Specification. Any user of this Specification is advised to seek appropriate legal, engineering or other professional advice regarding the use, interpretation or effect of this Specification on any matters discussed in this Specification.

Use of Bluetooth Specifications and any related intellectual property is governed by the Promoters Membership Agreement among the Promoter Members and Bluetooth SIG (the "Promoters Agreement"), certain membership agreements between Bluetooth SIG and its Adopter and Associate Members, including, but not limited to, the Membership Application, the Bluetooth Patent/Copyright License Agreement and the Bluetooth Trademark License Agreement (collectively, the "Membership Agreements") and the Bluetooth Specification Early Adopters Agreements (1.2 Early Adopters Agreements) among Early Adopter members of the unincorporated Bluetooth SIG and the Promoter Members (the "Early Adopters Agreement"). Certain rights and obligations of the Promoter Members under the Early Adopters Agreements have been assigned to Bluetooth SIG by the Promoter Members.

Use of the Specification by anyone who is not a member of Bluetooth SIG or a party to an Early Adopters Agreement (each such person or party, a "Member") is prohibited. The use of any portion of a Bluetooth Specification may involve the use of intellectual property rights ("IPR"), including pending or issued patents, or copyrights or other rights. Bluetooth SIG has made no search or investigation for such rights and disclaims any undertaking or duty to do so. The legal rights and obligations of each Member are governed by the applicable Membership Agreements, Early Adopters Agreement or Promoters Agreement. No license, express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

Any use of the Specification not in compliance with the terms of the applicable Membership Agreements, Early Adopters Agreement or Promoters Agreement is prohibited and any such prohibited use may result in (i) termination of the applicable Membership Agreements or Early Adopters Agreement and (ii) liability claims by Bluetooth SIG or any of its Members for patent, copyright and/or trademark infringement claims permitted by the applicable agreement or by applicable law.



THE SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, SATISFACTORY QUALITY, OR REASONABLE SKILL OR CARE, OR ANY WARRANTY ARISING OUT OF ANY COURSE OF DEALING, USAGE, TRADE PRACTICE, PROPOSAL, SPECIFICATION OR SAMPLE.

Each Member hereby acknowledges that products equipped with the Bluetooth wireless technology ("Bluetooth Products") may be subject to various regulatory controls under the laws and regulations applicable to products using wireless non licensed spectrum of various governments worldwide. Such laws and regulatory controls may govern, among other things, the combination, operation, use, implementation and distribution of Bluetooth Products. Examples of such laws and regulatory controls include, but are not limited to, airline regulatory controls, telecommunications regulations, technology transfer controls and health and safety regulations. Each Member is solely responsible for the compliance by their Bluetooth Products with any such laws and regulations and for obtaining any and all required authorizations, permits, or licenses for their Bluetooth Products related to such regulations within the applicable jurisdictions. Each Member acknowledges that nothing in the Specification provides any information or assistance in connection with securing such compliance, authorizations or licenses. **NOTHING IN THE SPECIFICATION CREATES ANY WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING SUCH LAWS OR REGULATIONS.**

ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHTS OR FOR NONCOMPLIANCE WITH LAWS, RELATING TO USE OF THE SPECIFICATION IS EXPRESSLY DISCLAIMED. To the extent not prohibited by law, in no event will Bluetooth SIG or its Members or their affiliates be liable for any damages, including without limitation, lost revenue, profits, data or programs, or business interruption, or for special, indirect, consequential, incidental or punitive damages, however caused and regardless of the theory of liability, arising out of or related to any furnishing, practicing, modifying, use or the performance or implementation of the contents of this Specification, even if Bluetooth SIG or its Members or their affiliates have been advised of the possibility of such damages. **BY USE OF THE SPECIFICATION, EACH MEMBER EXPRESSLY WAIVES ANY CLAIM AGAINST BLUETOOTH SIG AND ITS MEMBERS OR THEIR AFFILIATES RELATED TO USE OF THE SPECIFICATION.**

If this Specification is an intermediate draft, it is for comment only. No products should be designed based on it except solely to verify the prototyping specification at SIG sponsored IOP events and it does not represent any commitment to release or implement any portion of the intermediate draft, which may be withdrawn, modified, or replaced at any time in the adopted Specification.

Bluetooth SIG reserves the right to adopt any changes or alterations to the Specification it deems necessary or appropriate.

Copyright © 1999-2014. The Bluetooth word mark and logos are owned by Bluetooth SIG, Inc. All copyrights in the Bluetooth Specifications themselves are owned by Ericsson AB, Lenovo (Singapore) Pte. Ltd., Intel Corporation, Microsoft Corporation, Motorola Mobility, LLC, Nokia Corporation and Toshiba Corporation. Other third-party brands and names are the property of their respective owners.



TABLE OF CONTENTS

Part A

LOGICAL LINK CONTROL AND ADAPTATION PROTOCOL SPECIFICATION

1	Introduction	29
1.1	L2CAP Features	29
1.2	Assumptions	32
1.3	Scope	33
1.4	Terminology	33
2	General Operation	37
2.1	Channel Identifiers.....	37
2.2	Operation Between Devices	40
2.3	Operation Between Layers	41
2.4	Modes of Operation	42
2.5	Mapping Channels to Logical Links	44
3	Data Packet Format	45
3.1	Connection-oriented Channels in Basic L2CAP Mode	45
3.2	Connectionless Data Channel in Basic L2CAP Mode	46
3.3	Connection-oriented Channel in Retransmission/Flow Control/ Streaming Modes.....	47
3.3.1	L2CAP header fields	48
3.3.2	Control field (2 or 4 octets).....	49
3.3.3	L2CAP SDU Length Field (2 octets)	51
3.3.4	Information Payload Field	52
3.3.5	Frame Check Sequence (2 octets)	52
3.3.6	Invalid Frame Detection	53
3.3.7	Invalid Frame Detection Algorithm.....	53
3.4	Connection-Oriented Channels in LE Credit Based Flow Control Mode.....	55
3.4.1	L2CAP Header Fields	55
3.4.2	L2CAP SDU Length Field (2 octets)	55
3.4.3	Information Payload Field	55
4	Signaling Packet Formats	57
4.1	Command Reject (code 0x01).....	60
4.2	Connection Request (code 0x02)	61
4.3	Connection Response (code 0x03)	63
4.4	Configuration Request (code 0x04).....	65
4.5	Configuration Response (code 0x05)	67
4.6	Disconnection Request (code 0x06).....	69



4.7	Disconnection Response (code 0x07)	70
4.8	Echo Request (code 0x08)	70
4.9	Echo Response (code 0x09)	71
4.10	Information Request (code 0x0A)	71
4.11	Information Response (code 0x0B)	72
4.12	Extended Feature Mask	74
4.13	Fixed Channels Supported	75
4.14	Create Channel Request (code 0x0C)	76
4.15	Create Channel Response (code 0x0D)	77
4.16	Move Channel Request (code 0x0E)	78
4.17	Move Channel Response (code 0x0F)	80
4.18	Move Channel Confirmation (code 0x10)	81
4.19	Move Channel Confirmation Response (code 0x11)	82
4.20	Connection Parameter Update Request (code 0x12)	82
4.21	Connection Parameter Update Response (code 0x13)	84
4.22	LE Credit Based Connection Request (Code 0x14)	85
4.23	LE Credit Based Connection Response (Code 0x15)	86
4.24	LE Flow Control Credit (Code 0x16)	88
5	Configuration Parameter Options	89
5.1	Maximum Transmission Unit (MTU)	89
5.2	Flush Timeout Option	91
5.3	Quality of Service (QoS) Option	92
5.4	Retransmission and Flow Control Option	96
5.5	Frame Check Sequence (FCS) Option	101
5.6	Extended Flow Specification Option	102
5.7	Extended Window Size Option	108
6	State Machine	110
6.1	General rules for the state machine:	110
6.1.1	CLOSED state	112
6.1.2	WAIT_CONNECT_RSP state	113
6.1.3	WAIT_CONNECT state	114
6.1.4	CONFIG state	114
6.1.5	OPEN state	120
6.1.6	WAIT_DISCONNECT state	121
6.1.7	WAIT_CREATE_RSP state	122
6.1.8	WAIT_CREATE state	122
6.1.9	WAIT_MOVE_RSP state	123
6.1.10	WAIT_MOVE state	124
6.1.11	WAIT_MOVE_CONFIRM state	124
6.1.12	WAIT_CONFIRM_RSP state	125



6.2	Timers events	126
6.2.1	RTX.....	126
6.2.2	ERTX	127
7	General Procedures.....	131
7.1	Configuration Process	131
7.1.1	Request Path	132
7.1.2	Response Path	133
7.1.3	Lockstep Configuration Process	133
7.1.4	Standard Configuration Process	136
7.2	Fragmentation and Recombination.....	138
7.2.1	Fragmentation of L2CAP PDUs	138
7.2.2	Recombination of L2CAP PDUs	139
7.3	Encapsulation of SDUs.....	140
7.3.1	Segmentation of L2CAP SDUs	140
7.3.2	Reassembly of L2CAP SDUs	141
7.3.3	Segmentation and fragmentation.....	141
7.4	Delivery of Erroneous L2CAP SDUs	142
7.5	Operation with Flushing On ACL-U Logical Links.....	142
7.6	Connectionless Data Channel	143
7.7	Operation Collision Resolution	145
7.8	Aggregating Best Effort Extended Flow Specifications.....	145
7.9	Prioritizing Data over HCI	147
7.10	Supporting Extended Flow Specification for BR/EDR and BR/ EDR/LE Controllers	147
8	Procedures for Flow Control and Retransmission.....	149
8.1	Information Retrieval	149
8.2	Function of PDU Types for Flow Control and Retransmission.	149
8.2.1	Information frame (I-frame)	149
8.2.2	Supervisory Frame (S-frame)	149
8.3	Variables and Sequence Numbers	150
8.3.1	Sending peer.....	151
8.3.2	Receiving peer.....	152
8.4	Retransmission Mode	154
8.4.1	Transmitting frames	154
8.4.2	Receiving I-frames	156
8.4.3	I-frames pulled by the SDU reassembly function.....	157
8.4.4	Sending and receiving acknowledgements.....	157
8.4.5	Receiving REJ frames	158
8.4.6	Waiting acknowledgements	159



8.4.7	Exception conditions	159
8.5	Flow Control Mode	161
8.5.1	Transmitting I-frames	161
8.5.2	Receiving I-frames	162
8.5.3	I-frames pulled by the SDU reassembly function.....	162
8.5.4	Sending and receiving acknowledgements.....	162
8.5.5	Waiting acknowledgements	163
8.5.6	Exception conditions	164
8.6	Enhanced Retransmission Mode.....	165
8.6.1	Function Of PDU Types	165
8.6.2	Rules For Timers	167
8.6.3	General Rules for the State Machine	169
8.6.4	State Diagram	171
8.6.5	States Tables.....	171
8.7	Streaming Mode	197
8.7.1	Transmitting I-frames	197
8.7.2	Receiving I-frames	197
8.7.3	Exception Conditions	198
9	Procedure for AMP Channel Creation and Handling.....	199
9.1	Create Channel	199
9.2	Move Channel	202
9.2.1	Move Channel Protocol Procedure with Enhanced Retransmission Mode	203
9.2.2	Move Channel Protocol Procedure with Streaming Mode (Initiator is Data Source)	206
9.2.3	Move Channel Protocol Procedure with Streaming Mode (Initiator is Data Sink).....	207
9.3	Disconnect Channel	210
10	Procedures for Credit Based Flow Control	211
10.1	LE Credit Based Flow Control Mode	211
Appendix A Configuration MSCs	212	
Part B		
SERVICE DISCOVERY PROTOCOL (SDP) SPECIFICATION		
1	Introduction	219
1.1	General Description	219
1.2	Motivation	219
1.3	Requirements	219
1.4	Non-requirements and Deferred Requirements.....	220



1.5	Conventions.....	221
1.5.1	Bit And Byte Ordering Conventions	221
2	Overview	222
2.1	SDP Client-Server Architecture	222
2.2	Service Record	223
2.3	Service Attribute	225
2.3.1	Attribute ID	225
2.3.2	Attribute Value.....	226
2.4	Service Class.....	226
2.4.1	A Printer Service Class Example	227
2.5	Searching for Services	227
2.5.1	UUID	227
2.5.2	Service Search Patterns	228
2.6	Browsing for Services	228
2.6.1	Example Service Browsing Hierarchy.....	229
3	Data Representation	231
3.1	Data Element.....	231
3.2	Data Element Type Descriptor.....	231
3.3	Data Element Size Descriptor.....	232
3.4	Data Element Examples	233
4	Protocol Description	234
4.1	Transfer Byte Order	234
4.2	Protocol Data Unit Format	234
4.3	Partial Responses and Continuation State	236
4.4	Error Handling	236
4.4.1	SDP_ErrorResponse PDU	237
4.5	ServiceSearch Transaction.....	238
4.5.1	SDP_ServiceSearchRequest PDU	238
4.5.2	SDP_ServiceSearchResponse PDU	239
4.6	ServiceAttribute Transaction.....	241
4.6.1	SDP_ServiceAttributeRequest PDU	241
4.6.2	SDP_ServiceAttributeResponse PDU	243
4.7	ServiceSearchAttribute Transaction	244
4.7.1	SDP_ServiceSearchAttributeRequest PDU	244
4.7.2	SDP_ServiceSearchAttributeResponse PDU	246
5	Service Attribute Definitions.....	248
5.1	Universal Attribute Definitions	248
5.1.1	ServiceRecordHandle Attribute.....	248
5.1.2	ServiceClassIDList Attribute	249



5.1.3	ServiceRecordState Attribute	249
5.1.4	ServiceID Attribute	249
5.1.5	ProtocolDescriptorList Attribute	250
5.1.6	AdditionalProtocolDescriptorList Attribute.....	251
5.1.7	BrowseGroupList Attribute	252
5.1.8	LanguageBaseAttributeIDLList Attribute.....	252
5.1.9	ServiceInfoTimeToLive Attribute	253
5.1.10	ServiceAvailability Attribute.....	254
5.1.11	BluetoothProfileDescriptorList Attribute	254
5.1.12	DocumentationURL Attribute	255
5.1.13	ClientExecutableURL Attribute	255
5.1.14	IconURL Attribute.....	256
5.1.15	ServiceName Attribute	256
5.1.16	ServiceDescription Attribute.....	257
5.1.17	ProviderName Attribute.....	257
5.1.18	Reserved Universal Attribute IDs.....	257
5.2	ServiceDiscoveryServer Service Class Attribute Definitions ...	258
5.2.1	ServiceRecordHandle Attribute.....	258
5.2.2	ServiceClassIDList Attribute	258
5.2.3	VersionNumberList Attribute	258
5.2.4	ServiceDatabaseState Attribute	259
5.2.5	Reserved Attribute IDs.....	259
5.3	BrowseGroupDescriptor Service Class Attribute Definitions ...	260
5.3.1	ServiceClassIDList Attribute	260
5.3.2	GroupID Attribute	260
5.3.3	Reserved Attribute IDs.....	260
6	Security.....	261
	Appendix A Background Information	262
A.1	Service Discovery	262
A.2	Bluetooth Service Discovery	262
	Appendix B Example SDP Transactions	263
B.1	SDP Example 1 – ServiceSearchRequest	263
B.2	SDP Example 2 – ServiceAttributeTransaction	265
B.3	SDP Example 3 – ServiceSearchAttributeTransaction	267



Part C GENERIC ACCESS PROFILE

1	Introduction	286
1.1	Scope	286
1.2	Symbols and Conventions	287
1.2.1	Requirement Status Symbols.....	287
1.2.2	Signaling diagram conventions	288
1.2.3	Notation for Timers and Counters	288
2	Profile Overview.....	289
2.1	Profile Stack.....	289
2.2	Profile Roles	289
2.2.1	Roles when Operating over BR/EDR Physical Transport.....	289
2.2.2	Roles when Operating over an LE Physical Transport	290
2.3	User Requirements and Scenarios	293
2.4	Profile Fundamentals.....	293
2.5	Conformance	293
3	User Interface Aspects	294
3.1	The User Interface Level	294
3.2	Representation of Bluetooth Parameters.....	294
3.2.1	Bluetooth Device Address (BD_ADDR).....	294
3.2.2	Bluetooth Device Name (the user-friendly name).....	295
3.2.3	Bluetooth Passkey (Bluetooth PIN).....	296
3.2.4	Class of Device	297
3.2.5	Appearance Characteristic.....	298
3.3	Pairing	299
4	Modes – BR/EDR Physical Transport.....	300
4.1	Discoverability Modes.....	300
4.1.1	Non-discoverable Mode	301
4.1.2	Limited Discoverable Mode.....	301
4.1.3	General Discoverable Mode	303
4.2	Connectability Modes	304
4.2.1	Non-connectable Mode.....	304
4.2.2	Connectable Mode.....	304
4.3	Bondable Modes	306
4.3.1	Non-bondable Mode	306
4.3.2	Bondable Mode.....	306
4.4	Synchronizability Modes	307
4.4.1	Non-synchronizable Mode	307



4.4.2	Synchronizable Mode	307
5	Security Aspects – BR/EDR Physical Transport.....	308
5.1	Authentication	308
5.1.1	Purpose.....	308
5.1.2	Term on UI level.....	308
5.1.3	Procedure	309
5.1.4	Conditions	309
5.2	Security Modes.....	310
5.2.1	Legacy Security Modes.....	311
5.2.2	Security Mode 4 (service level enforced security)	312
6	Idle Mode Procedures – BR/EDR Physical Transport.....	328
6.1	General Inquiry	328
6.1.1	Purpose.....	328
6.1.2	Term on UI level.....	328
6.1.3	Description.....	329
6.1.4	Conditions	329
6.2	Limited Inquiry	329
6.2.1	Purpose.....	329
6.2.2	Term on UI level.....	330
6.2.3	Description.....	330
6.2.4	Conditions	330
6.3	Name Discovery	331
6.3.1	Purpose.....	331
6.3.2	Term on UI level.....	331
6.3.3	Description.....	331
6.3.4	Conditions	332
6.4	Device Discovery	332
6.4.1	Purpose.....	332
6.4.2	Term on UI Level.....	332
6.4.3	Description.....	333
6.4.4	Conditions	333
6.5	Bonding	334
6.5.1	Purpose.....	334
6.5.2	Term on UI level.....	334
6.5.3	Description.....	334
6.5.4	Conditions	336
7	Establishment Procedures – BR/EDR Physical Transport.....	337
7.1	Link Establishment.....	337



7.1.1	Purpose.....	337
7.1.2	Term on UI Level.....	337
7.1.3	Description.....	338
7.1.4	Conditions	339
7.2	Channel Establishment.....	340
7.2.1	Purpose.....	340
7.2.2	Term on UI level.....	340
7.2.3	Description.....	340
7.2.4	Conditions	341
7.3	Connection Establishment.....	342
7.3.1	Purpose.....	342
7.3.2	Term on UI level.....	342
7.3.3	Description.....	342
7.3.4	Conditions	343
7.4	Establishment of Additional Connection	343
7.5	Synchronization Establishment.....	344
7.5.1	Purpose.....	344
7.5.2	Term on UI Level.....	344
7.5.3	Description.....	344
7.5.4	Conditions	344
8	Extended Inquiry Response Data Format.....	346
9	Operational Modes and Procedures – LE Physical Transport....	348
9.1	Broadcast Mode and Observation Procedure	348
9.1.1	Broadcast Mode.....	348
9.1.2	Observation Procedure.....	349
9.2	Discovery Modes and Procedures.....	349
9.2.1	Requirements.....	350
9.2.2	Non-Discoverable Mode	350
9.2.3	Limited Discoverable Mode.....	351
9.2.4	General Discoverable Mode	352
9.2.5	Limited Discovery Procedure	354
9.2.6	General Discovery Procedure.....	355
9.2.7	Name Discovery Procedure	356
9.3	Connection Modes and Procedures	357
9.3.1	Requirements.....	357
9.3.2	Non-Connectable Mode	358
9.3.3	Directed Connectable Mode	358
9.3.4	Undirected Connectable Mode	359



9.3.5	Auto Connection Establishment Procedure	359
9.3.6	General Connection Establishment Procedure.....	360
9.3.7	Selective Connection Establishment Procedure	362
9.3.8	Direct Connection Establishment Procedure	364
9.3.9	Connection Parameter Update Procedure.....	365
9.3.10	Terminate Connection Procedure	366
9.3.11	Connection Establishment Timing Parameters	366
9.3.12	Connection Interval Timing Parameters.....	367
9.4	Bonding Modes and Procedures	368
9.4.1	Requirements.....	368
9.4.2	Non-Bondable Mode	369
9.4.3	Bondable Mode.....	369
9.4.4	Bonding Procedure	370
10	Security Aspects – LE Physical Transport.....	371
10.1	Requirements	371
10.2	LE Security Modes	371
10.2.1	LE Security Mode 1.....	372
10.2.2	LE Security Mode 2.....	372
10.2.3	Mixed Security Modes Requirements	373
10.2.4	Secure Connections Only Mode	373
10.3	Authentication Procedure	374
10.3.1	Responding to a Service Request	374
10.3.2	Initiating a Service Request	378
10.4	Data Signing	381
10.4.1	Connection Data Signing Procedure.....	381
10.4.2	Authenticate Signed Data Procedure.....	382
10.5	Authorization Procedure	383
10.6	Encryption Procedure	383
10.7	Privacy Feature	384
10.7.1	Privacy Feature in a Peripheral.....	385
10.7.2	Privacy Feature in a Central	386
10.7.3	Privacy Feature in a Broadcaster.....	386
10.7.4	Privacy Feature in an Observer	387
10.8	Random Device Address	387
10.8.1	Static Address	388
10.8.2	Private address	388
11	Advertising and Scan Response Data Format	389
12	GAP Service and Characteristics for GATT Server	390
12.1	Device Name Characteristic	391



12.2	Appearance Characteristic	391
12.3	Peripheral Preferred Connection Parameters Characteristic...	392
12.4	Central Address Resolution	393
13	BR/EDR/LE Operation.....	394
13.1	Modes, Procedures and Security Aspects	394
13.1.1	Discoverable Mode Requirements.....	395
13.2	Bonding for BR/EDR/LE Device Type	395
13.3	Relationship between Physical Transports.....	395
14	BR/EDR/LE Security Aspects	396
14.1	Cross-transport Key Derivation.....	396
14.2	Collision Handling	396
15	Bluetooth Device Requirements.....	397
15.1	Bluetooth Device Address	397
15.1.1	Bluetooth Device Address Types.....	397
15.2	GATT Profile Requirements	397
15.3	SDP Requirements	398
15.4	SDP Service Record Requirement	398
16	Definitions	399
16.1	General Definitions	399
16.2	Connection-related Definitions	399
16.3	Device-related Definitions	400
16.4	Procedure-related Definitions	401
16.5	Security-related Definitions.....	401
17	References	403
Appendix A (Normative): Timers and Constants		404
Appendix B (Informative): Information Flows of Related Procedures		408
B.1	LMP – Authentication	408
B.2	LMP – Pairing	409
B.3	Service Discovery	410
B.4	Generating a Resolvable Private Address	410
B.5	Resolving a Resolvable Private Address	410
Part D		
TEST SUPPORT		
1	Test Methodology	414
1.1	BR/EDR Test Scenarios	414
1.1.1	Test Setup	414
1.1.2	Transmitter Test	415



1.1.3	LoopBack Test	419
1.1.4	Pause Test	423
1.2	AMP Test Scenarios	424
1.2.1	Methodology Overview	424
1.2.2	Control and Configuration.....	426
1.2.3	AMP Test Manager	426
1.2.4	Test Commands/Events Format.....	427
1.2.5	AMP Test Manager Commands/Events	429
1.3	References	432
2	Test Control Interface (TCI).....	433
2.1	Introduction	433
2.1.1	Terms Used.....	433
2.1.2	Usage of the Interface.....	433
2.2	TCI Configurations.....	434
2.2.1	Bluetooth RF Requirements.....	434
2.2.2	Bluetooth Protocol Requirements	435
2.2.3	Bluetooth Profile Requirements	436
2.3	TCI Configuration and Usage	437
2.3.1	Transport Layers.....	437
2.3.2	Baseband and Link Manager Qualification	438
2.3.3	HCI Qualification	440

Part E**AMP MANAGER PROTOCOL SPECIFICATION**

1	Introduction	443
1.1	General Description	443
2	General Operation	444
2.1	Basic Capabilities	444
2.2	AMP Manager Channel Over L2CAP	445
2.3	Using the AMP Manager Protocol	446
2.3.1	Discovering a Remote AMP Manager.....	446
2.3.2	Discovering Available Controllers on a Remote Device	446
2.3.3	Creation of AMP Physical Links.....	447
2.4	Controller IDs.....	448
2.5	Controller Types	448
3	Protocol Description	449
3.1	Packet Formats.....	449
3.2	AMP Command Reject (Code 0x01)	451



3.3	AMP Discover Request (Code 0x02).....	452
3.4	AMP Discover Response (Code 0x03).....	453
3.5	AMP Change Notify (Code 0x04)	456
3.6	AMP Change Response (Code 0x05)	457
3.7	AMP Get Info Request (Code 0x06).....	457
3.8	AMP Get Info Response (Code 0x07)	458
3.9	AMP Get AMP Assoc Request (Code 0x08)	460
3.10	AMP Get AMP Assoc Response (Code 0x09).....	460
3.11	AMP Create Physical Link Request (Code 0x0A).....	461
3.12	AMP Create Physical Link Response (Code 0x0B).....	462
3.13	AMP Disconnect Physical Link Request (Code 0x0C)	465
3.14	AMP Disconnect Physical Link Response (Code 0x0D).....	465
3.15	Create Physical Link Collision Resolution	467
3.16	Response Timeout.....	467
3.17	Unexpected BR/EDR Physical Link Disconnect	467

Part F**ATTRIBUTE PROTOCOL (ATT)**

1	Introduction	471
1.1	Scope	471
1.2	Conformance	471
2	Protocol Overview	472
3	Protocol Requirements	473
3.1	Introduction	473
3.2	Basic Concepts.....	473
3.2.1	Attribute Type.....	473
3.2.2	Attribute Handle	473
3.2.3	Attribute Handle Grouping	474
3.2.4	Attribute Value.....	474
3.2.5	Attribute Permissions	474
3.2.6	Control-Point Attributes.....	476
3.2.7	Protocol Methods	476
3.2.8	Exchanging MTU Size	476
3.2.9	Long Attribute Values.....	476
3.2.10	Atomic Operations	477
3.3	Attribute PDU.....	477
3.3.1	Attribute PDU Format.....	478
3.3.2	Sequential Protocol.....	479
3.3.3	Transaction	480
3.4	Attribute Protocol PDUs.....	481



3.4.1	Error Handling.....	481
3.4.2	MTU Exchange	483
3.4.3	Find Information.....	485
3.4.4	Reading Attributes	489
3.4.5	Writing Attributes.....	499
3.4.6	Queued Writes	503
3.4.7	Server Initiated.....	507
3.4.8	Attribute Opcode Summary.....	509
3.4.9	Attribute PDU Response Summary	511
4	Security Considerations	514

Part G GENERIC ATTRIBUTE PROFILE (GATT)

1	Introduction	519
1.1	Scope	519
1.2	Profile Dependency	519
1.3	Conformance	519
1.4	Bluetooth Specification Release Compatibility.....	520
1.5	Conventions.....	520
2	Profile Overview.....	521
2.1	Protocol Stack.....	521
2.2	Configurations and Roles	521
2.3	User Requirements and Scenarios	522
2.4	Profile Fundamentals.....	522
2.5	Attribute Protocol	523
2.5.1	Overview	523
2.5.2	Attribute Caching	524
2.5.3	Attribute Grouping.....	525
2.5.4	UUIDs	526
2.6	GATT Profile Hierarchy.....	527
2.6.1	Overview	527
2.6.2	Service	528
2.6.3	Included Services.....	528
2.6.4	Characteristic.....	529
2.7	Configured Broadcast.....	529
3	Service Interoperability Requirements	530
3.1	Service Definition.....	530
3.2	Include Definition	531
3.3	Characteristic Definition.....	531



3.3.1	Characteristic Declaration.....	532
3.3.2	Characteristic Value Declaration.....	534
3.3.3	Characteristic Descriptor Declarations.....	534
3.4	Summary of GATT Profile Attribute Types	543
4	GATT Feature Requirements	544
4.1	Overview.....	544
4.2	Feature Support and Procedure Mapping	544
4.3	Server Configuration.....	546
4.3.1	Exchange MTU	546
4.4	Primary Service Discovery.....	547
4.4.1	Discover All Primary Services.....	547
4.4.2	Discover Primary Service by Service UUID	548
4.5	Relationship Discovery	550
4.5.1	Find Included Services.....	550
4.6	Characteristic Discovery	551
4.6.1	Discover All Characteristics of a Service	551
4.6.2	Discover Characteristics by UUID.....	553
4.7	Characteristic Descriptor Discovery.....	554
4.7.1	Discover All Characteristic Descriptors	554
4.8	Characteristic Value Read	556
4.8.1	Read Characteristic Value	556
4.8.2	Read Using Characteristic UUID	556
4.8.3	Read Long Characteristic Values.....	557
4.8.4	Read Multiple Characteristic Values	558
4.9	Characteristic Value Write	559
4.9.1	Write Without Response	559
4.9.2	Signed Write Without Response	560
4.9.3	Write Characteristic Value.....	560
4.9.4	Write Long Characteristic Values	561
4.9.5	Reliable Writes.....	562
4.10	Characteristic Value Notification	564
4.10.1	Notifications	564
4.11	Characteristic Value Indications.....	565
4.11.1	Indications.....	565
4.12	Characteristic Descriptors.....	566
4.12.1	Read Characteristic Descriptors	566
4.12.2	Read Long Characteristic Descriptors	566
4.12.3	Write Characteristic Descriptors	567
4.12.4	Write Long Characteristic Descriptors	568



4.13	GATT Procedure Mapping to ATT Protocol Opcodes	569
4.14	Procedure Timeouts	571
5	L2CAP Interoperability Requirements	572
5.1	BR/EDR L2CAP Interoperability Requirements	572
5.1.1	ATT_MTU.....	572
5.1.2	BR/EDR Channel Requirements.....	572
5.1.3	BR/EDR Channel Establishment Collisions	573
5.2	LE L2CAP Interoperability Requirements	573
5.2.1	ATT_MTU.....	573
5.2.2	LE Channel Requirements	574
6	GAP Interoperability Requirements	575
6.1	BR/EDR GAP Interoperability Requirements.....	575
6.1.1	Connection Establishment	575
6.2	LE GAP Interoperability Requirements.....	575
6.2.1	Connection Establishment	575
6.2.2	Profile Roles.....	575
6.3	Disconnected Events	576
6.3.1	Notifications and Indications While Disconnected	576
7	Defined Generic Attribute Profile Service	577
7.1	Service Changed	577
8	Security Considerations	579
8.1	Authentication Requirements.....	579
8.2	Authorization Requirements	580
9	SDP Interoperability Requirements.....	581
10	References	582
Appendix A Example Attribute Server Attributes		583

Part H SECURITY MANAGER SPECIFICATION

1	Introduction	591
1.1	Scope	591
1.2	Conventions.....	591
1.2.1	Bit and Byte Ordering Conventions.....	591
2	Security Manager.....	592
2.1	Introduction	592
2.2	Cryptographic Toolbox	594
2.2.1	Security function e	595
2.2.2	Random Address Hash function ah	595



2.2.3	Confirm value generation function $c1$ for LE Legacy Pairing.....	596
2.2.4	Key generation function $s1$ for LE Legacy Pairing	598
2.2.5	Function AES-CMAC	599
2.2.6	LE Secure Connections Confirm Value Generation Function $f4$	599
2.2.7	LE Secure Connections Key Generation Function $f5$. 600	
2.2.8	LE Secure Connections Check Value Generation Function $f6$	602
2.2.9	LE Secure Connections Numeric Comparison Value Generation Function $g2$	603
2.2.10	Link Key Conversion Function $h6$	604
2.3	Pairing Methods.....	605
2.3.1	Security Properties.....	606
2.3.2	IO Capabilities.....	607
2.3.3	OOB Authentication Data.....	608
2.3.4	Encryption Key Size.....	608
2.3.5	Pairing Algorithms.....	609
2.3.6	Repeated Attempts	623
2.4	Security in Bluetooth low energy	624
2.4.1	Definition of Keys and Values	624
2.4.2	Generation of Distributed Keys.....	624
2.4.3	Distribution of Keys	627
2.4.4	Encrypted Session Setup.....	628
2.4.5	Signing Algorithm.....	629
2.4.6	Slave Security Request.....	631
3	Security Manager Protocol	633
3.1	Introduction	633
3.2	Security Manager Channel over L2CAP	633
3.3	Command Format.....	634
3.4	SMP Timeout	635
3.5	Pairing Methods.....	635
3.5.1	Pairing Request	635
3.5.2	Pairing Response.....	638
3.5.3	Pairing Confirm	639
3.5.4	Pairing Random	640
3.5.5	Pairing Failed	642
3.5.6	Pairing Public Key.....	644
3.5.7	Pairing DHKey Check	644



3.5.8	Keypress Notification	645
3.6	Security in Bluetooth low energy	646
3.6.1	Key Distribution and Generation	646
3.6.2	Encryption Information	649
3.6.3	Master Identification	649
3.6.4	Identity Information	650
3.6.5	Identity Address Information	651
3.6.6	Signing Information	651
3.6.7	Security Request	652
Appendix A	EDIV and Rand Generation	654
A.1	EDIV Masking	654
A.1.1	DIV Mask generation function dm	654
A.1.2	EDIV Generation	655
A.1.3	DIV Recovery	655
Appendix B	Key Management	656
B.1	Database Lookup	656
B.2	Key Hierarchy	656
B.2.1	Diversifying function d1	657
B.2.2	Generating Keys from ER	658
B.2.3	Generating Keys from IR	659
Appendix C	Message Sequence Charts	660
C.1	Phase 1: Pairing Feature Exchange	660
C.1.1	Slave Security Request – Master Requests Pairing ..	661
C.2	Phase 2: Authenticating and Encrypting	662
C.2.1	LE Legacy Pairing	662
C.2.2	LE Secure Connections	665
C.3	Phase 3: Transport Specific Key Distribution	678
C.4	Security Re-established using Previously Distributed LTK ..	678
C.4.1	Master Initiated Security - Master Initiated Link Layer Encryption	678
C.4.2	Slave Security Request - Master Initiated Link Layer Encryption	679
C.5	Failure Conditions	679
C.5.1	Pairing Not Supported by Slave	679
C.5.2	Master Rejects Pairing Because of Key Size	680
C.5.3	Slave Rejects Pairing Because of Key Size	680



C.5.4 Passkey Entry Failure on Master	681
C.5.5 Passkey Entry Failure on Slave	682
C.5.6 Slave Rejects Master's Confirm Value	683
C.5.7 Master Rejects Slaves Confirm Value	684
Appendix D Sample Data	685
D.1 AES-CMAC RFC4493 Test Vectors	685
D.1.1 Example 1: Len = 0	685
D.1.2 Example 2: Len = 16	685
D.1.3 Example 3: Len = 40	685
D.1.4 Example 4: Len = 64	685
D.2 f4 LE SC Confirm Value Generation Function	686
D.3 f5 LE SC Key Generation Function	687
D.4 f6 LE SC Check Value Generation Function	687
D.5 g2 LE SC Numeric Comparison Generation Function	688
D.6 h6 LE SC Link Key Conversion Function	688
D.7 ah Random Address Hash Functions	688
LIST OF FIGURES (ALL PARTS)	689
LIST OF TABLES (ALL PARTS)	697

Core System Package [Host volume]

Part A

LOGICAL LINK CONTROL AND ADAPTATION PROTOCOL SPECIFICATION

The Bluetooth logical link control and adaptation protocol (L2CAP) supports higher level protocol multiplexing, packet segmentation and reassembly, and the conveying of quality of service information. The protocol state machine, packet format, and composition are described in this document.

CONTENTS

1	Introduction	29
1.1	L2CAP Features	29
1.2	Assumptions	32
1.3	Scope	33
1.4	Terminology	33
2	General Operation	37
2.1	Channel Identifiers	37
2.2	Operation Between Devices	40
2.3	Operation Between Layers	41
2.4	Modes of Operation	42
2.5	Mapping Channels to Logical Links	44
3	Data Packet Format	45
3.1	Connection-oriented Channels in Basic L2CAP Mode	45
3.2	Connectionless Data Channel in Basic L2CAP Mode	46
3.3	Connection-oriented Channel in Retransmission/Flow Control/ Streaming Modes	47
3.3.1	L2CAP header fields	48
3.3.2	Control field (2 or 4 octets)	49
3.3.3	L2CAP SDU Length Field (2 octets)	51
3.3.4	Information Payload Field	52
3.3.5	Frame Check Sequence (2 octets)	52
3.3.6	Invalid Frame Detection	53
3.3.7	Invalid Frame Detection Algorithm	53
3.4	Connection-Oriented Channels in LE Credit Based Flow Control Mode	55
3.4.1	L2CAP Header Fields	55
3.4.2	L2CAP SDU Length Field (2 octets)	55
3.4.3	Information Payload Field	55
4	Signaling Packet Formats	57
4.1	Command Reject (code 0x01)	60
4.2	Connection Request (code 0x02)	61
4.3	Connection Response (code 0x03)	63
4.4	Configuration Request (code 0x04)	65
4.5	Configuration Response (code 0x05)	67
4.6	Disconnection Request (code 0x06)	69
4.7	Disconnection Response (code 0x07)	70
4.8	Echo Request (code 0x08)	70
4.9	Echo Response (code 0x09)	71

4.10	Information Request (code 0x0A).....	71
4.11	Information Response (code 0x0B).....	72
4.12	Extended Feature Mask	74
4.13	Fixed Channels Supported	75
4.14	Create Channel Request (code 0x0C)	76
4.15	Create Channel Response (code 0x0D).....	77
4.16	Move Channel Request (code 0x0E).....	78
4.17	Move Channel Response (code 0x0F)	80
4.18	Move Channel Confirmation (code 0x10)	81
4.19	Move Channel Confirmation Response (code 0x11)	82
4.20	Connection Parameter Update Request (code 0x12).....	82
4.21	Connection Parameter Update Response (code 0x13).....	84
4.22	LE Credit Based Connection Request (Code 0x14)	85
4.23	LE Credit Based Connection Response (Code 0x15)	86
4.24	LE Flow Control Credit (Code 0x16).....	88
5	Configuration Parameter Options	89
5.1	Maximum Transmission Unit (MTU)	89
5.2	Flush Timeout Option	91
5.3	Quality of Service (QoS) Option	92
5.4	Retransmission and Flow Control Option	96
5.5	Frame Check Sequence (FCS) Option.....	101
5.6	Extended Flow Specification Option	102
5.7	Extended Window Size Option	108
6	State Machine	110
6.1	General rules for the state machine:.....	110
6.1.1	CLOSED state	112
6.1.2	WAIT_CONNECT_RSP state	113
6.1.3	WAIT_CONNECT state	114
6.1.4	CONFIG state	114
6.1.5	OPEN state	120
6.1.6	WAIT_DISCONNECT state	121
6.1.7	WAIT_CREATE_RSP state	122
6.1.8	WAIT_CREATE state	122
6.1.9	WAIT_MOVE_RSP state	123
6.1.10	WAIT_MOVE state.....	124
6.1.11	WAIT_MOVE_CONFIRM state.....	124
6.1.12	WAIT_CONFIRM_RSP state	125
6.2	Timers events	126
6.2.1	RTX.....	126
6.2.2	ERTX	127

7	General Procedures.....	131
7.1	Configuration Process	131
7.1.1	Request Path	132
7.1.2	Response Path	133
7.1.3	Lockstep Configuration Process	133
7.1.4	Standard Configuration Process	136
7.2	Fragmentation and Recombination.....	138
7.2.1	Fragmentation of L2CAP PDUs	138
7.2.2	Recombination of L2CAP PDUs	139
7.3	Encapsulation of SDUs.....	140
7.3.1	Segmentation of L2CAP SDUs	140
7.3.2	Reassembly of L2CAP SDUs	141
7.3.3	Segmentation and fragmentation.....	141
7.4	Delivery of Erroneous L2CAP SDUs	142
7.5	Operation with Flushing On ACL-U Logical Links.....	142
7.6	Connectionless Data Channel	143
7.7	Operation Collision Resolution	145
7.8	Aggregating Best Effort Extended Flow Specifications.....	145
7.9	Prioritizing Data over HCI	147
7.10	Supporting Extended Flow Specification for BR/EDR and BR/EDR/LE Controllers	147
8	Procedures for Flow Control and Retransmission.....	149
8.1	Information Retrieval	149
8.2	Function of PDU Types for Flow Control and Retransmission. 149	149
8.2.1	Information frame (I-frame).....	149
8.2.2	Supervisory Frame (S-frame)	149
8.2.2.1	Receiver Ready (RR)	149
8.2.2.2	Reject (REJ)	150
8.3	Variables and Sequence Numbers	150
8.3.1	Sending peer.....	151
8.3.1.1	Send sequence number TxSeq.....	151
8.3.1.2	Send state variable NextTXSeq	151
8.3.1.3	Acknowledge state variable ExpectedAckSeq	151
8.3.2	Receiving peer.....	152
8.3.2.1	Receive sequence number ReqSeq.....	152
8.3.2.2	Receive state variable, ExpectedTxSeq.....	153
8.3.2.3	Buffer state variable BufferSeq.....	153
8.4	Retransmission Mode	154
8.4.1	Transmitting frames	154
8.4.1.1	Last received R was set to zero	154

8.4.1.2	Last received R was set to one	156
8.4.2	Receiving I-frames	156
8.4.3	I-frames pulled by the SDU reassembly function.....	157
8.4.4	Sending and receiving acknowledgements.....	157
8.4.4.1	Sending acknowledgements.....	157
8.4.4.2	Receiving acknowledgements	157
8.4.5	Receiving REJ frames	158
8.4.6	Waiting acknowledgements	159
8.4.7	Exception conditions	159
8.4.7.1	TxSeq Sequence error	159
8.4.7.2	ReqSeq Sequence error.....	160
8.4.7.3	Timer recovery error	160
8.4.7.4	Invalid frame	160
8.5	Flow Control Mode	161
8.5.1	Transmitting I-frames	161
8.5.2	Receiving I-frames	162
8.5.3	I-frames pulled by the SDU reassembly function.....	162
8.5.4	Sending and receiving acknowledgements.....	162
8.5.4.1	Sending acknowledgements.....	162
8.5.4.2	Receiving acknowledgements	163
8.5.5	Waiting acknowledgements	163
8.5.6	Exception conditions	164
8.5.6.1	TxSeq Sequence error	164
8.5.6.2	ReqSeq Sequence error.....	164
8.5.6.3	Invalid frame	165
8.6	Enhanced Retransmission Mode	165
8.6.1	Function Of PDU Types	165
8.6.1.1	Receiver Ready (RR)	165
8.6.1.2	Reject (REJ)	165
8.6.1.3	Receiver Not Ready (RNR)	166
8.6.1.4	Selective Reject (SREJ)	166
8.6.1.5	Functions of the Poll (P) and Final (F) bits... 166	166
8.6.2	Rules For Timers	167
8.6.2.1	Timer Rules for ACL-U Logical Links	167
8.6.2.2	Timer Rules for AMP Controllers.....	168
8.6.2.3	Timer Values used After a Move Operation.. 169	169
8.6.3	General Rules for the State Machine	169
8.6.4	State Diagram	171
8.6.5	States Tables.....	171
8.6.5.1	State Machines	171
8.6.5.2	States	172
8.6.5.3	Variables and Timers	172

8.6.5.4	Events.....	175
8.6.5.5	Conditions	176
8.6.5.6	Actions.....	178
8.6.5.7	XMIT State Table	184
8.6.5.8	WAIT_F State Table.....	185
8.6.5.9	RECV State Table.....	186
8.6.5.10	REJ_SENT State Table	189
8.6.5.11	SREJ_SENT State Table.....	193
8.7	Streaming Mode	197
8.7.1	Transmitting I-frames	197
8.7.2	Receiving I-frames	197
8.7.3	Exception Conditions	198
8.7.3.1	TxSeq Sequence error	198
9	Procedure for AMP Channel Creation and Handling.....	199
9.1	Create Channel	199
9.2	Move Channel	202
9.2.1	Move Channel Protocol Procedure with Enhanced Retransmission Mode	203
9.2.1.1	Enhanced Retransmission Mode Procedures During a Move Operation	205
9.2.2	Move Channel Protocol Procedure with Streaming Mode (Initiator is Data Source)	206
9.2.3	Move Channel Protocol Procedure with Streaming Mode (Initiator is Data Sink).....	207
9.3	Disconnect Channel	210
10	Procedures for Credit Based Flow Control	211
10.1	LE Credit Based Flow Control Mode	211
Appendix A	Configuration MSCs	212

1 INTRODUCTION

This section of the Bluetooth Specification defines the Logical Link Control and Adaptation Layer Protocol, referred to as L2CAP. L2CAP provides connection-oriented and connectionless data services to upper layer protocols with protocol multiplexing capability and segmentation and reassembly operation. L2CAP permits higher level protocols and applications to transmit and receive upper layer data packets (L2CAP Service Data Units, SDU) up to 64 kilobytes in length. L2CAP also permits per-channel flow control and retransmission.

The L2CAP layer provides logical channels, named L2CAP channels, which are multiplexed over one or more logical links.

1.1 L2CAP FEATURES

The functional requirements for L2CAP include protocol/channel multiplexing, segmentation and reassembly (SAR), per-channel flow control, and error control. L2CAP sits above a lower layer composed of one of the following:

1. BR/EDR Controller and zero or more AMP Controllers or
2. BR/EDR/LE Controller (supporting BR/EDR and LE) and zero or more AMP Controllers, or
3. LE Controller (supporting LE only)

L2CAP interfaces with upper layer protocols.

[Figure 1.1 on page 30](#) breaks down L2CAP into its architectural components. The Channel Manager provides the control plane functionality and is responsible for all internal signaling, L2CAP peer-to-peer signaling and signaling with higher and lower layers. It performs the state machine functionality described in [Section 6 on page 110](#) and uses message formats described in [Section 4 on page 57](#), and [Section 5 on page 89](#). The Retransmission and Flow Control block provides per-channel flow control and error recovery using packet retransmission. The Resource Manager is responsible for providing a frame relay service to the Channel Manager, the Retransmission and Flow Control block and those application data streams that do not require Retransmission and Flow Control services. It is responsible for coordinating the transmission and reception of packets related to multiple L2CAP channels over the facilities offered at the lower layer interface.

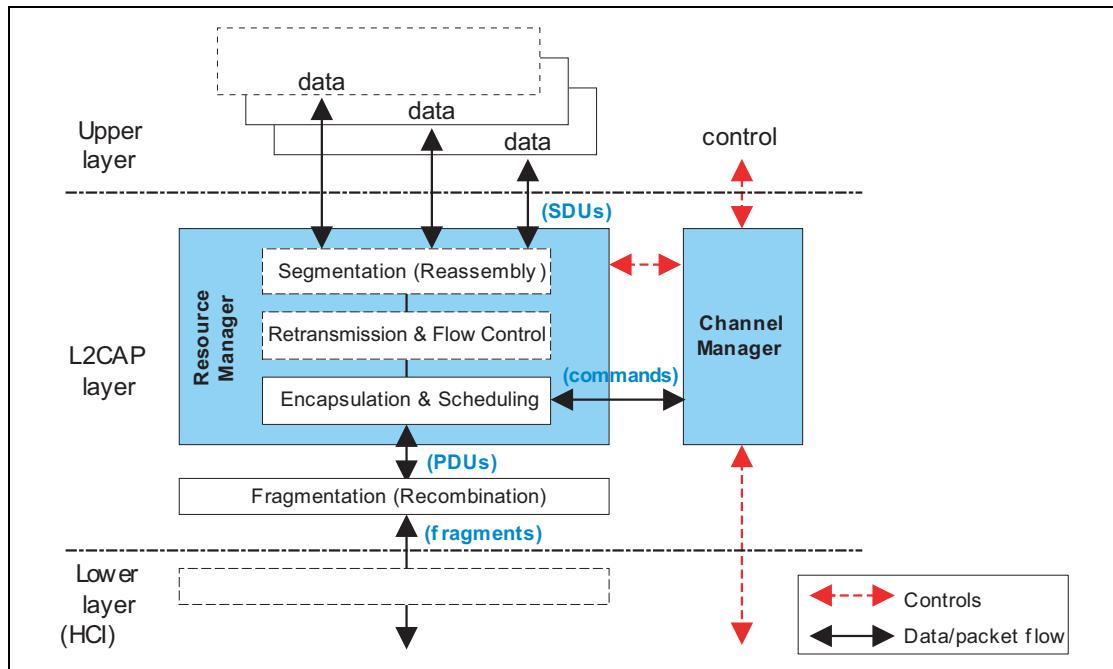


Figure 1.1: L2CAP architectural blocks

- *Protocol/channel multiplexing*

L2CAP supports multiplexing over individual Controllers and across multiple controllers. An L2CAP channel shall operate over one Controller at a time. During channel setup, protocol multiplexing capability is used to route the connection to the correct upper layer protocol.

For data transfer, logical channel multiplexing is needed to distinguish between multiple upper layer entities. There may be more than one upper layer entity using the same protocol.

- *Segmentation and reassembly*

With the frame relay service offered by the Resource Manager, the length of transport frames is controlled by the individual applications running over L2CAP. Many multiplexed applications are better served if L2CAP has control over the PDU length. This provides the following benefits:

- Segmentation will allow the interleaving of application data units in order to satisfy latency requirements.
- Memory and buffer management is easier when L2CAP controls the packet size.
- Error correction by retransmission can be made more efficient.
- The amount of data that is destroyed when an L2CAP PDU is corrupted or lost can be made smaller than the application's data unit.
- The application is decoupled from the segmentation required to map the application packets into the lower layer packets.

- *Flow control per L2CAP channel*

Controllers provide error and flow control for data going over the air and HCI flow control exists for data going over an HCI transport. When several data streams run over the same Controller using separate L2CAP channels, each channel requires individual flow control. A window based flow control scheme is provided.

- *Error control and retransmissions*

When L2CAP channels are moved from one Controller to another data can be lost. Also, some applications require a residual error rate much smaller than some Controllers can deliver. L2CAP provides error checks and retransmissions of L2CAP PDUs. The error checking in L2CAP protects against errors due to Controllers falsely accepting packets that contain errors but pass Controller-based integrity checks. L2CAP error checking and retransmission also protect against loss of packets due to flushing by the Controller. The error control works in conjunction with flow control in the sense that the flow control mechanism will throttle retransmissions as well as first transmissions.

- *Support for Streaming*

Streaming applications such as audio set up an L2CAP channel with an agreed-upon data rate and do not want flow control mechanisms, including those in the Controller, to alter the flow of data. A flush timeout is used to keep data flowing on the transmit side. Streaming mode is used to stop HCI and Controller based flow control from being applied on the receiving side.

- *Fragmentation and Recombination*

Some Controllers may have limited transmission capabilities and may require fragment sizes different from those created by L2CAP segmentation. Therefore layers below L2CAP may further fragment and recombine L2CAP PDUs to create fragments which fit each layer's capabilities. During transmission of an L2CAP PDU, many different levels of fragmentation and recombination may occur in both peer devices.

The HCI driver or controller may fragment L2CAP PDUs to honor packet size constraints of a host controller interface transport scheme. This results in HCI data packet payloads carrying start and continuation fragments of the L2CAP PDU. Similarly the Controller may fragment L2CAP PDUs to map them into Controller packets. This may result in Controller packet payloads carrying start and continuation fragments of the L2CAP PDU.

Each layer of the protocol stack may pass on different sized fragments of L2CAP PDUs, and the size of fragments created by a layer may be different in each peer device. However the PDU is fragmented within the stack, the receiving L2CAP entity still recombines the fragments to obtain the original L2CAP PDU.

- *Quality of Service*

The L2CAP connection establishment process allows the exchange of information regarding the quality of service (QoS) expected between two Bluetooth devices. Each L2CAP implementation monitors the resources used by the protocol and ensures that QoS contracts are honored.

For a BR/EDR or BR/EDR/LE Controller, L2CAP may support both isochronous (Guaranteed) and asynchronous (Best Effort) data flows over the same ACL logical link by marking packets as automatically-flushable or non-autonomously-flushable by setting the appropriate value for the `Packet_Boundary_Flag` in the HCI ACL Data Packet (see [\[vol.2, part E\] Section 5.4.2 on page 472](#)). Automatically-flushable L2CAP packets are flushed according to the automatic flush timeout set for the ACL logical link on which the L2CAP channels are mapped (see [\[vol.2, part E\] Section 6.19 on page 486](#)). Non-autonomously-flushable L2CAP packets are not affected by the automatic flush timeout and will not be flushed. All L2CAP packets can be flushed by using the HCI Flush command (see [\[vol.2, part E\] Section 7.3.4 on page 652](#)).

For AMP Controllers, L2CAP places all asynchronous data flows going to the same remote device over a single logical link (aggregation). L2CAP places each isochronous data flow over its own logical link.

1.2 ASSUMPTIONS

The protocol is designed based on the following assumptions:

1. Controllers provide orderly delivery of data packets, although there might be individual packet corruption and duplicates. For devices with a BR/EDR or BR/EDR/LE Controller, no more than one ACL-U logical link exists between any two devices. For devices with a given AMP Controller, more than one AMP-U logical link may exist between any two devices. For devices with a BR/EDR/LE or LE Controller, no more than one LE-U logical link exists between any two devices.
2. Controllers always provide the impression of full-duplex communication channels. This does not imply that all L2CAP communications are bi-directional. Unidirectional traffic does not require duplex channels.
3. The L2CAP layer provides a channel with a degree of reliability based on the mechanisms available in Controllers and with additional packet segmentation, error detection, and retransmission that can be enabled in the enhanced L2CAP layer. Some Controllers perform data integrity checks and resend data until it has been successfully acknowledged or a timeout occurs. Other Controllers will resend data up to a certain number of times whereupon the data is flushed. Because acknowledgments may be lost, timeouts may occur even after the data has been successfully sent. Note that the use of baseband broadcast packets in a BR/EDR or BR/EDR/LE Controller is unreliable and that all broadcasts start the first segment of an L2CAP packet with the same sequence bit.

4. Controllers provide error and flow control for data going over the air and HCI flow control exists for data going over an HCI transport but some applications will want better error control than some controllers provide. Also, moving channels between Controllers requires L2CAP flow and error control. The Flow and Error Control block provides four modes. Enhanced Retransmission mode and Retransmission Mode offer segmentation, flow control and L2CAP PDU retransmissions. Flow control mode offers just the segmentation and flow control functions. Streaming mode offers segmentation and receiver side packet flushing.

1.3 SCOPE

The following features are outside the scope of L2CAP's responsibilities:

- L2CAP does not transport synchronous data designated for SCO or eSCO logical transports.
- L2CAP does not support a reliable broadcast channel. See [Section 3.2 on page 46](#).

1.4 TERMINOLOGY

The following formal definitions apply:

Term	Description
Upper layer	The system layer above the L2CAP layer, which exchanges data with L2CAP in the form of SDUs. The upper layer may be represented by an application or higher protocol entity known as the Service Level Protocol. The interface of the L2CAP layer with the upper layer is not specified.
Lower layer	The system layer below the L2CAP layer, which exchanges data with the L2CAP layer in the form of PDUs, or fragments of PDUs. The lower layer is mainly represented within the Controller, however a Host Controller Interface (HCI) may be involved, such that an HCI host driver could also be seen as the lower layer. Except for the HCI functional specification (in case HCI is involved) the interface between L2CAP and the lower layer is not specified.
L2CAP channel	The logical connection between two endpoints in peer devices, characterized by their Channel Identifiers (CID), which is multiplexed over one Controller based logical link.
SDU, or L2CAP SDU	Service Data Unit: a packet of data that L2CAP exchanges with the upper layer and transports transparently over an L2CAP channel using the procedures specified here. The term SDU is associated with data originating from upper layer entities only, i.e. does not include any protocol information generated by L2CAP procedures.

Table 1.1: Terminology

Term	Description
Segment, or SDU segment	<p>A part of an SDU, as resulting from the Segmentation procedure. An SDU may be split into one or more segments.</p> <p>Note: this term is relevant only to Enhanced Retransmission mode, Streaming mode, Retransmission Mode and Flow Control Mode, not to the Basic L2CAP Mode.</p>
Segmentation	<p>A procedure used in the L2CAP Retransmission and Flow Control Modes, resulting in an SDU being split into one or more smaller units, called Segments, as appropriate for the transport over an L2CAP channel.</p> <p>Note: this term is relevant only to the Enhanced Retransmission mode, Streaming mode, Retransmission Mode and Flow Control Mode, not to the Basic L2CAP Mode.</p>
Reassembly	<p>The reverse procedure corresponding to Segmentation, resulting in an SDU being re-established from the segments received over an L2CAP channel, for use by the upper layer. Note that the interface between the L2CAP and the upper layer is not specified; therefore, reassembly may actually occur within an upper layer entity although it is conceptually part of the L2CAP layer.</p> <p>Note: this term is relevant only to Enhanced Retransmission mode, Streaming mode, Retransmission Mode and Flow Control Mode, not to the Basic L2CAP Mode.</p>
PDU, or L2CAP PDU	<p>Protocol Data Unit a packet of data containing L2CAP protocol information fields, control information, and/or upper layer information data. A PDU is always started by a Basic L2CAP header. Types of PDUs are: B-frames, I-frames, S-frames, C-frames and G-frames.</p>
Basic L2CAP header	<p>Minimum L2CAP protocol information that is present in the beginning of each PDU: a length field and a field containing the Channel Identifier (CID).</p>
Basic information frame (B-frame)	<p>A B-frame is a PDU used in the Basic L2CAP mode for L2CAP data packets. It contains a complete SDU as its payload, encapsulated by a Basic L2CAP header.</p>
Information frame (I-frame)	<p>An I-frame is a PDU used in Enhanced Retransmission Mode, Streaming mode, Retransmission mode, and Flow Control Mode. It contains an SDU segment and additional protocol information, encapsulated by a Basic L2CAP header</p>
Supervisory frame (S-frame)	<p>An S-frame is a PDU used in Enhanced Retransmission Mode, Retransmission Mode, and Flow Control Mode. It contains protocol information only, encapsulated by a Basic L2CAP header, and no SDU data.</p>
Control frame (C-frame)	<p>A C-frame is a PDU that contains L2CAP signaling messages exchanged between the peer L2CAP entities. C-frames are exclusively used on the L2CAP signaling channel.</p>

Table 1.1: Terminology

Term	Description
Group frame (G-frame)	A G-frame is a PDU exclusively used on the Connectionless L2CAP channel. It is encapsulated by a Basic L2CAP header and contains the PSM followed by the completed SDU. G-frames may be used to broadcast data to multiple slaves (either to all slaves via Picocell Broadcast or to only active slaves via Active Broadcast) or to send unicast data to a single remote device.
LE Information frame (LE-frame)	An LE-frame is a PDU used in LE Credit Based Flow Control Mode. It contains an SDU segment and additional protocol information, encapsulated by a Basic L2CAP header.
Fragment	<p>A part of a PDU, as resulting from a fragmentation operation. Fragments are used only in the delivery of data to and from the lower layer. They are not used for peer-to-peer transportation. A fragment may be a Start or Continuation Fragment with respect to the L2CAP PDU. A fragment does not contain any protocol information beyond the PDU; the distinction of start and continuation fragments is transported by lower layer protocol provisions.</p> <p>Note: Start Fragments always begin with the Basic L2CAP header of a PDU.</p>
Fragmentation	<p>A procedure used to split L2CAP PDUs to smaller parts, named fragments, appropriate for delivery to the lower layer transport. Although described within the L2CAP layer, fragmentation may actually occur in an HCI host driver, and/or in a Controller, to accommodate the L2CAP PDU transport to HCI data packet or Controller packet sizes.</p> <p>Fragmentation of PDUs may be applied in all L2CAP modes.</p> <p>Note: in version 1.1, Fragmentation and Recombination was referred to as "Segmentation and Reassembly".</p>
Recombination	The reverse procedure corresponding to fragmentation, resulting in an L2CAP PDU re-established from fragments. In the receive path, full or partial recombination operations may occur in the Controller and/or the Host, and the location of recombination does not necessarily correspond to where fragmentations occurs on the transmit side.
Maximum Transmission Unit (MTU)	The maximum size of payload data, in octets, that the upper layer entity is capable of accepting, i.e. the MTU corresponds to the maximum SDU size.
Maximum PDU payload Size (MPS)	<p>The maximum size of payload data in octets that the L2CAP layer entity is capable of accepting, i.e. the MPS corresponds to the maximum PDU payload size.</p> <p>Note: in the absence of segmentation, or in the Basic L2CAP Mode, the Maximum Transmission Unit is the equivalent to the Maximum PDU payload Size and shall be made equal in the configuration parameters.</p>
Signaling MTU (MTU _{sig})	The maximum size of command information that the L2CAP layer entity is capable of accepting. The MTU _{sig} , refers to the signaling channel only and corresponds to the maximum size of a C-frame, excluding the Basic L2CAP header. The MTU _{sig} value of a peer is discovered when a C-frame that is too large is rejected by the peer.

Table 1.1: Terminology

Term	Description
Connectionless MTU (MTU _{cnl})	The maximum size of the connection packet information that the L2CAP layer entity is capable of accepting. The MTU _{cnl} refers to the connectionless channel only and corresponds to the maximum G-frame, excluding the Basic L2CAP header and the PSM which immediately follows it. The MTU _{cnl} of a peer can be discovered by sending an Information Request.
MaxTransmit	<p>In Enhanced Retransmission mode and Retransmission mode, MaxTransmit controls the number of transmissions of a PDU that L2CAP is allowed to try before assuming that the PDU (and the link) is lost. The minimum value is 1 (only 1 transmission permitted). In Enhanced Retransmission mode a value 0 means infinite transmissions.</p> <p>Note: Setting MaxTransmit to 1 prohibits PDU retransmissions. Failure of a single PDU will cause the link to drop. By comparison, in Flow Control mode, failure of a single PDU will not necessarily cause the link to drop.</p>

Table 1.1: Terminology

2 GENERAL OPERATION

L2CAP is based around the concept of '*channels*'. Each one of the endpoints of an L2CAP channel is referred to by a *channel identifier (CID)*.

2.1 CHANNEL IDENTIFIERS

A channel identifier (CID) is the local name representing a logical channel endpoint on the device. The scope of CIDs is related to the logical link as shown in [Figure 2.1](#). The null identifier (0x0000) shall never be used as a destination endpoint. Identifiers from 0x0001 to 0x003F are reserved for specific L2CAP functions. These channels are referred to as Fixed Channels. At a minimum, the L2CAP Signaling channel (Fixed Channel 0x0001) or the L2CAP LE Signaling channel (Fixed Channel 0x0005) shall be supported. If Fixed Channel 0x0005 is supported, then Fixed Channels 0x0004 and 0x0006 shall be supported (see [Table 2.1](#)). Other fixed channels may be supported. The Information Request / Response mechanism (described in [Section 4.10](#) and [Section 4.11](#)) shall be used to determine which fixed channels a remote device supports over the ACL-U logical link.

The characteristics of each fixed channel are defined on a per channel basis. Fixed channel characteristics include configuration parameters (e.g., reliability, MTU size, QoS), security, and the ability to change parameters using the L2CAP configuration mechanism. [Table 2.1](#) lists the defined fixed channels, provides a reference to where the associated channel characteristics are defined and specifies the logical link over which the channel may operate. Fixed channels are available as soon as the ACL-U or LE-U logical link is set up. All initialization that is normally performed when a channel is created shall be performed for each of the supported fixed channels when the ACL-U or LE-U logical link is set up. Fixed channels shall only run over ACL-U or LE-U logical links and shall not be moved.

Implementations are free to manage the remaining CIDs in a manner best suited for that particular implementation, with the provision that two simultaneously active L2CAP channels shall not share the same CID. A different CID name space exists for each ACL-U and LE-U logical link. The AMP-U logical link shares the CID name space with its associated ACL-U logical link. [Table 2.1](#) and [Table 2.2](#) summarizes the definition and partitioning of the CID name space for each logical link.

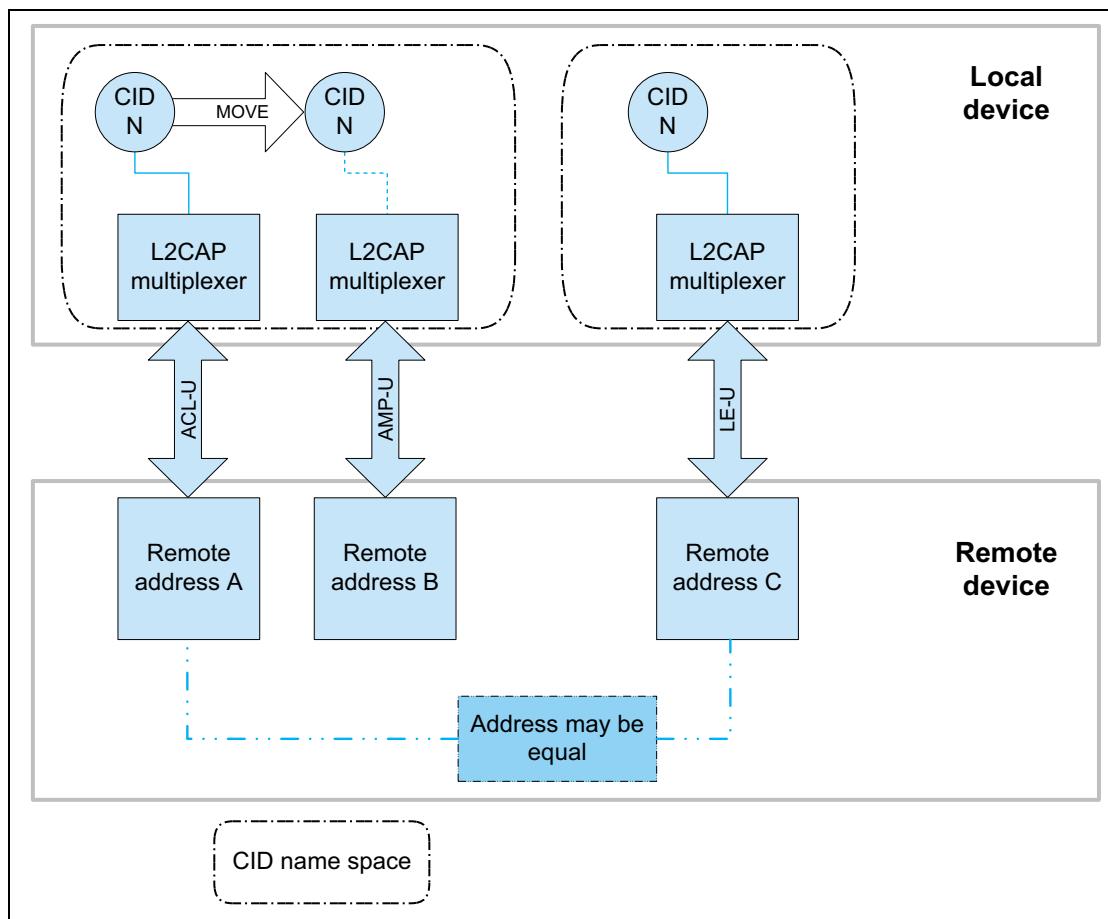


Figure 2.1: Dynamically allocated CID assignments

Assignment of dynamically allocated CIDs is relative to a particular logical link and a device can assign CIDs independently from other devices. Thus, even if the same CID value has been assigned to (remote) channel endpoints by several remote devices connected to a single local device, the local device can still uniquely associate each remote CID with a different device. Further, even if the same CID value has been assigned to (remote) channel endpoints by the same remote device, these can be distinguished because they will be bound to a different logical link.

The CID name space for the ACL-U and AMP-U logical links are as follows:

CID	Description	Channel Characteristics	Logical Link Supported
0x0000	Null identifier	Not allowed	
0x0001	L2CAP Signaling channel	See Section 4 on page 57	ACL-U
0x0002	Connectionless channel	See Section 7.6 on page 143	ACL-U

Table 2.1: CID name space on ACL-U logical link

CID	Description	Channel Characteristics	Logical Link Supported
0x0003	AMP Manager Protocol	See [Part E] Section 2.2 on page 445.	ACL-U
0x0004-0x0006	Reserved	Not applicable	
0x0007	BR/EDR Security Manager	See [Vol 3] Part H	ACL-U
0x0008-0x003E	Reserved	Not applicable	
0x003F	AMP Test Manager	See Part D, Section 1.2.3	ACL-U
0x0040-0xFFFF	Dynamically allocated	Communicated using L2CAP configuration mechanism (see Section 7.1 on page 131)	ACL-U, AMP-U

Table 2.1: CID name space on ACL-U logical link

The CID name space for the LE-U logical link is as follows:

CID	Description	Channel Characteristics	Logical Link Supported
0X0000	Null identifier	Not Allowed	
0x0001-0x0003	Reserved	Not applicable	
0x0004	Attribute Protocol	See [Vol 3] Part F	LE-U
0x0005	Low Energy L2CAP Signaling channel	See Section 4 on page 57	LE-U
0x0006	Security Manager Protocol	See [Vol 3] Part H	LE-U
0x0007-0x001F	Reserved	Not applicable	
0x0020-0x003E	Assigned Numbers		LE-U
0x003F	Reserved	Not applicable	
0x0040-0x007F	Dynamically allocated	Communicated using the L2CAP LE credit based create connection mechanism (see Section 4.22 on page 85)	LE-U
0x0080-0xFFFF	Reserved	Not applicable	

Table 2.2: CID name space on LE-U logical link

2.2 OPERATION BETWEEN DEVICES

[Figure 2.2 on page 40](#) illustrates the use of CIDs in a communication between corresponding peer L2CAP entities in separate devices. The connection-oriented data channels represent a connection between two devices, where a CID, combined with the logical link, identifies each endpoint of the channel. When used for broadcast transmissions, the connectionless channel restricts data flow to a single direction. The connectionless channel may be used to transmit data to all slaves in a piconet (utilizing Piconet Broadcast which includes both active and parked slaves) or to all active slaves (using Active Broadcast which includes only active slaves). When used for unicast transmissions the connectionless channel may be used in either direction between a master and a slave.

There are also a number of CIDs reserved for special purposes. The L2CAP signalling channels are examples of reserved channels. Fixed channel 0x0001 is used to create and establish connection-oriented data channels and to negotiate changes in the characteristics of connection-oriented channels and to discover characteristics of the connectionless channel operating over the ACL-U logical link.

The L2CAP signaling channel and all supported fixed channels are available immediately when the ACL-U logical link is established between two devices. Another CID (0x0002) is reserved for all incoming and outgoing connectionless data traffic, whether broadcast or unicast. Connectionless data traffic may flow immediately once the ACL-U logical link is established between two devices and once the transmitting device has determined that the remote device supports connectionless traffic.

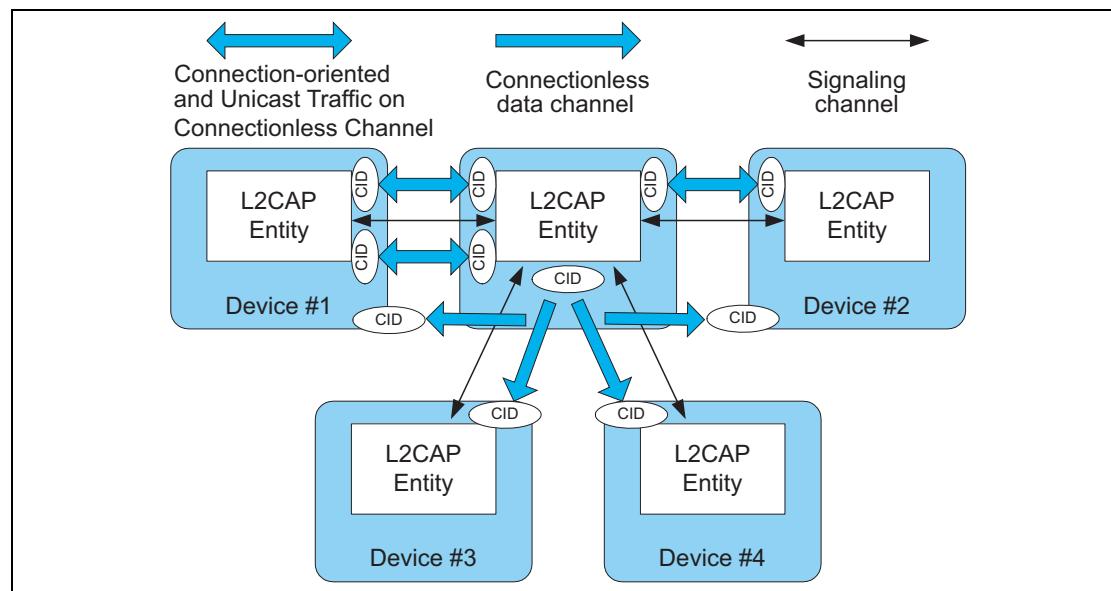


Figure 2.2: Channels between devices

Table 2.3 describes the various channel types and their source and destination identifiers. A dynamically allocated CID is allocated to identify, along with the logical link, the local endpoint and shall be in the range 0x0040 to 0xFFFF. [Section 6 on page 110](#) describes the state machine associated with each connection-oriented channel with a dynamically allocated CID. [Section 3.1 on page 45](#) and [Section 3.3 on page 47](#) describe the packet format associated with connection-oriented channels. [Section 3.2 on page 46](#) describes the packet format associated with the connectionless channel.

Channel Type	Local CID (sending)	Remote CID (receiving)
Connection-oriented	Dynamically allocated and fixed	Dynamically allocated and fixed
Connectionless data	0x0002 (fixed)	0x0002 (fixed)
L2CAP Signaling	0x0001 and 0x0005 (fixed)	0x0001 and 0x0005 (fixed)

Table 2.3: Types of Channel Identifiers

2.3 OPERATION BETWEEN LAYERS

L2CAP implementations should follow the general architecture described below. L2CAP implementations transfer data between upper layer protocols and the lower layer protocol. This document lists a number of services that should be exported by any L2CAP implementation. Each implementation shall also support a set of signaling commands for use between L2CAP implementations. L2CAP implementations should also be prepared to accept certain types of events from lower layers and generate events to upper layers. How these events are passed between layers is implementation specific.

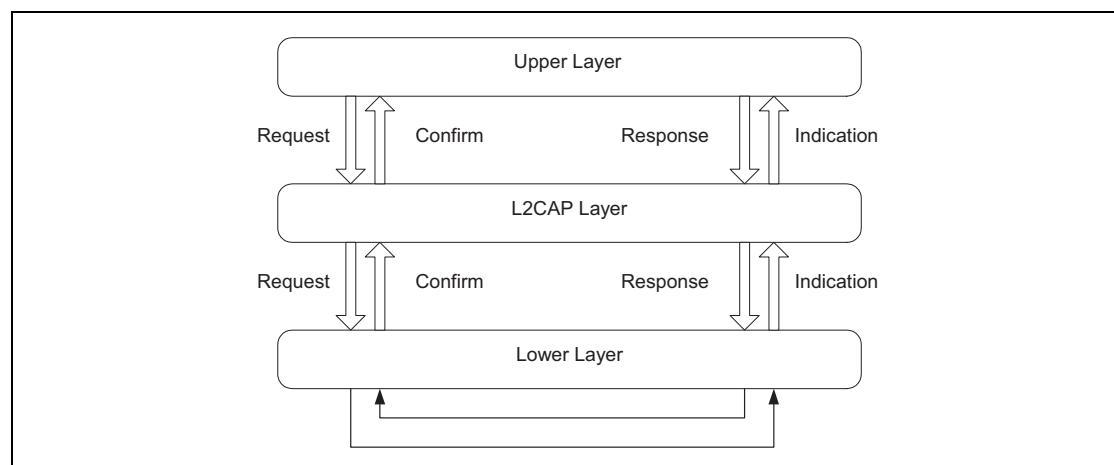


Figure 2.3: L2CAP transaction model

2.4 MODES OF OPERATION

L2CAP channels may operate in one of five different modes as selected for each L2CAP channel.

The modes are:

- Basic L2CAP Mode (equivalent to L2CAP specification in Bluetooth v1.1)¹
- Flow Control Mode
- Retransmission Mode
- Enhanced Retransmission Mode
- Streaming Mode
- LE Credit Based Flow Control Mode

The modes are enabled using the configuration procedure described in [Section 7.1](#). The Basic L2CAP Mode shall be the default mode, which is used when no other mode is agreed. Enhanced Retransmission mode shall be used for all reliable channels created over AMP-U logical links and for ACL-U logical links operating as described in [Section 7.10](#). Enhanced Retransmission mode should be enabled for reliable channels created over ACL-U logical links not operating as described in [Section 7.10](#). Streaming mode shall be used for streaming applications created over AMP-U logical links and ACL-U logical links operating as described in [Section 7.10](#). Streaming mode should be enabled for streaming applications created over ACL-U logical links not operating as described in [Section 7.10](#). Either Enhanced Retransmission mode or Streaming mode should be enabled when supported by both L2CAP entities. Flow Control Mode and Retransmission mode shall only be enabled when communicating with L2CAP entities that do not support either Enhanced Retransmission mode or Streaming mode.

In Flow Control mode, Retransmission mode, and Enhanced Retransmission mode, PDUs exchanged with a peer entity are numbered and acknowledged. The sequence numbers in the PDUs are used to control buffering, and a TxWindow size is used to limit the required buffer space and/or provide a method for flow control.

In Flow Control Mode no retransmissions take place, but missing PDUs are detected and can be reported as lost.

In Retransmission Mode a timer is used to ensure that all PDUs are delivered to the peer, by retransmitting PDUs as needed. A go-back-n repeat mechanism is used to simplify the protocol and limit the buffering requirements.

Enhanced Retransmission mode is similar to Retransmission mode. It adds the ability to set a POLL bit to solicit a response from a remote L2CAP entity, adds

1. Specification of the Bluetooth System v1.1 (Feb 22nd 2001): volume 1, part D.

the SREJ S-frame to improve the efficiency of error recovery and adds an RNR S-frame to replace the R-bit for reporting a local busy condition.

Streaming mode is for real-time isochronous traffic. PDUs are numbered but are not acknowledged. A finite flush timeout is set on the sending side to flush packets that are not sent in a timely manner. On the receiving side if the receive buffers are full when a new PDU is received then a previously received PDU is overwritten by the newly received PDU. Missing PDUs can be detected and reported as lost. TxWindow size is not used in Streaming mode.

Note: Although L2CAP Basic mode may be used for L2CAP channels operating over ACL-U logical links, only L2CAP channels which have been configured to use Enhanced Retransmission mode or Streaming mode may be moved to operate over an AMP-U logical link. (see [Section 4.16](#)).

LE Credit Based Flow Control Mode is used for LE L2CAP connection oriented channels for flow control using a credit based scheme for L2CAP data (i.e. not signaling packets). This is the only mode that shall be used for LE L2CAP connection oriented channels.

L2CAP channels used for multiplexing layers such as RFCOMM can serve a variety of higher layer protocols. In cases where the local device and remote device have mutual support for an AMP and complementary support for profiles on a multiplexing layer which requires reliability (e.g. RFCOMM), the multiplexing layer should be configured to use Enhanced Retransmission mode to ensure that profiles utilizing the multiplexer are not prevented from being moved to the AMP-U logical link. Similarly, in cases where the local device and remote device have mutual support for an AMP and complementary support for profiles on a multiplexing layer which does not require reliability, the multiplexing layer should be configured to use Streaming mode to ensure that profiles utilizing the multiplexer are not prevented from being moved to the AMP-U logical link.

Care should be taken in selecting the parameters used for Enhanced Retransmission mode and Streaming mode when they are used beneath legacy profile implementations to ensure that performance is not negatively impacted relative to the performance achieved when using the same profile with Basic mode on an ACL-U logical link. When there can never be a mutually supported AMP, nor complementary support for a profile which would benefit from AMP, it may be preferable to configure Basic mode to minimize the risk of negative performance impacts.

2.5 MAPPING CHANNELS TO LOGICAL LINKS

L2CAP maps channels to Controller logical links, which in turn run over Controller physical links. All logical links going between a local Controller and remote Controller run over a single physical link. There is one ACL-U logical link per BR/EDR physical link and one LE-U logical link per LE physical link, while there may be multiple AMP-U logical links per AMP physical link.

All Best Effort and Guaranteed channels going over a BR/EDR physical link between two devices shall be mapped to a single ACL-U logical link. All Best Effort channels going over an AMP physical link between two Controllers shall be mapped to a single AMP-U logical link while each Guaranteed channel going between two Controllers shall be mapped to its own AMP-U logical link with one AMP-U logical link per Guaranteed channel. All channels going over an LE physical link between two devices shall be treated as best effort and mapped to a single LE-U logical link.

When a Guaranteed channel is created or moved to a Controller, a corresponding Guaranteed logical link shall be created to carry the channel traffic. Creation of a Guaranteed logical link involves admission control. Admission control is verifying that the guarantee can be achieved without compromising existing guarantees. For an AMP Controller, L2CAP shall tell the controller to create a Guaranteed logical link and admission control shall be performed by the Controller. For a BR/EDR Controller, admission control (creation of a Guaranteed logical link) shall be performed by the L2CAP layer.

3 DATA PACKET FORMAT

L2CAP is packet-based but follows a communication model based on *channels*. A channel represents a data flow between L2CAP entities in remote devices. Channels may be connection-oriented or connectionless. Fixed channels other than the L2CAP connectionless channel (CID 0x0002) and the two L2CAP signaling channels (CIDs 0x0001 and 0x0005) are considered connection-oriented. All channels with dynamically assigned CIDs are connection-oriented. All L2CAP layer packet fields shall use Little Endian byte order with the exception of the information payload field. The endian-ness of higher layer protocols encapsulated within L2CAP information payload is protocol-specific.

3.1 CONNECTION-ORIENTED CHANNELS IN BASIC L2CAP MODE

[Figure 3.1 on page 45](#) illustrates the format of the L2CAP PDU used on connection-oriented channels. In basic L2CAP mode, the L2CAP PDU on a connection-oriented channel is also referred to as a "B-frame".

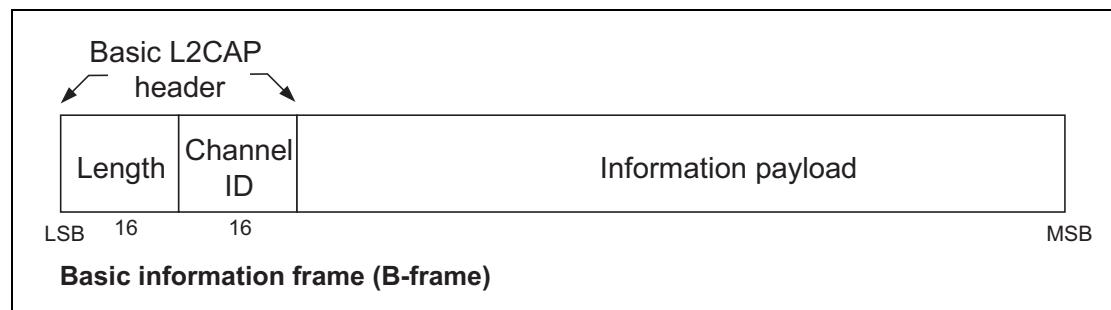


Figure 3.1: L2CAP PDU format in Basic L2CAP mode on connection-oriented channels (field sizes in bits)

The fields shown are:

- *Length: 2 octets (16 bits)*

Length indicates the size of the information payload in octets, excluding the length of the L2CAP header. The length of an information payload can be up to 65535 octets. The Length field is used for recombination and serves as a simple integrity check of the recombined L2CAP packet on the receiving end.

- *Channel ID: 2 octets*

The channel ID (CID) identifies the destination channel endpoint of the packet.

- *Information payload: 0 to 65535 octets*

This contains the payload received from the upper layer protocol (outgoing packet), or delivered to the upper layer protocol (incoming packet). The MTU for channels with dynamically allocated CIDs is determined during channel

configuration (see [Section 5.1 on page 89](#)). The minimum supported MTU values for the signaling PDUs are shown in [Table 4.1 on page 57](#)).

3.2 CONNECTIONLESS DATA CHANNEL IN BASIC L2CAP MODE

[Figure 3.2](#) illustrates the L2CAP PDU format within a connectionless data channel. Here, the L2CAP PDU is also referred to as a "G-frame".

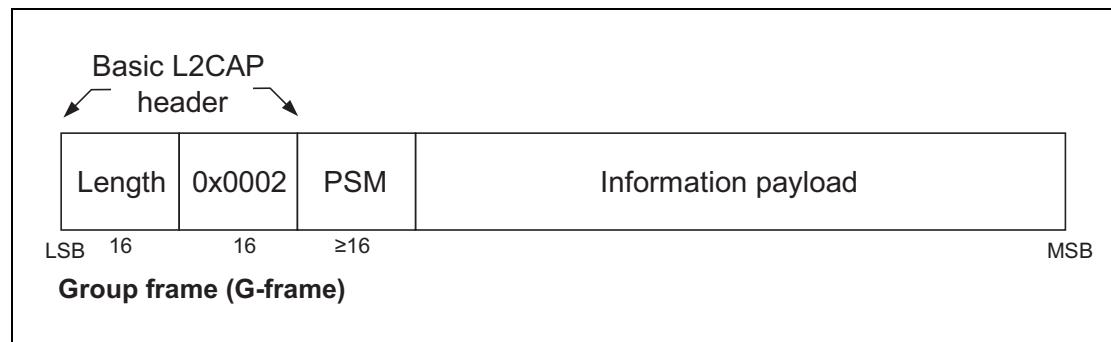


Figure 3.2: L2CAP PDU format on the Connectionless channel

The fields shown are:

- *Length: 2 octets*
Length indicates the size of information payload plus the PSM field in octets.
- *Channel ID: 2 octets*
Channel ID (0x0002) reserved for connectionless traffic.
- *Protocol/Service Multiplexer (PSM): 2 octets (minimum)*
For information on the PSM field see [Section 4.2 on page 61](#).
- *Information payload: 0 to 65533 octets*
This parameter contains the payload information to be distributed to all slaves in the piconet for broadcast connectionless traffic, or to a specific remote device for data sent via the L2CAP connectionless channel.
Implementations shall support a connectionless MTU (MTU_{cnl}) of 48 octets on the connectionless channel. Devices may also explicitly change to a larger or smaller connectionless MTU (MTU_{cnl}).
Note: the maximum size of the Information payload field decreases accordingly if the PSM field is extended beyond the two octet minimum.

3.3 CONNECTION-ORIENTED CHANNEL IN RETRANSMISSION/FLOW CONTROL/STREAMING MODES

To support flow control, retransmissions, and streaming, L2CAP PDU types with protocol elements in addition to the Basic L2CAP header are defined. The information frames (I-frames) are used for information transfer between L2CAP entities. The supervisory frames (S-frames) are used to acknowledge I-frames and request retransmission of I-frames.

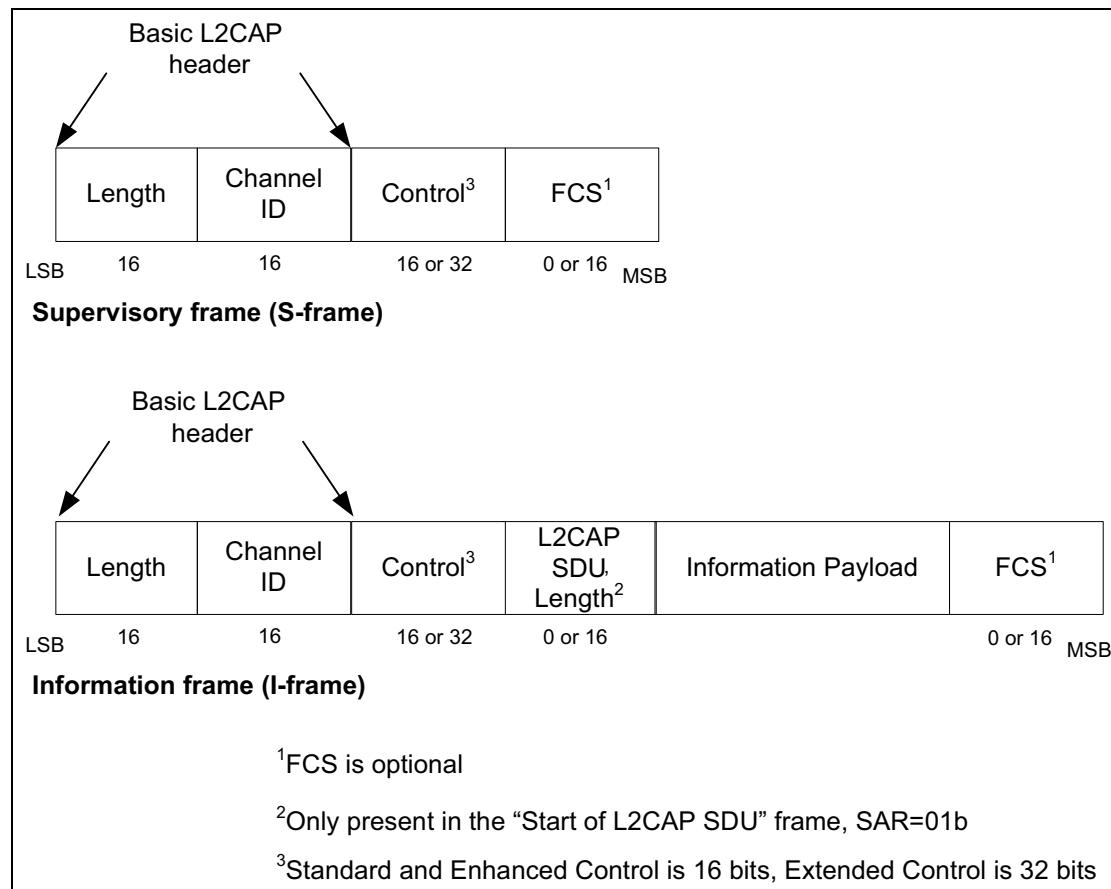


Figure 3.3: L2CAP PDU formats in Flow Control and Retransmission Modes

3.3.1 L2CAP header fields

- *Length: 2 octets*

The first two octets in the L2CAP PDU contain the length of the entire L2CAP PDU in octets, excluding the Length and CID field.

For I-frames and S-frames, the Length field therefore includes the octet lengths of the Control, L2CAP SDU Length (when present), Information octets and frame check sequence (FCS) (when present) fields.

The maximum number of Information octets in one I-frame is based on which fields are present and the type of the Control Field. The maximum number of Information octets in an I-frame with a Standard Control field is as follows:

L2CAP SDU Length present and FCS present	65529 octets
L2CAP SDU Length present and FCS not present	65531 octets
L2CAP SDU Length not present and FCS present	65531 octets
L2CAP SDU Length not present and FCS not present	65533 octets

The maximum number of Information octets in an I-frame with an Extended Control field is as follows:

L2CAP SDU Length present and FCS present	65527 octets
L2CAP SDU Length present and FCS not present	65529 octets
L2CAP SDU Length not present and FCS present	65529 octets
L2CAP SDU Length not present and FCS not present	65531 octets

- *Channel ID: 2 octets*

This field contains the Channel Identification (CID).

3.3.2 Control field (2 or 4 octets)

The Control Field identifies the type of frame. There are three different Control Field formats: the Standard Control Field, the Enhanced Control Field, and the Extended Control Field. The Standard Control Field shall be used for Retransmission mode and Flow Control mode. The Enhanced Control Field shall be used for Enhanced Retransmission mode and Streaming mode. The Extended Control Field may be used for Enhanced Retransmission mode and Streaming mode. The Control Field will contain sequence numbers where applicable. Its coding is shown in [Table 3.1 on page 49](#), [Table 3.2 on page 50](#), and [Table 3.3 on page 50](#). There are two different frame types, Information frame types and Supervisory frame types. Information and Supervisory frames types are distinguished by the least significant bit in the Control Field, as shown in [Table 3.1](#), [Table 3.2 on page 50](#), and [Table 3.3 on page 50](#).

- *Information frame format (I-frame)*

The I-frames are used to transfer information between L2CAP entities. Each I-frame has a TxSeq(Send sequence number), ReqSeq(Receive sequence number) which may or may not acknowledge additional I-frames received by the data link layer entity. Each I-frame with a Standard Control field has a retransmission bit (R bit) that affects whether I-frames are retransmitted. Each I-frame with an Enhanced Control Field or an Extended Control Field has an F-bit that is used in Poll/Final bit functions.

The SAR field in the I-frame is used for segmentation and reassembly control. The L2CAP SDU Length field specifies the length of an SDU, including the aggregate length across all segments if segmented.

- *Supervisory frame format (S-frame)*

S-frames are used to acknowledge I-frames and request retransmission of I-frames. Each S-frame has an ReqSeq sequence number which may acknowledge additional I-frames received by the data link layer entity. Each S-frame with a Standard Control Field has a retransmission bit (R bit) that affects whether I-frames are retransmitted. Each S-frame with an Enhanced Control field or an Extended Control Field has a Poll bit (P-bit) and a Final bit (F-bit) and does not have an R-bit.

Defined types of S-frames are RR (Receiver Ready), REJ (Reject), RNR (Receiver Not Ready) and SREJ (Selective Reject).

Frame type	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
I	SAR		ReqSeq						R	TxSeq						0
S	X	X	ReqSeq						R	X	X	X	S	0		1

X denotes reserved bits, which shall be set to 0.

Table 3.1: Standard Control Field formats

Frame type	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
I	SAR		ReqSeq									F	TxSeq				
S	X	X	ReqSeq									F	X	X	P	S	0

X denotes reserved bits, which shall be set to 0.

Table 3.2: Enhanced Control Field formats

Frame type	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
I	ReqSeq															
	TxSeq															
S	ReqSeq															
					X	X	X	X	X	X	P	S				

X denotes reserved bits, which shall be set to 0.

Table 3.3: Extended Control Field formats

- Send Sequence Number - TxSeq (6 bits or 14 bits)

The send sequence number is used to number each I-frame, to enable sequencing and retransmission.

- Receive Sequence Number - ReqSeq (6 bits or 14 bits)

The receive sequence number is used by the receiver side to acknowledge I-frames, and in the REJ and SREJ frames to request the retransmission of an I-frame with a specific send sequence number.

- Retransmission Disable Bit - R (1 bit)

The Retransmission Disable bit is used to implement Flow Control. The receiver sets the bit when its internal receive buffer is full, this happens when one or more I-frames have been received but the SDU reassembly function has not yet pulled all the frames received. When the sender receives a frame with the Retransmission Disable bit set it shall disable the RetransmissionTimer, this causes the sender to stop retransmitting I-frames.

R=0: Normal operation. Sender uses the RetransmissionTimer to control retransmission of I-frames. Sender does not use the MonitorTimer.

R=1: Receiver side requests sender to postpone retransmission of I-frames. Sender monitors signaling with the MonitorTimer. Sender does not use the RetransmissionTimer.

The functions of ReqSeq and R are independent.

- *Segmentation and Reassembly - SAR (2 bits)*

The SAR bits define whether an L2CAP SDU is segmented. For segmented SDUs, the SAR bits also define which part of an SDU is in this I-frame, thus allowing one L2CAP SDU to span several I-frames.

An I-frame with SAR="Start of L2CAP SDU" also contains a length indicator, specifying the number of information octets in the total L2CAP SDU. The encoding of the SAR bits is shown in [Table 3.4](#).

00b	Unsegmented L2CAP SDU
01b	Start of L2CAP SDU
10b	End of L2CAP SDU
11b	Continuation of L2CAP SDU

Table 3.4: SAR control element format

- *Supervisory function - S (2 bits)*

The S-bits mark the type of S-frame. There are four types defined: RR (Receiver Ready), REJ (Reject), RNR (Receiver Not Ready) and SREJ (Selective Reject). The encoding is shown in [Table 3.5](#).

00b	RR - Receiver Ready
01b	REJ - Reject
10b	RNR - Receiver Not Ready
11b	SREJ - Select Reject

Table 3.5: S control element format: type of S-frame

- *Poll - P (1 bit)*

The P-bit is set to 1 to solicit a response from the receiver. The receiver shall respond immediately with a frame with the F-bit set to 1.

- *Final - F (1 bit)*

The F-bit is set to 1 in response to an S-frame with the P bit set to 1.

3.3.3 L2CAP SDU Length Field (2 octets)

When an SDU spans more than one I-frame, the first I-frame in the sequence shall be identified by SAR=01b="Start of L2CAP SDU". The L2CAP SDU Length field shall specify the total number of octets in the SDU. The L2CAP SDU Length field shall be present in I-frames with SAR=01b (Start of L2CAP SDU) and shall not be used in any other I-frames. When the SDU is unsegmented (SAR=00b), the L2CAP SDU Length field is not needed and shall not be present.

3.3.4 Information Payload Field

The information payload field consists of an integral number of octets. The maximum number of octets in this field is the same as the negotiated value of the MPS configuration parameter. The maximum number of octets in this field is also limited by the range of the Basic L2CAP header length field. This ranges from 65533 octets for I-frames with a Standard or Enhanced Control Field, no SDU length field, and no FCS field to 65527 octets for I-frames with an Enhanced Control Field, an SDU length field, and FCS field. Thus, even if an MPS of 65533 has been negotiated, the range of the Basic L2CAP header length field will restrict the number of octets in this field. For example, when an Enhanced Control Field, an SDU length field, and FCS field are present the number of octets in this field is restricted to 65529.

3.3.5 Frame Check Sequence (2 octets)

The Frame Check Sequence (FCS) is 2 octets. The FCS is constructed using the generator polynomial $g(D) = D^{16} + D^{15} + D^2 + 1$ (see [Figure 3.4](#)). The 16 bit LFSR is initially loaded with the value 0x0000, as depicted in [Figure 3.5](#). The switch S is set in position 1 while data is shifted in, LSB first for each octet. After the last bit has entered the LFSR, the switch is set in position 2, and the register contents are transmitted from right to left (i.e. starting with position 15, then position 14, etc.). The FCS covers the Basic L2CAP header, Control, L2CAP-SDU length and Information payload fields, if present, as shown in [Figure 3.3 on page 47](#).

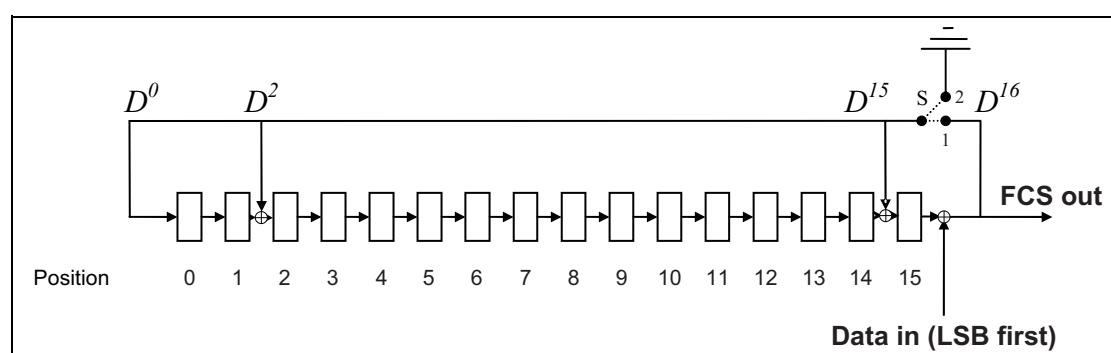


Figure 3.4: The LFSR circuit generating the FCS

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LFSR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.5: Initial state of the FCS generating circuit

Examples of FCS calculation, $g(D) = D^{16} + D^{15} + D^2 + 1$:

1. *I Frame*

Length = 14

Control = 0x0002 (SAR=00b, ReqSeq=000000b, R=0b, TxSeq=000001b)

Information Payload = 00 01 02 03 04 05 06 07 08 09 (10 octets, hexadecimal notation)

==> FCS = 0x6138

==> Data to Send = 0E 00 40 00 02 00 00 01 02 03 04 05 06 07 08 09 38 61
(hexadecimal notation)

2. *RR Frame*

Length = 4

Control = 0x0101 (ReqSeq=000001b, R=0b, S=00b)

==> FCS = 0x14D4

==> Data to Send = 04 00 40 00 01 01 D4 14 (hexadecimal notation)

3.3.6 Invalid Frame Detection

For Retransmission mode and Flow Control mode, a received PDU shall be regarded as invalid, if one of the following conditions occurs:

1. Contains an unknown CID.
2. Contains an FCS error.
3. Contains a length greater than the maximum PDU payload size (MPS).
4. I-frame that has fewer than 8 octets.
5. I-frame with SAR=01b (Start of L2CAP SDU) that has fewer than 10 octets.
6. I-frame with SAR bits that do not correspond to a normal sequence of either unsegmented or start, continuation, end for the given CID.
7. S-frame where the length field is not equal to 4.

These error conditions may be used for error reporting.

3.3.7 Invalid Frame Detection Algorithm

For Enhanced Retransmission mode and Streaming mode the following algorithm shall be used for received PDUs. It may be used for Retransmission mode and Flow Control mode:

1. Check the CID. If the PDU contains an unknown CID then it shall be ignored.
2. Check the FCS. If the PDU contains an FCS error then it shall be ignored. If the channel is configured to use "No FCS" then the PDU is considered to have a good FCS (no FCS error).

3. Check the following conditions. If one of the conditions occurs the channel shall be closed or in the case of fixed channels the ACL shall be disconnected.
 - a) PDU contains a length greater than the maximum PDU payload size (MPS)
 - b) I-frame that has fewer than the required number of octets. If the channel is configured to use a Standard or Enhanced Control Field then the required number of octets is 6 if "No FCS" is configured; otherwise, it is 8. If the channel is configured to use the Extended Control Field then the required number of octets is 8 if "No FCS" is configured; otherwise, it is 10.
 - c) S-frame where the length field is invalid. If the channel is configured to use a Standard or Enhanced Control Field then the length field shall be 2 if "No FCS" is configured; otherwise, the length field shall be 4. If the channel is configured to use the Extended Control Field then the length field shall be 4 if "No FCS" is configured; otherwise, the length field shall be 6.
4. Check the SAR bits. The SAR check is performed after the frame has been successfully received in the correct sequence. The PDU is invalid if one of the following conditions occurs:
 - a) I-frame with SAR=01b (Start of L2CAP SDU) that has fewer than the required number of octets. If the channel is configured to use a Standard or Enhanced Control field then the required number of octets is 8 if "No FCS" is configured; otherwise, the required number of octets is 10. If the channel is configured to use an Extended Control field then the required number of octets is 10 if "No FCS" is configured; otherwise, the required number of octets is 12.
 - b) I-frame with SAR bits that do not correspond to a normal sequence of either unsegmented or start, continuation, end for the given CID.
 - c) I-frame with SAR= 01b (Start of L2CAP SDU) where the value in the L2CAP SDU length field exceeds the configured MTU.
5. If the I-frame has been received in the correct sequence and is invalid as described in 4 then the channel shall be closed or in the case of fixed channels the ACL shall be disconnected. For Streaming mode and Flow Control mode if one or more I-frames are missing from a sequence of I-frames using SAR bits of start, continuation and end then received I-frames in the sequence may be ignored. For Flow Control mode and Streaming mode I-frames received out of sequence with SAR bits of unsegmented may be accepted.

If the algorithm is used for Retransmission mode or Flow control mode then it shall be used instead of Invalid Frame detection described in [Section 3.3.6](#).

These error conditions may be used for error reporting.

3.4 CONNECTION-ORIENTED CHANNELS IN LE CREDIT BASED FLOW CONTROL MODE

To support LE Credit Based Flow Control Mode, L2CAP PDU type with protocol elements in addition to the Basic L2CAP header are defined. In LE Credit Based Flow Control Mode, the L2CAP PDU on a connection-oriented channel is an LE information frame (LE-frame).

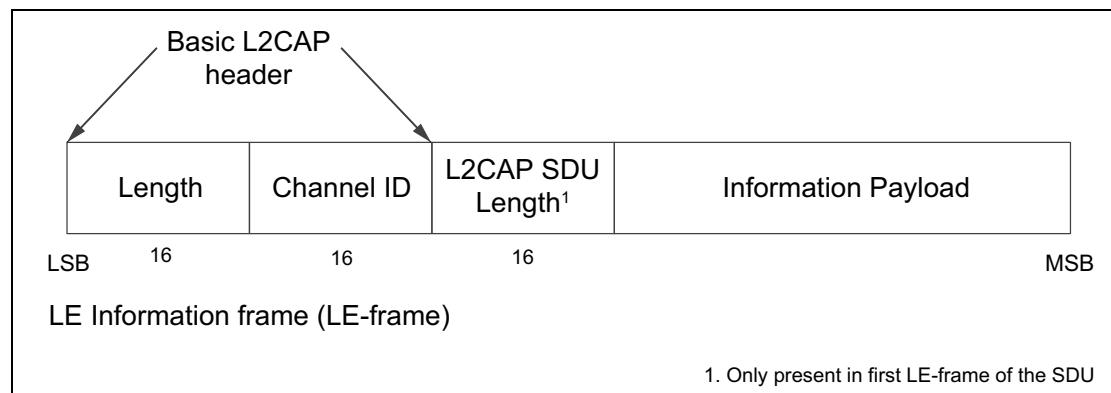


Figure 3.6: L2CAP PDU format in LE Credit Based Flow Control Mode

3.4.1 L2CAP Header Fields

- *Length: 2 octets*

The first two octets in the L2CAP PDU contain the length of the entire L2CAP PDU in octets, excluding the Length and CID fields.

For LE-frames, the Length field includes the octet lengths of the L2CAP SDU Length when present and Information Payload fields.

- *Channel ID: 2 octets*

The channel ID (CID) identifies the destination channel endpoint of the packet.

3.4.2 L2CAP SDU Length Field (2 octets)

The first LE-frame of the SDU shall contain the L2CAP SDU Length field that shall specify the total number of octets in the SDU. All subsequent LE-frames that are part of the same SDU shall not contain the L2CAP SDU Length field.

3.4.3 Information Payload Field

The information payload field consists of an integral number of octets. The maximum number of octets in this field is the same as the peer's MPS. The maximum number of octets in this field is also limited by the range of the Basic L2CAP header length field, of 65533 octets.

The number of octets contained in the first LE-frame information payload of the SDU is equal to the L2CAP header length field minus 2 octets. All subsequent

LE-frames of the same SDU contain the number of octets in the information payload equal to the L2CAP header length field.

If the SDU length field value exceeds the receiver's MTU, the receiver shall disconnect the channel. If the payload length of any LE-frame exceeds the receiver's MPS, the receiver shall disconnect the channel. If the sum of the payload lengths for the LE-frames exceeds the specified SDU length, the receiver shall disconnect the channel.

4 SIGNALING PACKET FORMATS

This section describes the signaling commands passed between two L2CAP entities on peer devices. All signaling commands are sent over a signaling channel. The signaling channel for managing channels over ACL-U logical links shall use CID 0x0001 and the signaling channel for managing channels over LE-U logical links shall use CID 0x0005. Signaling channels are available as soon as the lower layer logical transport is set up and L2CAP traffic is enabled. [Figure 4.1 on page 57](#) illustrates the general format of L2CAP PDUs containing signaling commands (C-frames). Multiple commands may be sent in a single C-frame over Fixed Channel CID 0x0001 while only one command per C-frame shall be sent over Fixed Channel CID 0x0005. Commands take the form of Requests and Responses. All L2CAP implementations shall support the reception of C-frames with a payload length that does not exceed the signaling MTU. The minimum supported payload length for the C-frame (MTU_{sig}) is defined in [Table 4.1 on page 57](#). L2CAP implementations should not use C-frames that exceed the MTU_{sig} of the peer device. If a device receives a C-frame that exceeds its MTU_{sig} then it shall send a Command Reject containing the supported MTU_{sig} . Implementations shall be able to handle the reception of multiple commands in an L2CAP packet sent over Fixed Channel CID 0x0001.

Logical Link	Minimum Supported Payload Length for the C-frame (MTU_{sig})
ACL-U not supporting Extended Flow Specification	48 octets
ACL-U supporting the Extended Flow Specification feature	672 octets
LE-U	23 octets

Table 4.1: Minimum Signaling MTU

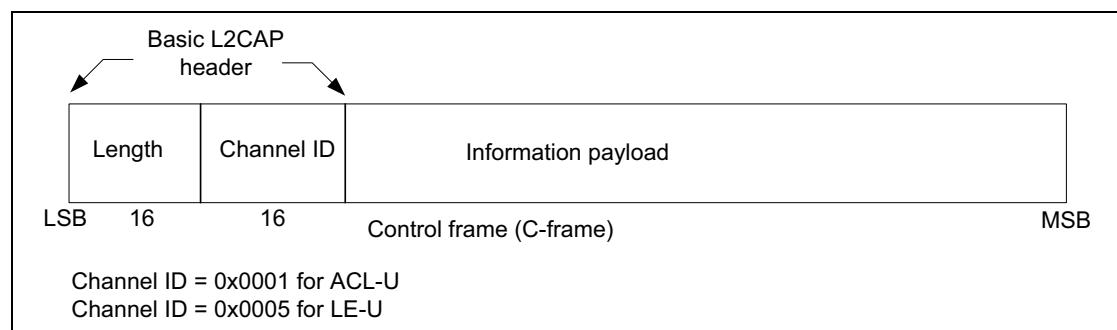


Figure 4.1: L2CAP PDU format on a signaling channel

[Figure 4.2](#) displays the general format of all signaling commands.

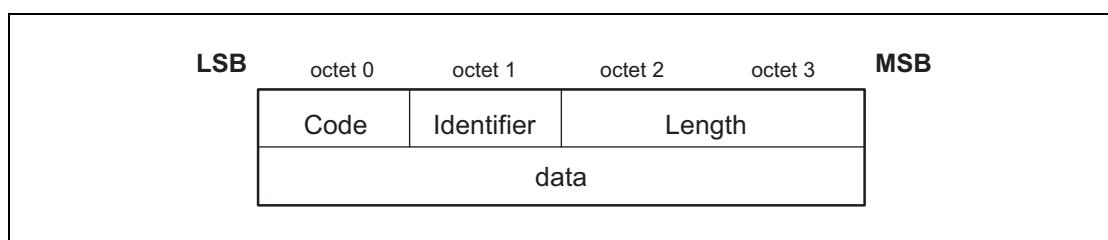


Figure 4.2: Command format

The fields shown are:

- *Code (1 octet)*

The Code field is one octet long and identifies the type of command. When a packet is received with a Code field that is unknown or disallowed on the signaling channel it is received on, a Command Reject packet (defined in [Section 4.1 on page 60](#)) is sent in response.

[Table 4.2 on page 58](#) lists the codes defined by this document. All codes are specified with the most significant bit in the left-most position.

Code	Description	CIDs on which Code is Allowed
0x00	Reserved	Any
0x01	Command reject	0x0001 and 0x0005
0x02	Connection request	0x0001
0x03	Connection response	0x0001
0x04	Configure request	0x0001
0x05	Configure response	0x0001
0x06	Disconnection request	0x0001 and 0x0005
0x07	Disconnection response	0x0001 and 0x0005
0x08	Echo request	0x0001
0x09	Echo response	0x0001
0x0A	Information request	0x0001
0x0B	Information response	0x0001
0x0C	Create Channel request	0x0001
0x0D	Create Channel response	0x0001
0x0E	Move Channel request	0x0001
0x0F	Move Channel response	0x0001
0x10	Move Channel Confirmation	0x0001
0x11	Move Channel Confirmation response	0x0001

Table 4.2: Signaling Command Codes

Code	Description	CIDs on which Code is Allowed
0x12	Connection Parameter Update request	0x0005
0x13	Connection Parameter Update response	0x0005
0x14	LE Credit Based Connection request	0x0005
0x15	LE Credit Based Connection response	0x0005
0x16	LE Flow Control Credit	0x0005
0x17 - 0xFF	Reserved	Any

Table 4.2: Signaling Command Codes (Continued)

- *Identifier (1 octet)*

The Identifier field is one octet long and matches responses with requests. The requesting device sets this field and the responding device uses the same value in its response. Within each signalling channel a different Identifier shall be used for each successive command. Following the original transmission of an Identifier in a command, the Identifier may be recycled if all other Identifiers have subsequently been used.

RTX and ERTX timers are used to determine when the remote end point is not responding to signaling requests. On the expiration of a RTX or ERTX timer, the same identifier shall be used if a duplicate Request is re-sent as stated in [Section 6.1.7 on page 122](#).

A device receiving a duplicate request on a particular signalling channel should reply with a duplicate response on the same signalling channel. A command response with an invalid identifier is silently discarded. Signaling identifier 0x00 is an illegal identifier and shall never be used in any command.

- *Length (2 octets)*

The Length field is two octets long and indicates the size in octets of the data field of the command only, i.e., it does not cover the Code, Identifier, and Length fields.

- *Data (0 or more octets)*

The Data field is variable in length. The Code field determines the format of the Data field. The length field determines the length of the data field.

4.1 COMMAND REJECT (CODE 0x01)

A Command Reject packet shall be sent in response to a command packet with an unknown command code or when sending the corresponding response is inappropriate. [Figure 4.3 on page 60](#) displays the format of the packet. The identifier shall match the identifier of the command packet being rejected. Implementations shall always send these packets in response to unidentified signaling packets. Command Reject packets should not be sent in response to an identified Response packet.

When multiple commands are included in an L2CAP packet and the packet exceeds the signaling MTU (MTU_{sig}) of the receiver, a single Command Reject packet shall be sent in response. The identifier shall match the first Request command in the L2CAP packet. If only Responses are recognized, the packet shall be silently discarded.

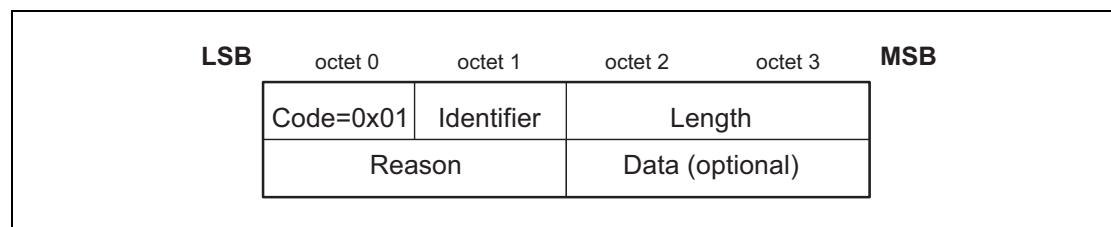


Figure 4.3: Command Reject Packet

[Figure 4.3](#) shows the format of the Command Reject packet. The data fields are:

- *Reason (2 octets)*

The Reason field describes why the Request packet was rejected, and is set to one of the Reason codes in [Table 4.3](#).

Reason value	Description
0x0000	Command not understood
0x0001	Signaling MTU exceeded
0x0002	Invalid CID in request
Other	Reserved

Table 4.3: Reason Code Descriptions

- *Data (0 or more octets)*

The length and content of the Data field depends on the Reason code. If the Reason code is 0x0000, “Command not understood”, no Data field is used. If the Reason code is 0x0001, “Signaling MTU Exceeded”, the 2-octet Data field represents the maximum signaling MTU the sender of this packet can accept.

If a command refers to an invalid channel then the Reason code 0x0002 will be returned. Typically a channel is invalid because it does not exist. The data

field shall be 4 octets containing the local (first) and remote (second) channel endpoints (relative to the sender of the Command Reject) of the disputed channel. The remote endpoint is the source CID from the rejected command. The local endpoint is the destination CID from the rejected command. If the rejected command contains only one of the channel endpoints, the other one shall be replaced by the null CID 0x0000.

Reason value	Data Length	Data value
0x0000	0 octets	N/A
0x0001	2 octets	Actual MTU _{sig}
0x0002	4 octets	Requested CID

Table 4.4: Reason Data values

4.2 CONNECTION REQUEST (CODE 0x02)

Connection request packets are sent to create an L2CAP channel between two devices. The L2CAP channel shall be established before configuration begins. [Figure 4.4](#) illustrates a Connection Request packet.

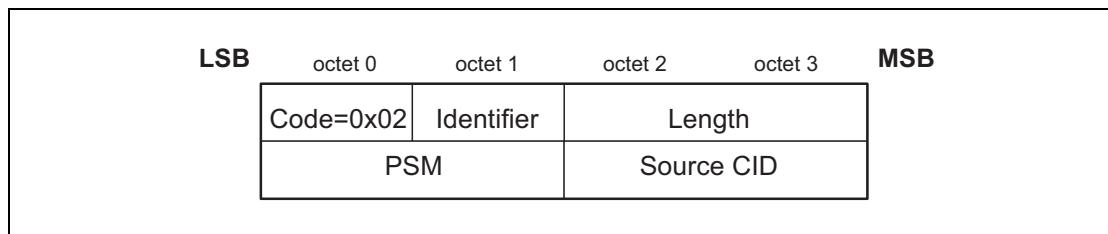


Figure 4.4: Connection Request Packet

The data fields are:

- *Protocol/Service Multiplexer - PSM (2 octets (minimum))*

The PSM field is at least two octets in length. The structure of the PSM field is based on the ISO 3309 extension mechanism for address fields. All PSM values shall be ODD, that is, the least significant bit of the least significant octet must be 1. Also, all PSM values shall have the least significant bit of the most significant octet equal to 0. This allows the PSM field to be extended beyond 16 bits. PSM values are separated into two ranges. Valid values in the first range are assigned by the Bluetooth SIG and indicate protocols. The second range of values are dynamically allocated and used in conjunction with the Service Discovery Protocol (SDP). The dynamically assigned values may be used to support multiple implementations of a particular protocol.

PSM values are defined in the [Assigned Numbers](#) document.

Range	Type	Server Usage	Client Usage
0x0001-0x0EFF (Note ¹)	Fixed, SIG assigned	PSM is fixed for all implementations.	PS may be obtained via SDP or may be assumed for a fixed service. Protocol used is indicated by the PSM as defined in the Assigned Numbers page.
>0x1000	Dynamic	PSM may be fixed for a given implementation or may be assigned at the time the service is registered in SDP.	PSM shall be obtained via SDP upon every reconnection. PSM for one direction will typically be different from the other direction.

Table 4.5: PSM ranges and usage

1. PSMs shall be odd and the least significant bit of the most significant byte shall be zero, hence the following ranges do not contain valid PSMS: 0x0100-0x01FF, 0x0300-0x03FF, 0x0500-0x05FF, 0x0700-0x07FF, 0x0900-0x09FF, 0x0B00-0x0BFF, 0x0D00-0x0DFF, 0x0F00-0x0FFF. All even values are also not valid as PSMs.
- **Source CID - SCID (2 octets)**
The source CID is two octets in length and represents a channel endpoint on the device sending the request. Once the channel has been configured, data packets flowing to the sender of the request shall be sent to this CID. Thus, the Source CID represents the channel endpoint on the device sending the request and receiving the response. The value of the Source CID shall be from the dynamically allocated range as defined in [Table 2.1 on page 38](#) and shall not be already allocated to a different channel on the device sending the request.

4.3 CONNECTION RESPONSE (CODE 0x03)

When a device receives a Connection Request packet, it shall send a Connection Response packet. Note: implementations conforming to previous versions of this specification may respond with a Command Reject (Reason 0x0002 – Invalid CID In Request) packet under conditions now covered by result codes of 0x0006 and 0x0007.

The format of the connection response packet is shown in [Figure 4.5](#).

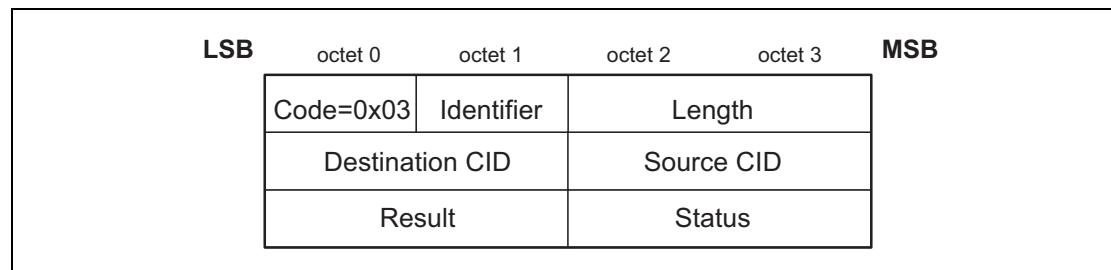


Figure 4.5: Connection Response Packet

The data fields are:

- *Destination Channel Identifier - DCID (2 octets)*

This field contains the channel endpoint on the device sending this Response packet. Thus, the Destination CID represents the channel endpoint on the device receiving the request and sending the response. The value of the Destination CID shall be from the dynamically allocated range as defined in [Table 2.1 on page 38](#) and shall not be already allocated to a different channel on the device sending the response.

- *Source Channel Identifier - SCID (2 octets)*

This field contains the channel endpoint on the device receiving this Response packet. This is copied from the SCID field of the connection request packet.

- *Result (2 octets)*

The result field indicates the outcome of the connection request. The result value of 0x0000 indicates success while a non-zero value indicates the connection request failed or is pending. A logical channel is established on the receipt of a successful result unless the DCID field is outside of the dynamically allocated range (see [Table 2.1 on page 38](#)) or is already allocated on the device sending the response. [Table 4.6 on page 63](#) defines values for this field. The DCID and SCID fields shall be ignored when the result field indicates the connection was refused.

Value	Description
0x0000	Connection successful.
0x0001	Connection pending

Table 4.6: Result values

Value	Description
0x0002	Connection refused – PSM not supported.
0x0003	Connection refused – security block.
0x0004	Connection refused – no resources available.
0x0005	Reserved
0x0006	Connection refused – Invalid Source CID
0x0007	Connection refused – Source CID already allocated
Other	Reserved.

Table 4.6: Result values (Continued)

- **Status (2 octets)**

Only defined for Result = Pending. Indicates the status of the connection. The status is set to one of the values shown in [Table 4.7 on page 64](#).

Value	Description
0x0000	No further information available
0x0001	Authentication pending
0x0002	Authorization pending
Other	Reserved

Table 4.7: Status values

4.4 CONFIGURATION REQUEST (CODE 0x04)

Configuration Request packets are sent to establish an initial logical link transmission contract between two L2CAP entities and also to re-negotiate this contract whenever appropriate. The contract consists of a set of configuration parameter options defined in [Section 5 on page 89](#). All parameter options have default values and can have previously agreed values which are values that were accepted in a previous configuration process or in a previous step in the current configuration process. The only parameters that should be included in the Configuration Request packet are those that require different values than the default or previously agreed values.

If no parameters need to be negotiated or specified then no options shall be inserted and the continuation flag (C) shall be set to zero. Any missing configuration parameters are assumed to have their most recently explicitly or implicitly accepted values. Even if all default values are acceptable, a Configuration Request packet with no options shall be sent. Implicitly accepted values are default values for the configuration parameters that have not been explicitly negotiated for the specific channel under configuration.

See [Section 7.1 on page 131](#) for details of the configuration procedure.

[Figure 4.6](#) defines the format of the Configuration Request packet.

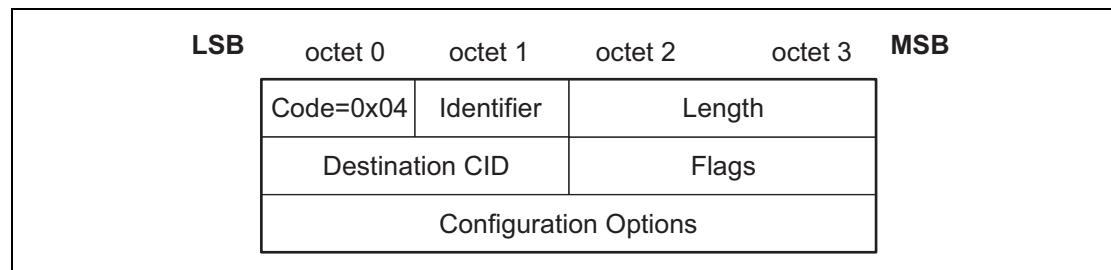


Figure 4.6: Configuration Request Packet

The data fields are:

- *Destination CID - DCID (2 octets)*

This field contains the channel endpoint on the device receiving this Request packet.

- *Flags (2 octets)*

[Figure 4.7](#) shows the two-octet Flags field. Note the most significant bit is shown on the left.

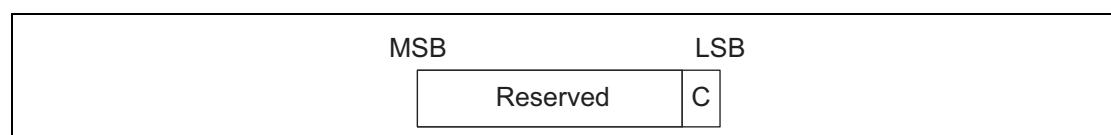


Figure 4.7: Configuration Request Flags field format

Only one flag is defined, the Continuation flag (C).

When both L2CAP entities support the Extended Flow Specification option, the Continuation flag shall not be used and shall be set to zero in all Configuration Request and Response packets.

When all configuration options cannot fit into a Configuration Request with length that does not exceed the receiver's MTU_{sig}, the options shall be passed in multiple configuration command packets. If all options fit into the receiver's MTU_{sig}, then they shall be sent in a single configuration request with the continuation flag set to zero. Each Configuration Request shall contain an integral number of options - partially formed options shall not be sent in a packet. Each Request shall be tagged with a different Identifier and shall be matched with a Response with the same Identifier.

When used in the Configuration Request, the continuation flag indicates the responder should expect to receive multiple request packets. The responder shall reply to each Configuration Request packet. The responder may reply to each Configuration Request with a Configuration Response containing the same option(s) present in the Request (except for those error conditions more appropriate for a Command Reject), or the responder may reply with a "Success" Configuration Response packet containing no options, delaying those options until the full Request has been received. The Configuration Request packet with the continuation flag cleared shall be treated as the Configuration Request event in the channel state machine.

When used in the Configuration Response, the continuation flag shall be set to one if the flag is set to one in the Request. If the continuation flag is set to one in the Response when the matching Request has the flag set to zero, it indicates the responder has additional options to send to the requestor. In this situation, the requestor shall send null-option Configuration Requests (with continuation flag set to zero) to the responder until the responder replies with a Configuration Response where the continuation flag is set to zero. The Configuration Response packet with the continuation flag set to zero shall be treated as the Configuration Response event in the channel state machine.

The result of the configuration transaction is the union of all the result values. All the result values must succeed for the configuration transaction to succeed.

Other flags are reserved and shall be set to zero. L2CAP implementations shall ignore these bits.

- *Configuration Options*

A list of the parameters and their values to be negotiated shall be provided in the Configuration Options field. These are defined in [Section 5 on page 89](#). A Configuration Request may contain no options (referred to as an empty or null configuration request) and can be used to request a response. For an empty configuration request the length field is set to 0x0004.

4.5 CONFIGURATION RESPONSE (CODE 0x05)

Configuration Response packets shall be sent in reply to Configuration Request packets except when the error condition is covered by a Command Reject response. Each configuration parameter value (if any is present) in a Configuration Response reflects an 'adjustment' to a configuration parameter value that has been sent (or, in case of default values, implied) in the corresponding Configuration Request. For example, if a configuration request relates to traffic flowing from device A to device B, the sender of the configuration response may adjust this value for the same traffic flowing from device A to device B, but the response cannot adjust the value in the reverse direction.

The options sent in the Response depend on the value in the Result field. [Figure 4.8 on page 67](#) defines the format of the Configuration Response packet. See also [Section 7.1 on page 131](#) for details of the configuration process.

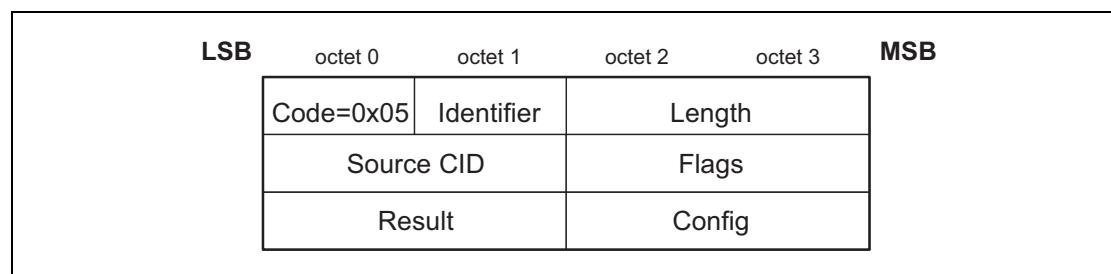


Figure 4.8: Configuration Response Packet

The data fields are:

- *Source CID - SCID (2 octets)*

This field contains the channel endpoint on the device receiving this Response packet. The device receiving the Response shall check that the Identifier field matches the same field in the corresponding configuration request command and the SCID matches its local CID paired with the original DCID.

- *Flags (2 octets)*

[Figure 4.9](#) displays the two-octet Flags field. Note the most significant bit is shown on the left.

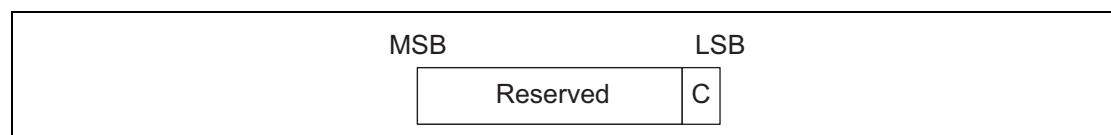


Figure 4.9: Configuration Response Flags field format

Only one flag is defined, the Continuation flag (C).

When both L2CAP entities support the Extended Flow Specification option, the Continuation flag shall not be used and shall be set to zero in all Configuration Request and Response packets.

More configuration responses will follow when C is set to one. This flag indicates that the parameters included in the response are a partial subset of parameters being sent by the device sending the Response packet.

The other flag bits are reserved and shall be set to zero. L2CAP implementations shall ignore these bits.

- *Result (2 octets)*

The Result field indicates whether or not the Request was acceptable. See [Table 4.8 on page 68](#) for possible result codes.

Result	Description
0x0000	Success
0x0001	Failure – unacceptable parameters
0x0002	Failure – rejected (no reason provided)
0x0003	Failure – unknown options
0x0004	Pending
0x0005	Failure - flow spec rejected
Other	Reserved

Table 4.8: Configuration Response Result codes

- *Configuration Options*

This field contains the list of parameters being configured. These are defined in [Section 5 on page 89](#). On a successful result (Result=0x0000) and pending result (Result=0x0004), these parameters contain the return values for any wild card parameter values (see [Section 5.3 on page 92](#)) and “adjustments” to non-negotiated configuration parameter values contained in the request. A response with the result code of 0x0000 (Success) is also referred to as a positive response.

On an unacceptable parameters failure (Result=0x0001) the rejected parameters shall be sent in the response with the values that would have been accepted if sent in the original request. Any missing configuration parameters in the Configuration Request are assumed to have their default value or previously agreed value and they too shall be included in the Configuration Response if they need to be changed. A response with the result code of 0x0001 is also referred to as a negative response.

On an unknown option failure (Result=0x0003), the option(s) that contain an option type field that is not understood by the recipient of the Request shall be included in the Response unless they are hints. Hints are those options in the Request that are skipped if not understood (see [Section 5 on page 89](#)). Hints shall not be included in the Response and shall not be the sole cause for rejecting the Request.

On a flow spec rejected failure (Result=0x0005), an Extended Flow Spec option may be included to reflect the QoS level that would be acceptable (see [Section 7.1.3 on page 133](#)).

The decision on the amount of time (or messages) spent arbitrating the channel parameters before terminating the negotiation is implementation specific.

4.6 DISCONNECTION REQUEST (CODE 0x06)

Terminating an L2CAP channel requires that a disconnection request be sent and acknowledged by a disconnection response. [Figure 4.10 on page 69](#) shows a disconnection request. The receiver shall ensure that both source and destination CIDs match before initiating a channel disconnection.

Once a Disconnection Request is issued, all incoming data in transit on this L2CAP channel shall be discarded and any new additional outgoing data shall be discarded. Once a disconnection request for a channel has been received, all data queued to be sent out on that channel shall be discarded.

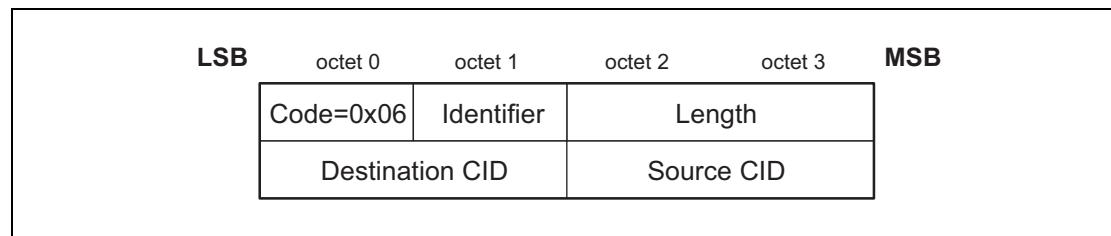


Figure 4.10: Disconnection Request Packet

The data fields are:

- *Destination CID - DCID (2 octets)*

This field specifies the endpoint of the channel to be disconnected on the device receiving this request.

- *Source CID - SCID (2 octets)*

This field specifies the endpoint of the channel to be disconnected on the device sending this request.

The SCID and DCID are relative to the sender of this request and shall match those of the channel to be disconnected. If the DCID is not recognized by the receiver of this message, a CommandReject message with 'invalid CID' result code shall be sent in response. If the receiver finds a DCID match but the SCID fails to find the same match, the request should be silently discarded.

4.7 DISCONNECTION RESPONSE (CODE 0x07)

Disconnection responses shall be sent in response to each valid disconnection request.

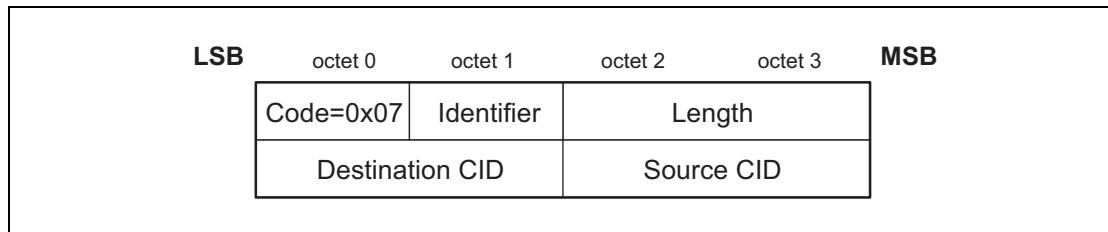


Figure 4.11: Disconnection Response Packet

The data fields are:

- *Destination CID - DCID (2 octets)*

This field identifies the channel endpoint on the device sending the response.

- *Source CID - SCID (2 octets)*

This field identifies the channel endpoint on the device receiving the response.

The DCID and the SCID (which are relative to the sender of the request), and the Identifier fields shall match those of the corresponding disconnection request command. If the CIDs do not match, the response should be silently discarded at the receiver.

4.8 ECHO REQUEST (CODE 0x08)

Echo requests are used to request a response from a remote L2CAP entity. These requests may be used for testing the link or for passing vendor specific information using the optional data field. L2CAP entities shall respond to a valid Echo Request packet with an Echo Response packet. The Data field is optional and implementation specific. L2CAP entities should ignore the contents of this field if present.

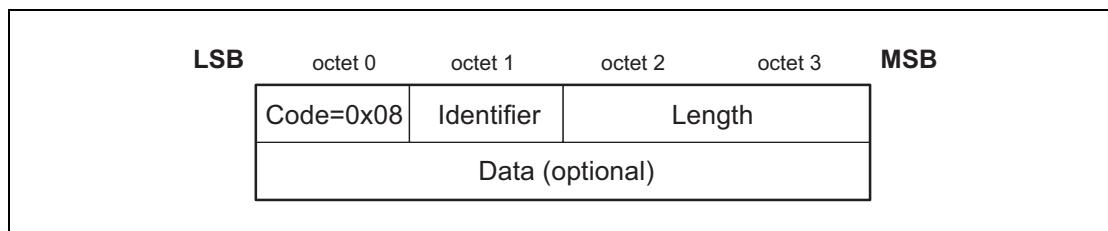


Figure 4.12: Echo Request Packet

4.9 ECHO RESPONSE (CODE 0x09)

An Echo response shall be sent upon receiving a valid Echo Request. The identifier in the response shall match the identifier sent in the Request. The optional and implementation specific data field may contain the contents of the data field in the Request, different data, or no data at all.

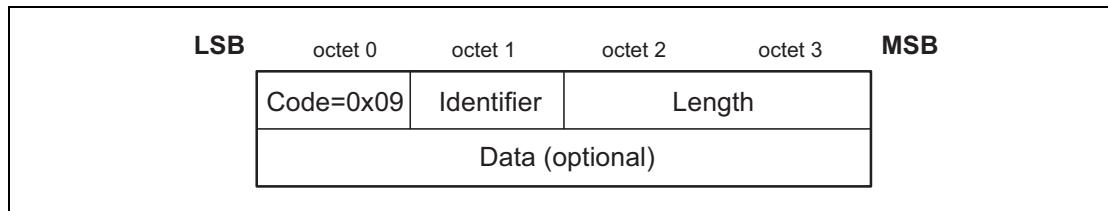


Figure 4.13: Echo Response Packet

4.10 INFORMATION REQUEST (CODE 0x0A)

Information requests are used to request implementation specific information from a remote L2CAP entity. L2CAP implementations shall respond to a valid Information Request with an Information Response. It is optional to send Information Requests.

An L2CAP implementation shall only use optional features or attribute ranges for which the remote L2CAP entity has indicated support through an Information Response. Until an Information Response which indicates support for optional features or ranges has been received only mandatory features and ranges shall be used.

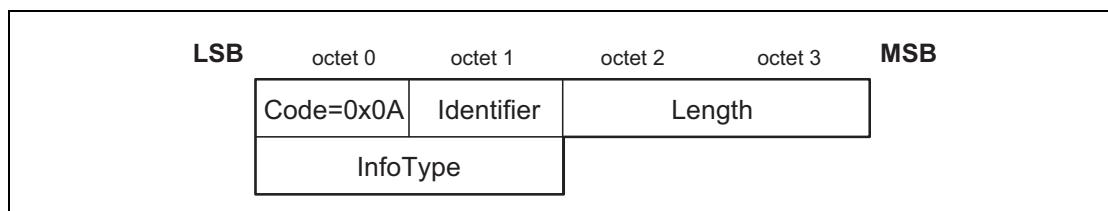


Figure 4.14: Information Request Packet

The data field is:

- *InfoType* (2 octets)

The *InfoType* defines the type of implementation specific information being requested. See [Section 4.11 on page 72](#) for details on the type of information requested.

Value	Description
0x0001	Connectionless MTU
0x0002	Extended features supported

Table 4.9: *InfoType* definitions

Value	Description
0x0003	Fixed Channels supported
Other	Reserved

Table 4.9: InfoType definitions (Continued)

L2CAP entities shall not send an Information Request packet with InfoType 0x0003 over Fixed Channel CID 0x0001 until first verifying that the Fixed Channels bit is set in the Extended feature mask of the remote device. Support for fixed channels is mandatory for devices with BR/EDR/LE or LE Controllers. Information Request and Information Response shall not be used over Fixed Channel CID 0x0005.

4.11 INFORMATION RESPONSE (CODE 0x0B)

An information response shall be sent upon receiving a valid Information Request. The identifier in the response shall match the identifier sent in the Request. The data field shall contain the value associated with the InfoType field sent in the Request, or shall be empty if the InfoType is not supported.

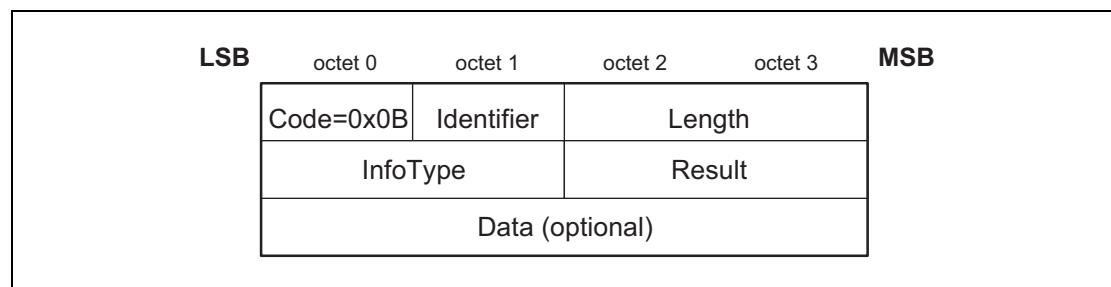


Figure 4.15: Information Response Packet

The data fields are:

- *InfoType* (2 octets)

The InfoType defines the type of implementation specific information that was requested. This value shall be copied from the InfoType field in the Information Request.

- *Result* (2 octets)

The Result contains information about the success of the request. If result is "Success," the data field contains the information as specified in [Table 4.11 on page 73](#). If result is "Not supported," no data shall be returned.

Value	Description
0x0000	Success
0x0001	Not supported

Table 4.10: Information Response Result values

Value	Description
Other	Reserved

Table 4.10: Information Response Result values (Continued)

- **Data (0 or more octets)**

The contents of the Data field depends on the InfoType.

For InfoType = 0x0001 the data field contains the remote entity's 2-octet acceptable connectionless MTU. The default value is defined in [Section 3.2 on page 46](#).

For InfoType = 0x0002, the data field contains the 4 octet L2CAP extended feature mask. The feature mask refers to the extended features that the L2CAP entity sending the Information Response supports. The feature bits contained in the L2CAP feature mask are specified in [Section 4.12 on page 74](#).

For InfoType = 0x0003, the data field contains an 8 octet bit map that indicates which Fixed L2CAP Channels (i.e., the L2CAP channels that use a CID from 0x0000 to 0x003F) are supported. A list of available fixed channels is provided in [Table 2.1 in Section 2.1](#). A detailed description of this InfoType data field is given in [Section 4.13 on page 75](#).

Note: L2CAP entities of versions prior to version 1.2, receiving an Information Request with InfoType = 0x0002 for an L2CAP feature discovery, return an Information Response with result code "Not supported." L2CAP entities at version 1.2 to 2.1 + EDR that have an all zero extended features mask may return an Information Response with result code "Not supported."

InfoType	Data	Data Length (octets)
0x0001	Connectionless MTU	2
0x0002	Extended feature mask	4
0x0003	Fixed Channels Supported	8

Table 4.11: Information Response Data fields

4.12 EXTENDED FEATURE MASK

The features are represented as a bit mask in the Information Response data field (see [Section 4.11 on page 72](#)). For each feature a single bit is specified which shall be set to 1 if the feature is supported and set to 0 otherwise. All unknown, reserved, or unassigned feature bits shall be set to 0.

The feature mask shown in [Table 4.12 on page 74](#) consists of 4 octets (numbered octet 0 ... 3), with bit numbers 0 ... 7 each. Within the Information Response packet data field, bit 0 of octet 0 is aligned leftmost, bit 7 of octet 3 is aligned rightmost.

Note: the L2CAP feature mask was introduced in Bluetooth v1.2 and contains features introduced after Bluetooth v1.1.

No.	Supported feature	Octet	Bit
0	Flow control mode	0	0
1	Retransmission mode	0	1
2	Bi-directional QoS ¹	0	2
3	Enhanced Retransmission Mode	0	3
4	Streaming Mode	0	4
5	FCS Option	0	5
6	Extended Flow Specification for BR/EDR	0	6
7	Fixed Channels	0	7
8	Extended Window Size	1	0
9	Unicast Connectionless Data Reception	1	1
31	Reserved for feature mask extension	3	7

Table 4.12: Extended feature mask

1. Peer side supports upper layer control of the Link Manager's Bi-directional QoS, see [Section 5.3 on page 92](#) for more details.

4.13 FIXED CHANNELS SUPPORTED

Each Fixed Channel is represented by a single bit in an 8 octet bit mask. The bit associated with a channel shall be set to 1 if the L2CAP entity supports signaling on that channel. The L2CAP signaling channel is mandatory and therefore the bit associated with that channel shall be set to 1. [Table 4.13](#) shows the format of the bit mask.

CID	Fixed Channel	Value	Octet	Bit
0x0000	Null identifier	Shall be set to 0	0	0
0x0001	L2CAP Signaling channel	Shall be set to 1	0	1
0x0002	Connectionless reception	0 – if not supported 1 – if supported	0	2
0x0003	AMP Manager Protocol Channel	0 – if not supported 1 – if supported	0	3
0x0004- 0x0006	Reserved	Shall be set to 0 and ignored upon receipt	0	4-6
0x0007	BR/EDR Security Manager	0 – if not supported 1 – if supported	0	7
0x0008- 0x003E	Reserved	Shall be set to 0 and ignored upon receipt.	1-6 7	0-7 0-6
0x003F	AMP Test Manager	0 – if not supported 1 – if supported	7	7

Table 4.13: Fixed Channels Supported bit mask

An L2CAP entity shall not transmit on any fixed channel (with the exception of the L2CAP signaling channel) until it has received a Fixed Channels Supported InfoType from the peer L2CAP entity indicating support for the channel, or has received a valid signaling packet from the remote device on the Fixed channel. All packets received on a non-supported fixed channel shall be ignored.

4.14 CREATE CHANNEL REQUEST (CODE 0x0C)

The Create Channel request packet, shown in [Figure 4.16](#), is sent to create an L2CAP channel between two devices over the Controller identified by the Controller ID.

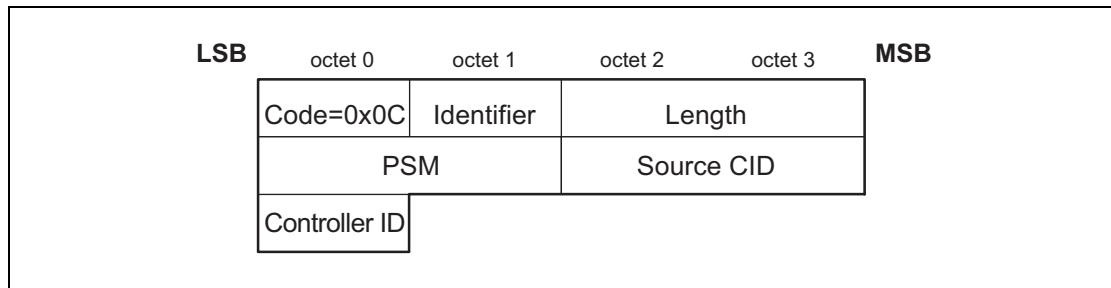


Figure 4.16: Create Channel Request Packet

The data fields are:

- *Protocol/Service Multiplexer –PSM (2 octets (minimum))*

The PSM field is at least two octets in length. The structure of the PSM field is based on the ISO 3309 extension mechanism for address fields. All PSM values shall be *odd*, that is, the least significant bit of the least significant octet must be 1. Also, all PSM values shall have the least significant bit of the most significant octet equal to 0. This allows the PSM field to be extended beyond 16 bits. PSM values are separated into two ranges. Values in the first range are assigned by the Bluetooth SIG and indicate protocols. The second range of values are dynamically allocated and used in conjunction with the Service Discovery Protocol (SDP). The dynamically assigned values may be used to support multiple implementations of a particular protocol. PSM values are defined in the [Assigned Numbers](#) document.

- *Source CID –SCID (2 octets)*

The source CID is two octets in length and represents a channel endpoint on the device sending the request. Once the channel has been configured, data packets flowing to the sender of the request shall be sent to this CID. Thus, the Source CID represents the channel endpoint on the device sending the request and receiving the response. The value of the Source CID shall be from the dynamically allocated range as defined in Table 2.1 on page [\[!XREF\]](#) and shall not be already allocated to a different channel on the device sending the request.

- *Controller Identifier –Controller ID (1 octet)*

The Controller ID is one octet in length and represents the Controller physical link over which the channel shall be created. The Controller ID is the identifier of the Controller on the remote device obtained via an AMP Manager Discover Available AMPs request. See [Part E, AMP Manager Protocol Specification on page 441](#) The Controller physical link shall already exist.

4.15 CREATE CHANNEL RESPONSE (CODE 0x0D)

When a device receives a Create Channel Request Packet, it shall send a Create Channel Response packet. Note: implementations conforming to versions prior to 4.2 may respond with a Command Reject (Reason 0x0002 – Invalid CID In Request) packet under conditions now covered by result codes of 0x0006 and 0x0007 of this packet.

The format of the Create Channel Response packet is shown in [Figure 4.17](#).

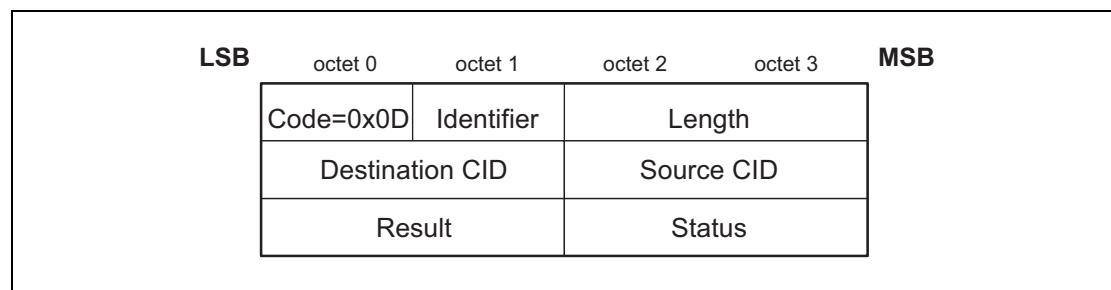


Figure 4.17: Create Channel Response Packet

The data fields are:

- *Destination Channel Identifier–DCID (2 octets)*

This field contains the channel endpoint on the device sending this Response packet. Thus, the Destination CID represents the channel endpoint on the device receiving the request and sending the response. The value of the Destination CID shall be from the dynamically allocated range as defined in [Table 2.1 on page 38](#) and shall not be already allocated to a different channel on the device sending the response.

- *Source Channel Identifier–SCID (2 octets)*

This field contains the channel endpoint on the device receiving this Response packet. This is copied from the SCID field of the Create Channel Request packet.

- *Result (2 octets)*

The result field indicates the outcome of the Create Channel Request. The result value of 0x0000 indicates success while a non-zero value indicates the Create Channel Request failed or is pending. A logical channel is established on the receipt of a successful result unless the DCID field is outside of the dynamically allocated range (see [Table 2.1 on page 38](#)) or is already allocated on the device sending the response. [Table 4.14](#) defines values for this field. The DCID and SCID fields shall be ignored when the result field indicates the connection was refused.

Result	Description
0x0000	Connection successful

Table 4.14: Result values

Result	Description
0x0001	Connection pending
0x0002	Connection refused - PSM not supported
0x0003	Connection refused - security block
0x0004	Connection refused - no resources available
0x0005	Connection refused - Controller ID not supported
0x0006	Connection refused – Invalid Source CID
0x0007	Connection refused – Source CID already allocated
Other	Reserved

Table 4.14: Result values (Continued)

- **Status (2 octets)**

Only defined for Result = Pending. Indicates the status of the connection. The status is set to one of the values shown in [Table 4.15](#).

Value	Description
0x0000	No further information available
0x0001	Authentication pending
0x0002	Authorization pending
Other	Reserved

Table 4.15: Status values

4.16 MOVE CHANNEL REQUEST (CODE 0x0E)

Move Channel request packets are sent to move an existing L2CAP channel from a physical link on one Controller to a physical link on another Controller. The CIDs for the L2CAP channel are not changed by a Move operation. [Figure 4.18](#) illustrates a Move Channel request packet. Channels shall only be moved if configured to use Enhanced Retransmission mode or Streaming mode. Fixed channels (see [Section 2.1](#)) shall not be moved.

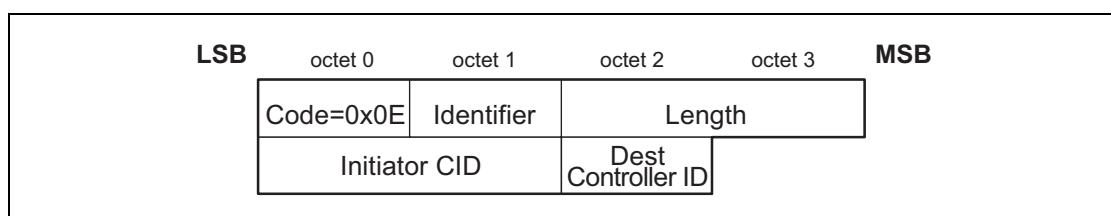


Figure 4.18: Move Channel Request Packet

The data fields are:

- *Initiator Channel Identifier - ICID (2 octets)*

This field contains the channel endpoint on the device sending this Request packet.

- *Destination Controller Identifier - Dest Controller ID (1 octet)*

The Dest Controller ID is one octet in length and represents the Controller physical link to which the channel shall be moved. The Controller ID is the identifier of the Controller on the remote device obtained via an AMP Manager Discover Available AMPs request. See [Part E, AMP Manager Protocol Specification on page 441](#). The Controller physical link shall already exist.

4.17 MOVE CHANNEL RESPONSE (CODE 0x0F)

When a device receives a Move Channel request packet it shall send a Move Channel response packet. If the device has sent its own Move Channel request then a collision has occurred. One of the requests shall be rejected while the other shall proceed. The Move Channel request that shall be rejected is based on the algorithm in [Section 7.7](#).

The format of Move Channel response packet is shown in [Figure 4.19](#).

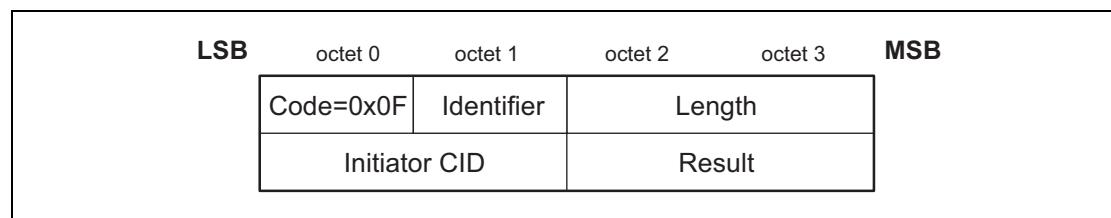


Figure 4.19: Move Channel Response Packet

The data fields are:

- *Initiator Channel Identifier – ICID (2 octets)*

This field contains the channel endpoint on the device that sent the Move Channel Request (device receiving this Response packet). This is the same value as was sent in the Move Channel Request packet.

- *Result (2 octets)*

The result field indicates the outcome of the move request. The result value of 0x0000 indicates success while a non-zero value indicates the move request failed. A logical channel is moved on the receipt of a successful result. [Table 4.16](#) defines values for this field. The ICID field shall be ignored when the result field indicates the move was refused.

Result	Description
0x0000	Move Success
0x0001	Move Pending
0x0002	Move refused - Controller ID not supported
0x0003	Move refused - new Controller ID is same as old Controller ID
0x0004	Move refused - Configuration not supported
0x0005	Move refused - Move Channel collision
0x0006	Move refused - Channel not allowed to be moved
Other	Reserved

Table 4.16: Result values

4.18 MOVE CHANNEL CONFIRMATION (CODE 0x10)

When the initiator of a Move Channel request receives a Move Channel response with a result code other than "pending" from the responder, it shall send a Move Channel confirmation packet, that will conclude the Move Channel procedure, informing the responder that all the QoS parameters were successfully (or not) negotiated. The format of Move Channel confirmation packet is shown in [Figure 4.20](#).

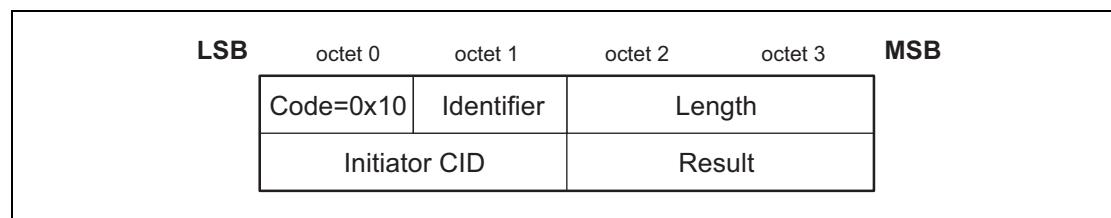


Figure 4.20: Move Channel Confirmation Packet

The data fields are:

- *Initiator Channel Identifier – ICID (2 octets)*

This field contains the channel endpoint on the device sending the Move Channel Confirmation packet. This is the same value as was sent in the original Move Channel Request packet.

- *Result (2 octets)*

The result field indicates the outcome of the move confirmation. The result value of 0x0000 indicates success while a non-zero value indicates the move failed.

Result	Description
0x0000	Move success - both sides succeed
0x0001	Move failure - one or both sides refuse
Other	Reserved

Table 4.17: Result values

4.19 MOVE CHANNEL CONFIRMATION RESPONSE (CODE 0x11)

When a device receives a Move Channel Confirmation packet it shall send a Move Channel Confirmation response packet. The purpose of the Move Channel Confirmation Response is so that Move Channel Confirmation is consistent with the command/response style. The format of Move Channel Confirmation response packet is shown in [Figure 4.21](#).

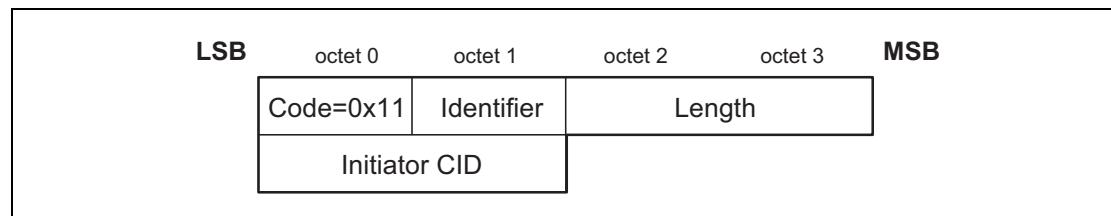


Figure 4.21: Move Channel Confirmation Response Packet

The data field is:

- *Initiator Channel Identifier – ICID (2 octets)*

This field contains the channel endpoint on the device that sent the Move Channel Confirmation packet.

4.20 CONNECTION PARAMETER UPDATE REQUEST (CODE 0x12)

This command shall only be sent from the LE slave device to the LE master device and only if one or more of the LE slave Controller, the LE master Controller, the LE slave Host and the LE master Host do not support the Connection Parameters Request Link Layer Control Procedure ([\[Vol. 6\] Part B, Section 5.1.7](#)). If an LE slave Host receives a Connection Parameter Update Request packet it shall respond with a Command Reject packet with reason 0x0000 (Command not understood).

The Connection Parameter Update Request allows the LE slave Host to request a set of new connection parameters. When the LE master Host receives a Connection Parameter Update Request packet, depending on the parameters of other connections, the LE master Host may accept the requested parameters and deliver the requested parameters to its Controller or reject the request. In devices supporting HCI, the LE master Host delivers the requested parameters to its Controller using the HCI_LE_Connection_Update command (see [\[Vol. 2\] Part E, Section 7.8.18](#)). If the LE master Host accepts the requested parameters it shall send the Connection Parameter Update Response packet with result 0x0000 (Parameters accepted) otherwise it shall set the result to 0x0001 (request rejected).

The LE slave Host will receive an indication from the LE slave Controller when the connection parameters have been updated. In devices supporting HCI, this

notification will be in the form of an LE Connection Update Complete event (see [Vol. 2] Part E, Section 7.7.65.1). If the LE master Controller rejects the updated connection parameters no indication from the LE slave Controller will be sent to the LE slave Host.

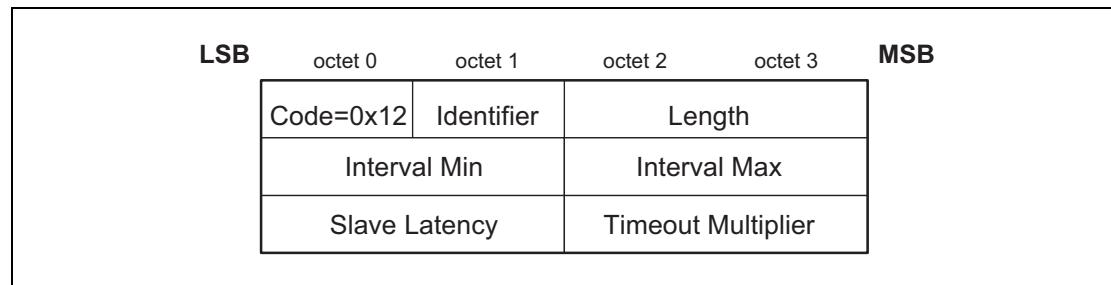


Figure 4.22: Connection Parameters Update Request Packet

The data fields are:

- *Interval Min (2 octets)*

Defines minimum value for the connection event interval in the following manner:

$\text{connIntervalMin} = \text{Interval Min} * 1.25 \text{ ms}$. Interval Min range: 6 to 3200 frames where 1 frame is 1.25 ms and equivalent to 2 BR/EDR slots. Values outside the range are reserved. Interval Min shall be less than or equal to Interval Max.

- *Interval Max (2 octets)*

Defines maximum value for the connection event interval in the following manner:

$\text{connIntervalMax} = \text{Interval Max} * 1.25 \text{ ms}$. Interval Max range: 6 to 3200 frames. Values outside the range are reserved. Interval Max shall be equal to or greater than the Interval Min.

- *Slave Latency (2 octets)*

Defines the slave latency parameter (as number of LL connection events) in the following manner:

$\text{connSlaveLatency} = \text{Slave Latency}$. The Slave Latency field shall have a value in the range of 0 to $(\text{connSupervisionTimeout} / (\text{connIntervalMax} * 2)) - 1$.

The Slave Latency field shall be less than 500.

- *Timeout Multiplier (2 octets)*

Defines connection timeout parameter in the following manner:

$\text{connSupervisionTimeout} = \text{Timeout Multiplier} * 10 \text{ ms}$

The Timeout Multiplier field shall have a value in the range of 10 to 3200.

4.21 CONNECTION PARAMETER UPDATE RESPONSE (CODE 0x13)

This response shall only be sent from the LE master device to the LE slave device.

The Connection Parameter Update Response packet shall be sent by the master Host when it receives a Connection Parameter Update Request packet. If the LE master Host accepts the request it shall send the connection parameter update to its Controller.

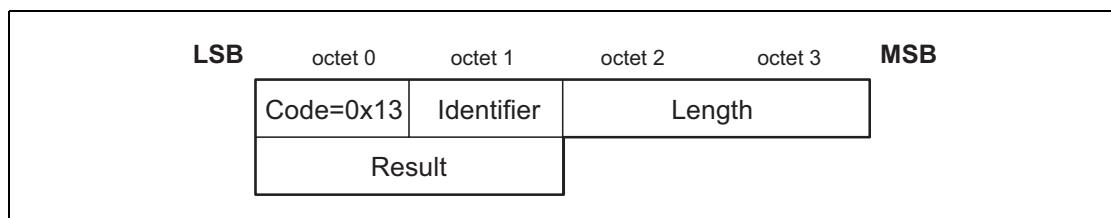


Figure 4.23: Connection Parameters Update Response Packet

The data field is:

- *Result (2 octets)*

The result field indicates the response to the Connection Parameter Update Request. The result value of 0x0000 indicates that the LE master Host has accepted the connection parameters while 0x0001 indicates that the LE master Host has rejected the connection parameters.

Result	Description
0x0000	Connection Parameters accepted
0x0001	Connection Parameters rejected
Other	Reserved

Table 4.18: Result values

4.22 LE CREDIT BASED CONNECTION REQUEST (CODE 0x14)

LE Credit Based Connection Request packets are sent to create and configure an L2CAP channel between two devices. [Figure 4.24](#) illustrates a Connection Request packet.

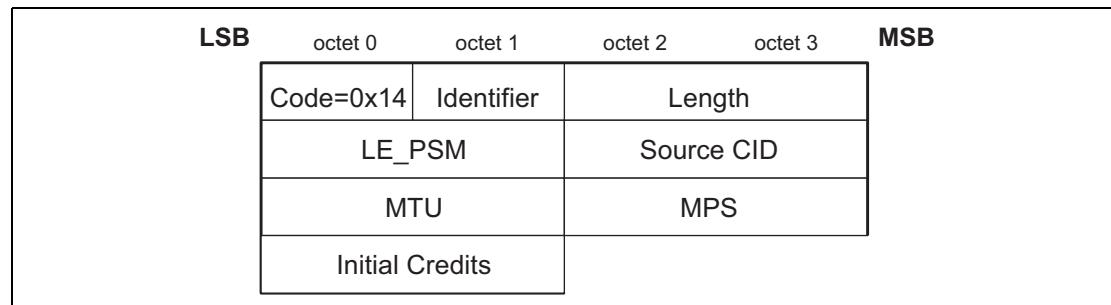


Figure 4.24: LE Credit Based Connection Request Packet

The data fields are:

- *LE Protocol/Service Multiplexer – LE_PSM (2 octets)*

The LE_PSM field is two octets in length. LE_PSM values are separated into two ranges. Values in the first range are assigned by the Bluetooth SIG and indicate protocols. Values in the second range are dynamically allocated and used in conjunction with services defined in the GATT server. The dynamically assigned values may be used to support multiple implementations of a particular protocol.

LE_PSM values are defined in the table below.

Range	Type	Server Usage	Client Usage
0x0001-0x007F	Fixed, SIG assigned	LE_PSM is fixed for all implementations	LE_PSM may be assumed for fixed service. Protocol used is indicated by the LE_PSM as defined in the Assigned Numbers page.
0x0080-0x00FF	Dynamic	LE_PSM may be fixed for a given implementation or may be assigned at the time the service is registered in GATT	LE_PSM shall be obtained from the service in GATT upon every reconnection. LE_PSM for one direction will typically be different from the other direction.
0x0100-0xFFFF	Reserved	Not applicable	Not applicable

Table 4.19: LE Credit Based Connection Request LE_PSM ranges

- *Source CID – SCID (2 octets)*

The source CID is two octets in length and represents a channel endpoint on the device sending the request. Once the channel has been created, data packets flowing to the sender of the request shall be sent to this CID. Thus,

the Source CID represents the channel endpoint on the device sending the request and receiving the response.

- *Maximum Transmission Unit – MTU (2 octets)*

The MTU field specifies the maximum SDU size (in octets) that the L2CAP layer entity sending the LE Credit Based Connection Request can receive on this channel. L2CAP implementations shall support a minimum MTU size of 23 octets.

- *Maximum PDU Size – MPS (2 octets)*

The MPS field specifies the maximum payload size (in octets) that the L2CAP layer entity sending the LE Credit Based Connection Request is capable of receiving on this channel. L2CAP implementations shall support a minimum MPS of 23 octets and may support an MPS up to 65533 octets.

- *Initial Credits – (2 octets)*

The initial credit value indicates the number of LE-frames that the peer device can send to the L2CAP layer entity sending the LE Credit Based Connection Request. The initial credit value shall be in the range of 0 to 65535.

4.23 LE CREDIT BASED CONNECTION RESPONSE (CODE 0x15)

When a device receives a LE Credit Based Connection Request packet, it shall send a LE Credit Based Connection Response packet. The format of the connection response packet is shown in [Figure 4.25](#).

LSB	octet 0	octet 1	octet 2	octet 3	MSB
	Code=0x15	Identifier	Length		
	Destination CID		MTU		
	MPS		Initial Credits		
	Result				

Figure 4.25: LE Credit Based Connection Response Packet

The data fields are:

- *Destination CID – DCID (2 octets)*

The destination CID is two octets in length and represents a channel endpoint on the device sending the response. Once the channel has been created, data packets flowing to the sender of the response shall be sent to this CID. Thus, the destination CID represents the channel endpoint on the device receiving the request and sending the response.

- **Maximum Transmission Unit – MTU (2 octets)**

The MTU field specifies the maximum SDU size (in octets) that the L2CAP layer entity sending the LE Credit Based Connection Response can receive on this channel. L2CAP implementations shall support a minimum MTU size of 23 octets.

- **Maximum PDU Size – MPS (2 octets)**

The MPS field specifies the maximum payload size (in octets) that the L2CAP layer entity sending the LE Credit Based Connection Response is capable of receiving on this channel. L2CAP implementations shall support a minimum MPS of 23 octets and may support an MPS up to 65533 octets.

- **Initial Credits – (2 octets)**

The initial credit value indicates the number of LE-frames that the peer device can send to the L2CAP layer entity sending the LE Credit Based Connection Response. The initial credit value shall be in the range of 0 to 65535.

- **Result – (2 octets)**

The result field indicates the outcome of the connection request. A result value of 0x0000 indicates success while a non-zero value indicates the connection request was refused. A logical channel is established on the receipt of a successful result. [Table 4.20](#) defines values for this field. The DCID, MTU, MPS and Initial Credits fields shall be ignored when the result field indicates the connection was refused.

Value	Description
0x0000	Connection successful
0x0001	Reserved
0x0002	Connection refused – LE_PSM not supported
0x0003	Reserved
0x0004	Connection refused – no resources available
0x0005	Connection refused – insufficient authentication
0x0006	Connection refused – insufficient authorization
0x0007	Connection refused – insufficient encryption key size
0x0008	Connection Refused – insufficient encryption
0x0009	Connection refused - Invalid Source CID
0x000A	Connection refused - Source CID already allocated
0x000B-0xFFFF	Reserved

Table 4.20: Result values for LE Credit Based Connection Response

4.24 LE FLOW CONTROL CREDIT (CODE 0X16)

A device shall send a LE Flow Control Credit packet when it is capable of receive additional LE-frames (for example after it has processed one or more LE-frames).

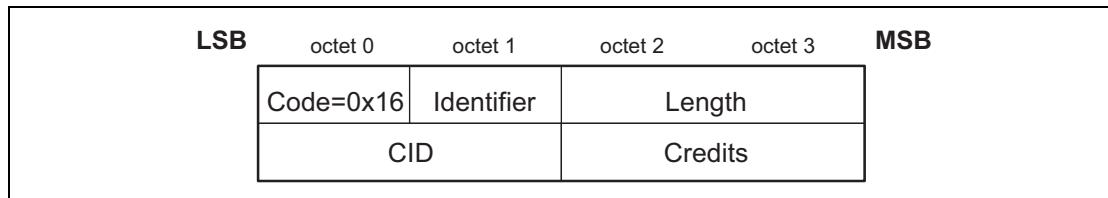


Figure 4.26: LE Flow Control Credit Packet

The data fields are:

- *CID – (2 octets)*

The CID is two octets in length and represents the source channel endpoint of the device sending the LE Flow Control Credit packet. For example, a received LE Flow Control Credit packet with a given CID (0x0042) would provide credits for the receiving device's destination CID (0x0042).

- *Credits – (2 octets)*

The credit value field represents number of credits the receiving device can increment, corresponding to the number of LE-frames that can be sent to the peer device sending the LE Flow Control Credit packet. The credit value field shall be a number between 1 and 65535.

5 CONFIGURATION PARAMETER OPTIONS

Options are a mechanism to extend the configuration parameters. Options shall be transmitted as information elements containing an option type, an option length, and one or more option data fields. [Figure 5.1](#) illustrates the format of an option.

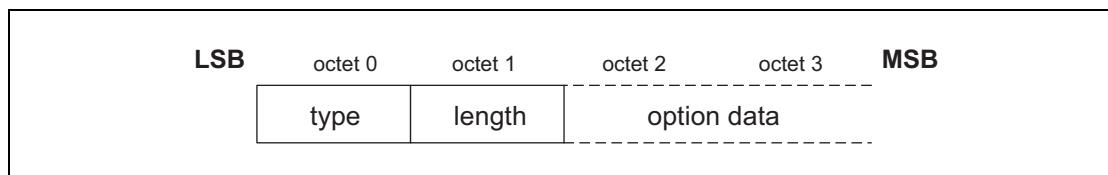


Figure 5.1: Configuration option format

The configuration option fields are:

- *Type (1 octet)*

The option type field defines the parameters being configured. The most significant bit of the type determines the action taken if the option is not recognized.

0 - option must be recognized; if the option is not recognized then refuse the configuration request

1 - option is a hint; if the option is not recognized then skip the option and continue processing

- *Length (1 octet)*

The length field defines the number of octets in the option data. Thus an option type without option data has a length of 0.

- *Option data*

The contents of this field are dependent on the option type.

5.1 MAXIMUM TRANSMISSION UNIT (MTU)

This option specifies the maximum SDU size the sender of this option is capable of accepting for a channel. The type is 0x01, and the payload length is 2 octets, carrying the two-octet MTU size value as the only information element (see [Figure 5.2 on page 90](#)). Unlike the B-Frame length field, the I-frame length field may be greater than the configured MTU because it includes the octet lengths of the Control, L2CAP SDU Length (when present), and frame check sequence fields as well as the Information octets.

MTU is not a negotiated value, it is an informational parameter that each device can specify independently. It indicates to the remote device that the local device can receive, in this channel, an MTU larger than the minimum required. All L2CAP implementations shall support a minimum MTU of 48 octets over the ACL-U logical link and 23 octets over the LE-U logical link; however, some protocols

and profiles explicitly require support for a larger MTU. The minimum MTU for a channel is the larger of the L2CAP minimum 48 octet MTU and any MTU explicitly required by the protocols and profiles using that channel. (Note: the MTU is only affected by the profile directly using the channel. For example, if a service discovery transaction is initiated by a non service discovery profile, that profile does not affect the MTU of the L2CAP channel used for service discovery).

The following rules shall be used when responding to a configuration request specifying the MTU for a channel:

- A request specifying any MTU greater than or equal to the minimum MTU for the channel shall be accepted.
- A request specifying an MTU smaller than the minimum MTU for the channel may be rejected.

The signaling described in [Section 4.5 on page 67](#) may be used to reject an MTU smaller than the minimum MTU for a channel. The "failure-unacceptable parameters" result sent to reject the MTU shall include the proposed value of MTU that the remote device intends to transmit. It is implementation specific whether the local device continues the configuration process or disconnects the channel.

If the remote device sends a positive configuration response it should include the actual MTU to be used on this channel for traffic flowing into the local device. Following the above rules, the actual MTU cannot be less than 48 bytes. This is the minimum of the MTU in the configuration request and the outgoing MTU capability of the device sending the configuration response. The new agreed value (the default value in a future re-configuration) is the value specified in the response.

Note: For backwards compatibility reception of the MTU option in a negative Configuration Response where the MTU option is not in error should be interpreted in the same way as it is in a positive Configuration Response (e.g. the case where another configuration option value is unacceptable but the negative Configuration Response contains the MTU option in addition to the unacceptable option).

The MTU to be used on this channel for the traffic flowing in the opposite direction will be established when the remote device sends its own Configuration Request as explained in [Section 4.4 on page 65](#).

If the configured mode is Enhanced Retransmission mode or Streaming mode then MTU shall not be reconfigured to a smaller size.

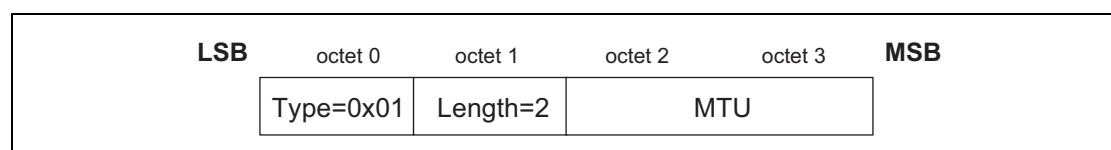


Figure 5.2: MTU Option Format

The option data field is:

- *Maximum Transmission Unit - MTU (2 octets)*

The MTU field is the maximum SDU size, in octets, that the originator of the Request can accept for this channel. The MTU is asymmetric and the sender of the Request shall specify the MTU it can receive on this channel if it differs from the default value. L2CAP implementations shall support a minimum MTU size of 48 octets. The default value is 672 octets¹.

5.2 FLUSH TIMEOUT OPTION

This option is used to inform the recipient of the Flush Timeout the sender is going to use. This option shall not be used if the Extended Flow Specification is used. The Flush Timeout is defined in the BR/EDR Baseband specification [vol.2, part B] Section 7.6.3 on page 156. The type is 0x02 and the payload size is 2 octets. The Flush Timeout option is negotiable.

If the remote device returns a negative response to this option and the local device cannot honor the proposed value, then it shall either continue the configuration process by sending a new request with the original value, or disconnect the channel. The flush timeout applies to all channels on the same ACL logical transport but may be overridden on a packet by packet basis by marking individual L2CAP packets as non-automatically-flushable via the Packet_Boundary_Flag in the HCI ACL Data Packet (see Section 1.1 on page 29).

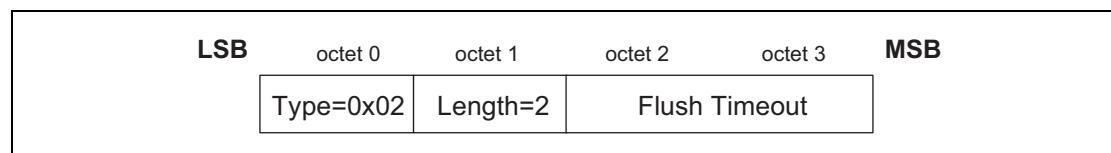


Figure 5.3: Flush Timeout option format

The option data field is:

- *Flush Timeout*

This value is the Flush Timeout in milliseconds. This is an asymmetric value and the sender of the Request shall specify its flush timeout value if it differs from the default value of 0xFFFF.

Possible values are:

0x0001 - no retransmissions at the baseband level should be performed since the minimum polling interval is 1.25 ms.

0x0002 to 0xFFFF - Flush Timeout used by the baseband.

1. The default MTU was selected based on the payload carried by two baseband DH5 packets ($2 \times 341 = 682$ octets) minus the baseband ACL headers ($2 \times 2 = 4$ octets) and a 6-octet L2CAP header. Note that the L2CAP header length is 4 octets (see Section 3.3.1) but for historical reasons an L2CAP header length of 6 bytes is used.

0xFFFF - an infinite amount of retransmissions. This is also referred to as a 'reliable channel'. In this case, the baseband shall continue retransmissions until physical link loss is declared by link manager timeouts.

5.3 QUALITY OF SERVICE (QOS) OPTION

This option specifies a flow specification similar to RFC 1363¹. Although the RFC flow specification addresses only the transmit characteristics, the Bluetooth QoS interface can handle the two directions (Tx and Rx) in the negotiation as described below.

If no QoS configuration parameter is negotiated the link shall assume the default parameters. The QoS option is type 0x03. This option shall not be used if the Extended Flow Specification option is used. The QoS option is negotiable.

In a configuration request, this option describes the outgoing traffic flow from the device sending the request. In a positive Configuration Response, this option describes the incoming traffic flow agreement to the device sending the response. In a negative Configuration Response, this option describes the preferred incoming traffic flow to the device sending the response.

L2CAP implementations are only required to support 'Best Effort' service, support for any other service type is optional. Best Effort does not require any guarantees. If no QoS option is placed in the request, Best Effort shall be assumed. If any QoS guarantees are required then a QoS configuration request shall be sent.

The remote device's Configuration Response contains information that depends on the value of the result field (see [Section 4.5 on page 67](#)). If the request was for Guaranteed Service, the response shall include specific values for any wild card parameters (see Token Rate and Token Bucket Size descriptions) contained in the request. If the result is "Failure – unacceptable parameters", the response shall include a list of outgoing flow specification parameters and parameter values that would make a new Connection Request from the local device acceptable by the remote device. Both explicitly referenced in a Configuration Request or implied configuration parameters can be included in a Configuration Response. Recall that any missing configuration parameters from a Configuration Request are assumed to have their most recently accepted values.

If a configuration request contains any QoS option parameters set to "do not care" then the configuration response shall set the same parameters to "do not care". This rule applies for both Best Effort and Guaranteed Service.

1. Internet Engineering Task Force, "A Proposed Flow Specification", RFC 1363, September 1992.

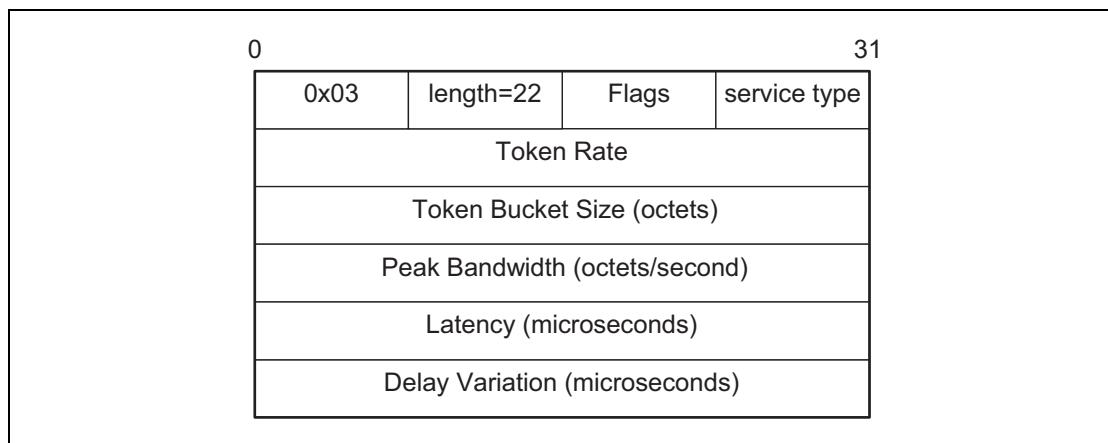


Figure 5.4: Quality of Service (QoS) option format containing Flow Specification

The option data fields are:

- *Flags (1 octet)*

Reserved for future use and shall be set to 0 and ignored by the receiver.

- *Service Type (1 octet)*

This field indicates the level of service required. [Table 5.1 on page 93](#) defines the different services available. The default value is ‘Best effort’.

If ‘Best effort’ is selected, the remaining parameters should be treated as optional by the remote device. The remote device may choose to ignore the fields, try to satisfy the parameters but provide no response (QoS option omitted in the Response message), or respond with the settings it will try to meet.

If ‘No traffic’ is selected, the remainder of the fields shall be ignored because there is no data being sent across the channel in the outgoing direction.

Value	Description
0x00	No traffic
0x01	Best effort (Default)
0x02	Guaranteed
Other	Reserved

Table 5.1: Service type definitions

- *Token Rate (4 octets)*

The value of this field represents the average data rate with which the application transmits data. The application may send data at this rate continuously. On a short time scale the application may send data in excess of the average data rate, dependent on the specified Token Bucket Size and Peak Bandwidth (see below). The Token Bucket Size and Peak Bandwidth allow the application to transmit data in a ‘bursty’ fashion.

The Token Rate signalled between two L2CAP peers is the data transmitted by the application and shall exclude the L2CAP protocol overhead. The Token Rate signalled over the interface between L2CAP and the Link Manager shall include the L2CAP protocol overhead. Furthermore the Token Rate value signalled over this interface may also include the aggregation of multiple L2CAP channels onto the same ACL logical transport.

The Token Rate is the rate with which traffic credits are provided. Credits can be accumulated up to the Token Bucket Size. Traffic credits are consumed when data is transmitted by the application. When traffic is transmitted, and there are insufficient credits available, the traffic is non-conformant. The Quality of Service guarantees are only provided for conformant traffic. For non-conformant traffic there may not be sufficient resources such as bandwidth and buffer space. Furthermore non-conformant traffic may violate the QoS guarantees of other traffic flows.

The Token Rate is specified in octets per second. The value 0x00000000 indicates no token rate is specified. This is the default value and means "do not care". When the Guaranteed service is selected, the default value shall not be used. The value 0xFFFFFFFF is a wild card matching the maximum token rate available. The meaning of this value depends on the service type. For best effort, the value is a hint that the application wants as much bandwidth as possible. For Guaranteed service the value represents the maximum bandwidth available at the time of the request.

- *Token Bucket Size (4 octets)*

The Token Bucket Size specifies a limit on the 'burstiness' with which the application may transmit data. The application may offer a burst of data equal to the Token Bucket Size instantaneously, limited by the Peak Bandwidth (see below). The Token Bucket Size is specified in octets.

The Token Bucket Size signalled between two L2CAP peers is the data transmitted by the application and shall exclude the L2CAP protocol overhead. The Token Bucket Size signalled over the interface between L2CAP and Link Manager shall include the L2CAP protocol overhead. Furthermore the Token Bucket Size value over this interface may include the aggregation of multiple L2CAP channels onto the same ACL logical transport.

The value of 0x00000000 means that no token bucket is needed; this is the default value. When the Guaranteed service is selected, the default value shall not be used. The value 0xFFFFFFFF is a wild card matching the maximum token bucket available. The meaning of this value depends on the service type. For best effort, the value indicates the application wants a bucket as big as possible. For Guaranteed service the value represents the maximum L2CAP SDU size.

The Token Bucket Size is a property of the traffic carried over the L2CAP channel. The Maximum Transmission Unit (MTU) is a property of an L2CAP implementation. For the Guaranteed service the Token Bucket Size shall be smaller or equal to the MTU.

- *Peak Bandwidth (4 octets)*

The value of this field, expressed in octets per second, limits how fast packets from applications may be sent back-to-back. Some systems can take advantage of this information, resulting in more efficient resource allocation.

The Peak Bandwidth signalled between two L2CAP peers specifies the data transmitted by the application and shall exclude the L2CAP protocol overhead. The Peak Bandwidth signalled over the interface between L2CAP and Link Manager shall include the L2CAP protocol overhead. Furthermore the Peak Bandwidth value over this interface may include the aggregation of multiple L2CAP channels onto the same ACL logical transport.

The value of 0x00000000 means "don't care." This states that the device has no preference on incoming maximum bandwidth, and is the default value. When the Guaranteed service is selected, the default value shall not be used.

- *Access Latency (4 octets)*

The value of this field is the maximum acceptable delay of an L2CAP packet to the air-interface. The precise interpretation of this number depends on over which interface this flow parameter is signalled. When signalled between two L2CAP peers, the Access Latency is the maximum acceptable delay between the instant when the L2CAP SDU is received from the upper layer and the start of the L2CAP SDU transmission over the air. When signalled over the interface between L2CAP and the Link Manager, it is the maximum delay between the instant the first fragment of an L2CAP PDU is stored in the Host Controller buffer and the initial transmission of the L2CAP packet on the air.

Thus the Access Latency value may be different when signalled between L2CAP and the Link Manager to account for any queuing delay at the L2CAP transmit side. Furthermore the Access Latency value may include the aggregation of multiple L2CAP channels onto the same ACL logical transport.

The Access Latency is expressed in microseconds. The value 0xFFFFFFFF means "do not care" and is the default value. When the Guaranteed service is selected, the default value shall not be used.

- *Delay Variation (4 octets)*

The value of this field is the difference, in microseconds, between the maximum and minimum possible delay of an L2CAP SDU between two L2CAP peers. The Delay Variation is a purely informational parameter. The value 0xFFFFFFFF means "do not care" and is the default value.

5.4 RETRANSMISSION AND FLOW CONTROL OPTION

This option specifies whether retransmission and flow control is used. If the feature is used both incoming and outgoing parameters are specified by this option. The Retransmission and Flow Control option contains both negotiable parameters and non-negotiable parameters. The mode parameter controls both incoming and outgoing data flow (i.e. both directions have to agree) and is negotiable. The other parameters control incoming data flow and are non-negotiable.

0	31		
0x04	Length=9	Mode	TxWindow size
Max Transmit	Retransmission time-out	Monitor time-out (least significant byte)	
Monitor time-out (most significant byte)	Maximum PDU size (MPS)		

Figure 5.5: Retransmission and Flow Control option format

The option data fields are:

- **Mode (1 octet)**

The field contains the requested mode of the link. Possible values are shown in [Table 5.2 on page 96](#).

Value	Description
0x00	L2CAP Basic Mode
0x01	Retransmission mode
0x02	Flow control mode
0x03	Enhanced Retransmission mode
0x04	Streaming mode
Other values	Reserved for future use

Table 5.2: Mode definitions

The Basic L2CAP mode is the default. If Basic L2CAP mode is requested then all other parameters shall be ignored.

Enhanced Retransmission mode should be enabled if a reliable channel has been requested. Enhanced Retransmission mode shall only be sent to an L2CAP entity that has previously advertised support for the mode in its Extended Feature Mask (see [section 4.12](#)).

Streaming mode should be enabled if a finite L2CAP Flush Time-Out is set on an L2CAP connection. Streaming mode shall only be sent to a device that has previously advertised support for the mode in the Extended Feature Mask (see [section 4.12](#)).

Flow Control mode and Retransmission mode shall only be used for backwards compatibility with L2CAP entities that do not support Enhanced Retransmission mode or Streaming mode.

- *TxWindow size (1 octet)*

This field specifies the size of the transmission window for Flow Control mode, Retransmission mode, and Enhanced Retransmission mode. The range is 1 to 32 for Flow Control mode and Retransmission mode. The range is 1 to 63 for Enhanced Retransmission mode.

In Retransmission mode and Flow Control mode this parameter should be negotiated to reflect the buffer sizes allocated for the connection on both sides. In general, the Tx Window size should be made as large as possible to maximize channel utilization. Tx Window size also controls the delay on flow control action. The transmitting device can send as many PDUs fit within the window.

In Enhanced Retransmission mode this value indicates the maximum number of I-frames that the sender of the option can receive without acknowledging some of the received frames. It is not negotiated. It is an informational parameter that each L2CAP entity can specify separately. In general, the TxWindow size should be made as large as possible to maximize channel utilization. The transmitting L2CAP entity can send as many PDUs as will fit within the receiving L2CAP entity's TxWindow.

TxWindow size values in a Configuration Response indicate the maximum number of packets the sender can send before it requires an acknowledgement. In other words it represents the number of unacknowledged packets the send can hold. The value sent in a Configuration Response shall be less than or equal to the TxWindow size sent in the Configuration Request. The receiver of this option in the Configuration Response may use this value as part of its acknowledgement algorithm.

In Streaming mode this value is not used and shall be set to 0 and ignored by the receiving device.

- *MaxTransmit (1 octet)*

This field controls the number of transmissions of a single I-frame that L2CAP is allowed to try in Retransmission mode and Enhanced Retransmission mode. The minimum value is 1 (one transmission is permitted).

MaxTransmit controls the number of retransmissions that L2CAP is allowed to try in Retransmission mode and Enhanced Retransmission mode before accepting that a packet and the channel is lost. When a packet is lost after being transmitted MaxTransmit times the channel shall be disconnected by sending a Disconnect request (see [Section 4.6](#)). In Enhanced Retransmission mode MaxTransmit controls the number of retransmissions for I-frames and S-frames with P-bit set to 1. The sender of the option in a Configuration Request specifies the value that shall be used by the receiver of the option. MaxTransmit values in a Configuration Response shall be ignored. Lower values might be appropriate for services requiring low latency. Higher values will be suitable for a link

requiring robust operation. A value of 1 means that no retransmissions will be made but also means that the channel will be disconnected as soon as a packet is lost. MaxTransmit shall not be set to zero in Retransmission mode. In Enhanced Retransmission mode a value of zero for MaxTransmit means infinite retransmissions.

In Streaming mode this value is not used and shall be set to 0 and ignored by the receiving L2CAP entity.

- *Retransmission time-out (2 octets)*

This is the value in milliseconds of the retransmission time-out (this value is used to initialize the RetransmissionTimer).

The purpose of this timer in retransmission mode is to activate a retransmission in some exceptional cases. In such cases, any delay requirements on the channel may be broken, so the value of the timer should be set high enough to avoid unnecessary retransmissions due to delayed acknowledgements. Suitable values could be 100's of milliseconds and up.

The purpose of this timer in flow control mode is to supervise I-frame transmissions. If an acknowledgement for an I-frame is not received within the time specified by the RetransmissionTimer value, either because the I-frame has been lost or the acknowledgement has been lost, the timeout will cause the transmitting side to continue transmissions. Suitable values are implementation dependent.

The purpose of this timer in Enhanced Retransmission mode is to detect lost I-frames and initiate appropriate error recovery. The value used for the Retransmission time-out is specified in [Section 8.6.2](#). The value sent in a Configuration Request is also specified in [Section 8.6.2](#). A value for the Retransmission time-out shall be sent in a positive Configuration Response and indicates the value that will be used by the sender of the Configuration Response.

In Streaming mode this value is not used and shall be set to 0 and ignored by the receiving L2CAP entity.

- *Monitor time-out (2 octets)*

In Retransmission mode this is the value in milliseconds of the interval at which S-frames should be transmitted on the return channel when no frames are received on the forward channel. (this value is used to initialize the MonitorTimer, see below).

This timer ensures that lost acknowledgements are retransmitted. Its main use is to recover Retransmission Disable Bit changes in lost frames when no data is being sent. The timer shall be started immediately upon transitioning to the open state. It shall remain active as long as the connection is in the open state and the retransmission timer is not active. Upon expiration of the Monitor timer an S-frame shall be sent and the timer shall be restarted. If the monitor timer is already active when an S-frame is sent, the timer shall be restarted. An idle connection will have periodic

monitor traffic sent in both directions. The value for this time-out should also be set to 100's of milliseconds or higher.

In Enhanced Retransmission mode the Monitor time-out is used to detect lost S-frames with P-bit set to 1. If the time-out occurs before a response with the F-bit set to 1 is received the S-frame is resent. The value used for the Monitor time-out is specified in [Section 8.6.3](#). The value sent in a Configuration Request is also specified in [Section 8.6.2](#). A value for the Monitor time-out shall be sent in a positive Configuration Response and indicates the value that will be used by the sender of the Configuration Response.

In Streaming mode this value is not used and shall be set to 0 and ignored by the receiving device.

- *Maximum PDU payload Size - MPS (2 octets)*

The maximum size of payload data in octets that the L2CAP layer entity sending the option in a Configuration Request is capable of accepting, i.e. the MPS corresponds to the maximum PDU payload size. Values used for MPS should take into account all possible Controllers to which the channel might be moved given that the MPS value cannot be renegotiated once the channel is created. Each device specifies the value separately. An MPS value sent in a positive Configuration Response is the actual MPS the receiver of the Configuration Request will use on this channel for traffic flowing into the local device. An MPS value sent in a positive Configuration Response shall be equal to or smaller than the value sent in the Configuration Request.

When using Retransmission mode and Flow Control mode the settings are configured separately for the two directions of an L2CAP connection. For example, in operating with an L2CAP entity implementing an earlier version of the core specification, an L2CAP connection can be configured as Flow Control mode in one direction and Retransmission mode in the other direction. If Basic L2CAP mode is configured in one direction and Retransmission mode or Flow control mode is configured in the other direction on the same L2CAP channel then the channel shall not be used.

Note: This asymmetric configuration only occurs during configuration.

When using Enhanced Retransmission mode or Streaming mode, both directions of the L2CAP connection shall be configured to the same mode. A precedence algorithm shall be used by both devices so a mode conflict can be resolved in a quick and deterministic manner.

There are two operating states:

- A device has a desired mode but is willing to use another mode (known as "state 1"), and
- A device requires a specific mode (known as "state 2"). This includes cases where channels are created over AMP-U logical links or ACL-U logical links operating as described in [Section 7.10](#).

In state 1, Basic L2CAP mode has the highest precedence and shall take precedence over Enhanced Retransmission mode and Streaming mode. Enhanced Retransmission mode has the second highest precedence and shall take precedence over all other modes except Basic L2CAP mode. Streaming mode shall have the next level of precedence after Enhanced Retransmission mode.

In state 2, a layer above L2CAP requires Enhanced Retransmission mode or Streaming mode. In this case, the required mode takes precedence over all other modes.

A device does not know in which state the remote device is operating so the state 1 precedence algorithm assumes that the remote device may be a state 2 device. If the mode proposed by the remote device has a higher precedence (according to the state 1 precedence) then the algorithm will operate such that creation of a channel using the remote device's mode has higher priority than disconnecting the channel.

The algorithm for state 1 devices is divided into two parts. Part one covers the case where the remote device proposes a mode with a higher precedence than the state 1 local device. Part two covers the case where the remote device proposes a mode with a lower precedence than the state 1 local device. Part one of the algorithm is as follows:

- When the remote device receives the Configuration Request from the local device it will either reject the local device's Configuration Request by sending a negative Configuration Response or disconnect the channel. The negative Configuration Response will contain the remote device's desired mode.
- Upon receipt of the negative Configuration Response the local device shall either send a second Configuration Request proposing the mode contained in the remote device's negative Configuration Response or disconnect the channel.
- When the local device receives the Configuration Request from the remote device it shall send a positive Configuration Response or disconnect the channel.
- If the mode in the remote Device's negative Configuration Response does not match the mode in the remote device's Configuration Request then the local device shall disconnect the channel.

Part two of the algorithm is as follows:

- When the local device receives the Configuration Request from the remote device it shall reject the Configuration Request by sending a negative Configuration Response proposing its desired mode. The local device's desired mode shall be the same mode it sent in its Configuration Request. Upon receiving the negative Configuration Response the remote device will either send a second Configuration Request or disconnect the channel.

- If the local device receives a second Configuration Request from the remote device that does not contain the desired mode then the local device shall disconnect the channel.
- If the local device receives a negative Configuration Response then it shall disconnect the channel.

An example of the algorithm for state 1 devices is as follows:

- The remote device proposes Basic L2CAP mode in a Configuration Request and the local device proposes Enhanced Retransmission mode or Streaming mode. The remote device rejects the local device's Configuration Request by sending a negative Configuration Response proposing Basic mode. The local device will send a second Configuration Request proposing Basic L2CAP mode or disconnect the channel. If the local device sends a second Configuration Request that does not propose Basic L2CAP mode then the remote device will disconnect the channel. If the local device rejects the remote device's Configuration Request then the remote device will disconnect the channel.

The algorithm for state 2 devices is as follows:

- If the local device proposes a mode in a Configuration Request and the remote device proposes a different mode or rejects the local device's Configuration Request then the local device shall disconnect the channel.

For Enhanced Retransmission mode and Streaming mode the Retransmission time-out and Monitor Time-out parameters of the Retransmission and Flow Control option parameters may be changed but all other parameters shall not be changed in a subsequent reconfiguration after the channel has reached the OPEN state.

5.5 FRAME CHECK SEQUENCE (FCS) OPTION

This option is used to specify the type of Frame Check Sequence (FCS) that will be included on S/I-frames that are sent. It is non-negotiable. The FCS option shall only be used when the mode is being, or is already configured to Enhanced Retransmission mode or Streaming mode. Note: The FCS option can be reconfigured only when the mode is Enhanced Retransmission mode or Streaming mode. Implementations may reconfigure the FCS when L2CAP channels are moved between Controllers. The Frame Check Sequence option is type 0x05. "No FCS" shall only be used if both L2CAP entities send the FCS Option with value 0x00 (No FCS) in a configuration request. If one L2CAP entity sends the FCS Option with "No FCS" in a configuration request and the other L2CAP sends the FCS Option with a value other than "No FCS" then the default shall be used. If one or both L2CAP entities do not send the FCS option in a configuration request then the default shall be used.

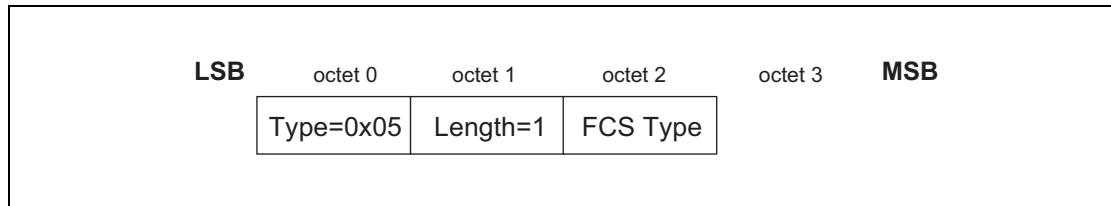


Figure 5.6: FCS option format

The FCS types are shown in [Table 5.3](#)

Value	Description
0x00	No FCS
0x01	16-bit FCS defined in section 3.3.5 (default)
0x02 - 0xFF	Reserved

Table 5.3: FCS types

Value of 0x00 is set when the sender wishes to omit the FCS from S/I-frames.

5.6 EXTENDED FLOW SPECIFICATION OPTION

This option specifies a flow specification for requesting a desired Quality of Service (QoS) on a channel. It is non-negotiable. The Extended Flow Specification shall be supported on all channels created over AMP-U logical links. Optionally, Extended Flow Specification may be supported on channels created over ACL-U logical links (see [Section 7.10](#)). If both devices show support for Extended Flow Specification for BR/EDR in the Extended Feature mask (see [Table 4.12](#)) then all channels created between the two devices shall use an Extended Flow Specification. The Quality of Service option and Flush Timeout option shall not be used if the Extended Flow Specification is used.

The parameters in the Extended Flow Specification option specify the traffic stream in the outgoing direction (transmitted traffic). Extended Flow Specification option is type 0x06.

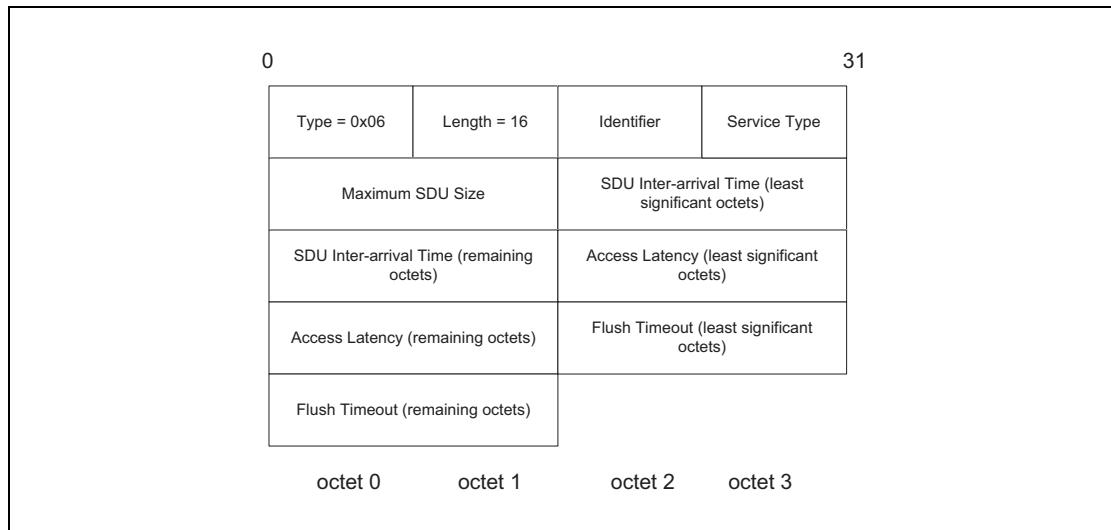


Figure 5.7: Extended Flow Specification option format

If no Extended Flow Specification is provided by the upper layer, an Extended Flow Specification with following default values shall be used:

QoS Parameter	Default Value
Identifier	0x01
Service Type	Best Effort
Maximum SDU size	0xFFFF
SDU Inter-arrival Time	0xFFFFFFFF
Access Latency	0xFFFFFFFF
Flush Timeout	0xFFFFFFFF

Table 5.4: Default Values for Extended Flow Specification

For a Best Effort channel no latency, air-time or bandwidth guarantees shall be assumed.

The parameters of the Extended Flow Specification are shown in [Table 5.5](#):

QoS parameter	Parameter Size in Octets	Unit
Identifier	1	na
Service Type	1	na
Maximum SDU Size	2	octets
SDU Inter-arrival Time	4	microseconds

Table 5.5: Traffic parameters for L2CAP QoS configuration

QoS parameter	Parameter Size in Octets	Unit
Access Latency	4	microseconds
Flush Timeout	4	microseconds

Table 5.5: Traffic parameters for L2CAP QoS configuration

- *Identifier (1 octet)*

This field provides a unique identifier for the flow specification. This identifier is used by some Controllers in the process of setting up the QoS request. Each active flow specification sent by a device to configure an L2CAP channel shall have a unique Identifier. An Identifier can be reused when the L2CAP channel associated with the flow spec is disconnected. The Identifier shall be unique within the scope of a physical link. Extended Flow Specifications for channels on different physical links may have the same Identifier. The Identifier for an Extended Flow Specification with Service Type Best Effort shall be 0x01.

Note: Since the Identifier for an Extended Flow Specification with Service Type Best Effort is fixed to 0x01 it is possible to generate a Best Effort Extended Flow Specification for the remote device without performing the Lockstep Configuration process. (The Lockstep Configuration process is described in [Section 7.1.3](#)). This allows a Best Effort channel created over the ACL-U logical link without using the Lockstep configuration procedure to be moved to an AMP-U logical link. It is not possible to move a Guaranteed channel created over the ACL-U logical link to an AMP-U logical link unless the Lockstep configuration procedure is used during channel creation. This is because the Identifier for the remote device's Extended Flow Specification with Service Type Guaranteed is not known.

- *Service Type (1 octet)*

This field indicates the level of service required. [Table 5.6](#) defines the different Service Types values. The default value is 'Best effort'. If 'Best effort' is selected then Access Latency and Flush Timeout shall both be set to 0xFFFFFFFF. Maximum SDU size and SDU Inter-arrival Time are used to indicate the maximum data rate that the application can deliver to L2CAP for transmission. This is useful for high speed Controllers so they do not reserve more bandwidth than the application can deliver. The remote device should respond with lower settings indicating the maximum rate at which it can receive data (for example, maximum rate data it can write to a mass storage device, etc.). Values of 0xFFFF for Maximum SDU size and 0xFFFFFFFF for SDU Inter-arrival time are used when the actual values are not known. If Maximum SDU size is set to 0xFFFF then SDU Inter-arrival time shall be set to 0xFFFFFFFF, if SDU Inter-arrival time is set to 0xFFFFFFFF then Maximum SDU size shall be set to 0xFFFF. This tells the Controller to allocate as much bandwidth as possible.

If “Guaranteed” is selected the QoS parameters can be used to identify different types of Guaranteed traffic.

- **Guaranteed bandwidth** traffic is traffic with a minimum data rate but no particular latency requested. Latency will be related to the link supervision timeout. For this type of traffic Access Latency is set to 0xFFFFFFFF.
- **Guaranteed Latency** traffic is traffic with only latency requirements. For this type of traffic SDU Inter-arrival time is set to 0xFFFFFFFF. HID interrupt channel and AVRCP are examples of this type of traffic.
- **Both Guaranteed Latency and Bandwidth** traffic has both a latency and bandwidth requirement. An example is Audio/Video streaming.

If 'No Traffic' is selected the remainder of the fields shall be ignored because there is no data being sent across the channel in the outgoing direction.

Value	Description
0x00	No Traffic
0x01	Best effort (Default)
0x02	Guaranteed
Other	Reserved

Table 5.6: Service Type definitions

A channel shall not be configured as “Best Effort” in one direction and “Guaranteed” in the other direction. If a channel is configured in this way it shall be disconnected. A channel may be configured as “No Traffic” in one direction and “Best Effort” in the other direction. The “No Traffic” refers to the application traffic not the Enhanced Retransmission mode supervisory traffic. A channel configured in this way is considered to have a service type of Best Effort. A channel may be configured as “No Traffic” in one direction and “Guaranteed” in the other direction. A channel configured in this way is considered to have a service type of Guaranteed.

Once configured the service type of a channel shall not be changed during reconfiguration.

- *Maximum SDU Size (2 octets)*

The Maximum SDU Size parameter specifies the maximum size of the SDUs transmitted by the application. If the Service Type is “Guaranteed” then traffic submitted to L2CAP with a larger size is considered non-conformant. QoS guarantees are only provided for conformant traffic.

- *SDU Inter-arrival time (4 octets)*

The SDU Inter-arrival time parameter specifies the time between consecutive SDUs generated by the application. For streaming traffic, SDU Inter-arrival time should be set to the average time between SDUs. For variable rate traffic and best effort traffic, SDU Inter-arrival time should be set to the minimum time between SDUs. If the Service Type is “Guaranteed”

then traffic submitted to L2CAP with a smaller interval, is considered non-conformant. QoS guarantees are only provided for conformant traffic.

- *Access Latency (4 octets)*

The Access Latency parameter specifies the maximum delay between consecutive transmission opportunities on the air-interface for the connection. Note that access latency is based on the time base of the Controller, which may not be synchronous to the time base being used by the host or the application.

For streaming traffic (for example, A2DP), the Access Latency should be set to indicate the time budgeted for transmission of the data over the air, and would normally be roughly equal to the Flush Timeout minus the duration of streaming data which can be stored in the receive side application buffers.

For non-streaming, bursty traffic (i.e. HID and AVRCP), the Access Latency parameter value sent in the L2CAP Configuration Request should be set to the desired latency, minus any HCI transport delays and any other stack delays that may be expected on the device and on target host systems. The remote device receiving the L2CAP Configuration Request may send an Access Latency parameter value in the L2CAP Configuration Response which is equal to or lower than the value it received. The remote device may send a lower value to account for other traffic it may be carrying, or overhead activities it may be carrying out.

If HCI is used then the host should take into account the latency of the HCI transport when determining the value for the Access Latency. For example if the application requires an Access Latency of 20ms and the HCI transport has a latency of 5 ms then the value for Access Latency should be 15ms.

- *Flush Timeout (4 Octets)*

The Flush Timeout defines a maximum period after which all segments of the SDU are flushed from L2CAP and the Controller. A Flush Timeout value of 0xFFFFFFFF indicates that data will not be discarded by the transmitting side, even if the link becomes congested. Note that in this case data is treated as reliable, and is never flushed. The device receiving the L2CAP Configuration Request with Flush Timeout set to 0xFFFFFFFF should not modify this parameter. The Flush Timeout for a “Best Effort” channel shall be set to 0xFFFFFFFF.

However, if the Traffic Type is “Guaranteed” and the transmit side buffer is limited, then the Flush Timeout parameter given in the L2CAP Configuration Request may be set to a value corresponding to the duration of streaming data which the transmit buffer can hold before it must begin discarding data. The side receiving the L2CAP Configuration Request may then set the Flush Timeout parameter in the L2CAP Configuration Response to a lower value if the receive side buffer is smaller than the transmit side buffer. Note that the total available buffer space is typically a combination of application buffers, any buffers maintained by the L2CAP implementation, and HCI buffers provided by the Controller.

The Flush Timeout should normally be set to a value which is larger than the Access Latency, and which also accounts for buffers maintained by the application on the receive side such as de-jitter buffers used in audio and video streaming applications. In general, the Flush Timeout value should be selected to ensure that the Flush Timeout expires when the application buffers are about to be exhausted.

Flush Timeout may be set to 0x00000000 to indicate that no retransmissions are to occur. Data may be flushed immediately after transmission in this case. This behavior is useful in applications such as gaming where the tolerable latency is on the order of a few milliseconds, and hence the information contained in a packet will become stale very rapidly. In such applications, it is preferable to send “fresher” data if the last SDU submitted for transmission was not transmitted as a result of interference.

5.7 EXTENDED WINDOW SIZE OPTION

This option is used to negotiate the maximum extended window size. The Extended Window Size option is type 0x07 and is non-negotiable.

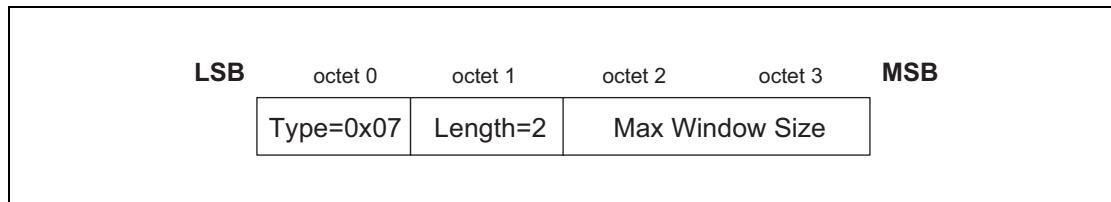


Figure 5.8: Extended Window Size option format

The allowable values for the Maximum Extended Window Size are:

Value	Description
0x0000	Invalid value for Enhanced Retransmission mode Valid for Streaming mode
0x0001 - 0x3FFF	Valid maximum window size (frames) for Enhanced Retransmission mode. Invalid for Streaming mode
0x4000 - 0xFFFF	Reserved

Table 5.7: Extended Window Size values

This option shall only be sent if the peer L2CAP entity has indicated support for the Extended Window size feature in the Extended Features Mask (see [Section 4.12](#)).

This option shall be ignored if the channel is configured to use Basic L2CAP Mode (see [Section 5.4](#)).

For Enhanced Retransmission mode, this option has the same directional semantics as the Retransmission and Flow Control option (see [Section 5.4](#)). The sender of a Configuration Request command containing this option is suggesting the maximum window size (possibly based on its own internal L2CAP receive buffer resources) that the peer L2CAP entity should use when sending data.

For Streaming mode, this option is used to enable use of the Extended Control Field and if sent, shall have a value of 0.

If this option is successfully configured then the Maximum Window Size negotiated using the Retransmission and Flow Control option (see [Section 5.4](#)) shall be ignored.

If this option is successfully configured in either direction then both L2CAP entities shall use the Extended Control Fields in all S/I-frames (see [Section 3.3.2](#)).

If the option is only configured in one direction then the maximum window size for the opposite direction shall be taken from the maximum window size value in the existing Retransmission and Flow Control option. In this configuration both L2CAP entities shall use the extended control fields in all S/I-frames.

6 STATE MACHINE

This section is informative. The state machine may not represent all possible scenarios.

6.1 GENERAL RULES FOR THE STATE MACHINE:

- It is implementation specific, and outside the scope of this specification, how the transmissions are triggered.
- “Ignore” means that the signal can be silently discarded.

The following states have been defined to clarify the protocol; the actual number of states and naming in a given implementation is outside the scope of this specification:

- CLOSED – channel not connected.
- WAIT_CONNECT – a connection request has been received, but only a connection response with indication “pending” can be sent.
- WAIT_CONNECT_RSP – a connection request has been sent, pending a positive connect response.
- CONFIG – the different options are being negotiated for both sides; this state comprises a number of substates, see [Section 6.1.3 on page 114](#)
- OPEN – user data transfer state.
- WAIT_DISCONNECT – a disconnect request has been sent, pending a disconnect response.
- WAIT_CREATE – a channel creation request has been received, but only a response with indication “pending” can be sent. This state is similar to WAIT_CONNECT.
- WAIT_CREATE_RSP – a channel creation request has been sent, pending a channel creation response. This state is similar to WAIT_CONNECT_RSP.
- WAIT_MOVE – a request to move the current channel to another Controller has been received, but only a response with indication “pending” can be sent.
- WAIT_MOVE_RSP – a request to move a channel to another Controller has been sent, pending a move response
- WAIT_MOVE_CONFIRM – a response to the move channel request has been sent, waiting for a confirmation of the move operation by the initiator side
- WAIT_CONFIRM_RSP – a move channel confirm has been sent, waiting for a move channel confirm response.

In the following state tables and descriptions, the L2CAP_Data message corresponds to one of the PDU formats used on connection-oriented data channels as described in section 3, including PDUs containing B-frames, I-frames, S-frames.

Some state transitions and actions are triggered only by internal events effecting one of the L2CAP entity implementations, not by preceding L2CAP signaling messages. It is implementation-specific and out of the scope of this specification, how these internal events are realized; just for the clarity of specifying the state machine, the following abstract internal events are used in the state event tables, as far as needed:

- *OpenChannel_Req* – a local L2CAP entity is requested to set up a new connection-oriented channel.
- *OpenChannel_Rsp* – a local L2CAP entity is requested to finally accept or refuse a pending connection request.
- *ConfigureChannel_Req* – a local L2CAP entity is requested to initiate an outgoing configuration request.
- *CloseChannel_Req* – a local L2CAP entity is requested to close a channel.
- *SendData_Req* – a local L2CAP entity is requested to transmit an SDU.
- *ReconfigureChannel_Req* – a local L2CAP entity is requested to reconfigure the parameters of a connection-oriented channel.
- *OpenChannelCntrl_Req* – a local L2CAP entity is requested to set up a new connection over a Controller identified by a Controller ID.
- *OpenChannelCntrl_Rsp* – a local L2CAP entity is requested to internally accept or refuse a pending connection over a Controller identified by a Controller ID.
- *MoveChannel_Req* – a local L2CAP entity is requested to move the current channel to another Controller.
- *ControllerLogicalLinkInd* – a Controller indicates the acceptance or rejection of a logical link request with an Extended Flow Specification.

There is a single state machine for each L2CAP connection-oriented channel that is active. A state machine is created for each new L2CAP_ConnectReq received. The state machine always starts in the CLOSED state.

To simplify the state event tables, the RTX and ERTX timers, as well as the handling of request retransmissions are described in [Section 6.1.7 on page 122](#) and not included in the state tables.

L2CAP messages not bound to a specific data channel and thus not impacting a channel state (e.g. L2CAP_InformationReq, L2CAP_EchoReq) are not covered in this section.

The following states and transitions are illustrated in [Figure 6.1 on page 128](#).

6.1.1 CLOSED state

Event	Condition	Action	Next State
OpenChannel_req	-	Send L2CAP_ConnectReq	WAIT_CONNECT_RSP
L2CAP_ConnectReq	Normal, connection is possible	Send L2CAP_ConnectRsp (success)	CONFIG (substate WAIT_CONFIG)
L2CAP_ConnectReq	Need to indicate pending	Send L2CAP_ConnectRsp (pending)	WAIT_CONNECT
L2CAP_ConnectReq	No resource, not approved, etc.	Send L2CAP_ConnectRsp (refused)	CLOSED
L2CAP_ConnectRsp	-	Ignore	CLOSED
L2CAP_ConfigReq	-	Send L2CAP_CommandReject (with reason Invalid CID)	CLOSED
L2CAP_ConfigRsp	-	Ignore	CLOSED
L2CAP_DisconnectReq	-	Send L2CAP_DisconnectRsp	CLOSED
L2CAP_DisconnectRsp	-	Ignore	CLOSED
L2CAP_Data	-	Ignore	CLOSED
OpenChannelCntrl_Req		Send L2CAP_CreateChanReq	WAIT_CREATE_RSP
L2CAP_CreateChanReq	Normal, connection is possible	Send L2CAP_CreateChanRsp (success)	CONFIG
L2CAP_CreateChanReq	Need to indicate pending	Send L2CAP_CreateChanRsp (pending)	WAIT_CREATE
L2CAP_CreateChanReq	No resource, not approved	Send L2CAP_CreateChanRsp (refused)	CLOSED
L2CAP_CreateChanRsp		Ignore	CLOSED
L2CAP_MoveChanReq		Ignore	CLOSED
L2CAP_MoveChanRsp		Ignore	CLOSED
L2CAP_MoveChanConfirm		Ignore	CLOSED
L2CAP_MoveChanConfirmRsp		Ignore	CLOSED

Table 6.1: CLOSED state event table

Notes. The L2CAP_ConnectReq message is not mentioned in any of the other states apart from the CLOSED state, as it triggers the establishment of a new channel, thus the branch into a new instance of the state machine.

6.1.2 WAIT_CONNECT_RSP state

Event	Condition	Action	Next State
L2CAP_ConnectRsp	Success indicated in result	Send L2CAP_ConfigReq	CONFIG (substate WAIT_CONNECT)
L2CAP_ConnectRsp	Result pending	-	WAIT_CONNECT_RSP
L2CAP_ConnectRsp	Remote side refuses connection	-	CLOSED
L2CAP_ConfigReq	-	Send L2CAP_CommandReject (with reason Invalid CID)	WAIT_CONNECT_RSP
L2CAP_ConfigRsp	-	Ignore	WAIT_CONNECT_RSP
L2CAP_DisconnectRsp	-	Ignore	WAIT_CONNECT_RSP
L2CAP_Data	-	Ignore	WAIT_CONNECT_RSP
L2CAP_CreateChanReq		Ignore	WAIT_CONNECT_RSP
L2CAP_CreateChanRsp		Ignore	WAIT_CONNECT_RSP
L2CAP_MoveChanReq		Ignore	WAIT_CONNECT_RSP
L2CAP_MoveChanRsp		Ignore	WAIT_CONNECT_RSP
L2CAP_MoveChanConfirm		Ignore	WAIT_CONNECT_RSP
L2CAP_MoveChanConfirmRsp		Ignore	WAIT_CONNECT_RSP

Table 6.2: WAIT_CONNECT_RSP state event table

Notes. An L2CAP_DisconnectReq message is not included here, since the Source and Destination CIDs are not available yet to relate it correctly to the state machine of a specific channel.

6.1.3 WAIT_CONNECT state

Event	Condition	Action	Next State
<i>OpenChannel_Rsp</i>	Pending connection request is finally acceptable	Send L2CAP_Connect_Rsp (success)	CONFIG (substate WAIT_CONFIG)
<i>OpenChannel_Rsp</i>	Pending connection request is finally refused	Send L2CAP_Connect_Rsp (refused)	CLOSED
L2CAP_ConnectRsp	-	Ignore	WAIT_CONNECT
L2CAP_ConfigRsp	-	Ignore	WAIT_CONNECT
L2CAP_DisconnectRsp	-	Ignore	WAIT_CONNECT
L2CAP_Data	-	Ignore	WAIT_CONNECT
L2CAP_CreateChanReq		Ignore	WAIT_CONNECT
L2CAP_CreateChanRsp		Ignore	WAIT_CONNECT
L2CAP_MoveChanReq		Ignore	WAIT_CONNECT
L2CAP_MoveChanRsp		Ignore	WAIT_CONNECT
L2CAP_MoveChanConfirm		Ignore	WAIT_CONNECT
L2CAP_MoveChanConfirmRsp		Ignore	WAIT_CONNECT

Table 6.3: WAIT_CONNECT state event table

Notes. An L2CAP_DisconnectReq or L2CAP_ConfigReq message is not included here, since the Source and Destination CIDs are not available yet to relate it correctly to the state machine of a specific channel.

6.1.4 CONFIG state

Two configuration processes exist as described in [Section 7.1 on page 131](#). The configuration processes are the Standard process and the Lockstep process.

In the Standard and Lockstep configuration processes both L2CAP entities initiate a configuration request during the configuration process. This means that each device adopts an initiator role for the outgoing configuration request, and an acceptor role for the incoming configuration request. Configurations in both directions may occur sequentially, but can also occur in parallel.

In the Lockstep configuration process both L2CAP entities send Configuration Request packets and receive Configuration Response packets with a pending result code before submitting the flow specifications to their local controllers. A final Configuration Response packet is sent by each L2CAP entity indicating the response from its local controller.

The following substates are distinguished within the CONFIG state:

- WAIT_CONFIG – a device has sent or received a connection response, but has neither initiated a configuration request yet, nor received a configuration request with acceptable parameters.
- WAIT_SEND_CONFIG – for the initiator path, a configuration request has not yet been initiated, while for the response path, a request with acceptable options has been received.
- WAIT_CONFIG_REQ_RSP – for the initiator path, a request has been sent but a positive response has not yet been received, and for the acceptor path, a request with acceptable options has not yet been received.
- WAIT_CONFIG_RSP – the acceptor path is complete after having responded to acceptable options, but for the initiator path, a positive response on the recent request has not yet been received.
- WAIT_CONFIG_REQ – the initiator path is complete after having received a positive response, but for the acceptor path, a request with acceptable options has not yet been received.
- WAIT_IND_FINAL_RSP – for both the initiator and acceptor, the Extended Flow Specification has been sent to the local controller but neither a response from controller nor the final configuration response has yet been received.
- WAIT_FINAL_RSP – the device received an indication from the Controller accepting the Extended Flow Specification and has sent a positive response. It is waiting for the remote device to send a configuration response.
- WAIT_CONTROL_IND – the device has received a positive response and is waiting for its Controller to accept or reject the Extended Flow Specification.

According to [Section 6.1.1 on page 112](#) and [Section 6.1.2 on page 113](#), the CONFIG state is entered via WAIT_CONFIG substate from either the CLOSED state, the WAIT_CONNECT state, or the WAIT_CONNECT_RSP state. The CONFIG state is left for the OPEN state if both the initiator and acceptor paths complete successfully.

For better overview, separate tables are given: [Table 6.4](#) shows the success transitions; therein, transitions on one of the minimum paths (no previous non-success transitions) are shaded. [Table 6.5 on page 118](#) shows the non-success transitions within the configuration process, and [Table 6.6 on page 118](#) shows further transition cause by events not belonging to the configuration process itself. The following configuration states and transitions are illustrated in [Figure 6.2 on page 129](#).

Previous state	Event	Condition	Action	Next State
WAIT_CONFIG	ConfigureChannel_Req	Standard process	Send L2CAP_ConfigReq	WAIT_CONFIG_REQ_RSP
WAIT_CONFIG	ConfigureChannel_Req	Lockstep process	Send L2CAP_Config Req (Ext Flow Spec plus other options)	WAIT_CONFIG_REQ_RSP
WAIT_CONFIG	L2CAP_ConfigReq	Options acceptable Standard process	Send L2CAP_ConfigRsp (success)	WAIT_SEND_CONFIG
WAIT_CONFIG	L2CAP_ConfigReq	Options acceptable Lockstep process	Send L2CAP_ConfigRsp (pending)	WAIT_SEND_CONFIG
WAIT_CONFIG_REQ_RSP	L2CAP_ConfigReq	Options acceptable	Send L2CAP_ConfigRsp (success)	WAIT_CONFIG_RSP
WAIT_CONFIG_REQ_RSP	L2CAP_ConfigRsp	Remote side accepts options	(continue waiting for configuration request)	WAIT_CONFIG_REQ
WAIT_CONFIG_REQ	L2CAP_ConfigReq	Options acceptable Standard process	Send L2CAP_ConfigRsp (success)	OPEN
WAIT_CONFIG_REQ	L2CAP_ConfigReq	Options acceptable Lockstep process	Send L2CAP_ConfigRsp (pending)	WAIT_IND_FINAL_RSP
WAIT_SEND_CONFIG	ConfigureChannel_Req	-	Send L2CAP_ConfigReq	WAIT_CONFIG_RSP
WAIT_CONFIG_RSP	L2CAP_ConfigRsp	Remote side accepts options Standard process	-	OPEN

Table 6.4: CONFIG state event table

Previous state	Event	Condition	Action	Next State
WAIT_CONFIG_RSP	L2CAP_ConfigRsp	Remote side accepts option Lockstep proc		WAIT_IND_FINAL_RSP
WAIT_IND_FINAL_RSP	ControllerLogical LinkInd	Reject	Send L2CAP_ConfigRsp (fail)	WAIT_CONFIG
WAIT_IND_FINAL_RSP	ControllerLogical LinkInd	Accept	Send L2CAP_ConfigRsp (success)	WAIT_FINAL_RSP
WAIT_IND_FINAL_RSP	L2CAP_ConfigRsp	Remote side fail		WAIT_CONFIG
WAIT_IND_FINAL_RSP	L2CAP_ConfigRsp	Remote side success		WAIT_CONTROL_IND
WAIT_FINAL_RSP	L2CAP_ConfigRsp	Remote side fail		WAIT_CONFIG
WAIT_FINAL_RSP	L2CAP_ConfigRsp	Remote side success		OPEN
WAIT_CONTROL_IND	ControllerLogical LinkInd	Reject	Send L2CAP_ConfigRsp (fail)	WAIT_CONFIG
WAIT_CONTROL_IND	ControllerLogical LinkInd	Accept	Send L2CAP_ConfigRsp (success)	OPEN

Table 6.4: CONFIG state event table (Continued)

Previous state	Event	Condition	Action	Next State
WAIT_CONFIG	L2CAP_ConfigReq	Options not acceptable	Send L2CAP_ConfigRsp (fail)	WAIT_CONFIG
WAIT_CONFIG	L2CAP_ConfigRsp	-	Ignore	WAIT_CONFIG
WAIT_SEND_CONFIG	L2CAP_ConfigRsp	-	Ignore	WAIT_SEND_CONFIG
WAIT_CONFIG_REQ_RSP	L2CAP_ConfigReq	Options not acceptable	Send L2CAP_ConfigRsp (fail)	WAIT_CONFIG_REQ_RSP
WAIT_CONFIG_REQ_RSP	L2CAP_ConfigRsp	Remote side rejects options	Send L2CAP_ConfigReq (new options)	WAIT_CONFIG_REQ_RSP
WAIT_CONFIG_REQ	L2CAP_ConfigReq	Options not acceptable	Send L2CAP_ConfigRsp (fail)	WAIT_CONFIG_REQ
WAIT_CONFIG_REQ	L2CAP_ConfigRsp	-	Ignore	WAIT_CONFIG_REQ
WAIT_CONFIG_RSP	L2CAP_ConfigRsp	Remote side rejects options	Send L2CAP_ConfigReq (new options)	WAIT_CONFIG_RSP

Table 6.5: CONFIG state/substates event table: non-success transitions within configuration process

Previous state	Event	Condition	Action	Next State
CONFIG (any substate)	CloseChannel_Req	Any internal reason to stop	Send L2CAP_Disconnect Req	WAIT_DISCONNECT
CONFIG (any substate)	L2CAP_DisconnectReq	-	Send L2CAP_Disconnect Rsp	CLOSED
CONFIG (any substate)	L2CAP_DisconnectRsp	-	Ignore	CONFIG (remain in sub-state)
CONFIG (any substate)	L2CAP_Data	-	Process the PDU	CONFIG (remain in sub-state)

Table 6.6: CONFIG state/substates event table: events not related to configuration process

Previous state	Event	Condition	Action	Next State
CONFIG (any substate)	L2CAP_ CreateChanReq		Ignore	CONFIG (remain in sub- state)
CONFIG (any substate)	L2CAP_ CreateChanRsp		Ignore	CONFIG (remain in sub- state)
CONFIG (any substate)	L2CAP_ MoveChanReq		Ignore	CONFIG (remain in sub- state)
CONFIG (any substate)	L2CAP_ MoveChanRsp		Ignore	CONFIG (remain in sub- state)
CONFIG (any substate)	L2CAP_ MoveChanConfirm		Ignore	CONFIG (remain in sub- state)
CONFIG (any substate)	L2CAP_ MoveChanCon- firmRsp		Ignore	CONFIG (remain in sub- state)

Table 6.6: CONFIG state/substates event table: events not related to configuration process

Notes:

- Receiving data PDUs (L2CAP_Data) in CONFIG state should be relevant only in case of a transition to a reconfiguration procedure (from OPEN state). Discarding the received data is allowed only in Retransmission Mode and Enhanced Retransmission Mode. Discarding an S-frame is allowed but not recommended. If a S-frame is discarded, the monitor timer will cause a new S-frame to be sent after a time out.
- Indicating a failure in a configuration response does not necessarily imply a failure of the overall configuration procedure; instead, based on the information received in the negative response, a modified configuration request may be triggered.

6.1.5 OPEN state

Event	Condition	Action	Next State
SendData_req	-	Send L2CAP_Data packet according to configured mode	OPEN
ReconfigureChannel_Req	-	Complete outgoing SDU Send L2CAP_ConfigReq	CONFIG (substate WAIT_CONFIG_REQ_RSP)
CloseChannel_Req	-	Send L2CAP_DisconnectReq	WAIT_DISCONNECT
L2CAP_ConnectRsp	-	Ignore	OPEN
L2CAP_ConfigReq	Incoming config options acceptable	Complete outgoing SDU Send L2CAP_ConfigRsp (ok)	CONFIG (substate WAIT_SEND_CONFIG)
L2CAP_ConfigReq	Incoming config options not acceptable	Complete outgoing SDU Send L2CAP_ConfigRsp (fail)	OPEN
L2CAP_DisconnectReq	-	Send L2CAP_DisconnectRsp	CLOSED
L2CAP_DisconnectRsp	-	Ignore	OPEN
L2CAP_Data	-	Process the PDU	OPEN
MoveChannel_Req		Send L2CAP_MoveChanReq	WAIT_MOVE_RSP
L2CAP_CreateChanReq		Ignore	OPEN
L2CAP_CreateChanRsp		Ignore	OPEN
L2CAP_MoveChanReq	Logical Link create/modify is not needed	Send L2CAP_MoveChanRsp (success)	WAIT_MOVE_CONFIRM
L2CAP_MoveChanReq	Need to indicate pending (logical link create/modify is needed)	Send L2CAP_MoveChanRsp (pending)	WAIT_MOVE
L2CAP_MoveChanReq	No resource, not approved	Send L2CAP_MoveChanRsp (refused)	WAIT_MOVE_CONFIRM
L2CAP_MoveChanRsp		Ignore	OPEN

Table 6.7: OPEN state event table

Event	Condition	Action	Next State
L2CAP_MoveChanConfirm		Ignore	OPEN
L2CAP_MoveChanConfirmRsp		Ignore	OPEN

Table 6.7: OPEN state event table

Note: The outgoing SDU shall be completed from the view of the remote entity. Therefore all PDUs forming the SDU shall have been reliably transmitted by the local entity and acknowledged by the remote entity, before entering the configuration state.

6.1.6 WAIT_DISCONNECT state

Event	Condition	Action	Next State
L2CAP_ConnectRsp	-	Ignore	WAIT_DISCONNECT
L2CAP_ConfigReq	-	Send L2CAP_CommandReject with reason Invalid CID	WAIT_DISCONNECT
L2CAP_ConfigRsp	-	Ignore	WAIT_DISCONNECT
L2CAP_DisconnectReq	-	Send L2CAP_DisconnectRsp	CLOSED
L2CAP_DisconnectRsp	-	-	CLOSED
L2CAP_Data	-	Ignore	WAIT_DISCONNECT
L2CAP_CreateChanReq		Ignore	WAIT_DISCONNECT
L2CAP_CreateChanRsp		Ignore	WAIT_DISCONNECT
L2CAP_MoveChanReq		Ignore	WAIT_DISCONNECT
L2CAP_MoveChanRsp		Ignore	WAIT_DISCONNECT
L2CAP_MoveChanConfirm		Ignore	WAIT_DISCONNECT
L2CAP_MoveChanConfirmRsp		Ignore	WAIT_DISCONNECT

Table 6.8: WAIT_DISCONNECT state event table

6.1.7 WAIT_CREATE_RSP state

Event	Condition	Action	Next State
L2CAP_CreateChanReq		Ignore	WAIT_CREATE_RSP
L2CAP_CreateChanRsp	Success indicated in result	Send L2CAP_ConfigReq	CONFIG (substate WAIT_CONFIG)
L2CAP_CreateChanRsp	Result pending		WAIT_CREATE_RSP
L2CAP_CreateChanRsp	Remote side refuses connection		CLOSED
L2CAP_MoveChanReq		Ignore	WAIT_CREATE_RSP
L2CAP_MoveChanRsp		Ignore	WAIT_CREATE_RSP
L2CAP_MoveChanConfirm		Ignore	WAIT_CREATE_RSP
L2CAP_MoveChanConfirmRsp		Ignore	WAIT_CREATE_RSP

Table 6.9: WAIT_CREATE_RSP state event table

6.1.8 WAIT_CREATE state

Event	Condition	Action	Next State
OpenChannelCntrl_Req	-	Ignore	WAIT_CREATE
OpenChannelCntrl_Rsp	Pending connection request is finally acceptable	Send L2CAP_CreateChanRsp (success)	CONFIG
OpenChannelCntrl_Rsp	Pending connection request is finally refused	Send L2CAP_CreateChanRsp (refused)	CLOSED
L2CAP_CreateChanReq	-	Ignore	WAIT_CREATE
L2CAP_CreateChanRsp	-	Ignore	WAIT_CREATE
L2CAP_MoveChanReq	-	Ignore	WAIT_CREATE
L2CAP_MoveChanRsp	-	Ignore	WAIT_CREATE
L2CAP_MoveChanConfirm	-	Ignore	WAIT_CREATE
L2CAP_MoveChanConfirmRsp	-	Ignore	WAIT_CREATE

Table 6.10: WAIT_CREATE state event table

6.1.9 WAIT_MOVE_RSP state

Event	Condition	Action	Next State
L2CAP_MoveChanReq	Prioritization algorithm (remain initiator)	Send L2CAP_MoveChanRsp (collision)	WAIT_MOVE_RSP
L2CAP_MoveChanReq	Prioritization algorithm (become responder) and logical link create/modify is needed)	Send L2CAP_MoveChanRsp (pending)	WAIT_MOVE
L2CAP_MoveChanReq	Prioritization algorithm (become responder) and logical link create/modify is not needed)	Send L2CAP_MoveChanRsp (success)	WAIT_MOVE_CONFIRM
L2CAP_CreateChanRsp	-	Ignore	WAIT_MOVE_RSP
L2CAP_CreateChanReq	-	Ignore	WAIT_MOVE_RSP
L2CAP_MoveChanRsp	Result pending	-	WAIT_MOVE_RSP
L2CAP_MoveChanRsp	Remote side refuses move and/or local side refuses move (create/modify logical link fails)	Send L2CAP_MoveChanConfirm (failure)	WAIT_CONFIRM_RSP
L2CAP_MoveChanRsp	Both remote side and local side accepts move (create/modify logical link succeeds)	Send L2CAP_MoveChanConfirm (success)	WAIT_CONFIRM_RSP
L2CAP_MoveChanConfirm	-	Ignore	WAIT_MOVE_RSP
L2CAP_MoveChanConfirmRsp	-	Ignore	WAIT_MOVE_RSP

Table 6.11: WAIT_MOVE_RSP state event table

6.1.10 WAIT_MOVE state

Event	Condition	Action	Next State
ControllerLogicalLinkInd	Create/modify logical link completed successfully	Send L2CAP_MoveChanRsp (success)	WAIT_MOVE_CONFIRM
ControllerLogicalLinkInd	Create/modify logical link completed with failure	Send L2CAP_MoveChanRsp (refused)	WAIT_MOVE_CONFIRM
L2CAP_CreateChanReq	-	Ignore	WAIT_MOVE
L2CAP_CreateChanRsp	-	Ignore	WAIT_MOVE
L2CAP_MoveChanReq	-	Ignore	WAIT_MOVE
L2CAP_MoveChanRsp	-	Ignore	WAIT_MOVE
L2CAP_MoveChanConfirm	-	Ignore	WAIT_MOVE
L2CAP_MoveChanConfirmRsp	-	Ignore	WAIT_MOVE

Table 6.12: WAIT_MOVE state event table

6.1.11 WAIT_MOVE_CONFIRM state

Event	Condition	Action	Next State
L2CAP_CreateChanReq	-	Ignore	WAIT_MOVE_CONFIRM
L2CAP_CreateChanRsp	-	Ignore	WAIT_MOVE_CONFIRM
L2CAP_MoveChanReq	-	Ignore	WAIT_MOVE_CONFIRM
L2CAP_MoveChanRsp	-	Ignore	WAIT_MOVE_CONFIRM
L2CAP_MoveChanConfirm	Result of move is success	Send L2CAP_MoveChanConfirmRsp	OPEN (on new Controller)
L2CAP_MoveChanConfirm	Result of move is failure	Send L2CAP_MoveChanConfirmRsp	OPEN (on old Controller)
L2CAP_MoveChanConfirmRsp	-	Ignore	WAIT_MOVE_CONFIRM

Table 6.13: WAIT_MOVE_CONFIRM state event table

6.1.12 WAIT_CONFIRM_RSP state

Event	Condition	Action	Next State
L2CAP_CreateChanReq	-	Ignore	WAIT_CONFIRM_RSP
L2CAP_CreateChanRsp	-	Ignore	WAIT_CONFIRM_RSP
L2CAP_MoveChanReq	-	Ignore	WAIT_CONFIRM_RSP
L2CAP_MoveChanRsp	-	Ignore	WAIT_CONFIRM_RSP
L2CAP_MoveChanConfirm	-	Ignore	WAIT_CONFIRM_RSP
L2CAP_MoveChanConfirmRsp	Move succeeded		OPEN (on new Controller)
L2CAP_MoveChanConfirmRsp	Move failed		OPEN (on old Controller)

Table 6.14: WAIT_CONFIRM_RSP state event table

6.2 TIMERS EVENTS

6.2.1 RTX

The Response Timeout eXpired (RTX) timer is used to terminate the channel when the remote endpoint is unresponsive to signaling requests. This timer is started when a signaling request (see [Section 7 on page 131](#)) is sent to the remote device. This timer is disabled when the response is received. If the initial timer expires, a duplicate Request message may be sent or the channel identified in the request may be disconnected. If a duplicate Request message is sent, the RTX timeout value shall be reset to a new value at least double the previous value. When retransmitting the Request message, the context of the same state shall be assumed as with the original transmission. If a Request message is received that is identified as a duplicate (retransmission), it shall be processed in the context of the same state which applied when the original Request message was received.

Implementations have the responsibility to decide on the maximum number of Request retransmissions performed at the L2CAP level before terminating the channel identified by the Requests. The exception is fixed channel CIDs since they can never be terminated. The LE Peripheral may disconnect the link on the expiry of the RTX timer. The decision should be based on the flush timeout of the signaling link. The longer the flush timeout, the more retransmissions may be performed at the physical layer and the reliability of the channel improves, requiring fewer retransmissions at the L2CAP level.

For example, if the flush timeout is infinite, no retransmissions should be performed at the L2CAP level. When terminating the channel, it is not necessary to send a L2CAP_DisconnectReq and enter WAIT_DISCONNECT state. Channels can be transitioned directly to the CLOSED state.

The value of this timer is implementation-dependent but the minimum initial value is 1 second and the maximum initial value is 60 seconds. One RTX timer shall exist for each outstanding signaling request, including each Echo Request. The timer disappears on the final expiration, when the response is received, or the physical link is lost. The maximum elapsed time between the initial start of this timer and the initiation of channel termination (if no response is received) is 60 seconds.

6.2.2 ERTX

The Extended Response Timeout eXpired (ERTX) timer is used in place of the RTX timer when it is suspected the remote endpoint is performing additional processing of a request signal. This timer is started when the remote endpoint responds that a request is pending, e.g., when an L2CAP_ConnectRsp event with a “connect pending” result (0x0001) is received. This timer is disabled when the formal response is received or the physical link is lost. If the initial timer expires, a duplicate Request may be sent or the channel may be disconnected.

If a duplicate Request is sent, the particular ERTX timer disappears, replaced by a new RTX timer and the whole timing procedure restarts as described previously for the RTX timer.

The value of this timer is implementation-dependent but the minimum initial value is 60 seconds and the maximum initial value is 300 seconds. Similar to RTX, there MUST be at least one ERTX timer for each outstanding request that received a Pending response. There should be at most one (RTX or ERTX) associated with each outstanding request. The maximum elapsed time between the initial start of this timer and the initiation of channel termination (if no response is received) is 300 seconds. When terminating the channel, it is not necessary to send a L2CAP_DisconnectReq and enter WAIT_DISCONNECT state. Channels should be transitioned directly to the CLOSED state.

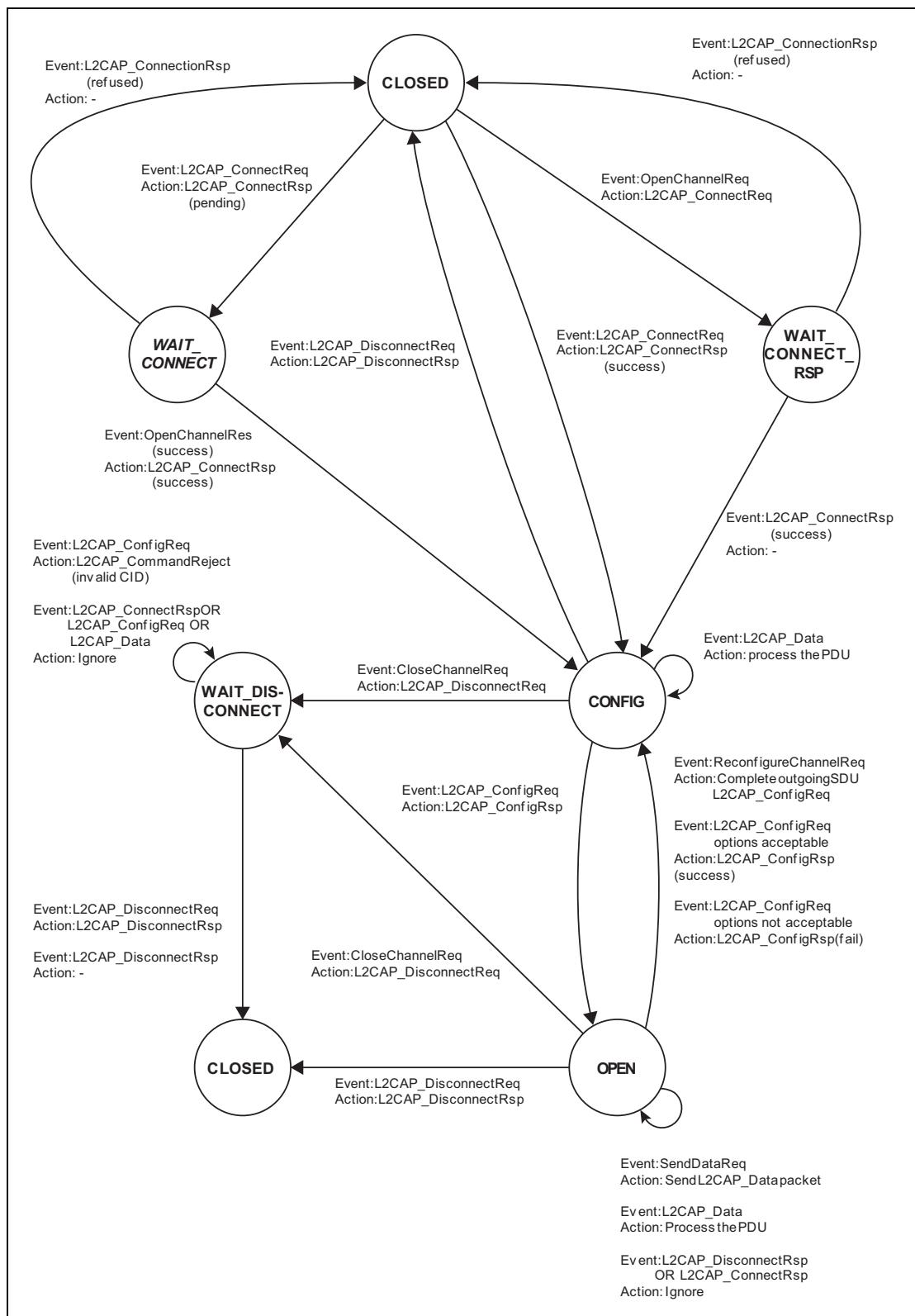


Figure 6.1: States and transitions

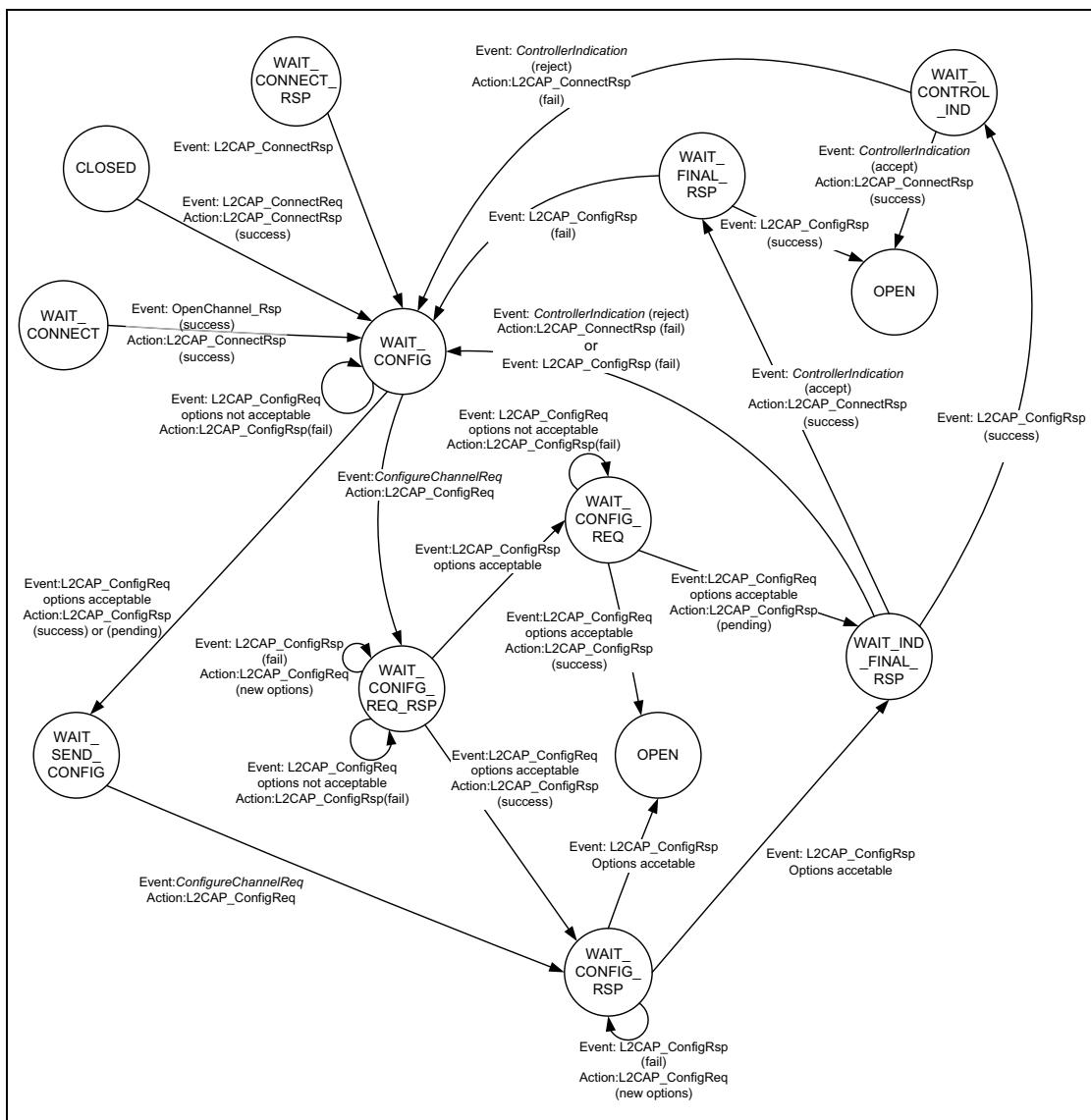


Figure 6.2: Configuration states and transitions

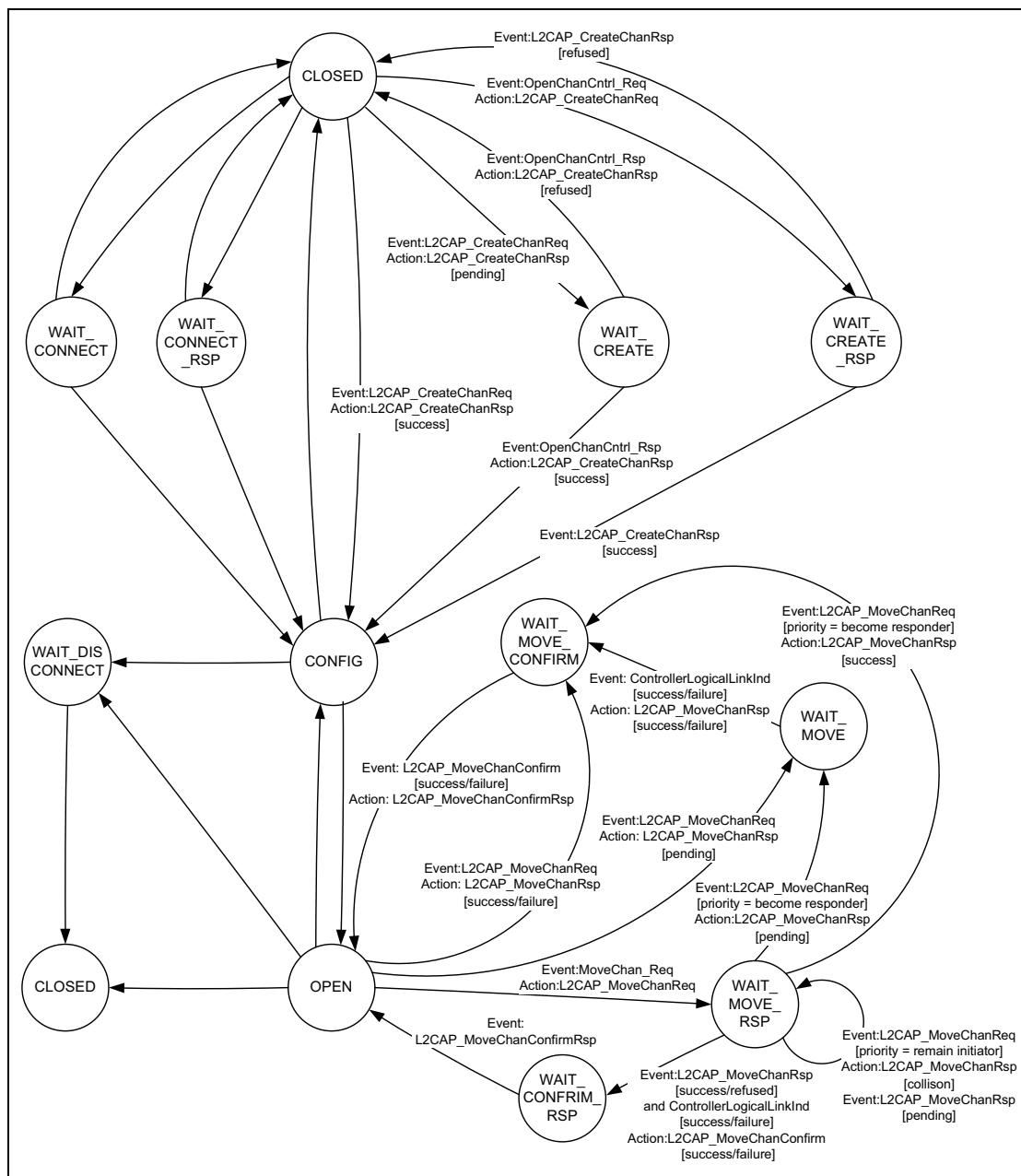


Figure 6.3: States and transitions—AMP enabled operation

Figure 6.3 is based on Figure 6.1 on page 128. Only the new transitions show event/action labels. Transitions without labels are the same as in Figure 6.1.

7 GENERAL PROCEDURES

This section describes the general operation of L2CAP, including the configuration process, the handling and the processing of user data for transportation over the air interface. This section also describes the operation of L2CAP features including the delivery of erroneous packets, the flushing of expired data and operation in connectionless mode, operation collision resolution, aggregation of best effort flow specifications, and prioritizing data over HCI.

Procedures for the flow control and retransmission modes are described in [Section 8 on page 149](#).

7.1 CONFIGURATION PROCESS

Configuration consists of two processes, the Standard process and the Lockstep process. The Lockstep process shall be used if both L2CAP entities support the Extended Flow Specification option otherwise the Standard process shall be used.

For both processes, configuring the channel parameters shall be done independently for both directions. Both configurations may be done in parallel.

Each configuration parameter is one-directional. The configuration parameters describe the non default parameters the device sending a Configuration Request will accept. The configuration request cannot request a change in the parameters the device receiving the request will accept.

If a device needs to establish the value of a configuration parameter the remote device will accept, then it must wait for a configuration request containing that configuration parameter to be sent from the remote device.

The Lockstep process is used when the channel parameters include an Extended Flow Specification option. The Lockstep process can be divided into two phases. In the first phase, each L2CAP entity shall receive an Extended Flow Specification option along with all non-default parameters from its peer L2CAP entity and then present the pair of Extended Flow specifications (local and remote) to its local Controller. In the second phase, each L2CAP entity shall report the result from its local Controller to the peer L2CAP entity. The Lockstep process is described in [Section 7.1.3](#). The Standard process is described in [Section 7.1.4](#).

Both the Lockstep and Standard processes can be abstracted into the initial Request configuration path and a Response configuration path, followed by the reverse direction phase. Reconfiguration follows a similar two-phase process by requiring configuration in both directions.

During a reconfiguration session, all data traffic on the channel should be suspended pending the outcome of the configuration. If an L2CAP entity receives a Configuration Request while it is waiting for a response it shall not block sending the Configuration Response, otherwise the configuration process may deadlock.

7.1.1 Request Path

The Request Path can configure the following:

- requester's incoming MTU
- requester's outgoing flush timeout
- requester's outgoing QoS
- requester's incoming and outgoing flow and error control information
- requester's incoming and outgoing Frame Check Sequence option
- requester's outgoing QoS using Extended Flow Specification option
- requester's incoming Extended Window size option plus incoming and outgoing frame format.

[Table 7.1 on page 132](#) defines the configuration options that may be placed in a Configuration Request.

Parameter	Description
MTU	Incoming MTU information
FlushTO	Outgoing flush timeout ¹
QoS	Outgoing QoS information ¹
RFCMode	Incoming and outgoing Retransmission and Flow Control Mode
FCS	Incoming and outgoing Frame Check Sequence
ExtFlowSpec	Outgoing QoS information ¹
ExtWindow	Incoming Extended Window size plus incoming and outgoing frame format

Table 7.1: Parameters allowed in Request

1. FlushTO, QoS and ExtFlowSpec are considered QoS information. ExtFlowSpec is used instead of FlushTO and QoS when both devices support ExtFlowSpec.

The state machine for the configuration process is described in [Section 6 on page 110](#).

7.1.2 Response Path

The Response Path can configure the following:

- responder's outgoing MTU, that is the remote side's incoming MTU
- remote side's flush timeout
- responder's incoming QoS Flow Specification (remote side's outgoing QoS Flow Specification)
- responder's incoming and outgoing Flow and Error Control information
- responder's incoming QoS Extended Flow Specification (remote side's outgoing QoS Flow Specification)
- responder's outgoing Extended Window size.

For the Standard process, if a request-oriented parameter is not present in the Request message (reverts to last agreed value), the remote side may negotiate for a non-default value by including the proposed value in a negative Response message. [Table 7.2](#) defines the configuration options that may be placed in a Configuration Response.

Parameter	Description
MTU	Outgoing MTU information
FlushTO	Incoming flush timeout ¹
QoS	Incoming QoS information ¹
RFCMode	Incoming and outgoing Retransmission and Flow Control Mode
ExtFlowSpec	Incoming QoS information ¹
ExtWindow	Outgoing Extended Window size

Table 7.2: Parameters allowed in Response

1. FlushTO, QoS and ExtFlowSpec are considered QoS information. ExtFlowSpec is used instead of FlushTO and QoS when both devices support ExtFlowSpec

7.1.3 Lockstep Configuration Process

Configuration Request packets are sent to establish or change the channel parameters including the QoS contract between two L2CAP entities. An Extended Flow Specification option along with any non-default parameters shall be sent in a Configuration Request following the sending or receipt of a Connect Response which accepts an L2CAP Connect Request for the channel being configured. Unlike the Standard process, negotiation involving the sending of multiple Configuration Request packets shall not be performed. In other words, only one Configuration Request packet shall be sent by each L2CAP entity during the Lockstep configuration process. The Lockstep process shall only be

used during channel reconfiguration when an Extended Flow Specification option is present in the Configuration Request packet used for reconfiguration.

The Extended Flow Specification shall be sent for all channels created on AMP-U logical links and shall only be sent for channels created on the ACL-U logical link if both the local and remote L2CAP entities have indicated support for the Extended Flow Specification for BR/EDR in their Extended Features masks. The Extended Features mask shall be obtained via the L2CAP Information Request/Response signaling mechanism (InfoType = 0x0002) prior to the issuance of the L2CAP Configuration Request. If a Configuration Request is sent containing the Extended Flow Specification option then the Quality of Service option and Flush Timeout option shall not be included.

Configuration Response packets shall be sent in reply to Configuration Request packets except when the error condition is covered by a Command Reject response. Although the L2CAP Configuration signaling mechanism allows for the use of wildcards, wildcard values are not supported in the Extended Flow Specification since each parameter represents a property of the traffic in one direction only, and no negotiation is intended to be performed at the L2CAP Configuration level.

The recipient of a Configuration Request shall perform all necessary checks with the Controller to validate that the requested QoS can be granted. In the case of a BR/EDR or BR/EDR/LE Controller the L2CAP layer performs the Controller checks. If HCI is used then the L2CAP entity should check that the requested QoS can be achieved over the HCI transport. In order to perform these checks, the recipient needs to have the QoS parameters for both directions. In order for each side to determine when to perform relevant Controller checks, each side will reply with a result “Pending” (0x0004). If no parameters are sent in the Configuration Response packet with result “Pending” then the parameters sent in the Configuration Request have been accepted without change. This Lockstep procedure results in both L2CAP entities performing the following:

- Receiving a Configuration Request containing the Extended Flow Specification option along with all non-default parameters
- Sending a Configuration Response with any modifications to the Extended Flow Specification option and allowed modifications to non-default parameters (result “Pending”)
- Sending a Configuration Request containing the Extended Flow Specification option along with all non-default parameters
- Receiving a Configuration Response containing any modifications to the Extended Flow Specification option and allowed modifications to non-default parameters (result “Pending”).

The ERTX timer is used when a Configuration Response is received with result “Pending” (see [Section 6.2.2](#)).

If a Configuration Response with result “success” is received before a Configuration Response with result “pending” is received the recipient shall disconnect the channel. This is a violation of the Lockstep configuration process.

If a device sends an Extended Flow Specification option in a Configuration Request with service type “Best Effort” and receives a Configuration Request with service type “Guaranteed,” the channel shall be disconnected. If a device sends an Extended Flow Specification in a Configuration Request with type “Guaranteed” and receives a Configuration Request with service type “Best Effort,” the channel shall be disconnected.

If the service type is “Best Effort” then values for certain parameters may be sent in a Configuration Response with result “Pending” to indicate the maximum bandwidth the sender of the Configuration Response is capable to receive. The values sent in a Configuration Response with result “Pending” and service type Best Effort shall be in accordance with [Table 7.3](#).

Parameter	Changes permitted by responder
Maximum SDU Size	May decrease only
SDU Inter-arrival Time	May increase only
Access Latency	None
Flush Timeout	None

Table 7.3: Permitted parameter in L2CAP Configuration Response for Service Type “Best Effort”

After the Configuration Response is received with result “Pending,” L2CAP may issue the necessary checks with the Controller.

If the Controller cannot support the Extended Flow Specifications with service type “Guaranteed,” then the recipient of the Configuration Request shall send a Configuration Response indicating a result code of “Failure - flow spec rejected” (0x0005). If the Controller indicates that it can support the Extended Flow Specifications, then the recipient of the Configuration Request shall send a Configuration Response with result code of “Success” (0x0000) with no parameters.

If the Result of the Configuration Response is Failure (0x0005) for service type “Guaranteed” then an Extended Flow Specification option may be sent in the Configuration Response. The Extended Flow Specification parameters sent in the Configuration Response may be changed to reflect a QoS level that would be acceptable, but shall only be changed in accordance with [Table 7.4](#).

If an L2CAP Configuration Response is received containing the Extended Flow Specification option with the same values sent earlier, the upper layer shall be notified of the error.

Parameter	Changes permitted by responder
Maximum SDU Size	None
SDU Inter-arrival Time	None
Access Latency	May decrease only
Flush Timeout	May decrease only (unless set to 0xFFFFFFFF, in which case no change is permitted)

Table 7.4: Permitted parameter changes in L2CAP Configuration Response for Service Type "Guaranteed"

7.1.4 Standard Configuration Process

For the Standard process the following general procedure shall be used for each direction:

1. Local device informs the remote device of the parameters that the local device will accept using a Configuration Request.
2. Remote device responds, agreeing or disagreeing with these values, including the default ones, using a Configuration Response.
3. The local and remote devices repeat steps (1) and (2) until agreement on all parameters is reached.

The decision on the amount of time (or messages) spent configuring the channel parameters before terminating the configuration is left to the implementation, but it shall not last more than 120 seconds.

There are two types of configuration parameters: negotiable and non-negotiable.

Negotiable parameters are those that a remote side receiving a Configuration Request can disagree with by sending a Configuration Response with the Unacceptable Parameters (0x0001) result code, proposing new values that can be accepted. Non-negotiable parameters are only informational and the recipient of a Configuration Request cannot disagree with them but can provide adjustments to the values in a positive Configuration Response. [Section 5](#) identifies each parameter as negotiable or non-negotiable. Note: MTU is non-negotiable but can be rejected if a value lower than the mandated minimum is proposed (See [Section 5.1](#)).

The following rules shall be used for parameter negotiation in the Request Path:

1. An L2CAP entity shall send at least one Configuration Request packet as part of initial configuration or reconfiguration. If all default or previously agreed values are acceptable, a Configuration Request packet with no options shall be sent.
2. When an L2CAP entity receives a positive Configuration Response from the remote device it shall consider all configuration parameters explicitly

contained in the Configuration Request along with the default and previously agreed values not explicitly contained in the Configuration Request as accepted by the remote device.

3. When an L2CAP entity receives a negative Configuration Response and sends a new Configuration Request it shall include all the options it sent in the previous Configuration Request with the same values except the negotiable options that were explicitly rejected in the negative Configuration Response which will have new values. Note: the resending of all the options provides backwards compatibility with remote implementations that don't assume rule 4 when responding.
4. Negotiable options not included in a negative Configuration Response are considered accepted by the remote device. Note: for backwards compatibility if non-negotiable options are included in a negative Configuration Response, they should be processed as if the response was positive.

The following rules shall be used for parameter negotiation in the Response Path:

1. A positive Configuration Response accepts the values of all configuration parameters explicitly contained in the received Configuration Request and the default and previously agreed values not explicitly provided.
2. An L2CAP Entity shall send a negative Configuration Response to reject negotiable parameter values that are unacceptable, be it values explicitly provided in the received Configuration Request or previously agreed or default values. The rejected parameters sent in a negative Configuration Response shall have values that are acceptable to the L2CAP entity sending the negative Configuration Response.
3. All negotiable options being rejected shall be rejected in the same negative Configuration Response.
4. The only options allowed in a negative Configuration Response are the negotiable options being rejected. No wildcards or adjustments to non-negotiable options shall be in a negative Configuration Response.
5. Negotiable options not included in the negative Configuration Response are considered accepted.

7.2 FRAGMENTATION AND RECOMBINATION

Fragmentation is the breaking down of PDUs into smaller pieces for delivery from L2CAP to the lower layer. Recombination is the process of reassembling a PDU from fragments delivered up from the lower layer. Fragmentation and Recombination may be applied to any L2CAP PDUs.

7.2.1 Fragmentation of L2CAP PDUs

An L2CAP implementation may fragment any L2CAP PDU for delivery to the lower layers. If L2CAP runs directly over the Controller without HCI, then an implementation may fragment the PDU into multiple Controller packets for transmission over the air. If L2CAP runs above the HCI, then an implementation may send HCI transport sized fragments to the Controller. All L2CAP fragments associated with an L2CAP PDU shall be processed for transmission by the Controller before any other L2CAP PDU for the same logical transport shall be processed.

For example, in the BR/EDR controller the two LLID bits defined in the first octet of baseband payload (also called the frame header) are used to signal the start and continuation of L2CAP PDUs. LLID shall be 10b for the first segment in an L2CAP PDU and 01b for a continuation segment. An illustration of fragmentation for a BR/EDR controller is shown in [Figure 7.1 on page 138](#). An example of how fragmentation might be used in a device with HCI is shown in [Figure 7.2 on page 139](#).

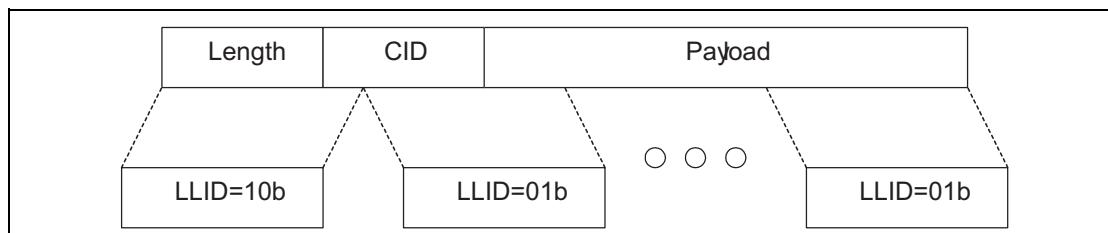


Figure 7.1: L2CAP fragmentation in a BR/EDR Controller

Note: The BR/EDR link controller is able to impose a different fragmentation on the PDU by using “start” and “continuation” indications as fragments are translated into baseband packets. Thus, both L2CAP and the BR/EDR link controller use the same mechanism to control the size of fragments.

7.2.2 Recombination of L2CAP PDUs

Controllers will attempt to deliver packets in sequence and without errors. Recombination of fragments may occur in the Controller but ultimately it is the responsibility of L2CAP to reassemble PDUs and SDUs and to check the length field of the SDUs. As the Controller receives packet fragments, it either signals the L2CAP layer on the arrival of each fragment, or accumulates a number of fragments (before the receive buffer fills up or a timer expires) before passing packets to the L2CAP layer.

An L2CAP implementation shall use the length field in the header of L2CAP PDUs, see [Section 3 on page 45](#), as a consistency check and shall discard any L2CAP PDUs that fail to match the length field. If channel reliability is not needed, packets with invalid lengths may be silently discarded. For reliable channels using Basic mode, an L2CAP implementation shall indicate to the upper layer that the channel has become unreliable. Reliable channels are defined by having an infinite flush timeout value as specified in [Section 5.2 on page 91](#). For higher data integrity L2CAP should be operated in the Enhanced Retransmission Mode.

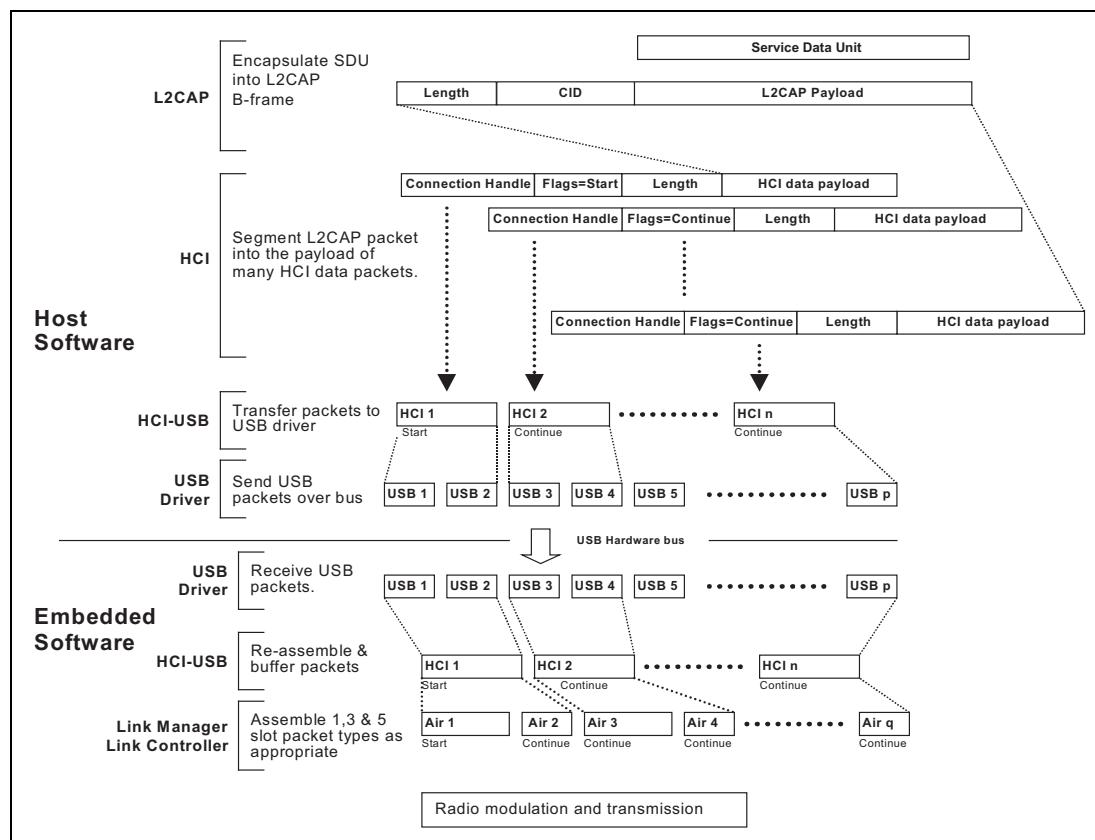


Figure 7.2: Example of fragmentation processes in a device with a BR/EDR Controller and USB HCI transport

7.3 ENCAPSULATION OF SDUs

All SDUs are encapsulated into one or more L2CAP PDUs.

In Basic L2CAP mode, an SDU shall be encapsulated with a minimum of L2CAP protocol elements, resulting in a type of L2CAP PDU called a Basic Information Frame (B-frame).

Segmentation and Reassembly operations are only used in Enhanced Retransmission mode, Streaming mode, Retransmission mode, and Flow Control mode. SDUs may be segmented into a number of smaller packets called SDU segments. Each segment shall be encapsulated with L2CAP protocol elements resulting in an L2CAP PDU called an Information Frame (I-frame).

The maximum size of an SDU segment shall be given by the Maximum PDU Payload Size (MPS). The MPS parameter may be exported using an implementation specific interface to the upper layer.

Note that this specification does not have a normative service interface with the upper layer, nor does it assume any specific buffer management scheme of a host implementation. Consequently, a reassembly buffer may be part of the upper layer entity. It is assumed that SDU boundaries shall be preserved between peer upper layer entities.

7.3.1 Segmentation of L2CAP SDUs

In Flow Control, Streaming, or Retransmission modes, incoming SDUs may be broken down into segments, which shall then be individually encapsulated with L2CAP protocol elements (header and checksum fields) to form I-frames. I-frames are subject to flow control and may be subject to retransmission procedures. The header carries a 2 bit SAR field that is used to identify whether the I-frame is a 'start', 'end' or 'continuation' packet or whether it carries a complete, unsegmented SDU. [Figure 7.3 on page 140](#) illustrates segmentation and fragmentation.

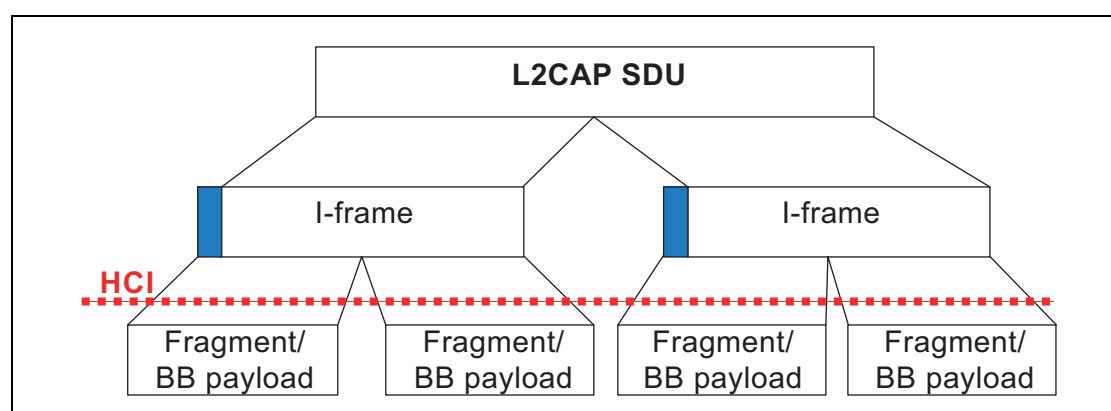


Figure 7.3: Segmentation and fragmentation of an SDU in a BR/EDR Controller

7.3.2 Reassembly of L2CAP SDUs

The receiving side uses the SAR field of incoming 'I-frames' for the reassembly process. The L2CAP SDU length field, present in the "start of SDU" I-frame, is an extra integrity check, and together with the sequence numbers may be used to indicate lost L2CAP SDUs to the application. [Figure 7.3 on page 140](#) illustrates segmentation and fragmentation.

7.3.3 Segmentation and fragmentation

[Figure 7.4 on page 141](#) illustrates the use of segmentation and fragmentation operations to transmit a single SDU. Note that while SDUs and L2CAP PDUs are transported in peer-to-peer fashion, the fragment size used by the Fragmentation and Recombination routines is implementation specific and may not be the same in the sender and the receiver. The over-the-air sequence of packet fragments as created by the sender is common to both devices.

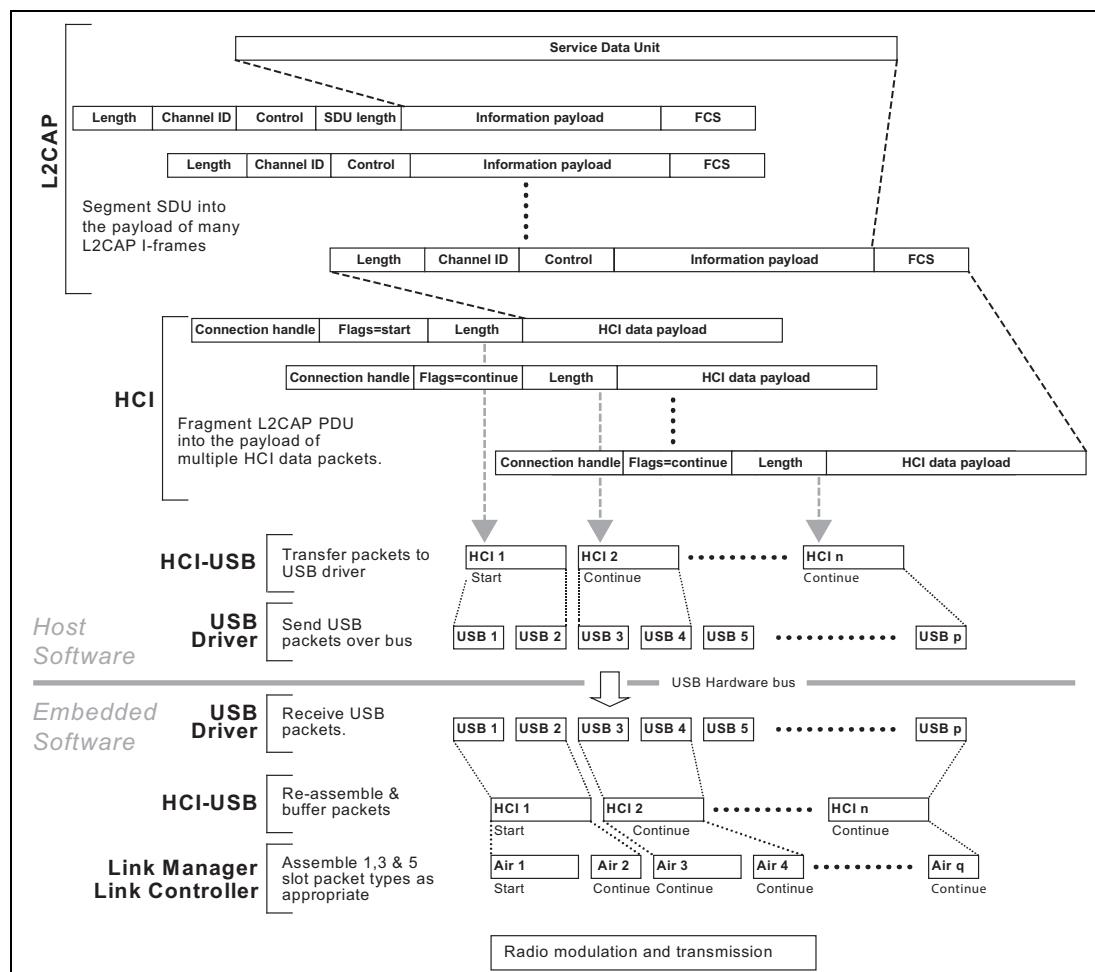


Figure 7.4: Example of segmentation and fragment processes in a device with a BR/EDR Controller and USB HCI Transport¹

1. For simplicity, the stripping of any additional HCI and USB specific information fields prior to the creation of the baseband packets (Air_1, Air_2, etc.) is not shown in the figure.

7.4 DELIVERY OF ERRONEOUS L2CAP SDUS

Some applications may require corrupted or incomplete L2CAP SDUs to be delivered to the upper layer. If delivery of erroneous L2CAP SDUs is enabled, the receiving side will pass information to the upper layer on which parts of the L2CAP SDU (i.e., which L2CAP frames) have been lost, failed the error check, or passed the error check. If delivery of erroneous L2CAP SDUs is disabled, the receiver shall discard any L2CAP SDU segment with any missing frames or any frames failing the error checks. L2CAP SDUs whose length field does not match the actual frame length shall also be discarded.

7.5 OPERATION WITH FLUSHING ON ACL-U LOGICAL LINKS

In the L2CAP configuration using either the Flush Timeout option or the Extended Flow Specification option, the Flush timeout may be set separately per L2CAP channel, but in the BR/EDR baseband, the flush mechanisms operate per ACL logical transport.

When there is more than one L2CAP channel mapped to the same ACL logical transport, the automatic flush time-out does not discriminate between L2CAP channels. The automatic flush time-out also applies to unicast data sent via the L2CAP connectionless channel. The exception is packets marked as non-automatically-flushable via the Packet_Boundary_Flag in the HCI ACL Data Packet (see [Section 1.1 on page 29](#)). The automatic flush time-out flushes a specific automatically-flushable L2CAP PDU. The HCI Flush command flushes all outstanding L2CAP PDUs for the ACL logical transport including L2CAP PDUs marked as non-automatically-flushable. Therefore, care has to be taken when using the Automatic Flush Time-out and the HCI Flush command. The HCI Enhanced Flush command should be used instead.

All packets associated with a reliable connection shall be marked as non-automatically-flushable (if it is mapped to an ACL logical transport with a finite automatic flush time-out) or L2CAP Enhanced Retransmission mode or Retransmission mode shall be used. In Enhanced Retransmission mode or Retransmission mode, loss of flushed L2CAP PDUs on the channel is detected by the L2CAP ARQ mechanism and they are retransmitted. Note that L2CAP Enhanced Retransmission mode or Retransmission mode might be used for other purposes such as the need for a residual error rate that is much smaller than the baseband can deliver. In this situation L2CAP Enhanced Retransmission mode or Retransmission mode and the Non-Flushable Packet Boundary Flag feature can be used at the same time.

If it is desired to send unicast data via the L2CAP connectionless channel which is not subject to automatic flushing, then the data should be marked as non-automatically flushable if it is mapped to an ACL logical transport with a finite automatic flush time-out. Unicast data sent via the L2CAP connectionless channel may be marked flushable.

There is only one automatic flush time-out setting per ACL logical transport. Therefore, all time bounded L2CAP channels on an ACL logical transport with an automatic flush time-out setting should configure the same flush time-out value at the L2CAP level. The flush time-out setting for the ACL logical transport also applies to unicast data sent via the L2CAP connectionless channel which are marked flushable.

If Automatic Flush Time-out is used, then it should be taken into account that it only flushes one L2CAP PDU. If one PDU has timed out and needs flushing, then other automatically-flushable packets on the same logical transport are also likely to need flushing. Therefore, flushing can be handled by the HCI Enhanced Flush command so that all outstanding automatically-flushable L2CAP PDUs are flushed.

When both reliable and isochronous data is to be sent over the same ACL logical transport, an infinite Automatic Flush Time-out can be used. In this case the isochronous data is flushed using the HCI Enhanced Flush command with Packet_Type set to “Automatically-Flushable Only,” thus preserving the reliable data.

7.6 CONNECTIONLESS DATA CHANNEL

In addition to connection-oriented channels, L2CAP also provides a connectionless channel. Data sent on the connectionless channel shall only be sent over the BR/EDR radio. The connectionless channel allows broadcast transmissions from the master device to all members of the piconet or unicast transmissions from either a master or slave to a single remote device. Data sent through the connectionless channel is sent in a best-effort manner. The connectionless channel provided by L2CAP has no quality of service.

While L2CAP itself does not provide retransmission for data sent via the connectionless channel, the baseband does provide an ARQ scheme for unicast data (see [\[vol.2, part B\] Section 7.6 on page 150](#)). If a higher degree of reliability is desired for unicast data sent via the connectionless L2CAP channel than is provided by the baseband ARQ scheme then an ARQ scheme should be implemented at a higher layer.

No acknowledgement is provided by the baseband for broadcast transmissions and hence broadcast transmissions sent via the connectionless L2CAP channel are unreliable and hence might or might not reach each member of the piconet.

The receiving L2CAP entity may silently discard data received via the connectionless L2CAP channel if the data packet is addressed to a PSM and no application is registered to receive data on that PSM.

Note that L2CAP does not provide flow control for either connection-oriented L2CAP channels operating in Basic mode or for traffic on the connectionless L2CAP channel. Hence if data received by L2CAP is not accepted by the target

applications in a timely manner, congestion can occur in a receiving L2CAP implementation. For connection-oriented channels, the receiving L2CAP entity may elect to close a channel if the target application for that channel does not accept the data in a timely manner. Since this option is not available for unicast data received via the connectionless L2CAP channel, the receiving L2CAP entity may instead elect to de-register the application receiving the data or to disconnect the underlying physical link. An application shall be notified if it is de-registered. If the underlying physical link is disconnected then all applications utilizing that physical link shall be notified.

Unicast data shall only be sent via the connectionless L2CAP channel to a remote device if the remote device indicates support for Unicast Connectionless Data Reception in the L2CAP Extended Features Mask.¹ The L2CAP Extended Features Mask should be retrieved using the L2CAP Information Request signal to determine if Unicast Connectionless Data Reception is supported. Optionally, support for Unicast Connectionless Data Reception can be inferred from information obtained via a SDP or EIR. For example, if a service is found via SDP or EIR that is known to mandate the support of UCD, then it may be assumed that the device indicating support for the service supports Unicast Connectionless Data Reception.

Unicast data sent via the L2CAP connectionless channel are subject to automatic flushing and hence the packet boundary flags should be set appropriately if the controller supports the Packet Boundary Flag feature. See [Section 7.5](#) for further details. Since the receiving L2CAP entity has no mechanism to enable it to know whether received packets were originally marked as flushable or non-flushable on the transmitting device, the receiving L2CAP entity should treat all unicast packets received via the connectionless L2CAP packet as non-flushable.

If encryption is required for a given profile, then the profile or application shall ensure that authentication is performed and encryption is enabled prior to sending any unicast data on the connectionless L2CAP channel by utilizing Security Mode 4 as defined in GAP ([\[Vol. 3\] Part C, Section 5.2.2](#)). There is no mechanism provided in this specification to prevent reception of unencrypted data on the connectionless L2CAP channel. An application which requires received data to be encrypted should ignore any unencrypted data it may receive over the connectionless channel.

Broadcast transmissions to the connectionless channel are sent with the broadcast LT_ADDR and hence may be received by any of the slaves in the piconet. If it is desirable to restrict the reception of the transmitted data to only a subset of the slaves in the piconet, then higher level encryption may be used to support private communication.

1. Note that the Unicast Connectionless Data Reception bit in the L2CAP Extended Features Mask does not in any way indicate support or lack of support for reception of broadcast data on the connectionless L2CAP channel even though both broadcast data and unicast data are sent and received using the same CID (0x0002). For historical reasons, there is no bit to indicate support for sending or receiving of broadcast data on the connectionless L2CAP channel.

The master will not receive transmissions broadcast on the connectionless channel. Therefore, higher layer protocols must loopback any broadcast data traffic being sent to the master device if required.

The connectionless data channel shall not be used with Enhanced Retransmission mode, Retransmission Mode or Flow Control Mode.

7.7 OPERATION COLLISION RESOLUTION

When two devices request the same operation by sending a request packet with the same code, a collision can occur. Some operations require collision resolution. The description of each operation in [Section 4](#) will indicate if collision resolution is required. Both devices must know which request will be rejected. The following algorithm shall be used by both devices to determine which request to reject.

1. Set i=0 (representing the least significant octet of the BD_ADDR).
2. Compare the octet[i] of the BD_ADDR of both devices. If the octets are not equal go to step 4.
3. Increment i by 1. Go to step 2.
4. The device with the larger BD_ADDR octet shall reject the request from the other device.

7.8 AGGREGATING BEST EFFORT EXTENDED FLOW SPECIFICATIONS

There shall only be one pair of Extended Flow Specifications per logical link and only one “Best Effort” logical link per physical link. Thus, all Best Effort L2CAP channels running over the same physical link are aggregated into one logical link and the Best Effort Flow Specifications describing the traffic for each channel running over an AMP-U logical link shall be aggregated to a single pair of Extended Flow Specifications (one for each direction). The purpose of a Best Effort Flow Specification is to specify the maximum data rate of the data that can be delivered to the Controller from the host and/or taken from the Controller by the host in order to help the Controller to not over allocate bandwidth on the physical link. Since L2CAP performs admission control for BR/EDR and BR/EDR/LE Controllers it is not necessary to aggregate Best Effort channels for BR/EDR and BR/EDR/LE Controllers.

The two Extended Flow Specification parameters that are affected by Best Effort are Maximum SDU size and SDU Inter-arrival Time. A data rate is determined by dividing the Maximum SDU size by the SDU Inter-arrival time giving a value in bytes per second. The important value is the data rate and not the Maximum SDU size or SDU Inter-arrival time.

Best Effort aggregation shall occur when performing any of the three operations:

- Configuring a new Best Effort channel
- Disconnecting a Best Effort channel
- Moving a Best Effort a channel

During L2CAP channel configuration, the Extended Flow Specifications are exchanged and each L2CAP entity is allowed to modify the other's flow specification (lower the values). If a Best Effort logical link already exists (meaning there is at least one other Best Effort channel) then the new flow specifications shall be aggregated with the existing Best Effort aggregate before sending it to the Controller. When a Best Effort channel is moved or disconnected a new aggregate shall be created. If the new aggregate is different from the previous aggregate it shall be given to the Controller. The following guidelines are provided for aggregating Best Effort Flow specification parameters. Keep in mind that the data rate is the important value so the description shows how to aggregate data rates.

1. The value 0xFFFF is used for Maximum SDU Size and the value 0xFFFFFFFF is used for SDU Inter-arrival time to indicate that the actual values are unknown and maximum bandwidth is assumed. Therefore, if these values exist in an Extended Flow Specification then the aggregate becomes 0xFFFF and 0xFFFFFFFF for Maximum SDU Size and SDU Inter-arrival time respectively.
2. In general it is assumed that data rates provided by the local applications can be added together. For example if one application indicates a data rate of 100 bytes/sec and a second application indicates a data rate of 100 bytes/sec then the aggregate data rate from the two applications is 200 bytes/sec. This is assumed for both directions.
3. The adjustments (sent in a Configuration Response) made by the remote device to an outgoing flow specification are assumed to be made on behalf of the application. These adjustments reduce the data rate of the outgoing flow specification. The reduced data rates can be added to data rates from other applications.
4. The final data rates shall be converted back into Maximum SDU Size and SDU Inter-arrival time to be passed to the Controller. Controllers may have a maximum PDU size or a "preferred" PDU size. When possible the "preferred" size should be used for the Maximum SDU size and the SDU Inter-arrival time should be set to a value that achieves the desired data rate. If the data rate cannot be achieved by using the "preferred" PDU size then an integer multiple of the preferred PDU size should be used.

7.9 PRIORITIZING DATA OVER HCI

In order for guaranteed channels to meet their guarantees, L2CAP should prioritize traffic over the HCI transport in devices that support HCI. Packets for Guaranteed channels should receive higher priority than packets for Best Effort channels.

7.10 SUPPORTING EXTENDED FLOW SPECIFICATION FOR BR/EDR AND BR/EDR/LE CONTROLLERS

If both the local L2CAP entity and the remote L2CAP entity indicate support for Extended Flow Specification for BR/EDR in the Extended Feature Mask then all channels created between the two devices shall send an Extended Flow Specification option and shall use the Lockstep configuration procedure. In addition all channels shall use Enhanced Retransmission mode or Streaming mode. If one or both L2CAP entities do not indicate support for Extended Flow Specification for BR/EDR in the Extended Feature Mask then the Lockstep configuration procedure shall not be used and the Extended Flow Specification option shall not be sent for channels created over the ACL-U logical link between the two devices.

The L2CAP entity shall perform admission control for Guaranteed channels during the Lockstep configuration procedure (see [Section 7.1.3](#)). Admission control is performed to determine if the requested Guaranteed QoS can be achieved by the BR/EDR or BR/EDR/LE Controller over an ACL-U logical link without compromising existing Guaranteed channels running on the Controller. If HCI is used then the L2CAP entity shall also verify that the QoS can be achieved over the HCI transport.

In performing admission control the L2CAP layer shall reject a Guaranteed QoS request that causes at least one of the following rules to be violated.

1. The total guaranteed data rate in both directions shall not exceed two times the highest Symmetric Max of an ACL-U logical link over the BR/EDR or BR/EDR/LE Controller (see [\[vol.2, part B\] Section 6.7 on page 141](#)). For example for a Basic Rate Controller the highest Symmetric Max Rate is 433.9kb/s (DH5) from [Table 6.9, “ACL packets,” on page 141 in vol. 2 Part B](#). Two times that value is 867.8kb/s.
2. The total guaranteed data rate in any one direction shall not exceed the highest Asymmetric Max Rate of an ACL-U logical link (see [\[Vol. 2\] Part B, Section 6.7](#)). For example the highest Asymmetric Max Rate for Basic Rate is 723.2kb/s (DH5 packet) from [Table 6.9, “ACL packets,” on page 141 in vol. 2 Part B](#).

The data rate of a Guaranteed channel is calculated by dividing the Maximum SDU size in an Extended Flow Specification by the SDU Inter-arrival time. Total guaranteed data rate in both directions is calculated by taking the sum of the

data rates in both directions for all existing Guaranteed channels plus the data rate in both directions of the requested Guaranteed channel (i.e. data rates from the both outgoing and incoming Extended Flow Specifications). Total guaranteed data rate for one direction is calculated by taking the sum of the data rates in one direction for all existing Guaranteed channels plus the data rate in the same direction of the requested Guaranteed channel.

An L2CAP entity should use additional information for admission control that may result in a Guaranteed QoS request being rejected even if none of the rules are violated. This allows the L2CAP entity to account for things such as CQDDR, SCO/eSCO channels, LMP traffic, page scanning, etc.

In order to meet the access latency and/or data rate required by a Guaranteed channel, the L2CAP entity should:

- a) Prioritize traffic over the HCI transport in devices that support HCI.
- b) Give conformant packets for Guaranteed channels higher priority than packets for Best Effort channels. Packets for Best Effort channels should have higher priority than non-conformant packets for Guaranteed channels. (See [Section 5.6](#) for a discussion of non-conformant and conformant traffic).
- c) Utilize the SAR feature to restrict the PDU sizes of Best Effort SDUs so that they do not prevent the Controller from sending the SDUs of the Guaranteed channel within the time periods required by its Extended Flow Specification.

When a channel is reconfigured the Lockstep configuration process is only used when an Extended Flow Specification option is present in the Configuration Request packet as described in [Section 7.1.3](#).

8 PROCEDURES FOR FLOW CONTROL AND RETRANSMISSION

When Enhanced Retransmission mode, Streaming mode, Flow Control mode, or Retransmission mode is used, the procedures defined in this chapter shall be used, including the numbering of information frames, the handling of SDU segmentation and reassembly, and the detection and notification of frames with errors. Retransmission mode and Enhanced Retransmission mode also allow the sender to resend frames with errors on request from the receiver.

8.1 INFORMATION RETRIEVAL

Before attempting to configure Enhanced Retransmission mode, Streaming mode, Flow Control mode, or Retransmission mode on a channel, support for the suggested mode shall be verified by performing an information retrieval for the “Extended features supported” information type (0x0002). If the information retrieval is not successful or the “Extended features mask” bit is not set for the wanted mode, the mode shall not be suggested in a configuration request.

8.2 FUNCTION OF PDU TYPES FOR FLOW CONTROL AND RETRANSMISSION

Two frame formats are defined for Enhanced Retransmission mode, Streaming Mode, Flow Control Mode, and Retransmission mode (see [Section 3.3 on page 47](#)). The I-frame is used to transport user information instead of the B-frame. The S-frame is used for supervision.

8.2.1 Information frame (I-frame)

I-frames are sequentially numbered frames containing information fields. I-frames also include some of the functionality of RR frames (see below).

8.2.2 Supervisory Frame (S-frame)

The S-frame is used to control the transmission of I-frames. For Retransmission mode and Flow Control mode, the S-frame has two formats: Receiver Ready (RR) and Reject (REJ). A description of how S-frames are used in Enhanced Retransmission mode is given in [Section 8.6.1](#). S-frames are not used in Streaming mode. The following description of S-frames only applies to Retransmission mode and Flow Control mode.

8.2.2.1 Receiver Ready (RR)

The receiver ready (RR) S-frame is used to:

1. Acknowledge I-frames numbered up to and including ReqSeq - 1.
2. Enable or disable retransmission of I-frames by updating the receiver with the current status of the Retransmission Disable Bit.

The RR frame has no information field.

8.2.2.2 Reject (REJ)

The reject (REJ) S-frame is used to request retransmission of all I-frames starting with the I-frame with TxSeq equal to ReqSeq specified in the REJ. The value of ReqSeq in the REJ frame acknowledges I-frames numbered up to and including ReqSeq - 1. I-frames that have not been transmitted, shall be transmitted following the retransmitted I-frames.

When a REJ is transmitted, it triggers a REJ Exception condition. A second REJ frame shall not be transmitted until the REJ Exception condition is cleared. The receipt of an I-frame with a TxSeq equal to the ReqSeq of the REJ frame clears the REJ Exception. The REJ Exception condition only applies to traffic in one direction. Note: this means that only valid I-frames can be rejected.

8.3 VARIABLES AND SEQUENCE NUMBERS

The sending peer uses the following variables and Sequence numbers:

- TxSeq – the send Sequence number used to sequentially number each new I-frame transmitted.
- NextTxSeq – the Sequence number to be used in the next new I-frame transmitted.
- ExpectedAckSeq – the Sequence number of the next I-frame expected to be acknowledged by the receiving peer.

The receiving peer uses the following variables and Sequence numbers:

- ReqSeq – The Sequence number sent in an acknowledgement frame to request transmission of I-frame with TxSeq = ReqSeq and acknowledge receipt of I-frames up to and including (ReqSeq-1).
- ExpectedTxSeq – the value of TxSeq expected in the next I-frame.
- BufferSeq – When segmented I-frames are buffered this is used to delay acknowledgement of received I-frame so that new I-frame transmissions do not cause buffer overflow.

All variables have the range 0 to 63. Arithmetic operations on state variables (NextTXSeq, ExpectedTxSeq, ExpectedAckSeq, BufferSeq) and sequence numbers (TxSeq, ReqSeq) contained in this document shall be taken modulo 64.

To perform Modulo 64 operation on negative numbers multiples of 64 shall be added to the negative number until the result becomes non-negative.

8.3.1 Sending peer

8.3.1.1 *Send sequence number TxSeq*

I-frames contain TxSeq, the send sequence number of the I-frame. When an I-frame is first transmitted, TxSeq is set to the value of the send state variable NextTXSeq. TxSeq is not changed if the I-frame is retransmitted.

8.3.1.2 *Send state variable NextTXSeq*

The CID sent in the information frame is the destination CID and identifies the remote endpoint of the channel. A send state variable NextTxSeq shall be maintained for each remote endpoint. NextTxSeq is the sequence number of the next in-sequence I-frame to be transmitted to that remote endpoint. When the link is created NextTXSeq shall be initialized to 0.

The value of NextTxSeq shall be incremented by 1 after each in-sequence I-frame transmission, and shall not exceed ExpectedAckSeq by more than the maximum number of outstanding I-frames (TxWindow). The value of TxWindow shall be in the range 1 to 32 for Retransmission mode and Flow Control mode. The value of TxWindow shall be in the range of 1 to 63 for Enhanced Retransmission mode.

8.3.1.3 *Acknowledge state variable ExpectedAckSeq*

The CID sent in the information frame is the destination CID and identifies the remote endpoint of the channel. An acknowledge state variable ExpectedAckSeq shall be maintained for each remote endpoint.

ExpectedAckSeq is the sequence number of the next in-sequence I-frame that the remote receiving peer is expected to acknowledge. (ExpectedAckSeq – 1 equals the TxSeq of the last acknowledged I-frame). When the link is created ExpectedAckSeq shall be initialized to 0.

Note that if the next acknowledgement acknowledges a single I-frame then its ReqSeq will be expectedAckSeq + 1.

If a valid ReqSeq is received from the peer then ExpectedAckSeq is set to ReqSeq. A valid ReqSeq value is one that is in the range $\text{ExpectedAckSeq} \leq \text{ReqSeq} \leq \text{NextTxSeq}$.

Note: The comparison with NextTXSeq must be \leq in order to handle the situations where there are no outstanding I-frames.

These inequalities shall be interpreted in the following way: ReqSeq is valid, if and only if $(\text{ReqSeq}-\text{ExpectedAckSeq}) \bmod 64 \leq (\text{NextTXSeq}-\text{ExpectedAckSeq}) \bmod 64$. Furthermore, from the description of NextTXSeq, it can be seen that $(\text{NextTXSeq}-\text{ExpectedAckSeq}) \bmod 64 \leq \text{TxWindow}$.

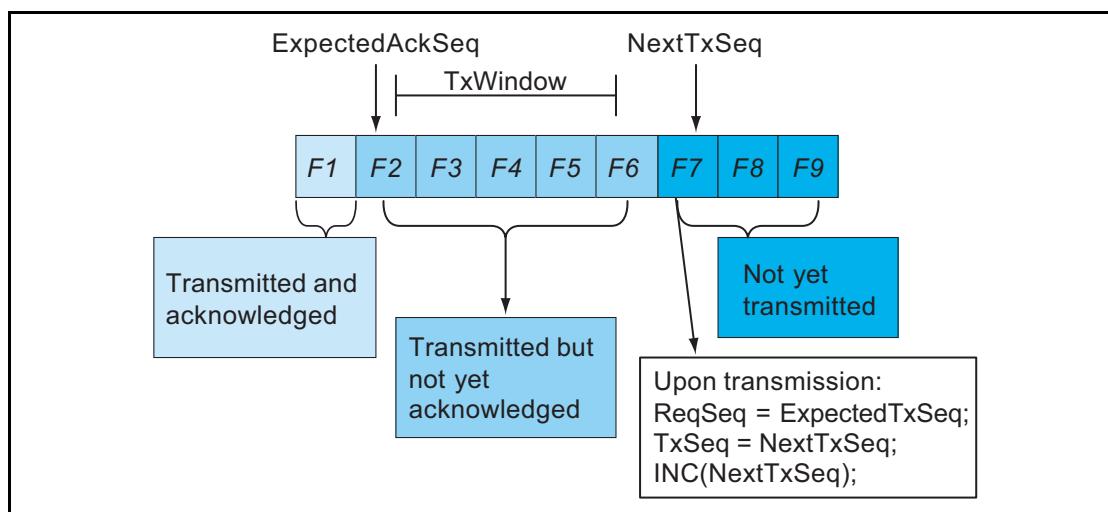


Figure 8.1: Example of the transmitter side

Figure 8.1 on page 152 shows $TxWindow=5$, and three frames awaiting transmission. The frame with number F_7 may be transmitted when the frame with F_2 is acknowledged. When the frame with F_7 is transmitted, $TxSeq$ is set to the value of $NextTXSeq$. After $TxSeq$ has been set, $NextTxSeq$ is incremented.

The sending peer expects to receive legal $ReqSeq$ values, these are in the range $ExpectedAckSeq$ up to and including $NextTxSeq$. Upon receipt of a $ReqSeq$ value equal to the current $NextTxSeq$ all outstanding I-frames have been acknowledged by the receiver.

8.3.2 Receiving peer

8.3.2.1 Receive sequence number $ReqSeq$

All I-frames and S-frames contain $ReqSeq$, the send Sequence number ($TxSeq$) that the receiving peer requests in the next I-frame.

When an I-frame or an S-frame is transmitted, the value of $ReqSeq$ shall be set to the current value of the receive state variable $ExpectedTxSeq$ or the buffer state variable $BufferSeq$. The value of $ReqSeq$ shall indicate that the data link layer entity transmitting the $ReqSeq$ has correctly received all I-frames numbered up to and including $ReqSeq - 1$.

Note: The option to set $ReqSeq$ to $BufferSeq$ instead of $ExpectedTxSeq$ allows the receiver to impose flow control for buffer management or other purposes. In this situation, if $BufferSeq <> ExpectedTxSeq$, the receiver should also set the retransmission disable bit to 1 to prevent unnecessary retransmissions.

8.3.2.2 Receive state variable, *ExpectedTxSeq*

Each channel shall have a receive state variable (*ExpectedTxSeq*). The receive state variable is the sequence number (TxSeq) of the next in-sequence I-frame expected.

The value of the receive state variable shall be the last in-sequence, valid I-frame received.

8.3.2.3 Buffer state variable *BufferSeq*

Each channel may have an associated *BufferSeq*. *BufferSeq* is used to delay acknowledgement of frames until they have been pulled by the upper layers, thus preventing buffer overflow. *BufferSeq* and *ExpectedTxSeq* are equal when there is no extra segmentation performed and frames are pushed to the upper layer immediately on reception. When buffer space is scarce, for example when frames reside in the buffer for a period, the receiver may choose to set *ReqSeq* to *BufferSeq* instead of *ExpectedTxSeq*, incrementing *BufferSeq* as buffer space is released. The windowing mechanism will ensure that transmission is halted when *ExpectedTxSeq* - *BufferSeq* is equal to *TxWindow*.

Note: Owing to the variable size of I-frames, updates of *BufferSeq* may be based on changes in available buffer space instead of delivery of I-frame contents.

I-frames shall have sequence numbers in the range $\text{ExpectedTxSeq} \leq \text{TxSeq} < (\text{BufferSeq} + \text{TxWindow})$.

On receipt of an I-frame with TxSeq equal to *ExpectedTxSeq*, *ExpectedTxSeq* shall be incremented by 1 regardless of how many I-frames with TxSeq greater than *ExpectedTxSeq* were previously received.

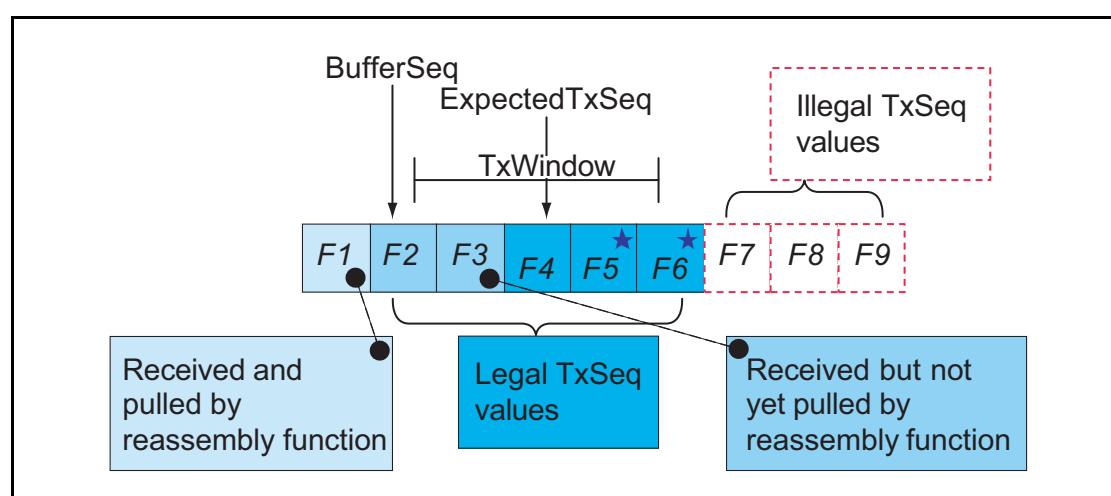


Figure 8.2: Example of the receiver side

[Figure 8.2 on page 153](#) shows TxWindow=5. F1 is successfully received and pulled by the upper layer. BufferSeq shows that F2 is the next I-frame to be pulled, and ExpectedTxSeq points to the next I-frame expected from the peer. An I-frame with TxSeq equal to 5 has been received thus triggering an SREJ or REJ exception. The star indicates I-frames received but discarded owing to the SREJ or REJ exception. They will be resent as part of the error recovery procedure.

In [Figure 8.2](#) there are several I-frames in a buffer awaiting the SDU reassembly function to pull them and the TxWindow is full. The receiver would usually disable retransmission in Retransmission mode or Flow Control mode by setting the Retransmission Disable Bit to 1 and send an RR back to the sending side. This tells the transmitting peer that there is no point in performing retransmissions. Both sides will send S-frames to make sure the peer entity knows the current value of the Retransmission Disable Bit.

8.4 RETRANSMISSION MODE

8.4.1 Transmitting frames

A new I-frame shall only be transmitted when the TxWindow is not full. No I-frames shall be transmitted if the last RetransmissionDisableBit (R) received is set to one.

A previously transmitted I-frame may be retransmitted as a result of an error recovery procedure, even if the TxWindow is full. When an I-frame is retransmitted it shall always be sent with the same TxSeq value used in its initial transmission.

The state of the RetransmissionDisableBit (R) is stored and used along with the state of the RetransmissionTimer to decide the actions when transmitting I-frames. The RetransmissionTimer is running whenever I-frames have been sent but not acknowledged.

8.4.1.1 *Last received R was set to zero*

If the last R received was set to zero, then I-frames may be transmitted. If there are any I-frames which have been sent and not acknowledged then they shall be retransmitted when the RetransmissionTimer elapses. If the retransmission timer has not elapsed then a retransmission shall not be sent and only new I-frames may be sent.

- a) If unacknowledged I-frames have been sent and the RetransmissionTimer has elapsed then an unacknowledged I-

frame shall be retransmitted. The RetransmissionTimer shall be restarted.

- b) If unacknowledged I-frames have been sent, the Retransmission timer has not elapsed then a new I-frame shall be sent if one is waiting and no timer action shall be taken.
- c) If no unacknowledged I-frames have been sent, and a new I-frame is waiting, then the new I-frame shall be sent, the RetransmissionTimer shall be started and if the Monitor Timer is running, it shall be stopped.
- d) If no unacknowledged I-frames have been sent and no new I-frames are waiting to be transmitted, and the RetransmissionTimer is running, then the retransmission timer shall be stopped and the monitor timer shall be started.

The table below summarizes actions when the RetransmissionTimer is in use and R=0.

Unacknowledged I-frames sent = Retransmission Timer is running	Retransmission Timer has elapsed	New I-frames are waiting	Transmit Action	Timer Action
True	True	True or False	Retransmit unacknowledged I-frame	Restart Retransmission Timer
True	False	True	Transmit new I-frame	No timer action
True	False	False	No transmit action	No Timer action
False	False	True	Transmit new I-frame	Restart Retransmission Timer
False	False	False	No Transmit action	If MonitorTimer is not running then restart MonitorTimer

Table 8.1: Summary of actions when the RetransmissionTimer is in use and R=0

If the RetransmissionTimer is not in use, no unacknowledged I-frames have been sent and no new I-frames are waiting to be transmitted

- a) If the MonitorTimer is running and has not elapsed then no transmit action shall be taken and no timer action shall be taken.
- b) If the MonitorTimer has elapsed then an S-frame shall be sent and the MonitorTimer shall be restarted.

If any I-frames become available for transmission then the MonitorTimer shall be stopped, the RetransmissionTimer shall be started and the rules for when the RetransmissionTimer is in use shall be applied.

When an I-frame is sent ReqSeq shall be set to ExpectedTxSeq, TxSeq shall be set to NextTxSeq and NextTxSeq shall be incremented by one.

8.4.1.2 Last received R was set to one

If the last R received was set to one, then I-frames shall not be transmitted. The only frames which may be sent are S-frames. An S-frame shall be sent according to the rules below:

- a) If the MonitorTimer is running and has not elapsed then no transmit action shall be taken and no timer action shall be taken.
- b) If the MonitorTimer has elapsed then an S-frame shall be sent and the MonitorTimer shall be restarted.

8.4.2 Receiving I-frames

Upon receipt of a valid I-frame with TxSeq equal to ExpectedTxSeq, the frame shall be accepted for the SDU reassembly function. ExpectedTxSeq is used by the reassembly function.

The first valid I-frame received after an REJ was sent, with a TxSeq of the received I-frame equal to ReqSeq of the REJ, shall clear the REJ Exception condition.

The ReqSeq shall be processed according to [Section 8.4.6 on page 159](#).

If a valid I-frame with TxSeq ≠ ExpectedTxSeq is received then an exception condition shall be triggered which is handled according to [Section 8.4.7 on page 159](#).

8.4.3 I-frames pulled by the SDU reassembly function

When the L2CAP layer has removed one or more I-frames from the buffer, BufferSeq may be incremented in accordance with the amount of buffer space released. If BufferSeq is incremented, an acknowledgement shall be sent to the peer entity.

Note: Since the primary purpose of BufferSeq is to prevent buffer overflow, an implementation may choose to set BufferSeq in accordance with how many new incoming I-frames could be stored rather than how many have been removed.

The acknowledgement may either be an RR or an I-frame. The acknowledgement shall be sent to the peer L2CAP entity with ReqSeq equal to BufferSeq. When there are no I-frames buffered for pulling ExpectedTxSeq is equal to BufferSeq.

If the MonitorTimer is active then it shall be restarted to indicate that a signal has been sent to the peer L2CAP entity.

8.4.4 Sending and receiving acknowledgements

Either the MonitorTimer or the RetransmissionTimer shall be active while in Retransmission Mode. Both timers shall not be active concurrently.

8.4.4.1 Sending acknowledgements

Whenever an L2CAP entity transmits an I-frame or an S-frame, ReqSeq shall be set to ExpectedTxSeq or BufferSeq.

8.4.4.2 Receiving acknowledgements

On receipt of a valid S-frame or I-frame, the ReqSeq contained in the frame shall acknowledge previously transmitted I-frames. ReqSeq acknowledges I-frames with a TxSeq up to and including ReqSeq – 1.

The following rules shall be applied:

1. If the RetransmissionDisableBit changed value from 0 to 1 (stop retransmissions) then the receiving entity shall
 - a) If the RetransmissionTimer is running then stop it and start the MonitorTimer.
 - b) Store the state of the RetransmissionDisableBit received.

2. If the RetransmissionDisableBit changed value from 1 to 0 (start retransmissions) then the receiving entity shall
 - a) Store the state of the RetransmissionDisableBit received.
 - b) If there are any I-frames that have been sent but not acknowledged, then stop the MonitorTimer and start the RetransmissionTimer.
 - c) Any buffered I-frames shall be transmitted according to [Section 8.4.1 on page 154](#).
3. If any unacknowledged I-frames were acknowledged by the ReqSeq contained in the frame, and the RetransmissionDisableBit equals 1 (retransmissions stopped), then the receiving entity shall
 - a) Follow the rules in [Section 8.4.1 on page 154](#).
4. If any unacknowledged I-frames were acknowledged by the ReqSeq contained in the frame and the RetransmissionDisableBit equals 0 (retransmissions started) then the receiving entity shall
 - a) If the RetransmissionTimer is running, then stop it.
 - b) If any unacknowledged I-frames have been sent then the RetransmissionTimer shall be restarted.
 - c) Follow the rules in [Section 8.4.1 on page 154](#).
 - d) If the RetransmissionTimer is not running and the MonitorTimer is not running, then start the MonitorTimer.

On receipt of a valid S-frame or I-frame the ReqSeq contained in the frame shall acknowledge previously transmitted I-frames. ExpectedAckSeq shall be set to ReqSeq to indicate that the I-frames with TxSeq up to and including (ReqSeq - 1) have been acknowledged.

8.4.5 Receiving REJ frames

Upon receipt of a valid REJ frame, where ReqSeq identifies an I-frame not yet acknowledged, the ReqSeq acknowledges I-frames with TxSeq up to and including ReqSeq - 1. Therefore the REJ acknowledges all I-frames before the I-frame it is rejecting.

ExpectedAckSeq shall be set equal to ReqSeq to mark I-frames up to and including ReqSeq - 1 as received.

NextTXSeq shall be set to ReqSeq to cause transmissions of I-frames to resume from the point where TxSeq equals ReqSeq.

If ReqSeq equals ExpectedAckSeq then the REJ frame shall be ignored.

8.4.6 Waiting acknowledgements

A counter, TransmitCounter, counts the number of times an L2CAP PDU has been transmitted. This shall be set to 1 after the first transmission. If the RetransmissionTimer expires the following actions shall be taken:

1. If the TransmitCounter is less than MaxTransmit then:
 - a) Increment the TransmitCounter
 - b) Retransmit the last unacknowledged I-frame, according to [Section 8.4.1 on page 154](#).
2. If the TransmitCounter is equal to MaxTransmit this channel to the peer entity shall be assumed lost. The channel shall move to the CLOSED state and appropriate action shall be taken to report this to the upper layers.

8.4.7 Exception conditions

Exception conditions may occur as the result of physical layer errors or L2CAP procedural errors. The error recovery procedures which are available following the detection of an exception condition at the L2CAP layer in Retransmission Mode are defined in this section.

8.4.7.1 TxSeq Sequence error

A TxSeq sequence error exception condition occurs in the receiver when a valid I-frame is received which contains a TxSeq value which is not equal to the expected value, thus TxSeq is not equal to ExpectedTxSeq.

The TxSeq sequence error may be due to three different causes:

- *Duplicated I-frame*

The duplicated I-frame is identified by a TxSeq in the range BufferSeq to ExpectedTxSeq – 1 ($\text{BufferSeq} \leq \text{TxSeq} < \text{ExpectedTxSeq}$). The ReqSeq and RetransmissionDisableBit shall be processed according to [Section 8.4.4 on page 157](#). The Information field shall be discarded since it has already been received.

- *Out-of-sequence I-frame*

The out-of-sequence I-frame is identified by a TxSeq within the legal range. The ReqSeq and RetransmissionDisableBit shall be processed according to [Section 8.4.4 on page 157](#).

A REJ exception is triggered, and an REJ frame with ReqSeq equal to ExpectedTxSeq shall be sent to initiate recovery. The received I-frame shall be discarded.

- *Invalid TxSeq*

An invalid TxSeq value is a value that does not meet either of the above conditions. An I-frame with an invalid TxSeq is likely to have errors in the control field and shall be silently discarded.

8.4.7.2 ReqSeq Sequence error

An ReqSeq sequence error exception condition occurs in the transmitter when a valid S-frame or I-frame is received which contains an invalid ReqSeq value. An invalid ReqSeq is one that is not in the range $\text{ExpectedAckSeq} \leq \text{ReqSeq} \leq \text{NextTxSeq}$.

The L2CAP entity shall close the channel as a consequence of an ReqSeq Sequence error.

8.4.7.3 Timer recovery error

If an L2CAP entity fails to receive an acknowledgement for the last I-frame sent, then it will not detect an out-of-sequence exception condition and therefore will not transmit an REJ frame.

The L2CAP entity that transmitted an unacknowledged I-frame shall, on the expiry of the RetransmissionTimer, take appropriate recovery action as defined in [Section 8.4.6 on page 159](#).

8.4.7.4 Invalid frame

Any frame received which is invalid (as defined in [Section 3.3.6 on page 53](#)) shall be discarded, and no action shall be taken as a result of that frame.

8.5 FLOW CONTROL MODE

When a link is configured to work in flow control mode, the flow control operation is similar to the procedures in retransmission mode, but all operations dealing with CRC errors in received packets are not used. Therefore

- REJ frames shall not be used in Flow Control Mode.
- The RetransmissionDisableBit shall always be set to zero in the transmitter, and shall be ignored in the receiver.

The behavior of flow control mode is specified in this section.

Assuming that the TxWindow size is equal to the buffer space available in the receiver (counted in number of I-frames), in flow control mode the number of unacknowledged frames in the transmitter window is always less than or equal to the number of frames for which space is available in the receiver. Note that a missing frame still occupies a place in the window.

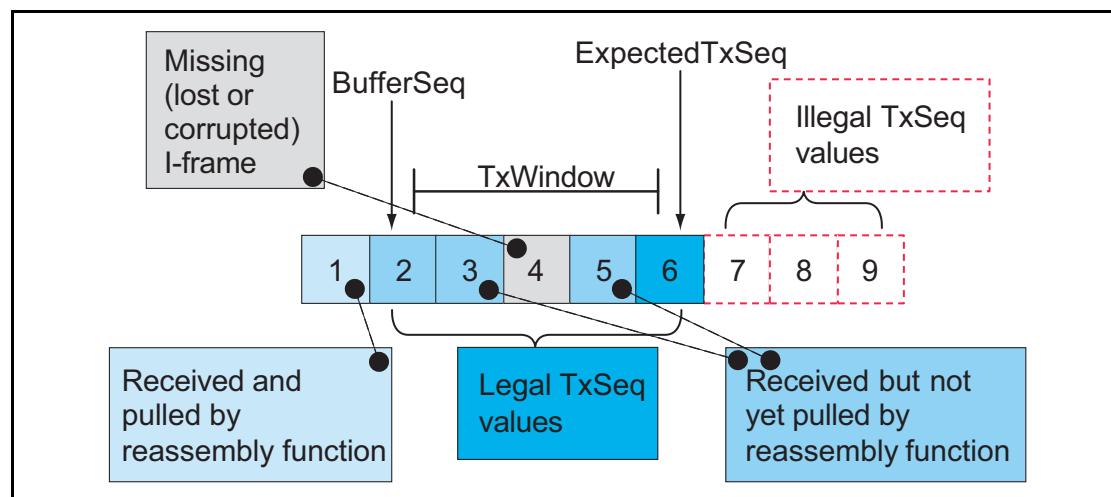


Figure 8.3: Overview of the receiver side when operating in flow control mode

8.5.1 Transmitting I-frames

A new I-frame shall only be transmitted when the TxWindow is not full.

Upon transmission of the I-frame the following actions shall be performed:

- If no unacknowledged I-frames have been sent then the MonitorTimer shall be stopped and the RetransmissionTimer shall be started.
- If any I-frames have been sent and not acknowledged then the RetransmissionTimer remains active and no timer operation is performed.

The control field parameter ReqSeq shall be set to ExpectedTxSeq, TxSeq shall be set to NextTXSeq and NextTXSeq shall be incremented by one.

8.5.2 Receiving I-frames

Upon receipt of a valid I-frame with TxSeq equal to ExpectedTxSeq, the frame shall be made available to the reassembly function. ExpectedTxSeq shall be incremented by one. An acknowledgement shall not be sent until the SDU reassembly function has pulled the I-frame.

Upon receipt of a valid I-frame with an out-of-sequence TxSeq (see [Section 8.5.6 on page 164](#)) all frames with a sequence number less than TxSeq shall be assumed lost and marked as missing. The missing I-frames are in the range from ExpectedTxSeq (the frame that the device was expecting to receive) up to TxSeq-1, (the frame that the device actually received). ExpectedTxSeq shall be set to TxSeq +1. The received I-frame shall be made available for pulling by the reassembly function. The acknowledgement shall not occur until the SDU reassembly function has pulled the I-frame. The ReqSeq shall be processed according to [Section 8.5.4 on page 162](#).

8.5.3 I-frames pulled by the SDU reassembly function

When the L2CAP layer has removed one or more I-frames from the buffer, BufferSeq may be incremented in accordance with the amount of buffer space released. If BufferSeq is incremented, an acknowledgement shall be sent to the peer entity. If the MonitorTimer is active then it shall be restarted to indicate that a signal has been sent to the peer L2CAP entity.

Note: Since the primary purpose of BufferSeq is to prevent buffer overflow, an implementation may choose to set BufferSeq in accordance with how many new incoming I-frames could be stored rather than how many have been removed.

The acknowledgement may be an RR or an I-frame. The acknowledgement shall be sent to the peer L2CAP entity with ReqSeq equal to BufferSeq. When there is no I-frame buffered for pulling, ExpectedTxSeq is equal to BufferSeq.

8.5.4 Sending and receiving acknowledgements

One of the timers MonitorTimer or RetransmissionTimer shall always be active while in Flow Control mode. Both timers shall never be active concurrently.

8.5.4.1 *Sending acknowledgements*

Whenever a data link layer entity transmits an I-frame or a S-frame, ReqSeq shall be set to ExpectedTxSeq or BufferSeq.

8.5.4.2 Receiving acknowledgements

On receipt of a valid S-frame or I-frame the ReqSeq contained in the frame shall be used to acknowledge previously transmitted I-frames. ReqSeq acknowledges I-frames with a TxSeq up to and including ReqSeq – 1.

1. If any outstanding I-frames were acknowledged then
 - a) Stop the RetransmissionTimer
 - b) If there are still unacknowledged I-frames then restart the RetransmissionTimer, otherwise start the MonitorTimer.
 - c) Transmit any I-frames awaiting transmission according to [Section 8.5.1 on page 161](#).

ExpectedAckSeq shall be set to ReqSeq to indicate that the I-frames with TxSeq up to and including ExpectedAckSeq have been acknowledged.

8.5.5 Waiting acknowledgements

If the RetransmissionTimer expires the following actions shall be taken:
The I-frame supervised by the RetransmissionTimer shall be considered lost, and ExpectedAckSeq shall be incremented by one.

1. If I-frames are waiting to be sent
 - a) the RetransmissionTimer is restarted.
 - b) I-frames awaiting transmission are transmitted according to [Section 8.5.1 on page 161](#).
2. If there are no I-frames waiting to be sent
 - a) If there are still unacknowledged I-frames the RetransmissionTimer is restarted, otherwise the MonitorTimer is started.

8.5.6 Exception conditions

Exception conditions may occur as the result of physical layer errors or L2CAP procedural errors. The error recovery procedures which are available following the detection of an exception condition at the L2CAP layer in flow control only mode are defined in this section.

8.5.6.1 TxSeq Sequence error

A TxSeq sequence error exception condition occurs in the receiver when a valid I-frame is received which contains a TxSeq value which is not equal to the expected value, thus TxSeq is not equal to ExpectedTxSeq.

The TxSeq sequence error may be due to three different causes:

- *Duplicated I-frame*

The duplicated I-frame is identified by a TxSeq in the range BufferSeq to ExpectedTxSeq – 1. The ReqSeq shall be processed according to [Section 8.5.4 on page 162](#). The Information field shall be discarded since it has already been received.

- *Out-of-sequence I-frame*

The out-of-sequence I-frame is identified by a TxSeq within the legal range ExpectedTxSeq < TxSeq < (BufferSeq + TxWindow). The ReqSeq shall be processed according to [Section 8.5.4 on page 162](#).

The missing I-frame(s) are considered lost and ExpectedTXSeq is set equal to TxSeq+1 as specified in [Section 8.5.2 on page 162](#). The missing I-frame(s) are reported as lost to the SDU reassembly function.

- *Invalid TxSeq*

An invalid TxSeq value is a value that does not meet either of the above conditions and TxSeq is not equal to ExpectedTxSeq. An I-frame with an invalid TxSeq is likely to have errors in the control field and shall be silently discarded.

8.5.6.2 ReqSeq Sequence error

An ReqSeq sequence error exception condition occurs in the transmitter when a valid S-frame or I-frame is received which contains an invalid ReqSeq value. An invalid ReqSeq is one that is not in the range $\text{ExpectedAckSeq} \leq \text{ReqSeq} \leq \text{NextTXSeq}$.

The L2CAP entity shall close the channel as a consequence of an ReqSeq Sequence error.

An L2CAP entity that fails to receive an acknowledgement for an I-frame shall, on the expiry of the RetransmissionTimer, take appropriate recovery action as defined in [Section 8.5.5 on page 163](#).

8.5.6.3 Invalid frame

Any frame received that is invalid (as defined in [Section 3.3.6 on page 53](#)) shall be discarded, and no action shall be taken as a result of that frame, unless the receiving L2CAP entity is configured to deliver erroneous frames to the layer above L2CAP. In that case, the data contained in invalid frames may also be added to the receive buffer and made available for pulling from the SDU reassembly function.

8.6 ENHANCED RETRANSMISSION MODE

Enhanced Retransmission mode operates as an HDLC balanced data link operational mode. Either L2CAP entity may send frames at any time without receiving explicit permission from the other L2CAP entity. A transmission may contain single or multiple frames and shall be used for I-frame transfer and/or to indicate status change.

8.6.1 Function Of PDU Types

Enhanced Retransmission mode uses I-frames to transfer upper layer information and S-frames for supervision. There are four S-frames defined: Receiver Ready (RR), Reject (REJ), Receiver Not Ready (RNR), and Selective Reject (SREJ). All frames formats in Enhanced Retransmission mode shall use the Enhanced Control Field.

8.6.1.1 Receiver Ready (RR)

The RR frame shall be used by an L2CAP entity to

1. Indicate that it is ready to receive I-frames
2. Acknowledge previously received I-frames numbered up to ReqSeq - 1 inclusive.

An RR with P-bit set to 1 (P=1) is used to indicate the clearance of any busy condition that was initiated by an earlier transmission of an RNR frame by the same L2CAP entity.

8.6.1.2 Reject (REJ)

The REJ frame shall be used by an L2CAP entity to request retransmission of I-frames starting with the frame numbered ReqSeq. I-frames numbered ReqSeq - 1 and below shall be considered acknowledged. Additional I-frames awaiting initial transmission may be transmitted following the retransmitted I-frame(s) up to the TxWindow size of the receiver.

At most only one REJ exception from a given L2CAP entity to another L2CAP entity shall be established at any given time. A REJ frame shall not be transmitted until an earlier REJ exception condition or all earlier SREJ

exception conditions have been cleared. The REJ exception condition shall be cleared upon the receipt of an I-frame with TxSeq equal to the ReqSeq of the REJ frame.

Two L2CAP entities may be in REJ exception conditions with each other at the same time.

8.6.1.3 Receiver Not Ready (RNR)

The RNR frame shall be used by an L2CAP entity to indicate a busy condition (i.e. temporary inability to receive I-frames). I-frames numbered up to ReqSeq - 1 inclusive shall be considered acknowledged. The I-frame numbered ReqSeq and any subsequent I-frames sent shall not be considered acknowledged. The acceptance status of these I-frames shall be indicated in subsequent transfers.

8.6.1.4 Selective Reject (SREJ)

The SREJ frame shall be used by an L2CAP entity to request retransmission of one I-frame. The ReqSeq shall indicate the TxSeq of the earliest I-frame to be retransmitted (not yet reported by a SREJ). If the P-bit is set to 1 then I-frames numbered up to ReqSeq -1 inclusive shall be considered acknowledged. If the P-bit is set to 0 then the ReqSeq field in the SREJ shall not indicate acknowledgement of I-frames.

Each SREJ exception condition shall be cleared upon receipt of an I-frame with TxSeq equal to the ReqSeq sent in the SREJ frame.

An L2CAP entity may transmit one or more SREJ frames with the P=0 before one or more earlier SREJ exception conditions initiated with SREJ(P=0) have been cleared. An L2CAP entity shall not transmit more than one SREJ with P=1 before all earlier SREJ exception conditions have been cleared. A SREJ frame shall not be transmitted if an earlier REJ exception condition has not been cleared. Likewise a REJ frame shall not be transmitted if one or more SREJ exception conditions have not been cleared. Only one I-frame shall be retransmitted in response to receiving a SREJ frame with P=0. Additional I-frames awaiting initial transmission may be transmitted following the retransmission of the specific I-frame requested by SREJ with P=1.

8.6.1.5 Functions of the Poll (P) and Final (F) bits.

P-bit set to 1 shall be used to solicit a response frame with the F-bit set to 1 from the remote L2CAP entity at the earliest respond opportunity. At most only one frame with a P=1 shall be outstanding in a given direction at a given time. Before an L2CAP entity issues another frame with P=1, it shall have received a response frame from the remote L2CAP entity with F=1. If no valid frame is received with F=1 within Monitor time-out period, the frame with P=1 may be retransmitted.

The Final bit shall be used to indicate the frame as a response to a soliciting poll (S-frame with P=1). The frame with F=1 shall not be retransmitted. The Monitor-timeout is not used to monitor lost frames with F=1. Additional frames with F=0 may be transmitted following the frame with F=1.

S-frames shall not be transmitted with both the F-bit and the P-bit set to 1 at the same time.

8.6.2 Rules For Timers

Timers are started upon transmission of a packet. Timers should be started when the corresponding packet leaves the controller (transmitted or flushed). If the timer is not started when the packet leaves the controller then it shall be started when the packet is delivered to the controller. The specific rules for BR/EDR Controllers, BR/EDR/LE Controllers, and AMP Controllers are described in the following sections.

8.6.2.1 Timer Rules for ACL-U Logical Links

If a flush timeout does not exist on the ACL-U logical link for the channel using Enhanced Retransmission mode then the value for the Retransmission time-out shall be at least two seconds and the value for the Monitor time-out shall be at least twelve seconds.

If a flush timeout exists on the link for Enhanced Retransmission mode then the value for the Retransmission time-out shall be three times the value of flush timeout, subject to a minimum of 1 second and maximum of 2 seconds.

If a flush timeout exists on the link for Enhanced Retransmission mode and both sides of the link are configured to the same flush timeout value then the monitor time-out shall be set to a value at least as large as the Retransmission time-out otherwise the value of the Monitor time-out shall be six times the value of flush timeout, subject to a minimum of the retransmission timeout value and a maximum of 12 seconds.

If an L2CAP entity knows that a specific packet has been flushed instead of transmitted then it may execute proper error recovery procedures immediately.

When configuring a channel over an ACL-U logical link the values sent in a Configuration Request packet for Retransmission timeout and Monitor timeout shall be 0.

Note: If the link has a flush timeout and the Non-Flushable Packet Boundary Flag feature is used to mark the Enhanced Retransmission mode packets as non-flushable then the link does not have a flush timeout with regards to Enhanced Retransmission mode.

8.6.2.2 Timer Rules for AMP Controllers

AMP Controllers can be classified by their behavior as follows:

1. Attempt to send a packet until LSTO disconnects the physical link.
2. Attempt to send a packet until a Controller based retry counter is exceeded whereupon the packet is flushed.

Class 1 AMP controllers are reliable. They will not lose packets though packets can be lost during a move operation. Class 2 AMP controllers are not reliable and can lose packets. The Best Effort Flush Timeout field in the AMP Info (see [\[Vol. 2\] Part E, Section 7.5.8](#)) can be used to determine the behavior of the controller (i.e. class 1 or class 2). If the Best Effort Flush Timeout field in the AMP info is 0xFFFFFFFF then the class is 1 otherwise the class is 2. The Best Effort Flush Timeout field indicates the flush timeout that may be set on the Best Effort logical link.

For class 1 AMP controllers the Retransmission and Monitor timers are not really needed. Packets should not be lost. Packets lost during the move operation will be detected by L2CAP and retransmitted as part of the move operation. Therefore, the Retransmission timeout and Monitor timeout shall be at least the value of the Link supervision timeout set on the link or can be turned off altogether. The values sent in a Configuration Request packet for Retransmission timeout and Monitor timeout shall be 0.

When configuring a channel the AMP-U logical link of a over a class 2 AMP Controller non-0 values for Retransmission timeout and Monitor timeout shall be sent in a Configuration Request packet. The non-0 values specify the “processing” time of received packets. Processing time includes the time it takes for a received packet to be passed from the device's Controller to the L2CAP layer plus the time to be processed by the L2CAP layer and a response submitted back to the Controller. The local device's processing time is used by the remote device in calculating the Retransmission timeout and Monitor timeout it will use. The timeout values that will actually be used by a device shall be sent in a Configuration Response packet.

As with BR/EDR and BR/EDR/LE Controllers, the method used for setting timer values for class 2 AMP Controllers depends on when timers are started. Timers are either started when the packet leaves the Controller or when the packet is delivered to the Controller.

The timeout values used for Class 2 AMP Controllers where timers are started when the packet leaves the controller shall be at least the values received in the Configuration Request packet from the remote device (remote device's “processing” time).

For Class 2 AMP Controller where timers are started when the packet is delivered to the Controller the following rules shall be used:

1. The local L2CAP Entity shall set a flush timeout on the Best Effort logical link equal to the value provided by the local Controller in the Best Effort Flush Timeout field of AMP info structure.
2. The value of the Retransmission timeout and Monitor Timeout shall be the value of the flush timeout multiplied by three plus the corresponding processing time received in the Configuration Request packet from the remote device.

If an L2CAP entity knows that a specific packet has been flushed instead of transmitted then it may execute proper error recovery procedures immediately.

8.6.2.3 Timer Values used After a Move Operation

When a channel is moved from one Controller to another Controller the timeout values used after the move operation are based on the capabilities of the new Controller. The timeout values shall be set according to the following table:

Controller	Timeout Values
BR/EDR and BR/EDR/LE	Retransmission timeout – at least 2 seconds Monitor timeout – at least 12 seconds
Class 1 AMP Controller	Retransmission timeout – at least LSTO (or turned off) Monitor timeout – at least LSTO (or turned off)
Class 2 AMP Controller where timers are started when packets leave the controller	Retransmission timeout – at least 500ms Monitor timeout – at least 500ms Note: 500ms is the default value for the “processing” time of received packets.
Class 2 AMP Controller where timers are started when packets are delivered to the controller	Retransmission timeout – at least (local controller “Best Effort” Flush Timeout * 3) + 500ms Monitor timeout – at least (local controller “Best Effort” Flush Timeout * 3) + 500ms Note: 500ms is the default value for the “processing” time of received packets

Table 8.2: AMP Controller timeout values

When moving to a Class 2 AMP Controller, the Retransmission Timeout and Monitor timeout should be reconfigured after the move operation to get the actual value for the “processing” time of received packets.

8.6.3 General Rules for the State Machine

Enhanced Retransmission mode is specified using a pair of state machines, a Transmitter state machine and a Receiver state machine. The following rules apply to the state machine pair.

1. The state machine pair is informative but described using normative text in order to clearly specify the behavior of the protocol. Designers and implementers may choose any design / implementation technique they wish, but it shall behave in a manner identical to the external behavior of the specified state machines.
2. There is a single state machine pair for each active L2CAP channel configured to use Enhanced Retransmission mode.
3. Variables are used to limit the number of states by maintaining the state of particular conditions. The variables are defined in [section 8.6.5.2](#).
4. For some combinations of Event and Condition, the state tables provide alternative groups of actions. These alternatives are separated by horizontal lines in the Actions and Next State columns. The alternatives are mutually exclusive; selection of an alternate is done based upon (i) local status, (ii) a layer management action, or (iii) an implementation decision. There is no relationship between the order of the alternatives between events, nor is it implied that the same alternative must be selected every time the event occurs.
5. The state tables use timers. Any Start Timer action restarts the specified timer from its initial value, even if the timer is already running. When the timer reaches 0 the appropriate timer expired event is set and the timer stops. The Stop Timer action stops a timer if it is running.
6. Events not recognized in a particular state are assumed to remain pending until any masking flag is modified or a transition is made to a state where they can be recognized.
7. Some state transitions and actions are triggered by internal events (e.g. requests from the upper layer). It is implementation specific how these internal events are realized. They are used for clarity in specifying the state machine. All events including Internal events are described in [Section 8.6.5.3 on page 172](#).
8. The state machines specify the exact frames to be sent by transmitters but are relaxed on what receivers are allowed to accept as valid. For example there are cases where the transmitter is required to send a frame with P=1. The correct response is a frame with F=1 but in some cases the receiver is allowed to accept a frame with F=0 in addition to F=1.

8.6.4 State Diagram

The state diagram shows the states and the main transitions. Not all events are shown on the state diagram.

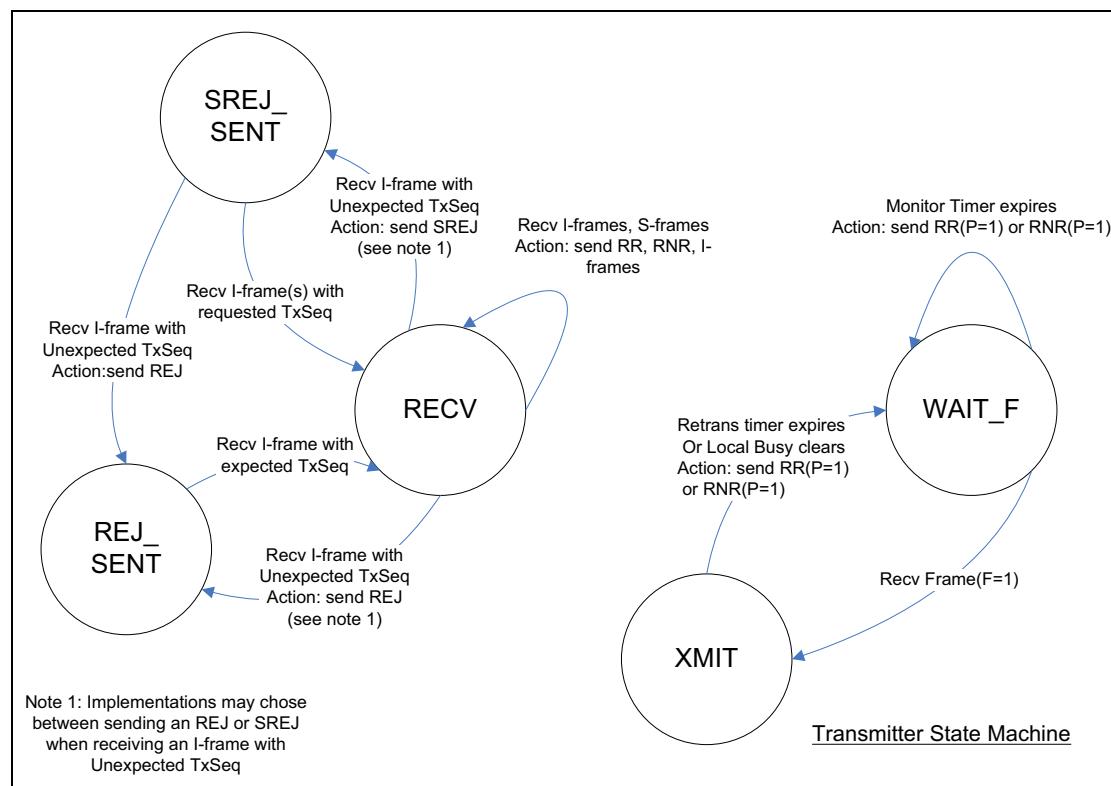


Figure 8.4: Receiver state machine

8.6.5 States Tables

8.6.5.1 State Machines

Enhanced Retransmission mode is described as a pair of state machine. The Receiver state machine handles all received frames while the Transmitter State machine handles all asynchronous events including requests from the upper layer and the expiration of timers.

The Receiver state machine “calls” the Transmitter state machine using the PassToTx action. This shows up in the Transmitter state machine as an event. When the Transmitter state machine is called it runs to completion before returning to the Receiver state machine. Running to completion means that all actions are executed and the Transmitter state is changed to the new state.

The Receiver and Transmitter state machine share variables and timers.

8.6.5.2 States

The following states have been defined to specify the protocol; the actual number of states and naming in a given implementation is outside the scope of this specification:

RECV—This is the main state of the Receiver state machine.

REJ_SENT—The L2CAP entity has sent a REJ frame to cause the remote L2CAP entity to resend I-frame(s). The L2CAP entity is waiting for the I-frame with a TxSeq that matches the ReqSeq sent in the REJ. Whether to send a REJ versus a SREJ is implementation dependent.

SREJ_SENT—The L2CAP entity has sent one or more SREJ frames to cause the remote L2CAP entity to resend missing I-frame(s). The local L2CAP entity is waiting for all requested I-frames to be received. If additional missing I-frames are detected while in SREJ_SENT then additional SREJ frames or a REJ frame can be sent to request those I-frames. Whether to send a SREJ versus a REJ is implementation dependent.

XMIT—This is the main state of the Transmitter state machine.

WAIT_F—Local busy has been cleared or the Retransmission timer has expired and an S-frame with P=1 has been sent. The local L2CAP entity is waiting for a frame with F=1. I-frames cannot be sent while in the WAIT_F state to prevent the situation where retransmission of I-frames could result in the channel being disconnected.

8.6.5.3 Variables and Timers

Variables are used to limit the number of states and help clarify the state chart tables. Variables can be set to values, evaluated in conditions and compared in conditional statements. They are also used in the action descriptions. Below is a list of the operators, connectives and statements that can be used with variables.

Operator, connective or statement	Description
<code>:=</code>	Assignment operator. Used to set a variable to a value
<code>=</code>	Relational operator "equal"
<code>></code>	Relational operator "greater than"
<code><</code>	Relational operator "less than"
<code>≥</code>	Relational operator "greater than or equal"
<code>≤</code>	Relational operator "less than or equal"
<code>+</code>	Arithmetic operator "plus"

Table 8.3: Operators, connectives and statements used with variables

Operator, connective or statement	Description
mod	Modulo operator - returns the remainder of division of one number by another.
and	logical connective "and." It returns TRUE if both operands are TRUE otherwise it returns FALSE.
or	logical connective "or." It returns TRUE if either of its operands are TRUE otherwise it returns FALSE.
if (expression) then { statement }	Conditional Statement. If expression is TRUE then the statement is executed otherwise the statement is not executed. The statement is composed of one or more actions. All the actions in the statement are indented under the if ... then clause and contained within braces "{}".
if (expression) then{ statement1 } else{ statement2 }	Conditional Statement. If expression is TRUE then statement1 is executed otherwise statement2 is executed. A statement is composed of one or more actions. All the actions in the statement1 are indented under the if ... then clause and contained within braces "{}". All the actions of statement2 are indented under the else clause and contained within braces "{}".

Table 8.3: Operators, connectives and statements used with variables (Continued)

Enhanced Retransmission mode uses the following variables and sequence numbers described in [Section 8.3](#):

- TxSeq
- NextTxSeq
- ExpectedAckSeq
- ReqSeq
- ExpectedTxSeq
- BufferSeq

In addition to the variables above the following variables and timers are used:

RemoteBusy—when set to TRUE RemoteBusy indicates that the local L2CAP entity has received an RNR from the remote L2CAP entity and considers the remote L2CAP entity as busy. When the remote device is busy it will likely discard I-frames sent to it. The RemoteBusy flag is set to FALSE when the local L2CAP Entity receives an RR, REJ or SREJ. When set to FALSE the local L2CAP entity considers the remote L2CAP entity able to accept I-frames. When the channel is created RemoteBusy shall be set to FALSE.

LocalBusy—when set to TRUE, LocalBusy indicates the local L2CAP entity is busy and will discard received I-frames. When set to FALSE the local L2CAP entity is not busy and is able to receive I-frames. When the channel is created LocalBusy shall be set to FALSE.

UnackedFrames—holds the number of unacknowledged I-frames. When the channel is created UnackedFrames shall be set to 0.

UnackedList—holds the unacknowledged I-frames so they can be retransmitted if necessary. I-frames in the list are accessed via their TxSeq number. For example UnackedList[5] accesses the I-frame with TxSeq 5.

PendingFrames—holds the number of pending I-frames. I-frames passed to L2CAP from the upper layer may not be able to be sent immediately because the remote L2CAP entity's TxWindow is full, is in a busy condition or the local L2CAP is in the incorrect state. When I-frames cannot be sent they are stored in a queue until conditions allow them to be sent. When the channel is created PendingFrames shall be set to 0.

SrejList—is a list of TxSeq values for I-frames that are missing and need to be retransmitted using SREJ. A SREJ has already been sent for each TxSeq on the list. When SrejList is empty it equals 0 (i.e. SrejList = 0). If SrejList is not empty it is greater than 0 (i.e. SrejList > 0).

RetryCount—holds the number of times an S-frame operation is retried. If an operation is tried MaxTransmit times without success the channel shall be closed.

RetryIframes[]—holds a retry counter for each I-frame that is sent within the receiving device's TxWindow. Each time an I-frame is retransmitted the corresponding counter within RetryIframes is incremented. When an attempt to retransmit the I-frame is made and the counter is equal to MaxTransmit then the channel shall be closed.

RNRsent—when set to TRUE it means that the local L2CAP entity has sent an RNR frame. It is used to determine if the L2CAP entity needs to send an RR to the remote L2CAP entity to clear the busy condition. When the channel is created RNRsent shall be set to FALSE.

RejActioned—is used to prohibit a frame with F=1 from causing I-frames already retransmitted in response to a REJ from being retransmitted again. RejActioned is set to TRUE if a received REJ is actioned when a frame sent with P=1 is unanswered. When the channel is created RejActioned shall be set to FALSE.

SrejActioned—is used in conjunction with SrejSaveReqSeq to prohibit a frame with F=1 from causing an I-frame already retransmitted in response to a SREJ from being retransmitted again. SrejActioned is set to TRUE if a received SREJ is actioned when a frame sent with P=1 is unanswered. When the channel is created SrejActioned shall be set to FALSE.

SrejSaveReqSeq—is used to save the ReqSeq of a SREJ frame that causes SrejActioned to be set to TRUE.

SendRej—when set to TRUE it indicates that the local L2CAP entity has determined that a REJ should be sent in the SREJ_SENT state while processing received I-frames. The sending of new SREJ frames is stopped. When the channel is created SendRej shall be set to FALSE.

BufferSeqSrej—is used while in the SREJ_SENT state to keep track of the value to which BufferSeq will be set upon exit of the SREJ_SENT state.

FramesSent—is used to keep track of the number I-frames sent by the Send-Data and Retransmit-I-frames actions.

MaxTxWin—contains the maximum window size plus 1. It is used in TxWindow modulo operations as the divisor and in state table conditions. This value shall be set to 16384 (0x4000) if the Extended Window Size option is used; otherwise it shall be set to 64.

RetransTimer—The Retransmission Timer is used to detect lost I-frames. When the channel is created the RetransTimer shall be off.

MonitorTimer—The Monitor Timer is used to detect lost S-frames. When the channel is created the MonitorTimer shall be off.

8.6.5.4 Events

Data-Request—The upper layer has requested that an SDU be sent. The SDU may need to be broken into multiple I-frames by L2CAP based on the MPS of the remote device and/or the maximum PDU allowed by the HCI or QoS requirements of the system.

Local-Busy-Detected—A local busy condition occurs when the local L2CAP entity is temporarily unable to receive, or unable to continue to receive, I-frames due to internal constraints. For example, the upper layer has not pulled received I-frames and the local L2CAP entity needs to send an acknowledgment to the remote L2CAP entity. The method for handling the detection of local busy is implementation specific. An implementation may wait to send an RNR to see if the busy condition will clear before the remote L2CAP entity's Retransmission timer expires. If the busy condition clears then frames can be acknowledged with an RR or I-frame. If the busy condition does not clear before the remote L2CAP entity's Retransmission timer expires then an RNR shall be sent in response to the RR or RNR poll sent by the remote L2CAP entity. Optionally an implementation may send an RNR as soon as the local busy condition is detected.

Local-Busy-Clear—The local busy condition clears when L2CAP has buffer space to receive more I-frames (i.e. SDU Reassembly function and/or upper layer has pulled I-frames) and if necessary the upper layer has cleared the busy condition.

Recv ReqSeqAndFbit—This is an event generated by the Receiver state machine. It contains the ReqSeq and F-bit value of a received frame. The value of the F-bit can be checked in a condition.

Recv Fbit—This is an event generated by the Receiver state machine. It contains the F-bit value of a received frame. The value of the F-bit can be checked in a condition.

RetransTimer-Expires—The Retransmission Timer has counted down to 0 and stopped.

MonitorTimer-Expires—The Monitor Timer has counted down to 0 and stopped.

Recv I-frame—Receive an I-frame with any value for the F-bit.

Recv RR, REJ, RNR, SREJ (P=x) or (F=x)—Receive a specific S-frame (RR, REJ, etc.) with a specific value for the P and/or F bit. The F-bit and the P-bit shall not both be set to 1 in a transmitted S-frame so received S-frames with both P and F set to 1 should be ignored. If the P and/or F bit value is not specified in the event then either value is accepted.

Recv RRorRNR—Receive an RR or RNR with any value for the P-bit and F-bit.

Recv REJorSREJ—Receive an REJ or SREJ with any value for the P-bit and F-bit.

Recv frame—This is catch-all for all frames that are not explicitly declared as events in the state table.

8.6.5.5 Conditions

RemoteBusy = TRUE or FALSE—TRUE indicates the remote L2CAP entity is in a busy condition and FALSE indicates the remote L2CAP entity is not busy.

LocalBusy = TRUE or FALSE—TRUE indicates the local L2CAP entity is in a busy condition and FALSE indicates the local L2CAP entity is not busy.

RemWindow-Not-Full—The number of unacknowledged I-frames sent by L2CAP has not yet reached the TxWindow size of the remote L2CAP entity.

RemWindow-Full—The number of unacknowledged I-frames sent by the L2CAP entity has reached the TxWindow size of the remote L2CAP entity. No more I-frames shall be sent until one or more I-frames have been acknowledged.

RNRsent = TRUE or FALSE—TRUE indicates an RNR has been sent while a local busy condition exists. It is set to FALSE when the local busy condition clears.

F = 0 or 1—the F-bit of a received frame is checked. The F-bit of the received frame is available as part of the Recv ReqSeqAndFbit and Recv Fbit events.

RetryIframes[i] < or ≥ MaxTransmit—Compare the appropriate counter in RetryIframes after processing the ReqSeq in the receive frame to determine if it has reached MaxTransmit or not.

RetryCount < or ≥ MaxTransmit—Compare RetryCount to determine if it has reached MaxTransmit or not.

With-Expected-TxSeq—The TxSeq of a received I-frame is equal to ExpectedTxSeq.

With-Valid-ReqSeq—The ReqSeq of the received frame is in the range ExpectedAckSeq ≤ ReqSeq < NextTxSeq.

With-Valid-ReqSeq-Retrans—The ReqSeq of the received frame is in the range ExpectedAckSeq ≤ ReqSeq < NextTxSeq.

With-Valid-F-bit—The F-bit of a received frame is valid if it is 0 or if it is 1 and a frame sent with P=1 by the local L2CAP entity is unanswered (i.e. the local L2CAP entity send a frame with P=1 and has not yet received a frame with F=1 until receiving this one). If the Transmitter state machine is in the WAIT_F state then a frame sent with P=1 is unanswered.

With-unexpected-TxSeq—The TxSeq of the received I-frame is within the TxWindow of the L2CAP entity receiving the I-frame but has a TxSeq “greater” than ExpectedTxSeq where “greater” means later in sequence than ExpectedTxSeq.

With-duplicate-TxSeq—The TxSeq of the received I-frame is within the TxWindow of the L2CAP entity receiving the I-frame but has a TxSeq “less” than ExpectedTxSeq where “less” means earlier in the sequence than ExpectedTxSeq. In other words this is a frame that has already been received.

With-Invalid-TxSeq—The TxSeq of the received I-frame is not within the TxWindow of the L2CAP entity receiving the frame.

With-Invalid-ReqSeq—The ReqSeq of the received frame is not in the range ExpectedAckSeq ≤ ReqSeq < NextTxSeq.

With-Invalid-ReqSeq-Retrans—The ReqSeq of the received frame is not in the range ExpectedAckSeq ≤ ReqSeq < NextTxSeq.

Not-With-Expected-TxSeq—The TxSeq of the received I-frame is within the TxWindow of the L2CAP entity receiving the frame but is not equal to ExpectedTxSeq. It is either unexpected or a duplicate.

With-Expected-TxSeq-Srej—The TxSeq of the received I-frame is equal to the TxSeq at the head of SrejList.

SendRej = TRUE or FALSE—TRUE indicates that a REJ will be sent after all frames requested using SREJ have been received.

SreqList = or > 1—Determine if the number of items in SreqList is equal to or greater than 1.

With-Unexpected-TxSeq-Sreq—The TxSeq of the received I-frame is equal to one of the values stored in SreqList but is not the TxSeq at the head. This indicates that one or more I-frames requested using SREJ are missing either because the SREJ was lost or the requested I-frame(s) were lost. Either way the SREJ frames must be resent to retrieve the missing I-frames.

With-duplicate-TxSeq-Sreq—The TxSeq of the received I-frame is equal to a TxSeq of one of the saved I-frames indicating it is a duplicate.

8.6.5.6 Actions

Send-Data—This action is executed as a result of a Data-Request event. The number of I-frames sent without being acknowledged shall not exceed the TxWindow size of the receiving L2CAP entity (UnackedFrames is less than or equal to the remote L2CAP entity's TxWindow). Any I-frames that cannot be sent because they would exceed the TxWindow size are queued for later transmission. For each I-frame the following actions shall be carried out:

```

Send I-frame with TxSeq set to NextTxSeq and ReqSeq set
to BufferSeq.

UnackedList [NextTxSeq] := I-frame
UnackedFrames := UnackedFrames + 1
FramesSent := FramesSent + 1
RetryIframes [NextTxSeq] := 1
NextTxSeq := (NextTxSeq + 1) mod MaxTxWin
Start-RetransTimer

```

Pend-Data—This action is executed as a result of a Data-Request when it is not possible to send I-frames because the window is full, the remote L2CAP entity is in a busy condition or the local L2CAP entity is not in a state where I-frames can be sent (e.g. WAIT_F). The I-frame(s) are queued for later transmission.

Process-ReqSeq—the ReqSeq contained in the received frame shall acknowledge previously transmitted I-frames. ExpectedAckSeq shall be set to ReqSeq to indicate that the I-frames with TxSeq up to and including (ReqSeq—1) have been acknowledged. The acknowledged I-frames shall be removed from UnackedList, the retry counters for each acknowledged frame shall be set to 0 and the number of acknowledged frames shall be subtracted from UnackedFrames so that UnackedFrames shall contain the number of the remaining unacknowledged I-frames. Pending I-frames are now available to be

transmitted by the Send-Ack action. If UnackedFrames equals 0 then Stop-RetransTimer.

Send RR, RNR (P=x) or (F=x)—Send the specified S-frame with the specified value for the P-bit or F-bit. If a value for the P-bit or F-bit is not specified the value shall be 0. For example Send RR(P=1) means send an RR with the P-bit set to 1 and the F-bit set to 0. The ReqSeq field shall be set to BufferSeq. If an RNR is sent, RNRsent shall be set to TRUE.

Send REJ (P =x) or (F=x)—Send a REJ with the specified value for the P-bit or F-bit. The ReqSeq field shall be set to ExpectedTxSeq. If a value for the P-bit or F-bit is not specified the value shall be 0. Note that this will acknowledge previously received I-frames up to ExpectedTxSeq—1 and may allow the remote L2CAP entity to transmit new I-frames. If the local L2CAP entity is not in a position to acknowledge the previously received I-frames it may use SREJ(P=0) or RNR. It may also wait to send the REJ until it is able to acknowledge the I-frames.

Send RRorRNR (P=x) or (F=1)—Send an RR or RNR with the specified value for the P-bit or F-bit based on the value of LocalBusy. If a value for the P-bit or F-bit is not specified the value shall be 0. An RNR shall be sent if LocalBusy equals TRUE. If LocalBusy equals FALSE then an RR shall be sent.

Send IorRRorRNR(F=1)—Send I-frames, an RR or an RNR with the F-bit set to 1. The following algorithm shall be used:

```

FramesSent := 0
if LocalBusy = TRUE then{
    Send RNR (F=1)
}
if RemoteBusy = TRUE and UnackedFrames > 0 then {
    Start-RetransTimer
}
RemoteBusy := FALSE
Send-Pending-I-frames (see note)
if LocalBusy = FALSE and FramesSent = 0 then {
    Send RR (F=1)
}

```

Note: The SendIorRRorRNR(F=1) sends frames by invoking other actions. During the execution of SendIorRRorRNR multiple actions may be invoked. The first action invoked shall send the first or only frame with the F-bit set to 1. All other frames sent shall have the F-bit set to 0.

Send SREJ—Send one or more SREJ frames with P=0. For each missing I-frame starting with ExpectedTxSeq up to but not including the TxSeq of the

received I-frame, an SREJ frame is sent with ReqSeq set to the TxSeq of the missing frame. The TxSeq is inserted into the tail of SrejList. For example if ExpectedTxSeq is 3 and the received I-frame has a TxSeq of 5 there are two missing I-frames. An SREJ with ReqSeq 3 is sent followed by an SREJ with ReqSeq 4. TxSeq 3 is inserted first into SrejList followed by TxSeq 4. After all SREJ frames have been sent ExpectedTxSeq shall be set to the TxSeq of the received I-frame + 1 mod MaxTxWin.

Send SREJ(SrejList)—Send one or more SREJ frames with P=0. An I-frame was received that matches one of the TxSeq values in the SrejList but does not match the head of SrejList. This means I-frames requested via SREJ are still missing. For each TxSeq value starting with the head of SrejList and going backwards (i.e., from the head towards the tail) through the SrejList up to but not including the TxSeq of the received frame, an SREJ frame is sent with ReqSeq set to the TxSeq from SrejList. The TxSeq is removed from SrejList and reinserted into the tail of SrejList. Finally, remove the TxSeq of the received frame from the head of the list.

Send SREJ(SrejList-tail)(F=1)—Send a SREJ frame with F=1 and ReqSeq equal to the TxSeq at the tail of SrejList.

Start-RetransTimer—If the Monitor timer is not running then start the Retransmission Timer from its initial value (see Retransmission time-out in [Section 5.4](#)). If the Retransmission timer is already running it is restarted from its initial value. If the Monitor timer is running then the Retransmission timer is not started.

Start-MonitorTimer—Start the Monitor Timer from its initial value (see Monitor time-out in [Section 5.4 on page 96](#)). If the timer is already running it is restarted from its initial value.

PassToTx—Pass the ReqSeq and F-bit value of a received frame to the Transmitter state machine. This will show up as a Recv ReqSeqAndFbit event in the Transmitter state machine.

PassToTxFbit—Pass the F-bit value of a received frame to the Transmitter state machine. This will show up as a Recv Fbit event in the Transmitter state machine.

Data-Indication—A received I-frame is passed to the SDU reassembly function. For the purpose of the state machine this operation is completed immediately so the Send_Ack action should be executed as one of the next actions. In some cases the SDU reassembly function cannot accept the I-frame so the I-frame will be stored within the L2CAP Entity consuming a portion of its TxWindow. When the I-frame is pulled by the SDU reassembly function the Send_Ack action should be executed. Before the Send_Ack action is executed BufferSeq is advanced as follows:

```
BufferSeq := (BufferSeq + 1) mod MaxTxWin
```

Increment-ExpectedTxSeq—ExpectedTxSeq is incremented as follows:

```
ExpectedTxSeq := (ExpectedTxSeq + 1) mod MaxTxWin
```

Stop-RetransTimer—the Retransmission timer is stopped.

Stop-MonitorTimer —the Monitor timer is stopped.

Send-Ack (F=x)—an acknowledgement with the specified value for the F-bit may be sent. Note that this action may occur in an action block with other actions that also send frames. If a frame has already been sent then it is not necessary to send additional frames. If the value for the F-bit is not specified it shall be set to 0. If the value specified is P then the F-bit shall be set equal to the value of the P-bit of the received frame being acknowledged. If more than one frame is sent in the acknowledgment only the first frame shall have an F-bit set to 1. An acknowledgement is an RR, RNR, or pending I-frame(s) (I-frames that have not been transmitted yet). If pending I-frames are available and are allowed to be sent then as many as allowed should be sent as an acknowledgement. Sending an RR or RNR as an acknowledgement for each received I-frame is not required. An implementation may wait to send an RR or RNR until a specific number of I-frames have been received, after a certain period of time has elapsed or some other algorithm. To keep data flowing it is recommended that an acknowledgment be sent before the TxWindow is full. It should also be noted that the maximum size of a remote L2CAP entity's unacknowledged I-frame list may be smaller than the local L2CAP entity's TxWindow. Therefore the local L2CAP entity should not expect the remote L2CAP entity to send enough frames to fill its TxWindow and should acknowledge I-frames accordingly. The following algorithm shall be used when sending an acknowledgment.

```

if LocalBusy == TRUE then {
    Send_RNR (F=x)
}
else if (RemoteBusy == FALSE) and Pending I-frames Exist and RemWindow-Not-Full then {
    Send-Pending-I-frames (F=x)
}
else {
    Send_RR (F=x)
}
```

InitSrej—Initialize the variables used for processing SREJ as follows:

```

Clear SrejList - (remove all values)
SendRej := FALSE
BufferSeqSrej := BufferSeq
```

SaveIframeSrej—Save the received I-frame. Missing I-frame(s) will be retransmitted in response to SREJ frames. Implementations may want to save

the I-frame in its proper sequence order by leaving room for the missing I-frames.

StoreOrIgnore—If the local L2CAP entity has room to store the received I-frame then it may store it otherwise it shall discard it. If the received I-frame is stored, ExpectedTxSeq is advanced as follows:

$$\text{ExpectedTxSeq} := (\text{ExpectedTxSeq} + 1) \bmod \text{MaxTxWin}$$

PbitOutstanding—If the Transmitter state machine of the local L2CAP entity is in the WAIT_F state then return TRUE otherwise return FALSE.

Retransmit-I-frames—All the unacknowledged I-frames starting with the I-frame with TxSeq equal to the ReqSeq field of the received S-frame (REJ or RR) is retransmitted. If the P-bit of the received S-frame is 1 then the F-bit of the first I-frame sent shall be 1. If the P-bit of the received S-frame is 0 then the F-bit of the first I-frame sent shall be 0. The F-bit of all other unacknowledged I-frames sent shall be 0. The retry counter in RetryIframes[] for each retransmitted I-frame is incremented by 1. If a retry counter in RetryIframes[] is equal to MaxTransmit then the channel shall be closed. FramesSent shall be incremented by 1 for each frame sent. If the RetransTimer is not already running then perform the Start-RetransTimer action.

Retransmit-Requested-I-frame—The unacknowledged I-frame with TxSeq equal to the ReqSeq field of the received S-frame (SREJ) is retransmitted. If the P-bit of the received S-frame is 1 then the F-bit of the retransmitted I-frame shall be 1. If the P-bit of the received S-frame is 0 then the F-bit of the retransmitted I-frame shall be 0. The retry counter in RetryIframes[] corresponding to the retransmitted I-frame is incremented by 1. If the RetransTimer is not already running then perform the Start-RetransTimer action.

Send-Pending-I-frames (F=x)—send all pending I-frames that can be sent without exceeding the receiver's TxWindow using the Send-Data action. If a value for the F-bit is specified then the F-bit of the first I-frame sent shall be set to the specified value and the F-bit of all other I-frames sent shall be set to 0. If no value for the F-bit is specified then all I-frames sent shall have the F-bit set to 0. Pending I-frames are I-frames that have been given to the L2CAP entity by the upper layer but have not yet been transmitted. If one or more I-frames are sent and the RetransTimer is not already running then perform the Start-RetransTimer action.

Close Channel—Close the L2CAP channel as described in [Section 4.6](#).

Ignore—the event may be silently discarded.

PopSrejList—Remove and discard the TxSeq from the head of SrejList.

Data-IndicationSrej—If the received I-frame fills a gap in a sequence of saved I-frames then all the saved I-frames in the sequence are passed to the SDU

reassembly function. For the purpose of the state machine this operation is completed immediately. For example if the TxSeq of saved I-frames before receiving an I-frame is 2, 3, 5, 6, 9 and the received I-frame has a TxSeq of 4 then it fills the gap between 3 and 5 so the sequence 2, 3, 4, 5, 6 can be passed to the SDU reassembly function. When the I-frames are actually removed from the L2CAP entity receive buffers either by being processed immediately or when pulled by the SDU reassembly function, BufferSeqSrej is advanced as follows:

BufferSeqSrej := (BufferSeqSrej + 1) mod MaxTxWin

8.6.5.7 XMIT State Table

Event	Condition	Action	Next State
Data-Request	RemoteBusy = FALSE and RemWindow-Not-Full	Send-Data	XMIT
Data-Request	RemoteBusy = TRUE or RemWindow-Full	Pend-Data	XMIT
Local-Busy-Detected		LocalBusy := TRUE	XMIT
		LocalBusy := TRUE Send RNR	XMIT
Local-Busy-Clear	RNRsent = TRUE	LocalBusy := FALSE RNRsent := FALSE Send RR(P=1) RetryCount := 1 Stop-RetransTimer Start-MonitorTimer	WAIT_F
Local-Busy-Clear	RNRsent = FALSE	LocalBusy := FALSE RNRsent := FALSE	XMIT
Recv ReqSeqAndFbit		Process-ReqSeq	XMIT
Recv Fbit			XMIT
RetransTimer-Expires		Send RRorRNR(P=1) RetryCount := 1 Start-MonitorTimer	WAIT_F

Table 8.4: XMIT state table

8.6.5.8 WAIT_F State Table

Event	Condition	Action	Next State
Data-Request		Pend-Data	WAIT_F
Recv ReqSeqAndFbit	F = 1	Process-ReqSeq Stop-MonitorTimer If UnackedFrames > 0 then { Start-RetransTimer }	XMIT
Recv ReqSeqAndFbit	F = 0	Process-ReqSeq	WAIT_F
Recv Fbit	F = 1	Stop-MonitorTimer If UnackedFrames > 0 then { Start-RetransTimer }	XMIT
Recv Fbit	F = 0		WAIT_F
MonitorTimer-Expires	RetryCount < Max-Transmit	RetryCount := RetryCount+1 Send RR or RNR(P=1) Start-MonitorTimer	WAIT_F
MonitorTimer-Expires	RetryCount ≥ Max-Transmit	Close Channel	

Table 8.5: WAIT_F State Table

8.6.5.9 RECV State Table

Event	Condition	Action	Next State
Recv I-frame (F=0)	With-Expected-TxSeq and With-Valid-ReqSeq and With-Valid-F-bit and LocalBusy = FALSE	Increment-ExpectedTxSeq PassToTx Data-Indication Send-Ack(F=0)	RECV
Recv I-frame (F=1)	With-Expected-TxSeq and With-Valid-ReqSeq and With-Valid-F-bit and LocalBusy = FALSE	Increment-ExpectedTxSeq PassToTx Data-Indication If RejActioned = FALSE then { Retransmit-I-frames Send-Pending-I-frames } else { RejActioned := FALSE } Send-Ack(F=0)	RECV
Recv I-frame	With-duplicate-TxSeq and With-Valid-ReqSeq and With-Valid-F-bit and LocalBusy = FALSE	PassToTx	RECV
Recv I-frame	With-unexpected-TxSeq and With-Valid-ReqSeq and With-Valid-F-bit and LocalBusy = FALSE	PassToTx SendREJ	REJ_SENT
		PassToTx InitSrej SaveIframeSrej SendSREJ	SREJ_SENT
Recv I-frame	With-Expected-TxSeq and With-Valid-ReqSeq and With-Valid-F-bit and LocalBusy = TRUE	PassToTx StoreOrIgnore	RECV
Recv I-frame	With-Valid-ReqSeq and Not-With_Expected_TxSeq and With-Valid-F-bit and LocalBusy = TRUE	PassToTx	RECV
Recv RNR (P=0)	With-Valid-ReqSeq and With-Valid-F-bit	RemoteBusy := TRUE PassToTx Stop-RetransTimer	RECV

Table 8.6: RECV_State table

Event	Condition	Action	Next State
Recv RNR (P=1)	With-Valid-ReqSeq and With-Valid-F-bit	RemoteBusy := TRUE PassToTx Stop-RetransTimer Send RRorRNR (F=1)	RECV
Recv RR(P=0)(F=0)	With-Valid-ReqSeq and With-Valid-F-bit	PassToTx If RemoteBusy = TRUE and UnackedFrames > 0 then { Start-RetransTimer } RemoteBusy := FALSE Send-Pending-I-frames	RECV
Recv RR(F=1)	With-Valid-ReqSeq and With-Valid-F-bit	RemoteBusy := FALSE PassToTx If RejActioned = FALSE then { Retransmit-I-frames } else { RejActioned := FALSE } Send-Pending-I-frames	RECV
Recv RR(P=1)	With-Valid-ReqSeq and With-Valid-F-bit	PassToTx Send IorRRorRNR(F=1)	RECV
Recv REJ (F=0)	With-Valid-ReqSeq- Retrans and RetryIframes[i] < Max- Transmit and With-Valid-F-bit	RemoteBusy := FALSE PassToTx Retransmit-I-frames Send-Pending-I-frames If PbitOutstanding then { RejActioned := TRUE }	RECV
Recv REJ (F=1)	With-Valid-ReqSeq- Retrans and RetryIframes[i] < Max- Transmit and With-Valid-F-bit	RemoteBusy := FALSE PassToTx If RejActioned = FALSE then { Retransmit-I-frames } else { RejActioned := FALSE } Send-Pending-I-frames]	RECV

Table 8.6: RECV_State table (Continued)

Logical Link Control and Adaptation Protocol Specification

Event	Condition	Action	Next State
Recv SREJ (P=0) (F=0)	With-Valid-ReqSeq- Retrans and Retryframes[i] < Max- Transmit and With-Valid-F-bit	RemoteBusy := FALSE PassToTxFbit Retransmit-Requested-I-frame If PbitOutstanding then { SrejActioned := TRUE SrejSaveReqSeq = ReqSeq }	RECV
Recv SREJ (P=0) (F=1)	With-Valid-ReqSeq- Retrans and Retryframes[i] < Max- Transmit and With-Valid-F-bit	RemoteBusy := FALSE PassToTxFbit If SrejActioned = TRUE and SrejSaveReqSeq = ReqSeq then { SrejActioned := FALSE } else { Retransmit-Requested-I- frame }	RECV
Recv SREJ(P=1)	With-Valid-ReqSeq- Retrans and Retryframes[i] < Max- Transmit and With-Valid-F-bit	RemoteBusy := FALSE PassToTx Retransmit-Requested-I-frame Send-Pending-I-frames If PbitOutstanding then { SrejActioned = TRUE SrejSaveReqSeq := ReqSeq }	RECV
Recv REJ	With-Valid-ReqSeq- Retrans and Retryframes[i] ≥ Max- Transmit	Close Channel	
RECV SREJ	With-Valid-ReqSeq- Retrans and Retryframes[i] >= Max- Transmit	Close Channel	
Recv I-frame	(With-Invalid-TxSeq and TxWindow >(MaxTxWin/2) or With-Invalid-ReqSeq	Close Channel	
Recv I-frame	With-Invalid-TxSeq and TxWindow ≤ (MaxTx- Win/2)	Close Channel Ignore	RECV
Recv RR or RNR	With-Invalid-ReqSeq	Close Channel	

Table 8.6: *RECV_State table (Continued)*

Event	Condition	Action	Next State
Recv REJorSREJ	With-Invalid-ReqSeq-Retrans	Close Channel	
Recv frame		Ignore	RECV

Table 8.6: RECV_State table (Continued)

8.6.5.10 REJ_SENT State Table

Event	Condition	Action	Next State
Recv I-frame (F=0)	With-Expected-TxSeq and With-Valid-ReqSeq and With-Valid-F-bit	Increment-ExpectedTxSeq PassToTx Data-Indication Send-Ack (F=0)	RECV
Recv I-frame (F=1)	With-Expected-TxSeq and With-Valid-ReqSeq and With-Valid-F-bit	Increment-ExpectedTxSeq PassToTx Data-Indication If RejActioned = FALSE then { Retransmit I-frames Send-Pending-I-frames } else { RejActioned := FALSE } Send-Ack (F=0)	RECV
Recv I-frame	With-Unexpected-TxSeq and With-Valid-ReqSeq and With-Valid-F-bit	PassToTx	REJ_SENT
Recv RR (F=1)	With-Valid-ReqSeq and With-Valid-F-bit	RemoteBusy := FALSE PassToTx If RejActioned = FALSE then { Retransmit-I-frames } else { RejActioned := FALSE } Send-Pending-I-frames	REJ_SENT

Table 8.7: REJ_SENT State table

Event	Condition	Action	Next State
Recv RR (P=0)(F=0)	With-Valid-ReqSeq and With-Valid-F-bit	PassToTx If RemoteBusy = TRUE and UnackedFrames > 0 then { Start-RetransTimer } RemoteBusy := FALSE Send-Ack (F=0)	REJ_SENT
Recv RR (P=1)	With-Valid-ReqSeq and With-Valid-F-bit	PassToTx If RemoteBusy = TRUE and UnackedFrames > 0 then { Start-RetransTimer } RemoteBusy := FALSE Send RR (F=1)	REJ_SENT
Recv RNR (P=1)	With-Valid-ReqSeq and With-Valid-F-bit	RemoteBusy := TRUE PassToTx Send RR (F=1)	REJ_SENT
Recv RNR (P=0)	With-Valid-ReqSeq and With-Valid-F-bit	RemoteBusy := TRUE PassToTx Send RR (F=0)	REJ_SENT
Recv REJ (F=0)	With-Valid-ReqSeq - Retrans and RetryIframes[i] < Max-Transmit and With-Valid-F-bit	RemoteBusy := FALSE PassToTx Retransmit-I-frames Send-Pending-I-frames If PbitOutstanding then { RejActioned := TRUE }	REJ_SENT
Recv REJ (F=1)	With-Valid-ReqSeq - Retrans and RetryIframes[i] < Max-Transmit and With-Valid-F-bit	RemoteBusy := FALSE PassToTx If RejActioned = FALSE then { Retransmit-I-frames } else { RejActioned := FALSE } Send-Pending-I-frames	REJ_SENT

Table 8.7: REJ_SENT State table (Continued)

Logical Link Control and Adaptation Protocol Specification

Event	Condition	Action	Next State
Recv SREJ (P=0) (F=0)	With-Valid-ReqSeq- Retrans and Retryframes[i] < Max- Transmit and With-Valid-F-bit	RemoteBusy := FALSE PassToTxFbit Retransmit-Requested-I-frame If PbitOutstanding then { SrejActioned := TRUE SrejSaveReqSeq := ReqSeq }	REJ_SENT
Recv SREJ (P=0) (F=1)	With-Valid-ReqSeq- Retrans and Retryframes[i] < Max- Transmit and With-Valid-F-bit	RemoteBusy := FALSE PassToTxFbit If SrejActioned = TRUE and SrejSaveReqSeq = ReqSeq then { SrejActioned := FALSE } else { Retransmit-Requested-I- frame }	REJ_SENT
Recv SREJ (P=1)	With-Valid-ReqSeq- Retrans and Retryframes[i] < Max- Transmit and With-Valid-F-bit	RemoteBusy := FALSE PassToTx Retransmit-Requested-I- frames Send-Pending-I-frames If PbitOutstanding then { SrejActioned := TRUE SrejSaveReqSeq := ReqSeq }	REJ_SENT
Recv REJ	With-Valid-ReqSeq- Retrans and Retryframes[i] ≥ Max- Transmit	Close Channel	
Recv SREJ (P=0)	With-Valid-ReqSeq- Retrans and Retryframes[i] ≥ Max- Transmit	Close Channel	
RECV SREJ (P=1)	With-Valid-ReqSeq- Retrans and Retryframes[i] ≥ Max- Transmit	Close Channel	

Table 8.7: REJ_SENT State table (Continued)

Event	Condition	Action	Next State
Recv I-frame	(With-Invalid-TxSeq and TxWindow > (MaxTx-Win/2) or With-Invalid-ReqSeq)	Close Channel	
Recv RRorRNR	With-Invalid-ReqSeq	Close Channel	
RecvREJorSREJ	With-Invalid-ReqSeq-Retrans	Close Channel	
Recv I-frame	With-Invalid-TxSeq and TxWindow £ (MaxTx-Win/2)	Close Channel Ignore	REJ_SENT
Recv frame		Ignore	REJ_SENT

Table 8.7: REJ_SENT State table (Continued)

8.6.5.11 SREJ_SENT State Table

Event	Condition	Action	Next State
Recv I-frame	With-Expected-TxSeq-Srej and With-Valid-ReqSeq and With-Valid-F-bit and SendRej = FALSE and SrejList = 1	SavelframeSrej PopSrejList PassToTx Data-IndicatioSrej BufferSeq := BufferSeqSrej Send-Ack (F=0)	RECV
Recv I-frame	With-Expected-TxSeq-Srej and With-Valid-ReqSeq and With-Valid-F-bit and SendRej = TRUE and SrejList = 1	SavelframeSrej PopSrejList PassToTx Data-IndicationSrej BufferSeq := BufferSeqSrej Send REJ	REJ_SENT
Recv I-frame	With-Expected-TxSeq-Srej and With-Valid-ReqSeq and With-Valid-F-bit and SrejList > 1	SavelframeSrej PopSrejList PassToTx Data-IndicationSrej	SREJ_SENT
Recv I-frame	With-Expected-TxSeq and With-Valid-ReqSeq and With-Valid-F-bit	SavelframeSrej Increment-ExpectedTxSeq PassToTx	SREJ_SENT
Recv I-frame	With-Unexpected-TxSeq and With-Valid-ReqSeq and With-Valid-F-bit and SendRej = FALSE	SavelframeSrej PassToTx Send SREJ PassToTx SendRej := TRUE	SREJ_SENT
Recv I-frame	With-Unexpected-TxSeq and With-Valid-ReqSeq and With-Valid-F-bit and SendRej = TRUE	PassToTx	SREJ_SENT
Recv I-frame	With-Unexpected-TxSeq-Srej and With-Valid-ReqSeq and With-Valid-F-bit	SavelframeSrej PassToTx Send SREJ(SrejList)	SREJ_SENT

Table 8.8: SREJ_SENT State Table

Logical Link Control and Adaptation Protocol Specification

Event	Condition	Action	Next State
Recv I-frame	With-duplicate-TxSeq-Srej and With-Valid-ReqSeq and With-Valid-F-bit	PassToTx	SREJ_SENT
Recv RR(F=1)	With-Valid-ReqSeq and With-Valid-F-bit	RemoteBusy := FALSE PassToTx If RejActioned = FALSE then { Retransmit-I-frames } else { RejActioned := FALSE } Send-Pending-I-frames	SREJ_SENT
Recv RR(P=1)	With-Valid-ReqSeq and With-Valid-F-bit	PassToTx If RemoteBusy = TRUE and UnackedFrames > 0 then { Start-RetransTimer } RemoteBusy := FALSE Send SREJ(SrejList-tail)(F=1)	SREJ_SENT
Recv RR(P=0)(F=0)	With-Valid-ReqSeq and With-Valid-F-bit	PassToTx If RemoteBusy = TRUE and UnackedFrames > 0 then { Start-RetransTimer } RemoteBusy := FALSE Send-Ack(F=0)	SREJ_SENT
Recv RNR(P=1)	With-Valid-ReqSeq and With-Valid-F-bit	RemoteBusy := TRUE PassToTx Send SREJ(SrejList-tail)(F=1)	SREJ_SENT
Recv RNR(P=0)	With-Valid-ReqSeq and With-Valid-F-bit	RemoteBusy := TRUE PassToTx Send RR(F=0)	SREJ_SENT
Recv REJ (F=0)	With-Valid-ReqSeq- Retrans and RetryIframes[i] < Max- Transmit and With-Valid-F-bit	RemoteBusy := FALSE PassToTx Retransmit-I-frames Send-Pending-I-frames If PbitOutstanding then { RejActioned := TRUE }	SREJ_SENT

Table 8.8: SREJ_SENT State Table (Continued)

Logical Link Control and Adaptation Protocol Specification

Event	Condition	Action	Next State
Recv REJ (F=1)	With-Valid-ReqSeq- Retrans and Retryframes[i] < Max- Transmit and With-Valid-F-bit	RemoteBusy := FALSE PassToTx If RejActioned = FALSE then { Retransmit-I-frames } else { RejActioned := FALSE } Send-Pending-I-frames	SREJ_SENT
Recv SREJ(P=0) (F=0)	With-Valid-ReqSeq- Retrans and Retryframes[i] < MaxTransmit and With-Valid-F-bit	RemoteBusy := FALSE PassToTxFbit Retransmit-Requested-I-frame If PbitOutstanding then { SrejActioned := TRUE SrejSaveReqSeq = ReqSeq }	SREJ_SENT
Recv SREJ(P=0) (F=1)	With-Valid-ReqSeq- Retrans and Retryframes[i] < MaxTransmit and With-Valid-F-bit	RemoteBusy := FALSE PassToTxFbit If SrejActioned = TRUE and SrejSaveReqSeq = ReqSeq then { SrejActioned := FALSE } else { Retransmit-Requested-I- frame }	SREJ_SENT
Recv SREJ(P=1)	With-Valid-ReqSeq- Retrans and Retryframes[i] < MaxTransmit and With-Valid-F-bit	RemoteBusy := FALSE PassToTx Retransmit-Requested-I-frame Send-Pending-I-frames If PbitOutstanding then { SrejActioned := TRUE SrejSaveReqSeq = ReqSeq }	SREJ_SENT
Recv REJ	With-Valid-ReqSeq- Retrans and Retryframes[i] ≥ MaxTransmit	Close Channel	

Table 8.8: SREJ_SENT State Table (Continued)

Event	Condition	Action	Next State
Recv SREJ(P=0)	With-Valid-ReqSeqRetrans and Retryframes[i] ≥ Max-Transmit	Close Channel	
Recv SREJ(P=1)	With-Valid-ReqSeqRetrans and Retryframes[i] ≥ Max-Transmit	Close Channel	
Recv I-frame	(With-Invalid-TxSeq and TxWindow > (Max-TxWin/2) or With-Invalid-ReqSeq	Close Channel	
Recv RR or RNR	With-Invalid-ReqSeq	Close Channel	
Recv REJ or SREJ	With-Invalid-ReqSeq-Retrans	Close Channel	
Recv I-frame	With-Invalid-TxSeq and TxWindow ≤ (Max-TxWin/2)	Close Channel Ignore	SREJ_SENT
Recv frame	-	Ignore	SREJ_SENT

Table 8.8: SREJ_SENT State Table (Continued)

8.7 STREAMING MODE

When a link is configured to work in Streaming Mode, the frame format for outgoing data is the same as for Enhanced Retransmission mode but frames are not acknowledged. Therefore

- RR, REJ, RNR and SREJ frames shall not be used in Streaming Mode.
- The F-bit shall always be set to zero in the transmitter, and shall be ignored in the receiver.
- the MonitorTimer and RetransmissionTimer shall not be used in Streaming mode.

A channel configured to work in Streaming mode shall be configured with a finite value for the Flush Timeout on the transmitter.

8.7.1 Transmitting I-frames

When transmitting a new I-frame the control field parameter ReqSeq shall be set to 0, TxSeq shall be set to NextTXSeq and NextTXSeq shall be incremented by one.

8.7.2 Receiving I-frames

Upon receipt of a valid I-frame with TxSeq equal to ExpectedTxSeq, the frame shall be made available to the reassembly function. ExpectedTxSeq shall be incremented by one.

Upon receipt of a valid I-frame with an out-of-sequence TxSeq (see [Section 8.7.3.1 on page 198](#)) all frames with a sequence number less than TxSeq shall be assumed lost and marked as missing. The missing I-frames are in the range from ExpectedTxSeq (the frame that the device was expecting to receive) up to and including TxSeq - 1. ExpectedTxSeq shall be set to TxSeq +1. The received I-frame shall be made available for pulling by the reassembly function. The ReqSeq shall be ignored.

Note: It is possible for a complete window size of I-frames to be missing and thus, no missing I-frames are detected. For example, when a window size of 63 is used this situation occurs when 63 I-frames in a row are missing. If the ability to not detect missing I-frames will cause problems for an application, it is recommended that the Extended Window Size option be used.

If there is no buffer space for the received I-frame an existing I-frame (i.e. the oldest) shall be discarded (flushed) freeing up buffer space for the new I-frame. The discarded I-frame shall be marked as missing.

8.7.3 Exception Conditions

Exception conditions may occur as the result of physical layer errors or L2CAP procedural errors. The error recovery procedures which are available following the detection of an exception condition at the L2CAP layer in Streaming mode are defined in this section.

8.7.3.1 *TxSeq Sequence error*

A TxSeq sequence error exception condition occurs in the receiver when a valid I-frame is received which contains a TxSeq value which is not equal to the expected value, thus TxSeq is not equal to ExpectedTxSeq.

The out-of-sequence I-frame is identified by a TxSeq that is greater than ExpectedTxSeq ($\text{TxSeq} > \text{ExpectedTXSeq}$). The ReqSeq shall be ignored. The missing I-frame(s) are considered lost and ExpectedTXSeq is set equal to $\text{TxSeq}+1$ as specified in [Section 8.7.2 on page 197](#). The missing I-frame(s) are reported as lost to the SDU reassembly function.

9 PROCEDURE FOR AMP CHANNEL CREATION AND HANDLING

When an AMP is used, the procedures defined in this chapter shall be used. Enhanced Retransmission mode is used on all reliable channels to ensure a reliable move channel operation and to ensure that user traffic with an infinite flush timeout is reliable even for AMPs that do not support infinite flush timeout. Streaming mode is used on all channels configured with a finite flush timeout.

9.1 CREATE CHANNEL

Create Channel Request is used to create a new L2CAP channel over a Controller. The channel will have the quality of service specified by a pair of Extended Flow Specifications exchanged during channel configuration. Channels shall only be created over the BR/EDR controller if both L2CAP entities support Extended Flow Specification for BR/EDR. After configuration each device will have an outgoing (transmit traffic) flow specification and an incoming (received traffic) flow specification. Note: Where Extended Flow Specification for BR/EDR is not supported by one or both L2CAP entities, an L2CAP channel can be established with Connection Request.

All Best Effort channels created over the same AMP Physical link are aggregated over a single AMP Logical link, so if a Best Effort channel is requested over an AMP Physical link where a Best Effort logical link already exists then the Best Effort Extended Flow Specifications are aggregated into one pair of flow specifications and the flow specification of the AMP Logical link is modified using the HCI Flow Spec Modify command (in devices that support HCI). [Section 7.8](#) describes how Best Effort Flow specifications are aggregated. A logical link is created for each Guaranteed channel and one for the first Best Effort channel of the AMP.

All channels created over the same BR/EDR physical link are aggregated over a single logical link. Aggregation of Best Effort Extended Flow Specifications is not necessary for channels created over ACL-U logical links so the term “modify the logical link” in the algorithm descriptions results in “no action” when the logical link is ACL-U. The L2CAP layer should perform admission control for Guaranteed channels created on ACL-U logical links (see [Section 7.10](#) for a description of L2CAP admission control). The term “create a logical link” in the algorithm description refers to L2CAP performing admission control when the logical link is ACL-U.

Basic Algorithm:

1. Extended Flow specifications and other configuration parameters shall be exchanged via the Lockstep Configuration procedure.
2. If the service type of the Extended Flow Specifications are Best Effort (including the case where one is Best Effort and the other is “No Traffic”) then the Best Effort Algorithm shall be used otherwise the Guaranteed Algorithm shall be used. (See [Section 5.6](#) for rules on setting the service type of Extended Flow Specifications).

Best Effort Algorithm:

1. If one or more Best Effort channels already exist for the Controller then Goto step 3
2. Tell the Controller to create a logical link passing the Extended Flow Specifications. Goto step 5
3. Aggregate the Extended Flow Specifications with all the other Best Effort Extended Flow specifications running on the same physical link as described in [Section 7.8](#).
4. Tell the controller to modify the logical link passing the aggregated Extended Flow Specifications.
5. If the Controller accepts the create/modify request and returns success then complete the L2CAP Configuration with result = success otherwise complete the L2CAP configuration with result = “Failure - flow spec rejected”

Guaranteed Algorithm

1. Tell the Controller to create a logical link passing the Extended Flow Specifications.
2. If the Controller accepts the create request and returns success, complete the L2CAP Configuration with result = success; otherwise, complete the L2CAP configuration with result = “Failure - flow spec rejected.”

If the Controller is a BR/EDR or BR/EDR/LE Controller and the Extended Flow Specification type is Guaranteed then L2CAP should perform admission control by determining if the requested QoS can be achieved by the Controller without compromising existing Guaranteed channels running on the Controller.

An example of the creation of the first Best Effort channel or a guaranteed channel is shown in [Figure 9.1](#). An example of creation of a subsequent Best Effort channel is shown in [Figure 9.2](#).

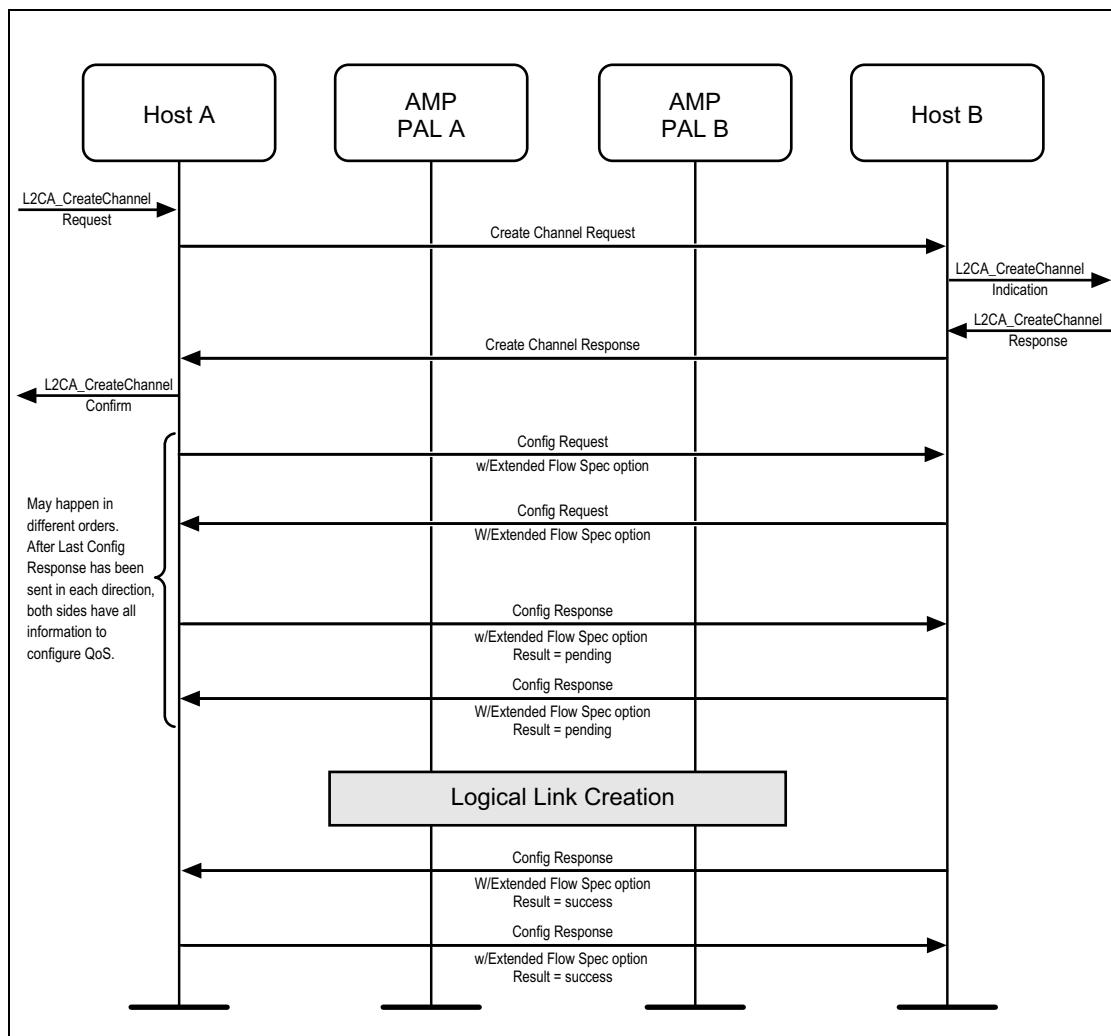


Figure 9.1: Creation of First Best Effort L2CAP channel or a Guaranteed L2CAP channel

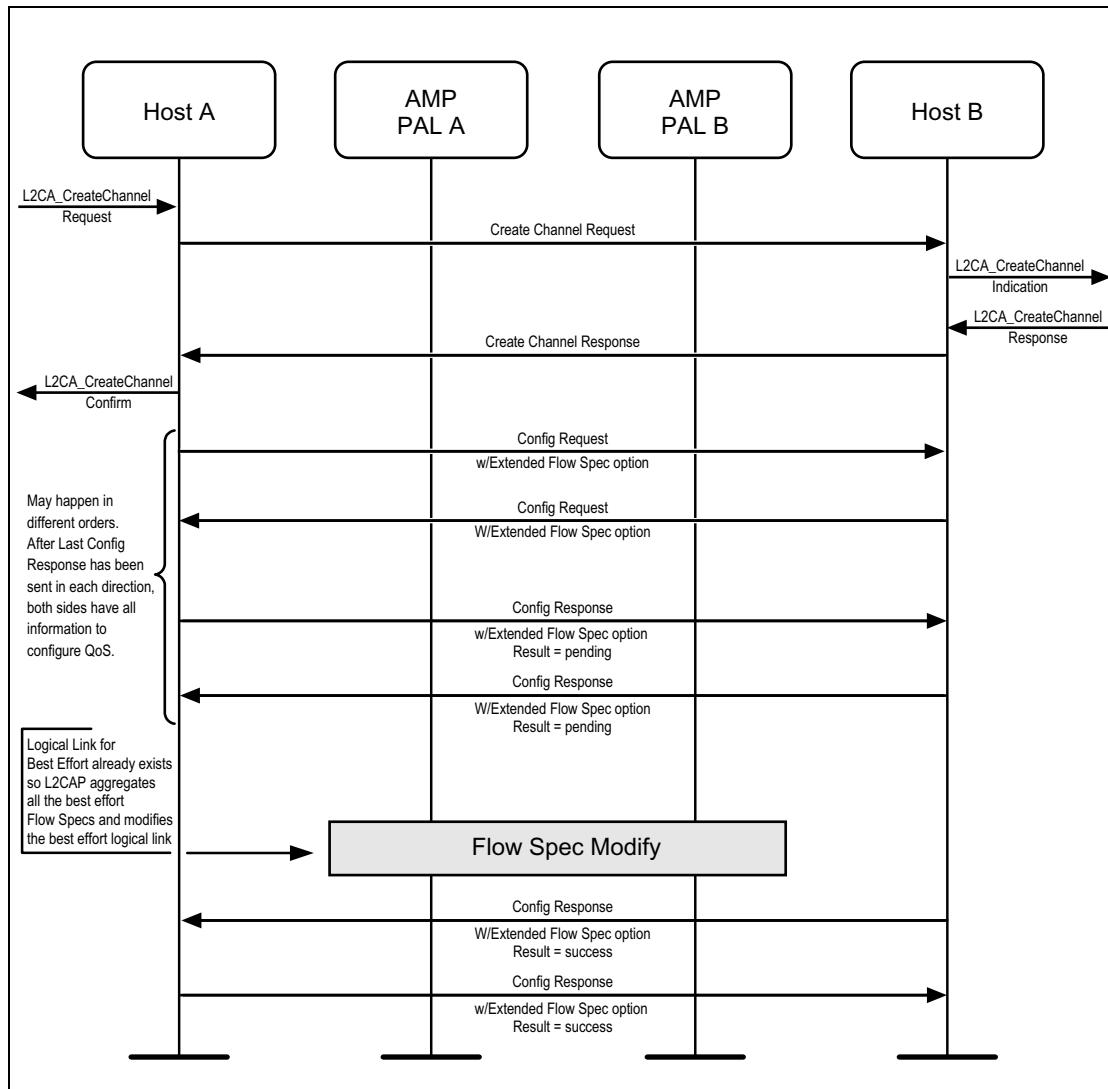


Figure 9.2: Creation of Subsequent Best Effort L2CAP channel

9.2 MOVE CHANNEL

Move Channel is used to move an L2CAP channel from one Controller to another. Moving a channel retains the existing channel configuration (MTU, QOS, etc.) and the CID. If reconfiguration is necessary then it may be done after a successful move channel operation. If a channel configured as Best Effort is moved, the Extended Flow Specification aggregate shall be recalculated (see [Section 7.8](#) for the aggregation algorithm) for both the old and new Controllers in cases where an aggregate already exists or where a channel is moved to a Controller with an existing Best Effort channel. If the Extended Flow Specification aggregate are modified for a Controller then a logical link modify operation shall be performed for that Controller.

The two procedures for moving channels are based on the mode configured for the channel. One procedure is used for channels configured with Enhanced

Retransmission mode. Another procedure is used for channels configured with Streaming mode. The procedures are described in the following sections.

Note that the terms such as “logical link create” and “logical link modify” are used in the Move procedures. For AMP Controllers these terms refer to logical link operations (e.g. QoS admission control) performed by the Controller. For BR/EDR and BR/EDR/LE Controllers these terms refer to QoS admission control performed by L2CAP on behalf of the Controller.

9.2.1 Move Channel Protocol Procedure with Enhanced Retransmission Mode

1. When the L2CAP layer on the initiating device receives the L2CA_Move_Channel.request from the upper layer (application) it shall stop sending I-frames and S-frames. It shall continue to listen on the old Controller and follow the procedures in [Section 9.2.1.1](#). The L2CAP layer on the initiating device shall send the Move Channel Request packet over the L2CAP signaling channel to the remote (responding) device.
2. When the L2CAP layer on the responding device receives the Move Channel Request packet it shall stop sending I-frames and S-frames. It shall still listen on the old Controller and follow the procedures in [Section 9.2.1.1](#). It shall send a Move Channel Response packet to the other side. If a logical link must be created or modified (or admission control is required) the Move Channel Response packet shall contain a result code of “pending.” If a logical link does not need to be created or modified and the move operation is allowed then the Move Channel Response packet shall contain a result code of “success.” Otherwise, it shall contain the appropriate “refused” result code. If the response code sent in the Move Channel Response packet is “pending” then the L2CAP layer on the responding device shall attempt to create or modify a logical link on the new Controller for the channel. When the logical link create/modify operation is complete, the L2CAP layer on the responding device shall send a Move Channel Response packet to the other side with the result code indicating the success/failure of the logical link create/modify.
3. When the L2CAP layer on the initiating device receives the Move Channel Response packet it shall check the result code. If the result code in the Move Channel Response packet is “refused” it shall send a Move Channel Confirmation packet with result code “failure” to the other side. If the result code in the Move Channel Response packet is “pending” or “success” the L2CAP layer shall stop listening on the old Controller and attempt to create/modify a logical link on the new Controller for the channel. If the result code in the Move Channel Response packet is “pending” then it shall use the ERTX timer while waiting for a Move Channel Response packet with a “non-pending” result code from the responding device. When the logical link create/modify operation is complete and the L2CAP layer on the initiating

device has received a Move Channel Response packet from the responding device with a “non-pending” result code, it shall send a Move Channel Confirmation packet to conclude the Move Channel procedure. The result code in the Move Channel Confirmation packet indicates the success/failure of its own logical channel create/modify operation and the result code in the Move Channel Response packet from the responding device. If both are success then the result code sent in the Move Channel Confirmation packet shall be “success” otherwise it shall be “failure.”

4. When the L2CAP layer on the responding device receives the Move Channel Confirmation packet it shall send a Move Channel Confirmation Response and send an L2CA_Move_Channel_Confirm.indication to the upper layer with the result code contained in the Move Channel Confirmation packet. If the result code of the Move Channel Confirmation is success the L2CAP layer on the responding device shall listen on the new Controller if the result code is failure then it shall listen on the old Controller.
5. When the L2CAP layer on the initiating device receives the Move Channel Confirmation response packet it shall send an L2CAP_Move_Channel.confirm to the upper layer with the result code passed in the Move Channel Confirmation packet. If the result code sent in the Move Channel Confirmation packet is success then the L2CAP layer on the initiating device shall send an RR(P=1) on the new Controller and shall start listening on the new Controller. Before sending the RR(P=1) the L2CAP layer on the initiating device may initiate reconfiguration by sending a Configuration Request packet. If the result sent is failure then the L2CAP layer on the initiating device shall send an RR(P=1) on the old Controller and start listening on the old Controller.
6. When the L2CAP layer on the responding device receives the RR(P=1) packet it shall send an I-frame(F=1) with a TxSeq matching the ReqSeq in the RR(P=1) or an RR(F=1) if there are no I-frames to send. If the I-frame is a retransmission and the PDU size used for the new Controller is smaller than the PDU size used for the old Controller, the responding device shall segment the retransmitted I-frame to make it fit in the new PDU size. If the L2CAP layer on the responding device receives a Configuration Request packet before the RR(P=1) it shall perform reconfiguration. When the reconfiguration is complete it shall wait for the RR(P=1). If the L2CAP layer on the responding device desires to perform reconfiguration but receives the RR(P=1) it may send a Configuration Request packet instead of the RR(F=1) or I-frame(F=1) to initiate reconfiguration. After the reconfiguration is complete it shall send the RR(F=1) or I-frame(F=1).

7. When the L2CAP layer on the initiating device receives the I-frame(F=1) or RR(F=1) from the responding device, the move is complete. If it receives a Configuration Request packet before receiving the RR(F=1) or I-frame(F=1) it shall perform reconfiguration. When reconfiguration is complete it shall wait for the RR(F=1) or I-frame(F=1).

9.2.1.1 Enhanced Retransmission Mode Procedures During a Move Operation

1. Stop sending I-frames and S-frames and cancel all timers. Set the transmitter to the XMIT state and clear all retry counters.
2. Clear or reset any SREJ_SENT and REJ_SENT state specific variables including flushing any received out-of-sequence I-frames saved as part of being in the SREJ_SENT state. Set the receiver to the RECV state.
3. Process received I-frames in the RECV state including processing the ReqSeq and passing completed SDUs to the upper layer. Only process the ReqSeq of received S-frames. Do not change state.
4. Ignore all out-of-sequence I-frames keeping note of the TxSeq of the last in-sequence I-frame received.
5. If the initiating device is in a local busy condition then it should wait to send the Move Channel Confirmation packet until the local busy condition has cleared. If the responding device is in a local busy condition then it should send a Move Channel Response packet with result “pending” upon receiving the Move Channel Request packet and wait until the local busy condition is cleared before sending a Move Channel Response packet with a “non-pending” result.
6. The ReqSeq of the RR(P=1) sent by the initiating device shall be set to the TxSeq of the next I-frame after the last in-sequence I-frame received (noted in step 4). The ReqSeq of the RR(F=1) or I-frame(F=1) sent by the responding device shall be set to the TxSeq of the next frame after the last in-sequence I-frame received (noted in step 4).
7. The new Controller or HCI transport may require smaller PDU size than the old Controller. If this is the case then the L2CAP layer shall segment all retransmitted I-frames to fit the new PDU size.
8. After sending the RR(P=1) the initiating device should start the Monitor Timer and go to the WAIT_F state.
9. Timers shall be stopped during channel reconfiguration and restarted when reconfiguration is complete.

9.2.2 Move Channel Protocol Procedure with Streaming Mode (Initiator is Data Source)

1. When the L2CAP layer on the initiating device (data source) receives the L2CA_Move_Channel.request it shall stop sending L2CAP PDUs. The L2CAP layer on the initiating device shall send a Move Channel Request packet over the L2CAP signaling channel to the remote (responding) device.
2. When the L2CAP layer on the responding device receives the Move Channel Request packet it shall still listen on the old Controller for receive frames (timing could be such that packets from the initiating device are still in transit). It shall pass completely received SDUs up to the upper layer. It shall send a Move Channel Response packet to the other side. If a logical link must be created (or admission control is required) the Move Channel Response packet shall contain a result code of “pending.” If a logical link does not need to be created and the move operation is allowed then the Move Channel Response packet shall contain a result code of “success.” Otherwise it shall contain the appropriate “refused” result code. If the response code sent in the Move Channel Response packet is “pending” then the L2CAP layer on the responding device shall attempt to create a logical link on the new Controller for the channel. When the logical link create operation is complete, the L2CAP layer on the responding device shall send a Move Channel Response packet to the other side with the result code indicating the success/failure of the logical link create operation. It shall continue to listen on the old Controller.
3. When the L2CAP layer on the initiating device receives the Move Channel Response packet it shall check the result code. If the result code in the Move Channel Response packet is “refused” it shall send a Move Channel Confirmation packet with result code “failure” to the other side. If the result code in the Move Channel Response packet is “pending” or “success” the L2CAP layer shall attempt to create a logical link on the new Controller for the channel. If the result code in the Move Channel Response packet is “pending” then it shall use the ERTX timer while waiting for a Move Channel Response packet with a “non-pending” result code from the responding device. When the logical link create operation is complete and it has received a Move Channel Response packet from the responding device with a “non-pending” result code it shall send a Move Channel Confirmation packet and wait for a Move Channel Confirmation response packet. The result code in the Move Channel Confirmation indicates the success/failure of its own logical link create operation and the result code in the Move Channel Response packet from the responding device. If both are success then the result code sent in the Move Channel Confirmation shall be success otherwise it shall be failure.

4. When the L2CAP layer on the responding device receives the Move Channel Confirmation packet it shall send a Move Channel Confirmation response packet and send an L2CA_Move_Channel_Confirm.indication to the upper layer with the result code passed in the Move Channel Confirmation packet. If the result code of the Move Channel Confirmation is success it shall listen on the new Controller. If the result code is failure it shall listen on the old Controller.
5. When the L2CAP layer on the initiating device receives the Move Channel Confirmation response packet it shall send an L2CAP_Move_Channel.confirm to the upper layer with the result code passed in the Move Channel Confirmation packet. If the result sent in the Move Channel Confirmation is success then the initiating device shall send I-frames for the channel on the new Controller otherwise it shall send I-frames for the channel on the old Controller.

9.2.3 Move Channel Protocol Procedure with Streaming Mode (Initiator is Data Sink)

1. When the L2CAP layer on initiating device (data sink) receives the L2CAP_Move_Channel.request it shall send a Move Channel Request packet over the L2CAP signaling channel to the remote (responding) device. It shall continue listening on the old Controller.
2. When the L2CAP layer on the responding device receives the Move Channel Request packet it shall stop sending L2CAP PDUs and send a Move Channel Response packet to the other side. If a logical link must be created (or admission control is required) the Move Channel Response packet shall contain a result code of “pending.” If a logical link does not need to be created and the move operation is allowed then the Move Channel Response packet shall contain a result code of “success.” Otherwise it shall contain the appropriate “refused” result code. If the result code sent in the Move Channel Response packet is “pending” then the L2CAP layer on the responding device shall attempt to create a logical link on the new Controller for the channel. When the logical link create operation is complete, the L2CAP layer on the responding device shall send a Move Channel Response packet to the other side with the result code indicating the success/failure of the logical link create.
3. When the L2CAP layer on the initiating device receives the Move Channel Response packet it shall check the result code. If the result code in the Move Channel Response packet is “refused” it shall send a Move Channel Confirmation packet with result code “failure” to the other side. If the result code in the Move Channel Response packet is “pending” or “success” the L2CAP layer shall attempt to create a logical link on the new Controller for the channel. If the result code in the Move Channel Response packet is “pending” then it shall use the ERTX timer while waiting for a Move Channel Response packet with

a “non-pending” result code from the responding device. When the logical link create operation is complete and it has received a Move Channel Response packet from the responding device with a “non-pending” result code it shall send a Move Channel Confirmation packet and wait for a Move Channel Confirmation response packet. The result code in the Move Channel Confirmation indicates the success/failure of its own logical link create operation and the result code in the Move Channel Response packet from the responding device. If both are success then the result code sent in the Move Channel Confirmation shall be success otherwise it shall be failure. If the result sent is success then the initiating device shall listen on the new Controller otherwise it shall listen on the old Controller.

4. When the L2CAP layer on the responding device receives the Move Channel Confirmation packet it shall send a Move Channel Confirmation response packet and send an L2CA_Move_Channel_Confirm.indication to the upper layer. If the result code in the Move Channel Confirmation packet is success it shall send I-frames for the channel on the new Controller. If the result code is failure it shall send I-frames on for the channel the old Controller.
5. When the L2CAP layer on the initiating device receives the Move Channel Confirmation response packet it shall send an L2CA_Move_Channel.confirm to the upper layer with the result code passed in the Move Channel Confirmation packet.

[Figure 9.3](#) and [Figure 9.4](#) show examples of the move operation with Enhanced Retransmission mode active.

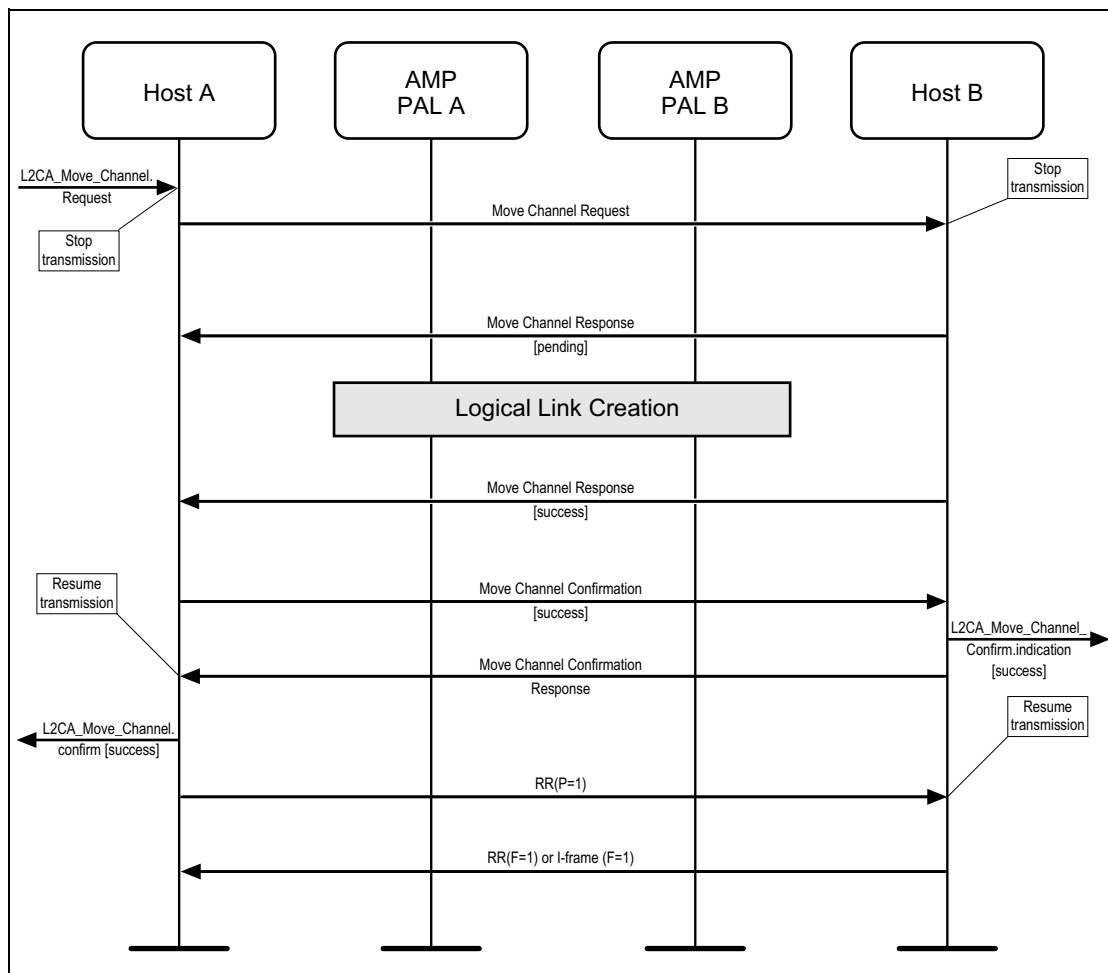


Figure 9.3: Move of a Best Effort L2CAP channel Enhanced Retransmission mode enabled to an AMP with no existing best effort logical link or move of any Guaranteed L2CAP channel

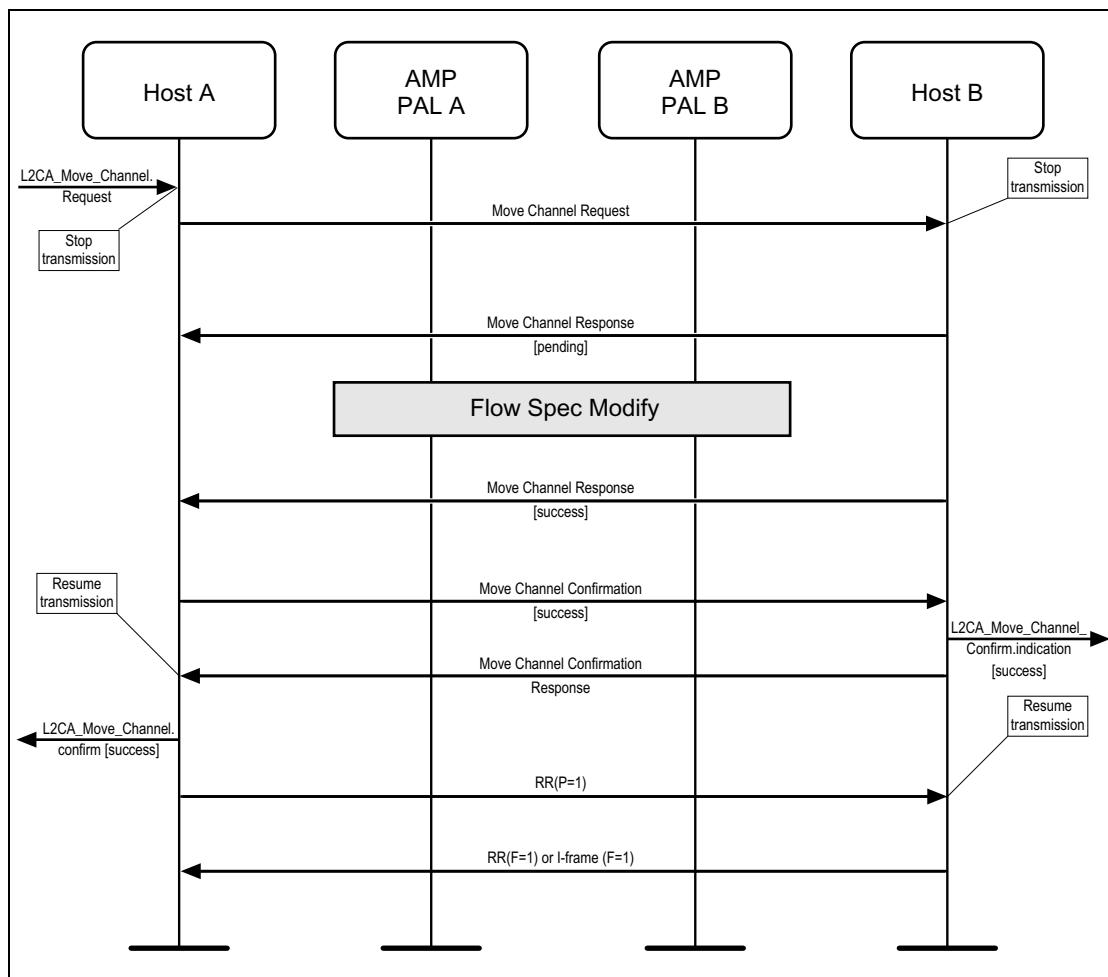


Figure 9.4: Move of a Best Effort L2CAP channel with Enhanced Retransmission mode enabled to an AMP where a best effort logical link exists

9.3 DISCONNECT CHANNEL

If a channel configured as Best Effort is disconnected the Extended Flow Specification aggregate shall be recalculated (see [Section 7.8](#) for the aggregation algorithm) unless it is the last or only channel. If this is not the last or only Best Effort channel for the logical link then a logical link modify operation shall be performed otherwise the logical link should be disconnected. The ACL-U logical link between two devices should not be disconnected until all the L2CAP channels running over all logical links between the two devices have been disconnected.

10 PROCEDURES FOR CREDIT BASED FLOW CONTROL

10.1 LE CREDIT BASED FLOW CONTROL MODE

LE Credit Based Flow Control Mode is used for LE L2CAP connection oriented channels with flow control using a credit based scheme for L2CAP data (i.e. not signaling packets).

The number of credits (LE-frames) that can be received by a device on an L2CAP channel is determined during connection establishment. LE-frames shall only be sent on an L2CAP channel if the device has a credit count greater than zero for that L2CAP channel. For each LE-frame sent the device decreases the credit count for that L2CAP channel by one. The peer device may return credits for an L2CAP channel at any time by sending an LE Flow Control Credit packet. When a credit packet is received by a device it shall increment the credit count for that L2CAP channel by the value of the Credits field in this packet. The number of credits returned for an L2CAP channel may exceed the initial credits provided in the LE Credit Based Connection Request or Response packet. The device sending the LE Flow Control Credit packet shall ensure that the number of credits returned for an L2CAP channel does not cause the credit count to exceed 65535. The device receiving the credit packet shall disconnect the L2CAP channel if the credit count exceeds 65535. The device shall also disconnect the L2CAP channel if it receives an LE-frame on an L2CAP channel from the peer device that has a credit count of zero. If a device receives an LE Flow Control Credit packet with credit value set to zero, the packet shall be ignored. A device shall not send credit values of zero in LE Flow Control Credit packets.

If a connection request is received and there is insufficient authentication between the two devices, the connection shall be rejected with a result value of “Connection refused - insufficient authentication”. If a connection request is received and there is insufficient authorization between the two devices, the connection shall be rejected with a result value of “Connection refused - insufficient authorization”. If a connection request is received and the encryption key size is too short, the connection shall be rejected with a result value of “Connection refused – insufficient encryption key size”.

Note: When encryption is not enabled, the result value “Connection refused – insufficient authentication” does not indicate that MITM protection is required.



APPENDIX A CONFIGURATION MSCs

The examples in this appendix describe a sample of the multiple possible configuration scenarios that might occur.

[Figure A.1](#) illustrates the basic configuration process. In this example, the devices exchange MTU information. All other values are assumed to be default.

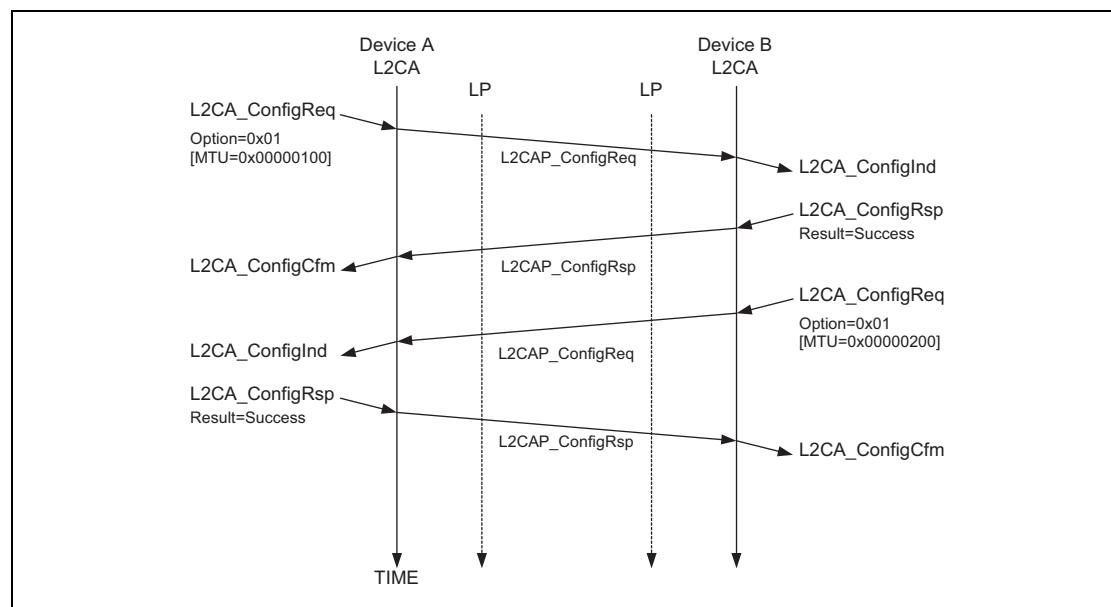


Figure A.1: Basic MTU exchange

Figure A.2 illustrates how two devices interoperate even though one device supports more options than the other does. Device A is an upgraded version. It uses a hypothetically defined option type 0x20 for link-level security. Device B rejects the command using the Configuration Response packet with result 'unknown parameter' informing Device A that option 0x20 is not understood. Device A then resends the request omitting option 0x20. Device B notices that it does not need to such a large MTU and accepts the request but includes in the response the MTU option informing Device A that Device B will not send an L2CAP packet with a payload larger than 0x80 octets over this channel. On receipt of the response, Device A could reduce the buffer allocated to hold incoming traffic.

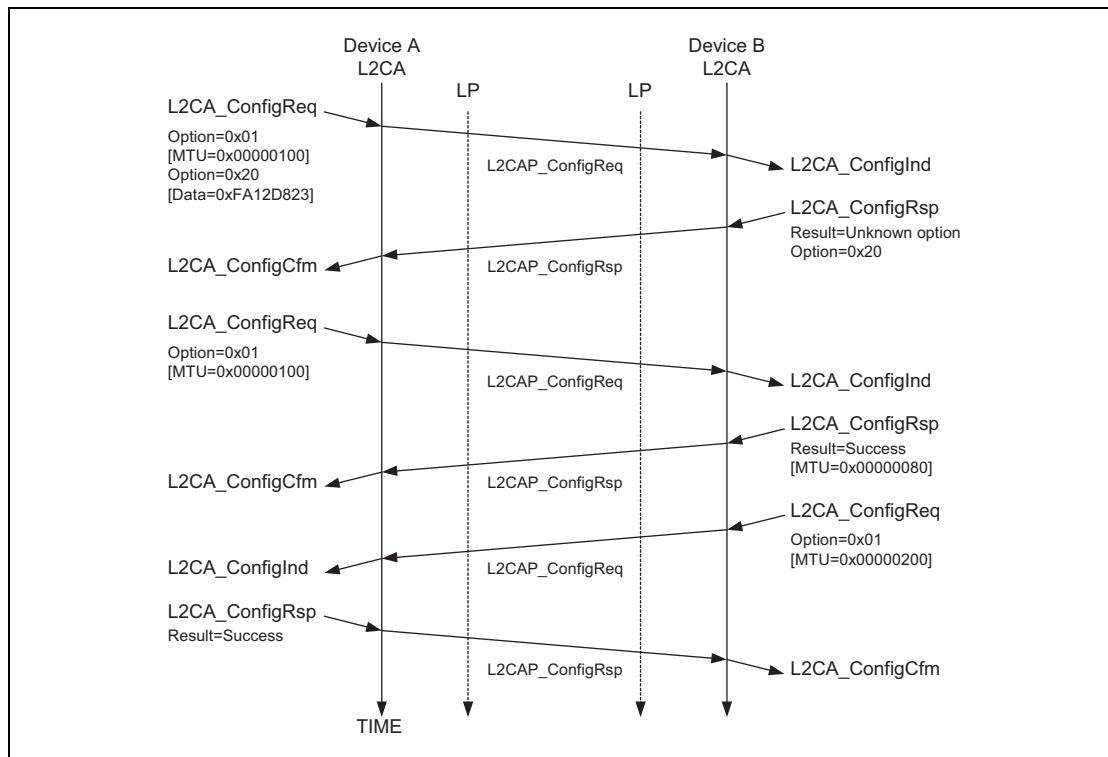


Figure A.2: Dealing with Unknown Options

Figure A.3 illustrates an unsuccessful configuration request. There are two problems described by this example. The first problem is that the configuration request is placed in an L2CAP packet that cannot be accepted by the remote device, due to its size. The remote device informs the sender of this problem using the Command Reject message. Device A then resends the configuration options using two smaller L2CAP_ConfigReq messages.

The second problem is an attempt to configure a channel with an invalid CID. For example device B may not have an open connection on that CID (0x01234567 in this example case).

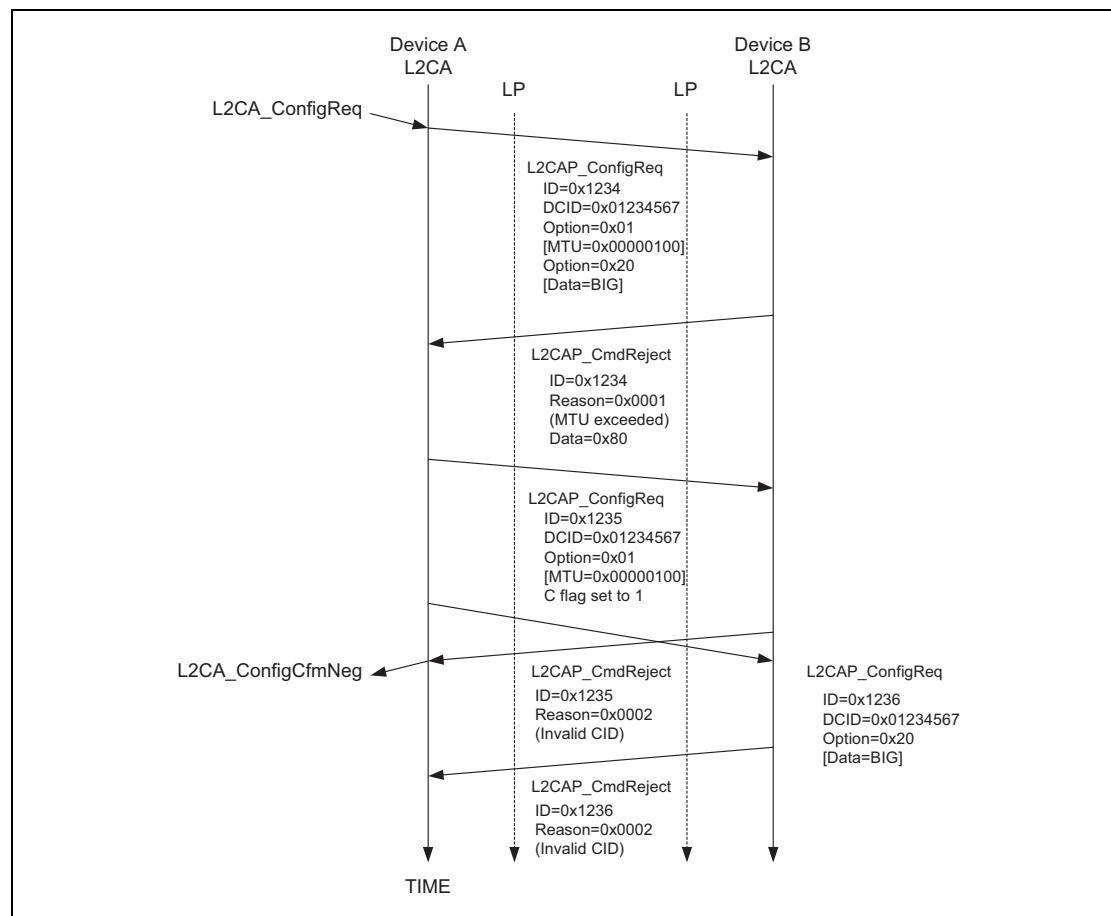


Figure A.3: Unsuccessful Configuration Request

Core System Package [Host volume]

Part B

SERVICE DISCOVERY PROTOCOL (SDP) SPECIFICATION

This specification defines a protocol for locating services provided by or available through a Bluetooth device.

CONTENTS

1	Introduction	219
1.1	General Description	219
1.2	Motivation	219
1.3	Requirements	219
1.4	Non-requirements and Deferred Requirements.....	220
1.5	Conventions.....	221
1.5.1	Bit And Byte Ordering Conventions	221
2	Overview	222
2.1	SDP Client-Server Architecture	222
2.2	Service Record	223
2.3	Service Attribute	225
2.3.1	Attribute ID	225
2.3.2	Attribute Value.....	226
2.4	Service Class.....	226
2.4.1	A Printer Service Class Example	227
2.5	Searching for Services	227
2.5.1	UUID	227
2.5.2	Service Search Patterns	228
2.6	Browsing for Services	228
2.6.1	Example Service Browsing Hierarchy.....	229
3	Data Representation	231
3.1	Data Element	231
3.2	Data Element Type Descriptor.....	231
3.3	Data Element Size Descriptor.....	232
3.4	Data Element Examples	233
4	Protocol Description	234
4.1	Transfer Byte Order	234
4.2	Protocol Data Unit Format	234
4.3	Partial Responses and Continuation State	236
4.4	Error Handling	236
4.4.1	SDP_ErrorResponse PDU	237
4.5	ServiceSearch Transaction.....	238
4.5.1	SDP_ServiceSearchRequest PDU	238
4.5.2	SDP_ServiceSearchResponse PDU	239
4.6	ServiceAttribute Transaction.....	241
4.6.1	SDP_ServiceAttributeRequest PDU	241
4.6.2	SDP_ServiceAttributeResponse PDU	243
4.7	ServiceSearchAttribute Transaction	244

4.7.1	SDP_ServiceSearchAttributeRequest PDU	244
4.7.2	SDP_ServiceSearchAttributeResponse PDU	246
5	Service Attribute Definitions.....	248
5.1	Universal Attribute Definitions	248
5.1.1	ServiceRecordHandle Attribute.....	248
5.1.2	ServiceClassIDList Attribute	249
5.1.3	ServiceRecordState Attribute.....	249
5.1.4	ServiceID Attribute	249
5.1.5	ProtocolDescriptorList Attribute	250
5.1.6	AdditionalProtocolDescriptorList Attribute.....	251
5.1.7	BrowseGroupList Attribute	252
5.1.8	LanguageBaseAttributeIDList Attribute.....	252
5.1.9	ServiceInfoTimeToLive Attribute	253
5.1.10	ServiceAvailability Attribute.....	254
5.1.11	BluetoothProfileDescriptorList Attribute	254
5.1.12	DocumentationURL Attribute	255
5.1.13	ClientExecutableURL Attribute	255
5.1.14	IconURL Attribute.....	256
5.1.15	ServiceName Attribute	256
5.1.16	ServiceDescription Attribute.....	257
5.1.17	ProviderName Attribute.....	257
5.1.18	Reserved Universal Attribute IDs.....	257
5.2	ServiceDiscoveryServer Service Class Attribute Definitions ...	258
5.2.1	ServiceRecordHandle Attribute.....	258
5.2.2	ServiceClassIDList Attribute	258
5.2.3	VersionNumberList Attribute	258
5.2.4	ServiceDatabaseState Attribute	259
5.2.5	Reserved Attribute IDs.....	259
5.3	BrowseGroupDescriptor Service Class Attribute Definitions ...	260
5.3.1	ServiceClassIDList Attribute	260
5.3.2	GroupID Attribute	260
5.3.3	Reserved Attribute IDs.....	260
6	Security.....	261
Appendix A	Background Information	262
A.1	Service Discovery	262
A.2	Bluetooth Service Discovery	262

**Appendix B Example SDP Transactions 263**

- B.1 SDP Example 1 – ServiceSearchRequest 263
- B.2 SDP Example 2 – ServiceAttributeTransaction 265
- B.3 SDP Example 3 – ServiceSearchAttributeTransaction 267

1 INTRODUCTION

1.1 GENERAL DESCRIPTION

The service discovery protocol (SDP) provides a means for applications to discover which services are available and to determine the characteristics of those available services.

1.2 MOTIVATION

Service Discovery in the Bluetooth environment, where the set of services that are available changes dynamically based on the RF proximity of devices in motion, is qualitatively different from service discovery in traditional network-based environments. The service discovery protocol defined in this specification is intended to address the unique characteristics of the Bluetooth environment. See [section A, “Background Information,” on page 262](#), for further information on this topic.

1.3 REQUIREMENTS

The following capabilities have been identified as requirements for version 1.0 of the Service Discovery Protocol.

1. SDP shall provide the ability for clients to search for needed services based on specific attributes of those services.
2. SDP shall permit services to be discovered based on the class of service.
3. SDP shall enable browsing of services without a priori knowledge of the specific characteristics of those services.
4. SDP shall provide the means for the discovery of new services that become available when devices enter RF proximity with a client device as well as when a new service is made available on a device that is in RF proximity with the client device.
5. SDP shall provide a mechanism for determining when a service becomes unavailable when devices leave RF proximity with a client device as well as when a service is made unavailable on a device that is in RF proximity with the client device.
6. SDP shall provide for services, classes of services, and attributes of services to be uniquely identified.
7. SDP shall allow a client on one device to discover a service on another device without consulting a third device.
8. SDP should be suitable for use on devices of limited complexity.
9. SDP shall provide a mechanism to incrementally discover information about the services provided by a device. This is intended to minimize the quantity

of data that must be exchanged in order to determine that a particular service is not needed by a client.

10. SDP should support the caching of service discovery information by intermediary agents to improve the speed or efficiency of the discovery process.
11. SDP should be transport independent.
12. SDP shall function while using L2CAP as its transport protocol.
13. SDP shall permit the discovery and use of services that provide access to other service discovery protocols.
14. SDP shall support the creation and definition of new services without requiring registration with a central authority.

1.4 NON-REQUIREMENTS AND DEFERRED REQUIREMENTS

The Bluetooth SIG recognizes that the following capabilities are related to service discovery. These items are not addressed in SDP version 1.0. However, some may be addressed in future revisions of the specification.

1. SDP 1.0 does not provide access to services. It only provides access to information about services.
2. SDP 1.0 does not provide brokering of services.
3. SDP 1.0 does not provide for negotiation of service parameters.
4. SDP 1.0 does not provide for billing of service use.
5. SDP 1.0 does not provide the means for a client to control or change the operation of a service.
6. SDP 1.0 does not provide an event notification when services, or information about services, become unavailable.
7. SDP 1.0 does not provide an event notification when attributes of services are modified.
8. This specification does not define an application programming interface for SDP.
9. SDP 1.0 does not provide support for service agent functions such as service aggregation or service registration.

1.5 CONVENTIONS

1.5.1 Bit And Byte Ordering Conventions

When multiple bit fields are contained in a single byte and represented in a drawing in this specification, the more significant (high-order) bits are shown toward the left and less significant (low-order) bits toward the right.

Multiple-byte fields are drawn with the more significant bytes toward the left and the less significant bytes toward the right. Multiple-byte fields are transferred in network byte order. See [Section 4.1 on page 234](#).

2 OVERVIEW

2.1 SDP CLIENT-SERVER ARCHITECTURE

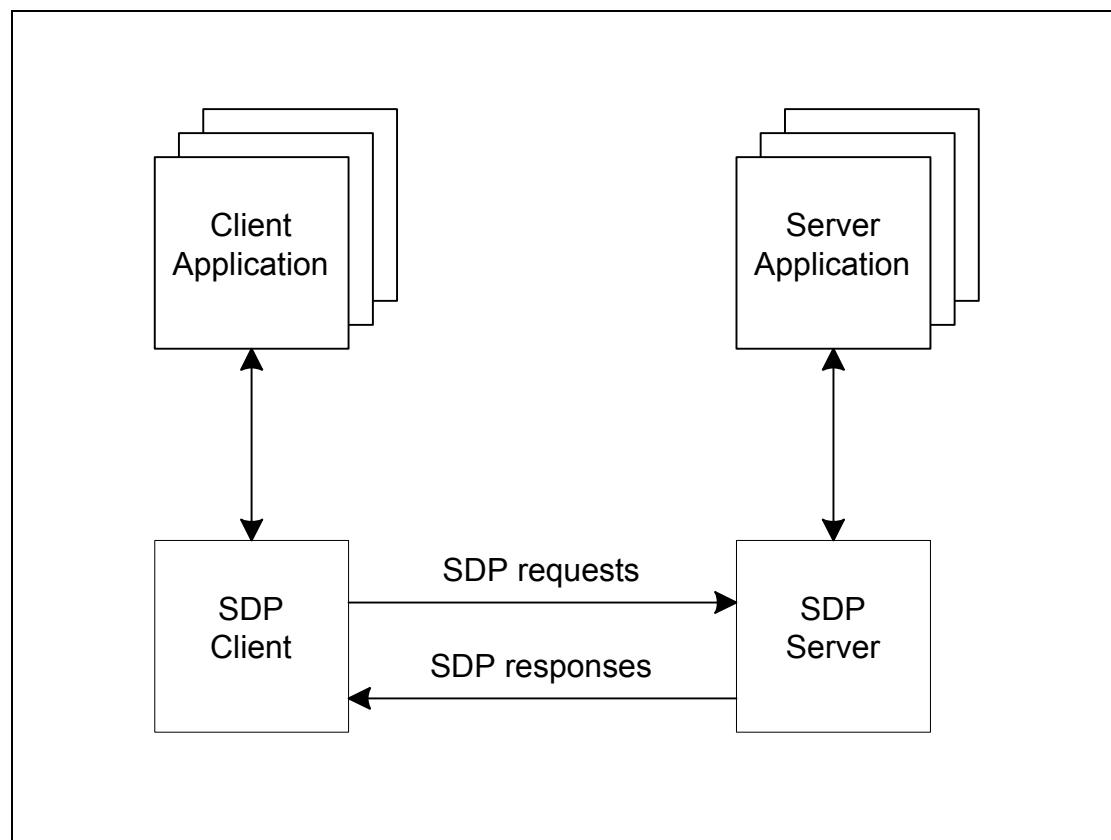


Figure 2.1: SDP Client-Server Interaction

The service discovery mechanism provides the means for client applications to discover the existence of services provided by server applications as well as the attributes of those services. The attributes of a service include the type or class of service offered and the mechanism or protocol information needed to utilize the service.

As far as the Service Discovery Protocol (SDP) is concerned, the configuration shown in Figure 2.1 can be simplified to that shown in Figure 2.2.

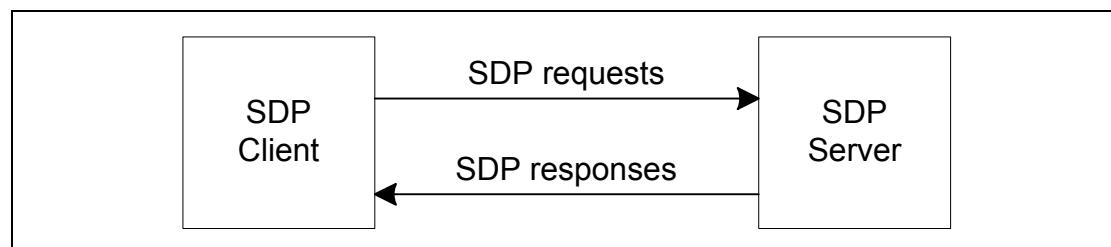


Figure 2.2: Simplified SDP Client-Server Interaction

SDP involves communication between an SDP server and an SDP client. The server maintains a list of service records that describe the characteristics of services associated with the server. Each service record contains information about a single service. A client can retrieve information from a service record maintained by the SDP server by issuing an SDP request.

If the client, or an application associated with the client, decides to use a service, it opens a separate connection to the service provider in order to utilize the service. SDP provides a mechanism for discovering services and their attributes (including associated service access protocols), but it does not provide a mechanism for utilizing those services (such as delivering the service access protocols).

If multiple applications on a device provide services, an SDP server can act on behalf of those service providers to handle requests for information about the services that they provide. Similarly, multiple client applications can utilize an SDP client to query servers on behalf of the client applications.

The set of SDP servers that are available to an SDP client will change dynamically based on the RF proximity of the servers to the client. When a server becomes available, a potential client must be notified by a means other than SDP so that the client can use SDP to query the server about its services. Similarly, when a server leaves proximity or becomes unavailable for any reason, there is no explicit notification via the service discovery protocol. However, the client can use SDP to poll the server and may infer that the server is not available if it no longer responds to requests.

Additional information regarding application interaction with SDP shall be contained in the Bluetooth Service Discovery Profile document.

2.2 SERVICE RECORD

A service is any entity that can provide information, perform an action, or control a resource on behalf of another entity. A service may be implemented as software, hardware, or a combination of hardware and software.

All of the information about a service that is maintained by an SDP server is contained within a single service record. The service record shall only be a list of service attributes.

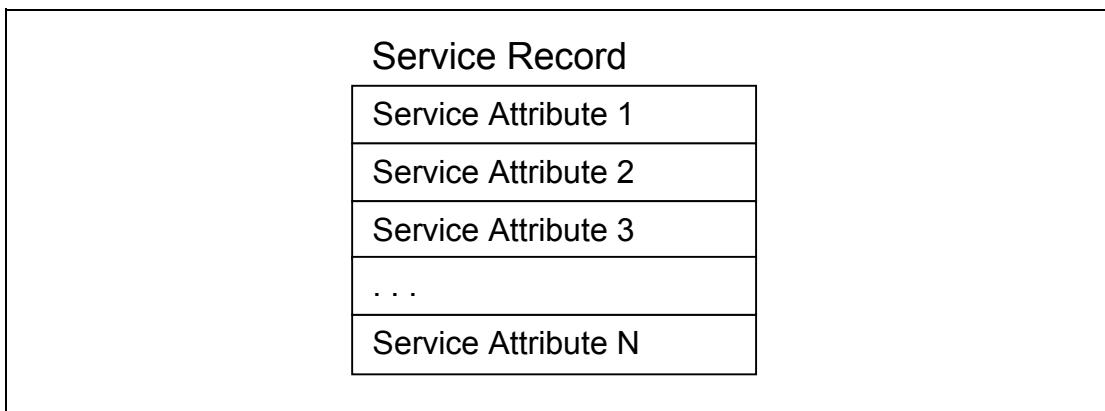


Figure 2.3: Service Record

A service record handle is a 32-bit number that shall uniquely identify each service record within an SDP server. It is important to note that, in general, each handle is unique only within each SDP server. If SDP server S1 and SDP server S2 both contain identical service records (representing the same service), the service record handles used to reference these identical service records are completely independent. The handle used to reference the service on S1 will be meaningless if presented to S2.

The service discovery protocol does not provide a mechanism for notifying clients when service records are added to or removed from an SDP server. While an L2CAP (Logical Link Control and Adaptation Protocol) connection is established to a server, a service record handle acquired from the server shall remain valid unless the service record it represents is removed. If a service is removed from the server, further requests to the server (during the L2CAP connection in which the service record handle was acquired) using the service's (now stale) record handle shall result in an error response indicating an invalid service record handle. An SDP server shall ensure that no service record handle values are re-used while an L2CAP connection remains established. Service record handles shall remain valid across successive L2CAP connections while the ServiceDatabaseState attribute value remains unchanged. Further, service record handles should remain valid until such time that the corresponding service is permanently removed or changes in an incompatible way. See the ServiceRecordState and ServiceDatabaseState attributes in [Section 5 on page 248](#).

A device may have a service record with a service record handle of 0x00000000 representing the SDP server itself. This service record contains attributes for the SDP server and the protocol it supports. For example, one of its attributes is the list of SDP protocol versions supported by the server.

2.3 SERVICE ATTRIBUTE

Each service attribute describes a single characteristic of a service. Some examples of service attributes are:

ServiceClassIDList	Identifies the type of service represented by a service record. In other words, the list of classes of which the service is an instance
ServiceID	Uniquely identifies a specific instance of a service
ProtocolDescriptorList	Specifies the protocol stack(s) that may be used to utilize a service
ProviderName	The textual name of the individual or organization that provides a service
IconURL	Specifies a URL that refers to an icon image that may be used to represent a service
ServiceName	A text string containing a human-readable name for the service
ServiceDescription	A text string describing the service

See [Section 5.1 on page 248](#), for attribute definitions that are common to all service records. Service providers can also define their own service attributes.

A service attribute consists of two components: an attribute ID and an attribute value.

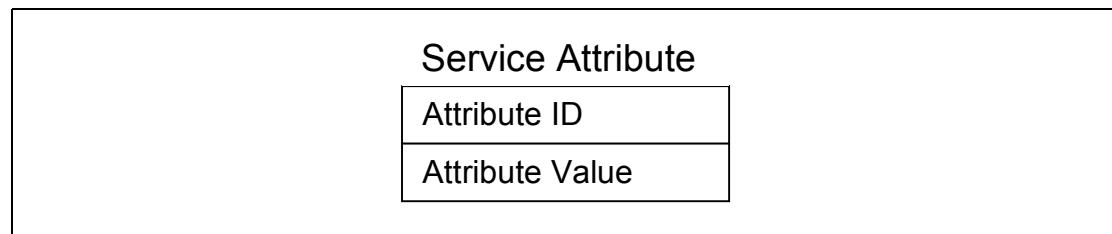


Figure 2.4: Service Attribute

2.3.1 Attribute ID

An attribute ID is a 16-bit unsigned integer that distinguishes each service attribute from other service attributes within a service record. The attribute ID also identifies the semantics of the associated attribute value.

A service class definition specifies each of the attribute IDs for a service class and assigns a meaning to the attribute value associated with each attribute ID. Each attribute ID is defined to be unique only within each service class.

All services belonging to a given service class assign the same meaning to each particular attribute ID. See [Section 2.4 on page 226](#).

In the Service Discovery Protocol, an attribute ID is represented as a data element. See [Section 3 on page 231](#).

2.3.2 Attribute Value

The attribute value is a variable length field whose meaning is determined by the attribute ID associated with it and by the service class of the service record in which the attribute is contained. In the Service Discovery Protocol, an attribute value is represented as a data element. (See [Section 3 on page 231](#).) Generally, any type of data element is permitted as an attribute value, subject to the constraints specified in the service class definition that assigns an attribute ID to the attribute and assigns a meaning to the attribute value. See [Section 5 on page 248](#), for attribute value examples.

2.4 SERVICE CLASS

Each service is an instance of a service class. The service class definition provides the definitions of all attributes contained in service records that represent instances of that class. Each attribute definition specifies the numeric value of the attribute ID, the intended use of the attribute value, and the format of the attribute value. A service record contains attributes that are specific to a service class as well as universal attributes that are common to all services.

Each service class is also assigned a unique identifier. This service class identifier is contained in the attribute value for the ServiceClassIDList attribute, and is represented as a UUID (see [Section 2.5.1 on page 227](#)). Since the format and meanings of many attributes in a service record are dependent on the service class of the service record, the ServiceClassIDList attribute is very important. Its value shall be examined or verified before any class-specific attributes are used. Since all of the attributes in a service record must conform to all of the service's classes, the service class identifiers contained in the ServiceClassIDList attribute are related. Typically, each service class is a subclass of another class whose identifier is contained in the list. A service subclass definition differs from its superclass in that the subclass contains additional attribute definitions that are specific to the subclass.

When a new service class is defined that is a subclass of an existing service class, the new service class retains all of the attributes defined in its superclass. Additional attributes may be defined that are specific to the new service class. In other words, the mechanism for adding new attributes to some of the instances of an existing service class is to create a new service class that is a subclass of the existing service class.

If a Service Class UUID is exposed in the SDP database of a product, then the product containing the SDP record shall comply with the specification which defines the service corresponding to the UUID.

2.4.1 A Printer Service Class Example

A color postscript printer with duplex capability might conform to 4 ServiceClass definitions and have a ServiceClassIDList with UUIDs (See [Section 2.5.1 on page 227](#).) representing the following ServiceClasses:

- DuplexColorPostscriptPrinterServiceClassID,
- ColorPostscriptPrinterServiceClassID,
- PostscriptPrinterServiceClassID,
- PrinterServiceClassID

Note that this example is only illustrative. This may not be a practical printer class hierarchy.

2.5 SEARCHING FOR SERVICES

The Service Search transaction allows a client to retrieve the service record handles for particular service records based on the values of attributes contained within those service records. Once an SDP client has a service record handle, it can request the values of specific attributes.

The capability search for service records based on the values of arbitrary attributes is not provided. Rather, the capability is provided to search only for attributes whose values are Universally Unique Identifiers¹ (UUIDs). Important attributes of services that can be used to search for a service are represented as UUIDs.

2.5.1 UUID

A UUID is a universally unique identifier that is guaranteed to be unique across all space and all time. UUIDs can be independently created in a distributed fashion. No central registry of assigned UUIDs is required. A UUID is a 128-bit value.

To reduce the burden of storing and transferring 128-bit UUID values, a range of UUID values has been pre-allocated for assignment to often-used, registered purposes. The first UUID in this pre-allocated range is known as the Bluetooth Base UUID and has the value 00000000-0000-1000-8000-00805F9B34FB, from the Bluetooth [Assigned Numbers](#) document. UUID values in the pre-allocated range have aliases that are represented as 16-bit or 32-bit values. These aliases are often called 16-bit and 32-bit UUIDs, but it is important to note that each actually represents a 128-bit UUID value.

The full 128-bit value of a 16-bit or 32-bit UUID may be computed by a simple arithmetic operation.

1. The format of UUIDs is specified in ITU-T Rec. X.667, alternatively known as ISO/IEC 9834-8:2005.

$128_bit_value = 16_bit_value * 2^{96} + \text{Bluetooth_Base_UUID}$

$128_bit_value = 32_bit_value * 2^{96} + \text{Bluetooth_Base_UUID}$

A 16-bit UUID may be converted to 32-bit UUID format by zero-extending the 16-bit value to 32-bits. An equivalent method is to add the 16-bit UUID value to a zero-valued 32-bit UUID.

Note that two 16-bit UUIDs may be compared directly, as may two 32-bit UUIDs or two 128-bit UUIDs. If two UUIDs of differing sizes are to be compared, the shorter UUID must be converted to the longer UUID format before comparison.

2.5.2 Service Search Patterns

A service search pattern is a list of UUIDs used to locate matching service records. A service search pattern matches a service record if each and every UUID in the service search pattern is contained within any of the service record's attribute values. The UUIDs need not be contained within any specific attributes or in any particular order within the service record. The service search pattern matches if the UUIDs it contains constitute a subset of the UUIDs in the service record's attribute values. The only time a service search pattern does not match a service record is if the service search pattern contains at least one UUID that is not contained within the service record's attribute values. Note also that a valid service search pattern must contain at least one UUID.

2.6 BROWSING FOR SERVICES

Normally, a client searches for services based on some desired characteristic(s) (represented by a UUID) of the services. However, there are times when it is desirable to discover which types of services are described by an SDP server's service records without any a priori information about the services. This process of looking for any offered services is termed browsing. In SDP, the mechanism for browsing for services is based on an attribute shared by all service classes. This attribute is called the *BrowseGroupList* attribute. The value of this attribute contains a list of UUIDs. Each UUID represents a browse group with which a service may be associated for the purpose of browsing.

When a client desires to browse an SDP server's services, it creates a service search pattern containing the UUID that represents the root browse group. All services that may be browsed at the top level are made members of the root browse group by having the root browse group's UUID as a value within the *BrowseGroupList* attribute.

Normally, if an SDP server has relatively few services, all of its services will be placed in the root browse group. However, the services offered by an SDP server may be organized in a browse group hierarchy, by defining additional

browse groups below the root browse group. Each of these additional browse groups is described by a service record with a service class of `BrowseGroupDescriptor`.

A browse group descriptor service record defines a new browse group by means of its `Group ID` attribute. In order for a service contained in one of these newly defined browse groups to be browsable, the browse group descriptor service record that defines the new browse group must in turn be browsable. The hierarchy of browsable services that is provided by the use of browse group descriptor service records allows the services contained in an SDP server to be incrementally browsed and is particularly useful when the SDP server contains many service records.

2.6.1 Example Service Browsing Hierarchy

Here is a fictitious service browsing hierarchy that illuminates the manner in which browse group descriptors are used. Browse group descriptor service records are identified with (G); other service records with (S).

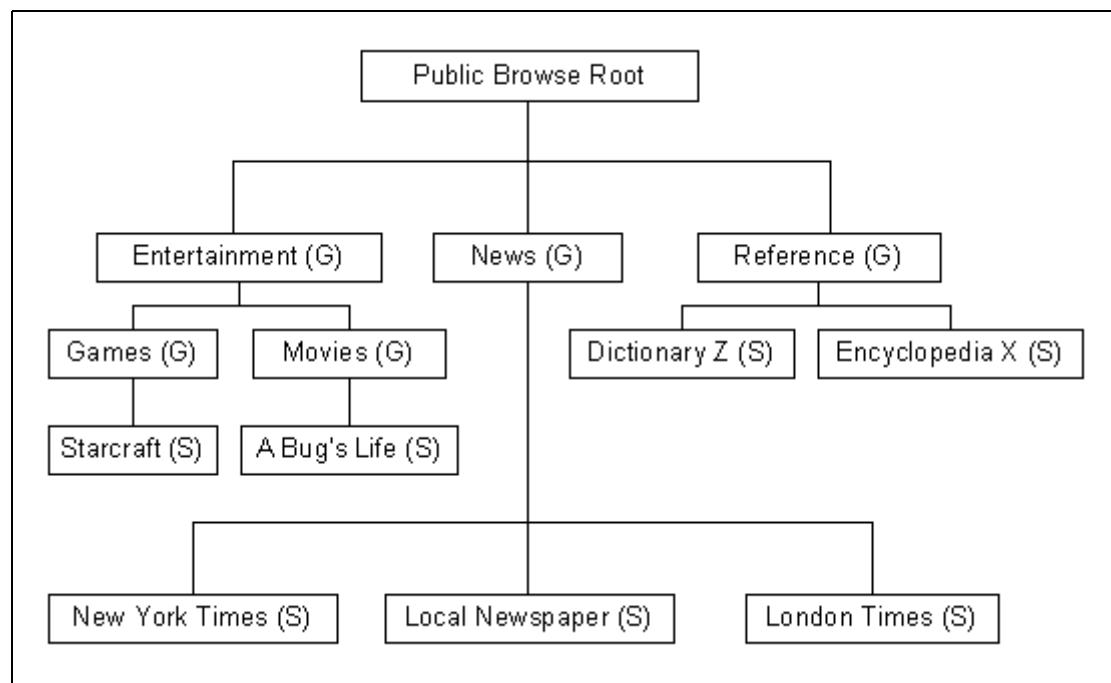


Figure 2.5: Service Browsing Hierarchy

This table shows the services records and service attributes necessary to implement the browse hierarchy.

Service Name	Service Class	Attribute Name	Attribute Value
Entertainment	BrowseGroupDescriptor	BrowseGroupList	PublicBrowseRoot
		GroupID	EntertainmentID
News	BrowseGroupDescriptor	BrowseGroupList	PublicBrowseRoot
		GroupID	NewsID
Reference	BrowseGroupDescriptor	BrowseGroupList	PublicBrowseRoot
		GroupID	ReferenceID
Games	BrowseGroupDescriptor	BrowseGroupList	EntertainmentID
		GroupID	GamesID
Movies	BrowseGroupDescriptor	BrowseGroupList	EntertainmentID
		GroupID	MoviesID
Starcraft	Video Game Class ID	BrowseGroupList	GamesID
A Bug's Life	Movie Class ID	BrowseGroupList	MovielD
Dictionary Z	Dictionary Class ID	BrowseGroupList	ReferenceID
Encyclopedia X	Encyclopedia Class ID	BrowseGroupList	ReferenceID
New York Times	Newspaper ID	BrowseGroupList	NewspaperID
London Times	Newspaper ID	BrowseGroupList	NewspaperID
Local Newspaper	Newspaper ID	BrowseGroupList	NewspaperID

Table 2.1: Service Browsing Hierarchy

3 DATA REPRESENTATION

Attribute values can contain information of various types with arbitrary complexity; thus enabling an attribute list to be generally useful across a wide variety of service classes and environments.

SDP defines a simple mechanism to describe the data contained within an attribute ID, attribute ID range, and attribute value. The primitive construct used is the data element.

3.1 DATA ELEMENT

A data element is a typed data representation. It consists of two fields: a header field and a data field. The header field, in turn, is composed of two parts: a type descriptor and a size descriptor. The data is a sequence of bytes whose length is specified in the size descriptor (described in [Section 3.3 on page 232](#)) and whose meaning is (partially) specified by the type descriptor.

3.2 DATA ELEMENT TYPE DESCRIPTOR

A data element type is represented as a 5-bit type descriptor. The type descriptor is contained in the most significant (high-order) 5 bits of the first byte of the data element header. The following types have been defined.

Type Descriptor Value	Valid Size Descriptor Values	Type Description
0	0	Nil, the null type
1	0, 1, 2, 3, 4	Unsigned Integer
2	0, 1, 2, 3, 4	Signed twos-complement integer
3	1, 2, 4	UUID, a universally unique identifier
4	5, 6, 7	Text string
5	0	Boolean ¹
6	5, 6, 7	Data element sequence, a data element whose data field is a sequence of data elements
7	5, 6, 7	Data element alternative, data element whose data field is a sequence of data elements from which one data element is to be selected.
8	5, 6, 7	URL, a uniform resource locator
9-31		Reserved

Table 3.1: Data Element

1. False is represented by the value 0, and true is represented by the value 1. However, to maximize interoperability, any non-zero value received must be accepted as representing true.

3.3 DATA ELEMENT SIZE DESCRIPTOR

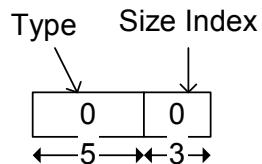
The data element size descriptor is represented as a 3-bit size index followed by 0, 8, 16, or 32 bits. The size index is contained in the least significant (low-order) 3 bits of the first byte of the data element header. The size index is encoded as follows.

Size Index	Additional bits	Data Size
0	0	1 byte. Exception: if the data element type is nil, the data size is 0 bytes.
1	0	2 bytes
2	0	4 bytes
3	0	8 bytes
4	0	16 bytes
5	8	The data size is contained in the additional 8 bits, which are interpreted as an unsigned integer.
6	16	The data size is contained in the additional 16 bits, which are interpreted as an unsigned integer.
7	32	The data size is contained in the additional 32 bits, which are interpreted as an unsigned integer.

Table 3.2: Data Element Size

3.4 DATA ELEMENT EXAMPLES

Nil is represented as:



A 16-bit signed integer is represented as:



The 3 character ASCII string "Hat" is represented as:

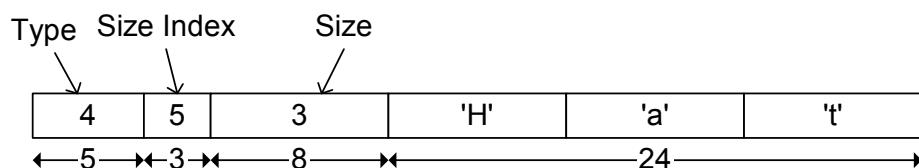


Figure 3.1: Data Element

4 PROTOCOL DESCRIPTION

SDP is a simple protocol with minimal requirements on the underlying transport. It can function over a reliable packet transport (or even unreliable, if the client implements timeouts and repeats requests as necessary).

SDP uses a request/response model where each transaction consists of one request protocol data unit (PDU) and one response PDU. In the case where SDP is used with the Bluetooth L2CAP transport protocol, no more than one SDP request PDU per connection to a given SDP server may be outstanding at a given instant. In other words, a client does not issue a further request to a server prior to receiving a response to the current request before issuing another request on the same L2CAP connection. Limiting SDP to sending one unacknowledged request PDU provides a simple form of flow control.

The protocol examples found in [Example SDP Transactions](#), could be helpful in understanding the protocol transactions.

4.1 TRANSFER BYTE ORDER

The service discovery protocol shall transfer multiple-byte fields in standard network byte order (Big Endian), with more significant (high-order) bytes being transferred before less-significant (low-order) bytes.

4.2 PROTOCOL DATA UNIT FORMAT

Every SDP PDU consists of a PDU header followed by PDU-specific parameters. The header contains three fields: a PDU ID, a Transaction ID, and a ParameterLength. Each of these header fields is described here. Parameters may include a continuation state parameter, described below; PDU-specific parameters for each PDU type are described later in separate PDU descriptions.

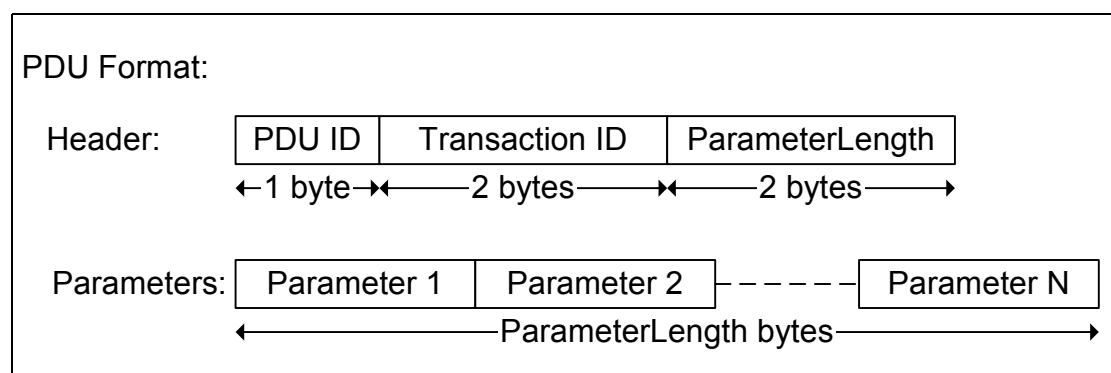


Figure 4.1: Protocol Data Unit Format

Service Discovery Protocol (SDP) Specification*PDU ID:**Size: 1 Byte*

Value	Parameter Description
N	The PDU ID field identifies the type of PDU. I.e. its meaning and the specific parameters.
0x00	Reserved
0x01	SDP_ErrorResponse
0x02	SDP_ServiceSearchRequest
0x03	SDP_ServiceSearchResponse
0x04	SDP_ServiceAttributeRequest
0x05	SDP_ServiceAttributeResponse
0x06	SDP_ServiceSearchAttributeRequest
0x07	SDP_ServiceSearchAttributeResponse
0x08-0xFF	Reserved

*TransactionID:**Size: 2 Bytes*

Value	Parameter Description
N	The TransactionID field uniquely identifies request PDUs and is used to match response PDUs to request PDUs. The SDP client can choose any value for a request's TransactionID provided that it is different from all outstanding requests. The TransactionID value in response PDUs is required to be the same as the request that is being responded to. Range: 0x0000 – 0xFFFF

*ParameterLength:**Size: 2 Bytes*

Value	Parameter Description
N	The ParameterLength field specifies the length (in bytes) of all parameters contained in the PDU. Range: 0x0000 – 0xFFFF

4.3 PARTIAL RESPONSES AND CONTINUATION STATE

Some SDP requests may require responses that are larger than can fit in a single response PDU. In this case, the SDP server shall generate a partial response along with a continuation state parameter. The continuation state parameter can be supplied by the client in a subsequent request to retrieve the next portion of the complete response. The continuation state parameter is a variable length field whose first byte contains the number of additional bytes of continuation information in the field. The format of the continuation information is not standardized among SDP servers. Each continuation state parameter is meaningful only to the SDP server that generated it.

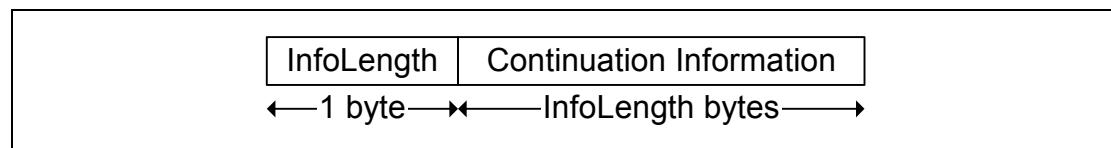


Figure 4.2: Continuation State Format

After a client receives a partial response and the accompanying continuation state parameter, it can re-issue the original request (with a new transaction ID) and include the continuation state in the new request indicating to the server that the remainder of the original response is desired. The maximum allowable value of the InfoLength field is 16 (0x10).

Note that, unless otherwise stated in a PDU response specification, an SDP server can split a response at any arbitrary boundary when it generates a partial response. The SDP server may select the boundary based on the contents of the reply, but is not required to do so. Therefore, length fields give the lengths as measured in the complete assembled record, not the length of the fields in the partial segment. When a service record is segmented into partial responses all attribute list values are relative to the complete record, not relevant to the partial record.

4.4 ERROR HANDLING

Each transaction consists of a request and a response PDU. Generally, each type of request PDU has a corresponding type of response PDU. However, if the server determines that a request is improperly formatted or for any reason the server cannot respond with the appropriate PDU type, it shall respond with an SDP_ErrorResponse PDU.

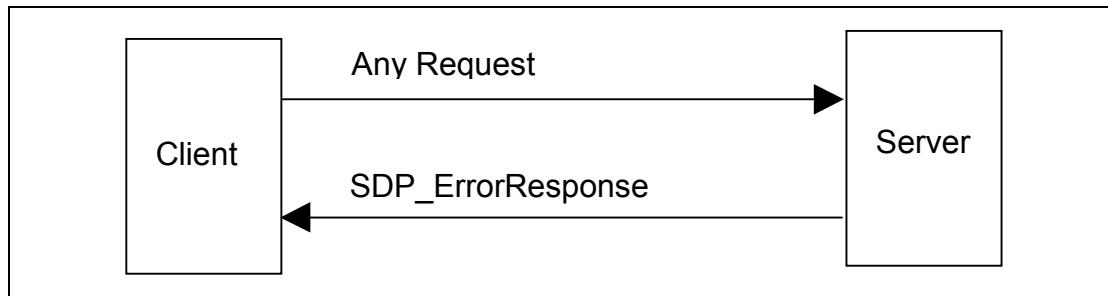


Figure 4.3: Error Handling

4.4.1 SDP_ErrorResponse PDU

PDU Type	PDU ID	Parameters
SDP_ErrorResponse	0x01	ErrorCode

Description:

The SDP server generates this PDU type in response to an improperly formatted request PDU or when the SDP server, for whatever reason, cannot generate an appropriate response PDU.

PDU Parameters:

ErrorCode: Size: 2 Bytes

Value	Parameter Description
N	The ErrorCode identifies the reason that an SDP_ErrorResponse PDU was generated.
0x0000	Reserved
0x0001	Invalid/unsupported SDP version
0x0002	Invalid Service Record Handle
0x0003	Invalid request syntax
0x0004	Invalid PDU Size
0x0005	Invalid Continuation State
0x0006	Insufficient Resources to satisfy Request
0x0007-0xFFFF	Reserved

Note: Invalid PDU size should be used, for example, if an incoming request PDU's length is inconsistent with the specification of that request PDU or the length parameter of an incoming request PDU is inconsistent with that request PDU's actual contents.

4.5 SERVICESEARCH TRANSACTION

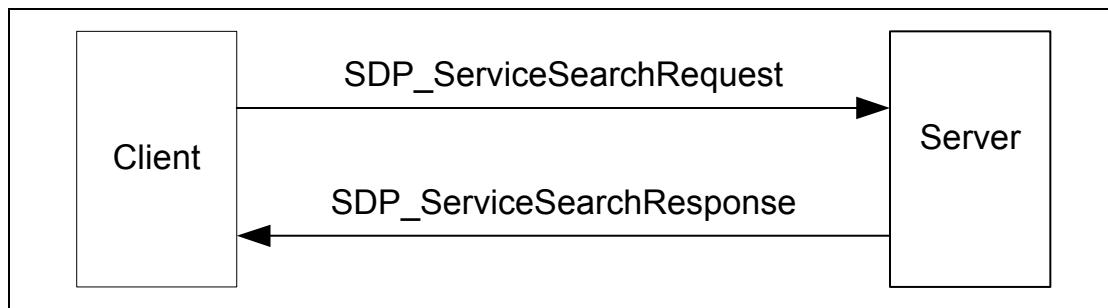


Figure 4.4: ServiceSearch Transaction

4.5.1 SDP_ServiceSearchRequest PDU

PDU Type	PDU ID	Parameters
SDP_ServiceSearchRequest	0x02	ServiceSearchPattern, MaximumServiceRecordCount, ContinuationState

Description:

The SDP client generates an SDP_ServiceSearchRequest to locate service records that match the service search pattern given as the first parameter of the PDU. Upon receipt of this request, the SDP server shall examine its service record data base and return an SDP_ServiceSearchResponse containing the service record handles of service records within its service record database that match the given service search pattern, or an appropriate error response.

Note that no mechanism is provided to request information for all service records. However, see [Section 2.6 on page 228](#) for a description of a mechanism that permits browsing for non-specific services without a priori knowledge of the services.

PDU Parameters:

ServiceSearchPattern: Size: Varies

Value	Parameter Description
Data Element Sequence	The ServiceSearchPattern is a data element sequence where each element in the sequence is a UUID. The sequence shall contain at least one UUID. The maximum number of UUIDs in the sequence is 12 ¹ . The list of UUIDs constitutes a service search pattern.

1. The value of 12 has been selected as a compromise between the scope of a service search and the size of a search request PDU. It is not expected that more than 12 UUIDs will be useful in a service search pattern.

Service Discovery Protocol (SDP) Specification**MaximumServiceRecordCount:****Size: 2 Bytes**

Value	Parameter Description
N	MaximumServiceRecordCount is a 16-bit count specifying the maximum number of service record handles to be returned in the response(s) to this request. The SDP server shall not return more handles than this value specifies. If more than N service records match the request, the SDP server determines which matching service record handles to return in the response(s). Range: 0x0001-0xFFFF

ContinuationState:**Size: 1 to 17 Bytes**

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation state information that were returned in a previous response from the server. N is required to be less than or equal to 16. If no continuation state is to be provided in the request, N is set to 0.

4.5.2 SDP_ServiceSearchResponse PDU

PDU Type	PDU ID	Parameters
SDP_ServiceSearchResponse	0x03	TotalServiceRecordCount, CurrentServiceRecordCount, ServiceRecordHandleList, ContinuationState

Description:

The SDP server generates an SDP_ServiceSearchResponse upon receipt of a valid SDP_ServiceSearchRequest. The response contains a list of service record handles for service records that match the service search pattern given in the request. If a partial response is generated, it shall contain an integral number of complete service record handles; a service record handle value shall not be split across multiple PDUs.

PDU Parameters:*TotalServiceRecordCount:*

Size: 2 Bytes

Value	Parameter Description
N	<p>The TotalServiceRecordCount is an integer containing the number of service records that match the requested service search pattern. If no service records match the requested service search pattern, this parameter is set to 0. N should never be larger than the MaximumServiceRecordCount value specified in the SDP_ServiceSearchRequest. When multiple partial responses are used, each partial response contains the same value for TotalServiceRecordCount.</p> <p>Range: 0x0000-0xFFFF</p>

CurrentServiceRecordCount:

Size: 2 Bytes

Value	Parameter Description
N	<p>The CurrentServiceRecordCount is an integer indicating the number of service record handles that are contained in the next parameter. If no service records match the requested service search pattern, this parameter is set to 0. N should never be larger than the TotalServiceRecordCount value specified in the current response.</p> <p>Range: 0x0000-0xFFFF</p>

ServiceRecordHandleList:

Size: (CurrentServiceRecordCount*4) Bytes

Value	Parameter Description
List of 32-bit handles	<p>The ServiceRecordHandleList contains a list of service record handles. The number of handles in the list is given in the CurrentServiceRecordCount parameter. Each of the handles in the list refers to a service record that matches the requested service search pattern. Note that this list of service record handles does not have the format of a data element. It contains no header fields, only the 32-bit service record handles.</p>

ContinuationState:

Size: 1 to 17 Bytes

Value	Parameter Description
Continuation State	<p>ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation information. If the current response is complete, this parameter consists of a single byte with the value 0. If a partial response is contained in the PDU, the ContinuationState parameter may be supplied in a subsequent request to retrieve the remainder of the response.</p>

4.6 SERVICEATTRIBUTE TRANSACTION

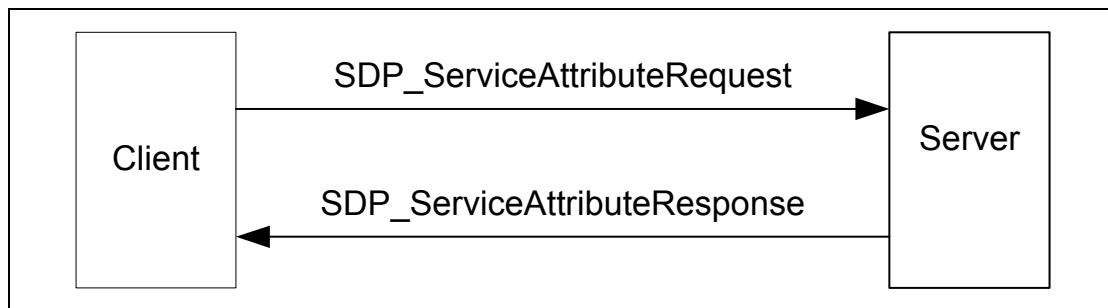


Figure 4.5: ServiceAttribute Transaction

4.6.1 SDP_ServiceAttributeRequest PDU

PDU Type	PDU ID	Parameters
SDP_ServiceAttributeRequest	0x04	ServiceRecordHandle, MaximumAttributeByteCount, AttributeIDList, ContinuationState

Description:

The SDP client generates an SDP_ServiceAttributeRequest to retrieve specified attribute values from a specific service record. The service record handle of the desired service record and a list of desired attribute IDs to be retrieved from that service record are supplied as parameters.

Command Parameters:

ServiceRecordHandle: Size: 4 Bytes

Value	Parameter Description
32-bit handle	The ServiceRecordHandle parameter specifies the service record from which attribute values are to be retrieved. The handle is obtained via a previous SDP_ServiceSearch transaction.

Service Discovery Protocol (SDP) Specification*MaximumAttributeByteCount:**Size: 2 Bytes*

Value	Parameter Description
N	<p>MaximumAttributeByteCount specifies the maximum number of bytes of attribute data to be returned in the response to this request. The SDP server shall not return more than N bytes of attribute data in the response PDU. If the requested attributes require more than N bytes, the SDP server determines how to segment the list. In this case the client may request each successive segment by issuing a request containing the continuation state that was returned in the previous response PDU. Note that in the case where multiple response PDUs are needed to return the attribute data, MaximumAttributeByteCount specifies the maximum size of the portion of the attribute data contained in each response PDU.</p> <p>Range: 0x0007-0xFFFF</p>

*AttributeIDList:**Size: Varies*

Value	Parameter Description
Data Element Sequence	<p>The AttributeIDList is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. Each attribute ID is encoded as a 16-bit unsigned integer data element. Each attribute ID range is encoded as a 32-bit unsigned integer data element, where the high order 16 bits are interpreted as the beginning attribute ID of the range and the low order 16 bits are interpreted as the ending attribute ID of the range. The attribute IDs contained in the AttributeIDList shall be listed in ascending order without duplication of any attribute ID values. Note that all attributes can be requested by specifying a range of 0x0000-0xFFFF.</p>

*ContinuationState:**Size: 1 to 17 Bytes*

Value	Parameter Description
Continuation State	<p>ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation state information that were returned in a previous response from the server. N shall be less than or equal to 16. If no continuation state is to be provided in the request, N shall be set to 0.</p>

4.6.2 SDP_ServiceAttributeResponse PDU

PDU Type	PDU ID	Parameters
SDP_ServiceAttributeResponse	0x05	AttributeListByteCount, AttributeList, ContinuationState

Description:

The SDP server generates an SDP_ServiceAttributeResponse upon receipt of a valid SDP_ServiceAttributeRequest. The response contains a list of attributes (both attribute ID and attribute value) from the requested service record.

PDU Parameters:

AttributeListByteCount: Size: 2 Bytes

Value	Parameter Description
N	The AttributeListByteCount contains a count of the number of bytes in the AttributeList parameter. N shall never be larger than the MaximumAttributeByteCount value specified in the SDP_ServiceAttributeRequest. Range: 0x0002-0xFFFF

AttributeList: Size: AttributeListByteCount

Value	Parameter Description
Data Element Sequence	The AttributeList is a data element sequence containing attribute IDs and attribute values. The first element in the sequence contains the attribute ID of the first attribute to be returned. The second element in the sequence contains the corresponding attribute value. Successive pairs of elements in the list contain additional attribute ID and value pairs. Only attributes that have non-null values within the service record and whose attribute IDs were specified in the SDP_ServiceAttributeRequest are contained in the AttributeList. Neither an attribute ID nor an attribute value is placed in the AttributeList for attributes in the service record that have no value. The attributes are listed in ascending order of attribute ID value.

ContinuationState: Size: 1 to 17 Bytes

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation information. If the current response is complete, this parameter consists of a single byte with the value 0. If a partial response is given, the ContinuationState parameter may be supplied in a subsequent request to retrieve the remainder of the response.

If a partial response is generated, the response may be arbitrarily segmented into multiple PDUs (subject to the constraint imposed by the allowed range of values for the AttributeListByteCount parameter). Attributes in partial responses are not required to be integral.

4.7 SERVICESEARCHATTRIBUTE TRANSACTION

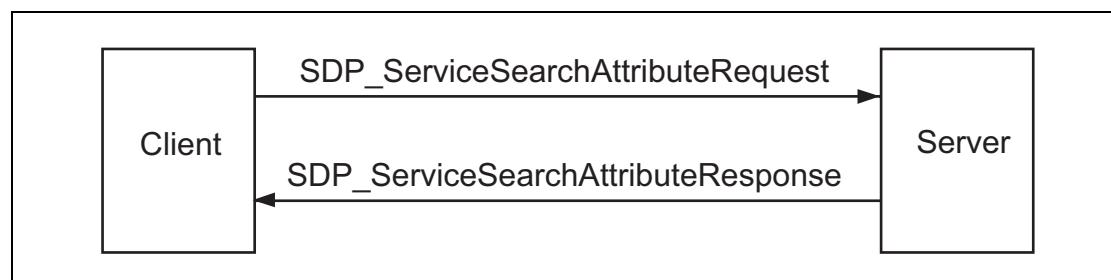


Figure 4.6: ServiceSearchAttribute Transaction

4.7.1 SDP_ServiceSearchAttributeRequest PDU

PDU Type	PDU ID	Parameters
SDP_ServiceSearchAttributeRequest	0x06	ServiceSearchPattern, MaximumAttributeByteCount, AttributeIDList, ContinuationState

Description:

The SDP_ServiceSearchAttributeRequest transaction combines the capabilities of the SDP_ServiceSearchRequest and the SDP_ServiceAttributeRequest into a single request. As parameters, it contains both a service search pattern and a list of attributes to be retrieved from service records that match the service search pattern. The SDP_ServiceSearchAttributeRequest and its response are more complex and can require more bytes than separate SDP_ServiceSearch and SDP_ServiceAttribute transactions. However, using SDP_ServiceSearchAttributeRequest can reduce the total number of SDP transactions, particularly when retrieving multiple service records.

Note that the service record handle for each service record is contained in the ServiceRecordHandle attribute of that service and may be requested along with other attributes.

PDU Parameters:*ServiceSearchPattern:*

Size: Varies

Value	Parameter Description
Data Element Sequence	The ServiceSearchPattern is a data element sequence where each element in the sequence is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in the sequence is 12 ¹ . The list of UUIDs constitutes a service search pattern.

1. The value of 12 has been selected as a compromise between the scope of a service search and the size of a search request PDU. It is not expected that more than 12 UUIDs will be useful in a service search pattern.

MaximumAttributeByteCount:

Size: 2 Bytes

Value	Parameter Description
N	MaximumAttributeByteCount specifies the maximum number of bytes of attribute data to be returned in the response to this request. The SDP server shall not return more than N bytes of attribute data in the response PDU. If the requested attributes require more than N bytes, the SDP server determines how to segment the list. In this case the client may request each successive segment by issuing a request containing the continuation state that was returned in the previous response PDU. Note that in the case where multiple response PDUs are needed to return the attribute data, MaximumAttributeByteCount specifies the maximum size of the portion of the attribute data contained in each response PDU. Range: 0x0007-0xFFFF

AttributeIDList:

Size: Varies

Value	Parameter Description
Data Element Sequence	The AttributeIDList is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. Each attribute ID is encoded as a 16-bit unsigned integer data element. Each attribute ID range is encoded as a 32-bit unsigned integer data element, where the high order 16 bits are interpreted as the beginning attribute ID of the range and the low order 16 bits are interpreted as the ending attribute ID of the range. The attribute IDs contained in the AttributeIDList shall be listed in ascending order without duplication of any attribute ID values. Note that all attributes may be requested by specifying a range of 0x0000-0xFFFF.

ContinuationState:

Size: 1 to 17 Bytes

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation state information that were returned in a previous response from the server. N shall be less than or equal to 16. If no continuation state is to be provided in the request, N shall set to 0.

4.7.2 SDP_ServiceSearchAttributeResponse PDU

PDU Type	PDU ID	Parameters
SDP_ServiceSearchAttributeResponse	0x07	AttributeListsByteCount, AttributeLists, ContinuationState

Description:

The SDP server generates an SDP_ServiceSearchAttributeResponse upon receipt of a valid SDP_ServiceSearchAttributeRequest. The response contains a list of attributes (both attribute ID and attribute value) from the service records that match the requested service search pattern.

PDU Parameters:

AttributeListsByteCount:

Size: 2 Bytes

Value	Parameter Description
N	The AttributeListsByteCount contains a count of the number of bytes in the AttributeLists parameter. N shall never be larger than the MaximumAttributeByteCount value specified in the SDP_ServiceSearchAttributeRequest. Range: 0x0002-0xFFFF

AttributeLists:

Size: Varies

Value	Parameter Description
Data Element Sequence	The AttributeLists is a data element sequence where each element in turn is a data element sequence representing an attribute list. Each attribute list contains attribute IDs and attribute values from one service record. The first element in each attribute list contains the attribute ID of the first attribute to be returned for that service record. The second element in each attribute list contains the corresponding attribute value. Successive pairs of elements in each attribute list contain additional attribute ID and value pairs. Only attributes that have non-null values within the service record and whose attribute IDs were specified in the SDP_ServiceSearchAttributeRequest are contained in the AttributeLists. Neither an attribute ID nor attribute value is placed in AttributeLists for attributes in the service record that have no value. Within each attribute list, the attributes are listed in ascending order of attribute ID value.

*ContinuationState:**Size: 1 to 17 Bytes*

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation information. If the current response is complete, this parameter consists of a single byte with the value 0. If a partial response is given, the ContinuationState parameter may be supplied in a subsequent request to retrieve the remainder of the response.

If a partial response is generated, the response may be arbitrarily segmented into multiple PDUs (subject to the constraint imposed by the allowed range of values for the AttributeListByteCount parameter). Attributes in partial responses are not required to be integral.

5 SERVICE ATTRIBUTE DEFINITIONS

The service classes and attributes contained in this document are necessarily a partial list of the service classes and attributes supported by SDP. Only service classes that directly support the SDP server are included in this document. Additional service classes will be defined in other documents and possibly in future revisions of this document. Also, it is expected that additional attributes will be discovered that are applicable to a broad set of services; these may be added to the list of Universal attributes in future revisions of this document.

5.1 UNIVERSAL ATTRIBUTE DEFINITIONS

Universal attributes are those service attributes whose definitions are common to all service records. Note that this does not mean that every service record contains values for all of these service attributes. However, if a service record has a service attribute with an attribute ID allocated to a universal attribute, the attribute value shall conform to the universal attribute's definition.

Only two attributes are required to exist in every service record instance. They are the ServiceRecordHandle (attribute ID 0x0000) and the ServiceClassIDList (attribute ID 0x0001). All other service attributes are optional within a service record.

5.1.1 ServiceRecordHandle Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceRecordHandle	0x0000	32-bit unsigned integer

Description:

A service record handle is a 32-bit number that uniquely identifies each service record within an SDP server. It is important to note that, in general, each handle is unique only within each SDP server. If SDP server S1 and SDP server S2 both contain identical service records (representing the same service), the service record handles used to reference these identical service records are completely independent. The handle used to reference the service on S1 will, in general, be meaningless if presented to S2. Service record handle values 0x00000001-0x0000FFFF are reserved.

5.1.2 ServiceClassIDList Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceClassIDList	0x0001	Data Element Sequence

Description:

The ServiceClassIDList attribute consists of a data element sequence in which each data element is a UUID representing the service classes that a given service record conforms to. The UUIDs should be listed in order from the most specific class to the most general class unless otherwise specified by the profile specifications defining the service classes. When profiles are enhanced, any new UUIDs should be added at the end of the ServiceClassIDList (after any existing UUIDs) to minimize interoperability issues with legacy implementations. The ServiceClassIDList shall contain at least one service class UUID.

5.1.3 ServiceRecordState Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceRecordState	0x0002	32-bit unsigned integer

Description:

The ServiceRecordState is a 32-bit integer that is used to facilitate caching of Service Attributes. If this attribute is contained in a service record, its value shall be changed when any other attribute value is added to, deleted from or changed within the service record. This permits a client to check the value of this single attribute. If its value has not changed since it was last checked, the client knows that no other attribute values within the service record have changed.

5.1.4 ServiceID Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceID	0x0003	UUID

Description:

The ServiceID is a UUID that universally and uniquely identifies the service instance described by the service record. This service attribute is particularly useful if the same service is described by service records in more than one SDP server.

5.1.5 ProtocolDescriptorList Attribute

Attribute Name	Attribute ID	Attribute Value Type
ProtocolDescriptorList	0x0004	Data Element Sequence or Data Element Alternative

Description:

The ProtocolDescriptorList attribute describes one or more protocol stacks that can be used to gain access to the service described by the service record.

If the ProtocolDescriptorList describes a single stack, it takes the form of a data element sequence in which each element of the sequence is a protocol descriptor. Each protocol descriptor is, in turn, a data element sequence whose first element is a UUID identifying the protocol and whose successive elements are protocol-specific parameters. Potential protocol-specific parameters are a protocol version number and a connection-port number. The protocol descriptors are listed in order from the lowest layer protocol to the highest layer protocol used to gain access to the service.

If it is possible for more than one kind of protocol stack to be used to gain access to the service, the ProtocolDescriptorList takes the form of a data element alternative where each member is a data element sequence as described in the previous paragraph.

Protocol Descriptors

A protocol descriptor identifies a communications protocol and provides protocol-specific parameters. A protocol descriptor is represented as a data element sequence. The first data element in the sequence shall be the UUID that identifies the protocol. Additional data elements optionally provide protocol-specific information, such as the L2CAP protocol/service multiplexer (PSM) and the RFCOMM server channel number (CN) shown below.

ProtocolDescriptorList Examples

These examples are intended to be illustrative. The parameter formats for each protocol are not defined within this specification.

In the first two examples, it is assumed that a single RFCOMM instance exists on top of the L2CAP layer. In this case, the L2CAP protocol specific information (PSM) points to the single instance of RFCOMM. In the last example, two different and independent RFCOMM instances are available on top of the L2CAP layer. In this case, the L2CAP protocol specific information (PSM) points to a distinct identifier that distinguishes each of the RFCOMM instances. See the L2CAP specification ([\[Vol. 3\] Part A, Section 4.2](#)) for the range of allowed PSM values.

IrDA-like printer

((L2CAP, PSM=RFCOMM), (RFCOMM, CN=1), (PostscriptStream))

IP Network Printing

((L2CAP, PSM=RFCOMM), (RFCOMM, CN=2), (PPP), (IP), (TCP),
 (IPP))

Synchronization Protocol Descriptor Example

((L2CAP, PSM=0x1001), (RFCOMM, CN=1), (Obex), (vCal))

((L2CAP, PSM=0x1003), (RFCOMM, CN=1), (Obex),
 (otherSynchronisationApplication))

5.1.6 AdditionalProtocolDescriptorList Attribute

Attribute Name	Attribute ID	Attribute Value Type
AdditionalProtocolDescriptorList	0x000D	Data Element Sequence

Description:

The AdditionalProtocolDescriptorLists attribute contains a sequence of ProtocolDescriptorList-elements. Each element having the same format as the ProtocolDescriptorList described in section 5.1.5. The ordering of the elements is significant and should be specified and fixed in Profiles that make use of this attribute.

The AdditionalProtocolDescriptorLists attribute supports services that require more channels in addition to the service described in [Section 5.1.5](#). If the AdditionalProtocolDescriptorLists attribute is included in a service record, the ProtocolDescriptorList attribute must be included.

AdditionalProtocolDescriptorList Examples

The following is just an illustrative example and is not meant to refer a real specified service or protocols.

Attribute	Attribute Value type	Attribute Value
ProtocolDescriptorList		
ProtocolDescriptor	#0DataElementSequence	
ProtocolID	UUID	L2CAP
Param: PSM	PSM	FooDataProtocol
ProtocolDescriptor	#1DataElementSequence	
ProtocolID	UUID	FooDataProtocol

AdditionalProtocolDescriptorLists

```

ProtocolDescriptorList #0 DataElementSequence
ProtocolDescriptor #0 DataElementSequence
  ProtocolID      UUID      L2CAP
  Param: PSM      PSM       FooControlProtocol
ProtocolDescriptor #1DataElementSequence
  ProtocolID      UUID      FooControlProtocol

```

5.1.7 BrowseGroupList Attribute

Attribute Name	Attribute ID	Attribute Value Type
BrowseGroupList	0x0005	Data Element Sequence

Description:

The BrowseGroupList attribute consists of a data element sequence in which each element is a UUID that represents a browse group to which the service record belongs. The top-level browse group ID, called PublicBrowseRoot and representing the root of the browsing hierarchy, has the value 00001002-0000-1000-8000-00805F9B34FB (UUID16: 0x1002) from the Bluetooth [Assigned Numbers](#) document.

5.1.8 LanguageBaseAttributeIDList Attribute

Attribute Name	Attribute ID	Attribute Value Type
LanguageBaseAttributeIDList	0x0006	Data Element Sequence

Description:

In order to support human-readable attributes for multiple natural languages in a single service record, a base attribute ID is assigned for each of the natural languages used in a service record. The human-readable universal attributes are then defined with an attribute ID offset from each of these base values, rather than with an absolute attribute ID.

The LanguageBaseAttributeIDList attribute is a list in which each member contains a language identifier, a character encoding identifier, and a base attribute ID for each of the natural languages used in the service record. The LanguageBaseAttributeIDList attribute consists of a data element sequence in which each element is a 16-bit unsigned integer. The elements are grouped as triplets (threes).

The first element of each triplet contains an identifier representing the natural language. The language is encoded according to ISO 639:1988 (E/F): “Code for the representation of names of languages”.

The second element of each triplet contains an identifier that specifies a character encoding used for the language. Values for character encoding can be found in IANA's database¹, and have the values that are referred to as MIBEnum values. The recommended character encoding is UTF-8.

The third element of each triplet contains an attribute ID that serves as the base attribute ID for the natural language in the service record. Different service records within a server may use different base attribute ID values for the same language.

To facilitate the retrieval of human-readable universal attributes in a principal language, the base attribute ID value for the primary language supported by a service record shall be 0x0100. Also, if a LanguageBaseAttributeIDList attribute is contained in a service record, the base attribute ID value contained in its first element shall be 0x0100. If one or more human readable attributes are contained in a service record, the LanguageBaseAttributeIDList attribute should be included in that service record.

5.1.9 ServiceInfoTimeToLive Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceInfoTimeToLive	0x0007	32-bit unsigned integer

Description:

The ServiceTimeToLive attribute is a 32-bit integer that contains the number of seconds for which the information in a service record is expected to remain valid and unchanged. This time interval is measured from the time that the attribute value is retrieved from the SDP server. This value does not imply a guarantee that the service record will remain available or unchanged. It is simply a hint that a client can use to determine a suitable polling interval to re-validate the service record contents.

1. See <http://www.isi.edu/in-notes/iana/assignments/character-sets>

5.1.10 ServiceAvailability Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceAvailability	0x0008	8-bit unsigned integer

Description:

The ServiceAvailability attribute is an 8-bit unsigned integer that represents the relative ability of the service to accept additional clients. A value of 0xFF indicates that the service is not currently in use and is thus fully available, while a value of 0x00 means that the service is not accepting new clients. For services that support multiple simultaneous clients, intermediate values indicate the relative availability of the service on a linear scale.

For example, a service that can accept up to 3 clients should provide ServiceAvailability values of 0xFF, 0xAA, 0x55, and 0x00 when 0, 1, 2, and 3 clients, respectively, are utilizing the service. The value 0xAA is approximately (2/3) * 0xFF and represents 2/3 availability, while the value 0x55 is approximately (1/3)*0xFF and represents 1/3 availability. Note that the availability value is approximated as

$$(1 - (\text{current_number_of_clients} / \text{maximum_number_of_clients})) * 0xFF$$

When the maximum number of clients is large, this formula must be modified to ensure that ServiceAvailability values of 0x00 and 0xFF are reserved for their defined meanings of unavailability and full availability, respectively.

Note that the maximum number of clients a service can support may vary according to the resources utilized by the service's current clients.

A non-zero value for ServiceAvailability does not guarantee that the service will be available for use. It should be treated as a hint or an approximation of availability status.

5.1.11 BluetoothProfileDescriptorList Attribute

Attribute Name	Attribute ID	Attribute Value Type
BluetoothProfileDescriptorList	0x0009	Data Element Sequence

Description:

The BluetoothProfileDescriptorList attribute consists of a data element sequence in which each element is a profile descriptor that contains information about a Bluetooth profile to which the service represented by this service record conforms. Each profile descriptor is a data element sequence whose first element is the UUID assigned to the profile and whose second element is a 16-bit profile version number.

Each version of a profile is assigned a 16-bit unsigned integer profile version number, which consists of two 8-bit fields. The higher-order 8 bits contain the major version number field and the lower-order 8 bits contain the minor version number field. The initial version of each profile has a major version of 1 and a minor version of 0. When upward compatible changes are made to the profile, the minor version number will be incremented. If incompatible changes are made to the profile, the major version number will be incremented.

5.1.12 DocumentationURL Attribute

Attribute Name	Attribute ID	Attribute Value Type
DocumentationURL	0x000A	URL

Description:

This attribute is a URL which points to documentation on the service described by a service record.

5.1.13 ClientExecutableURL Attribute

Attribute Name	Attribute ID	Attribute Value Type
ClientExecutableURL	0x000B	URL

Description:

This attribute contains a URL that refers to the location of an application that can be used to utilize the service described by the service record. Since different operating environments require different executable formats, a mechanism has been defined to allow this single attribute to be used to locate an executable that is appropriate for the client device's operating environment. In the attribute value URL, the first byte with the value 0x2A (ASCII character '*'') is to be replaced by the client application with a string representing the desired operating environment before the URL is to be used.

The list of standardized strings representing operating environments is contained in the Bluetooth [Assigned Numbers](#) document.

For example, assume that the value of the ClientExecutableURL attribute is http://my.fake/public/*/client.exe. On a device capable of executing SH3 WindowsCE files, this URL would be changed to <http://my.fake/public/sh3-microsoft-wince/client.exe>. On a device capable of executing Windows 98 binaries, this URL would be changed to <http://my.fake/public/i86-microsoft-win98/client.exe>.

5.1.14 IconURL Attribute

Attribute Name	Attribute ID	Attribute Value Type
IconURL	0x000C	URL

Description:

This attribute contains a URL that refers to the location of an icon that can be used to represent the service described by the service record. Since different hardware devices require different icon formats, a mechanism has been defined to allow this single attribute to be used to locate an icon that is appropriate for the client device. In the attribute value URL, the first byte with the value 0x2A (ASCII character '*') is to be replaced by the client application with a string representing the desired icon format before the URL is to be used.

The list of standardized strings representing icon formats is contained in the Bluetooth [Assigned Numbers](#) document.

For example, assume that the value of the IconURL attribute is `http://my.fake/public/icons/*`. On a device that prefers 24 x 24 icons with 256 colors, this URL would be changed to `http://my.fake/public/icons/24x24x8.png`. On a device that prefers 10 x 10 monochrome icons, this URL would be changed to `http://my.fake/public/icons/10x10x1.png`.

5.1.15 ServiceName Attribute

Attribute Name	Attribute ID Offset	Attribute Value Type
ServiceName	0x0000	String

Description:

The ServiceName attribute is a string containing the name of the service represented by a service record. It should be brief and suitable for display with an Icon representing the service. The offset 0x0000 is added to the attribute ID base (contained in the LanguageBaseAttributeIDList attribute) in order to compute the attribute ID for this attribute.

5.1.16 ServiceDescription Attribute

Attribute Name	Attribute ID Offset	Attribute Value Type
ServiceDescription	0x0001	String

Description:

This attribute is a string containing a brief description of the service. It should be less than 200 characters in length. The offset 0x0001 is added to the attribute ID base (contained in the LanguageBaseAttributeIDList attribute) in order to compute the attribute ID for this attribute.

5.1.17 ProviderName Attribute

Attribute Name	Attribute ID Offset	Attribute Value Type
ProviderName	0x0002	String

Description:

This attribute is a string containing the name of the person or organization providing the service. The offset 0x0002 is added to the attribute ID base (contained in the LanguageBaseAttributeIDList attribute) in order to compute the attribute ID for this attribute.

5.1.18 Reserved Universal Attribute IDs

Attribute IDs in the range of 0x000E – 0x01FF are reserved for use by SDP, if allocated the assigned numbers document will be updated.

5.2 SERVICEDISCOVERYSERVER SERVICE CLASS ATTRIBUTE DEFINITIONS

This service class describes service record that contains attributes of the service discovery server itself. The attributes listed in this section are only valid if the ServiceClassIDList attribute contains the ServiceDiscoveryServerServiceClassID. All of the universal attributes may be included in service records of the ServiceDiscoveryServer class.

5.2.1 ServiceRecordHandle Attribute

Described in the universal attribute definition for ServiceRecordHandle.

Value

A 32-bit integer with the value 0x00000000.

5.2.2 ServiceClassIDList Attribute

Described in the universal attribute definition for ServiceClassIDList.

Value

A UUID representing the ServiceDiscoveryServerServiceClassID.

5.2.3 VersionNumberList Attribute

Attribute Name	Attribute ID	Attribute Value Type
VersionNumberList	0x0200	Data Element Sequence

Description:

The VersionNumberList is a data element sequence in which each element of the sequence is a version number supported by the SDP server.

A version number is a 16-bit unsigned integer consisting of two fields. The higher-order 8 bits contain the major version number field and the low-order 8 bits contain the minor version number field. The initial version of SDP has a major version of 1 and a minor version of 0. When upward compatible changes are made to the protocol, the minor version number will be incremented. If incompatible changes are made to SDP, the major version number will be incremented. This guarantees that if a client and a server support a common major version number, they can communicate if each uses only features of the specification with a minor version number that is supported by both client and server.

5.2.4 ServiceDatabaseState Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceDatabaseState	0x0201	32-bit unsigned integer

Description:

The ServiceDatabaseState is a 32-bit integer that is used to facilitate caching of service records. If this attribute exists, its value shall be changed when any of the other service records are added to or deleted from the server's database. If this value has not changed since the last time a client queried its value, the client knows that a) none of the other service records maintained by the SDP server have been added or deleted; and b) any service record handles acquired from the server are still valid. A client shall query this attribute's value when a connection to the server is established, prior to using any service record handles acquired during a previous connection.

Note that the ServiceDatabaseState attribute does not change when existing service records are modified, including the addition, removal, or modification of service attributes. A service record's ServiceRecordState attribute indicates when that service record is modified.

5.2.5 Reserved Attribute IDs

Attribute IDs in the range of 0x0202-0x02FF are reserved.

5.3 BROWSEGROUPDESCRIPTOR SERVICE CLASS ATTRIBUTE DEFINITIONS

This service class describes the ServiceRecord provided for each BrowseGroupDescriptor service offered on a Bluetooth device. The attributes listed in this section are only valid if the ServiceClassIDList attribute contains the BrowseGroupDescriptorServiceClassID. Note that all of the universal attributes may be included in service records of the BrowseGroupDescriptor class.

5.3.1 ServiceClassIDList Attribute

Described in the universal attribute definition for ServiceClassIDList.

Value

A UUID representing the BrowseGroupDescriptorServiceClassID.

5.3.2 GroupID Attribute

Attribute Name	Attribute ID	Attribute Value Type
GroupID	0x0200	UUID

Description:

This attribute contains a UUID that can be used to locate services that are members of the browse group that this service record describes.

5.3.3 Reserved Attribute IDs

Attribute IDs in the range of 0x0201-0x02FF are reserved.

6 SECURITY

In Security Mode 4, SDP should use no security but may use security (an authenticated or unauthenticated link key with encryption). See [\[Part C\] Section 5.2.2 on page 312](#)).

APPENDIX A BACKGROUND INFORMATION

A.1 SERVICE DISCOVERY

As computing continues to move to a network-centric model, finding and making use of services that may be available in the network becomes increasingly important. Services can include common ones such as printing, paging, FAXing, and so on, as well as various kinds of information access such as teleconferencing, network bridges and access points, eCommerce facilities, and so on — most any kind of service that a server or service provider might offer. In addition to the need for a standard way of discovering available services, there are other considerations: getting access to the services (finding and obtaining the protocols, access methods, “drivers” and other code necessary to utilize the service), controlling access to the services, advertising the services, choosing among competing services, billing for services, and so on. This problem is widely recognized; many companies, standards bodies and consortia are addressing it at various levels in various ways. Service Location Protocol (SLP), Jini™, and Salutation™, to name just a few, all address some aspect of service discovery.

A.2 BLUETOOTH SERVICE DISCOVERY

Bluetooth Service Discovery Protocol (SDP) addresses service discovery specifically for the Bluetooth environment. It is optimized for the highly dynamic nature of Bluetooth communications. SDP focuses primarily on discovering services available from or through Bluetooth devices. SDP does not define methods for accessing services; once services are discovered with SDP, they can be accessed in various ways, depending upon the service. This might include the use of other service discovery and access mechanisms such as those mentioned above; SDP provides a means for other protocols to be used along with SDP in those environments where this can be beneficial. While SDP can coexist with other service discovery protocols, it does not require them. In Bluetooth environments, services can be discovered using SDP and can be accessed using other protocols defined by Bluetooth.

APPENDIX B EXAMPLE SDP TRANSACTIONS

The following are simple examples of typical SDP transactions. These are meant to be illustrative of SDP flows. The examples do not consider:

- Caching (in a caching system, the SDP client would make use of the ServiceRecordState and ServiceDatabaseState attributes);
- Service availability (if this is of interest, the SDP client should use the ServiceAvailability and/or ServiceTimeToLive attributes);
- SDP versions (the VersionNumberList attribute could be used to determine compatible SDP versions);
- SDP Error Responses (an SDP error response is possible for any SDP request that is in error); and
- Communication connection (the examples assume that an L2CAP connection is established).

The examples are meant to be illustrative of the protocol. The format used is `ObjectName [ObjectSizeInBytes] {SubObjectDefinitions}`, but this is not meant to illustrate an interface. The `ObjectSizeInBytes` is the size of the object in decimal. The `SubObjectDefinitions` (inside of {} characters) are components of the immediately enclosing object. Hexadecimal values shown as lower-case letters, such as for transaction IDs and service handles, are variables (the particular value is not important for the illustration, but each such symbol always represents the same value). Comments are included in this manner: /* comment text */

Numeric values preceded by “0x” are hexadecimal, while those preceded by “0b” are binary. All other numeric values are decimal.

B.1 SDP EXAMPLE 1 – SERVICESEARCHREQUEST

The first example is that of an SDP client searching for a generic printing service. The client does not specify a particular type of printing service. In the example, the SDP server has two available printing services. The transaction illustrates:

1. SDP client to SDP server: `SDP_ServiceSearchRequest`, specifying the `PrinterServiceClassID` (represented as a `DataElement` with a 32-bit UUID value of `ppp...ppp`) as the only element of the `ServiceSearchPattern`. The `PrinterServiceClassID` is assumed to be a 32-bit UUID and the data element type for it is illustrated. The `TransactionID` is illustrated as `tttt`.
2. SDP server to SDP client: `SDP_ServiceSearchResponse`, returning handles to two printing services, represented as `qqqqqqqq` for the first printing service and `rrrrrrrr` for the second printing service. The `Transaction ID` is the same value as supplied by the SDP client in the corresponding request (`tttt`).

Service Discovery Protocol (SDP) Specification

```

/* Sent from SDP Client to SDP server */
SDP_ServiceSearchRequest[15] {
    PDUID[1] {
        0x02
    }
    TransactionID[2] {
        0xtttt
    }
    ParameterLength[2] {
        0x000A
    }
    ServiceSearchPattern[7] {
        DataElementSequence[7] {
            0b00110 0b101 0x05
            UUID[5] {
                /* PrinterServiceClassID */
                0b00011 0b010 0xpppppppp
            }
        }
    }
    MaximumServiceRecordCount[2] {
        0x0003
    }
    ContinuationState[1] {
        /* no continuation state */
        0x00
    }
}

/* Sent from SDP server to SDP client */
SDP_ServiceSearchResponse[18] {
    PDUID[1] {
        0x03
    }
    TransactionID[2] {
        0xtttt
    }
    ParameterLength[2] {
        0x000D
    }
    TotalServiceRecordCount[2] {
        0x0002
    }
    CurrentServiceRecordCount[2] {
        0x0002
    }
    ServiceRecordHandleList[8] {
        /* print service 1 handle */
        0xqqqqqqqqq
        /* print service 2 handle */
        0xrrrrrrrrr
    }
    ContinuationState[1] {
        /* no continuation state */
    }
}

```

```

        0x00
    }
}

```

B.2 SDP EXAMPLE 2 – SERVICEATTRIBUTETRANSACTION

The second example continues the first example. In Example 1, the SDP client obtained handles to two printing services. In Example 2, the client uses one of those service handles to obtain the ProtocolDescriptorList attribute for that printing service. The transaction illustrates:

1. SDP client to SDP server: SDP_ServiceAttributeRequest, presenting the previously obtained service handle (the one denoted as `qqqqqqqq`) and specifying the ProtocolDescriptorList attribute ID (AttributeID 0x0004) as the only attribute requested (other attributes could be retrieved in the same transaction if desired). The TransactionID is illustrated as `uuuu` to distinguish it from the TransactionID of Example 1.
2. SDP server to SDP client: SDP_ServiceAttributeResponse, returning the ProtocolDescriptorList for the specified printing service. This protocol stack is assumed to be ((L2CAP), (RFCOMM, 2), (PostscriptStream)). The ProtocolDescriptorList is a data element sequence in which each element is, in turn, a data element sequence whose first element is a UUID representing the protocol, and whose subsequent elements are protocol-specific parameters. In this example, one such parameter is included for the RFCOMM protocol, an 8-bit value indicating RFCOMM server channel 2. The Transaction ID is the same value as supplied by the SDP client in the corresponding request (`uuuu`). The Attributes returned are illustrated as a data element sequence where the protocol descriptors are 32-bit UUIDs and the RFCOMM server channel is a data element with an 8-bit value of 2.

```

/* Sent from SDP Client to SDP server */
SDP_ServiceAttributeRequest[17] {
    PDUID[1] {
        0x04
    }
    TransactionID[2] {
        0xuuuu
    }
    ParameterLength[2] {
        0x000C
    }
    ServiceRecordHandle[4] {
        0xqqqqqqqqq
    }
    MaximumAttributeByteCount[2] {
        0x0080
    }
    AttributeIDList[5] {
        DataElementSequence[5] {

```

Service Discovery Protocol (SDP) Specification

```

    0b00110 0b101 0x03
    AttributeID[3] {
        0b00001 0b001 0x0004
    }
}
ContinuationState[1] {
    /* no continuation state */
    0x00
}
}

/* Sent from SDP server to SDP client */
SDP_ServiceAttributeResponse[38] {
    PDUID[1] {
        0x05
    }
    TransactionID[2] {
        0xuuuu
    }
    ParameterLength[2] {
        0x0021
    }
    AttributeListByteCount[2] {
        0x001E
    }
    AttributeList[30] {
        DataElementSequence[30] {
            0b00110 0b101 0x1C
            Attribute[28] {
                AttributeID[3] {
                    0b00001 0b001 0x0004
                }
               AttributeValue[25] {
                    /* ProtocolDescriptorList */
                    DataElementSequence[25] {
                        0b00110 0b101 0x17
                        /* L2CAP protocol descriptor */
                        DataElementSequence[7] {
                            0b00110 0b101 0x05
                            UUID[5] {
                                /* L2CAP Protocol UUID */
                                0b00011 0b010 <32-bit L2CAP UUID>
                            }
                        }
                    }
                    /* RFCOMM protocol descriptor */
                    DataElementSequence[9] {
                        0b00110 0b101 0x07
                        UUID[5] {
                            /* RFCOMM Protocol UUID */
                            0b00011 0b010 <32-bit RFCOMM UUID>
                        }
                    }
                    /* parameter for server 2 */
                    Uint8[2] {
                        0b00001 0b000 0x02
                    }
                }
            }
        }
    }
}

```



```
/* PostscriptStream protocol descriptor */
DataElementSequence[7] {
    0b00110 0b101 0x05
    UUID[5] {
        /* PostscriptStream Protocol UUID */
        0b00011 0b010 <32-bit PostscriptStream UUID>
    }
}
ContinuationState[1] {
    /* no continuation state */
    0x00
}
```

B.3 SDP EXAMPLE 3 – SERVICESEARCHATTRIBUTETRANSACTION

The third example is a form of service browsing, although it is not generic browsing in that it does not make use of SDP browse groups. Instead, an SDP client is searching for available Synchronization services that can be presented to the user for selection. The SDP client does not specify a particular type of synchronization service. In the example, the SDP server has three available synchronization services: an address book synchronization service and a calendar synchronization service (both from the same provider), and a second calendar synchronization service from a different provider. The SDP client is retrieving the same attributes for each of these services; namely, the data formats supported for the synchronization service (vCard, vCal, iCal, etc.) and those attributes that are relevant for presenting information to the user about the services. Also assume that the maximum size of a response is 400 bytes. Since the result is larger than this, the SDP client will repeat the request supplying a continuation state parameter to retrieve the remainder of the response. The transaction illustrates:

1. SDP client to SDP server: SDP_ServiceSearchAttributeRequest, specifying the generic SynchronisationServiceClassID (represented as a data element whose 32-bit UUID value is `sss...sss`) as the only element of the ServiceSearchPattern. The SynchronisationServiceClassID is assumed to be a 32-bit UUID. The requested attributes are the ServiceRecordHandle (Attribute ID 0x0000), ServiceClassIDList (AttributeID 0x0001), IconURL (AttributeID 0x000C), ServiceName (AttributeID 0x0100), ServiceDescription (AttributeID 0x0101), and ProviderName (AttributeID 0x0102) attributes; as well as the service-specific SupportedDataStores (AttributeID 0x0301). Since the first two attribute IDs (0x0000 and 0x0001) and three other attribute IDs(0x0100, 0x0101, and 0x0102 are consecutive, they are specified as attribute ranges. The TransactionID is illustrated as `vvvv` to distinguish it

from the TransactionIDs of the other Examples.

Note that values in the service record's primary language are requested for the text attributes (ServiceName, ServiceDescription and ProviderName) so that absolute attribute IDs may be used, rather than adding offsets to a base obtained from the LanguageBaseAttributeIDList attribute.

2. SDP server to SDP client: SDP_ServiceSearchAttributeResponse, returning the specified attributes for each of the three synchronization services. In the example, each ServiceClassIDList is assumed to contain a single element, the generic SynchronisationServiceClassID (a 32-bit UUID represented as sss...sss). Each of the other attributes contain illustrative data in the example (the strings have illustrative text; the icon URLs are illustrative, for each of the respective three synchronization services; and the SupportedDataStore attribute is represented as an unsigned 8-bit integer where 0x01 = vCard2.1, 0x02 = vCard3.0, 0x03 = vCal1.0 and 0x04 = iCal). Note that one of the service records (the third for which data is returned) has no ServiceDescription attribute. The attributes are returned as a data element sequence, where each element is in turn a data element sequence representing a list of attributes. Within each attribute list, the ServiceClassIDList is a data element sequence while the remaining attributes are single data elements. The Transaction ID is the same value as supplied by the SDP client in the corresponding request (0xvvvv). Since the entire result cannot be returned in a single response, a non-null continuation state is returned in this first response.
3. Note that the total length of the initial data element sequence (487 in the example) is indicated in the first response, even though only a portion of this data element sequence (368 bytes in the example, as indicated in the AttributeLists byte count) is returned in the first response. The remainder of this data element sequence is returned in the second response (without an additional data element header).
4. SDP client to SDP server: SDP_ServiceSearchAttributeRequest, with the same parameters as in step 1, except that the continuation state received from the server in step 2 is included as a request parameter. The TransactionID is changed to 0xwww to distinguish it from previous request.
5. SDP server to SDP client: SDP_ServiceSearchAttributeResponse, with the remainder of the result computed in step 2 above. Since all of the remaining result fits in this second response, a null continuation state is included.

```
/* Part 1 -- Sent from SDP Client to SDP server */
SdpSDP_ServiceSearchAttributeRequest[33] {
    PDUID[1] {
        0x06
    }
    TransactionID[2] {
        0xvvvv
    }
    ParameterLength[2] {
        0x001C
    }
}
```

Service Discovery Protocol (SDP) Specification

```

    }
    ServiceSearchPattern[7] {
        DataElementSequence[7] {
            0b00110 0b101 0x05
            UUID[5] {
                /* SynchronisationServiceClassID */
                0b00011 0b010 0xssssssss
            }
        }
    }
    MaximumAttributeByteCount[2] {
        0x0190
    }
    AttributeIDList[18] {
        DataElementSequence[18] {
            0b00110 0b101 0x10
            AttributeIDRange[5] {
                0b00001 0b010 0x00000001
            }
            AttributeID[3] {
                0b00001 0b001 0x000C
            }
            AttributeIDRange[5] {
                0b00001 0b010 0x01000102
            }
            AttributeID[3] {
                0b00001 0b001 0x0301
            }
        }
    }
    ContinuationState[1] {
        /* no continuation state */
        0x00
    }
}

/* Part 2 -- Sent from SDP server to SDP client */
SdpSDP_ServiceSearchAttributeResponse[384] {
    PDUID[1] {
        0x07
    }
    TransactionID[2] {
        0xvvvv
    }
    ParameterLength[2] {
        0x017B
    }
    AttributeListByteCount[2] {
        0x0170
    }
    AttributeLists[368] {
        DataElementSequence[487] {
            0b00110 0b110 0x01E4
            DataElementSequence[178] {
                0b00110 0b101 0xB0
                Attribute[8] {
                    AttributeID[3] {
                        0b00001 0b001 0x0000

```

Service Discovery Protocol (SDP) Specification

```

        }
        AttributeValue[5] {
            /* service record handle */
            0b00001 0b010 0xhhhhhhhh
        }
    }
    Attribute[10] {
        AttributeID[3] {
            0b00001 0b001 0x0001
        }
    }
    AttributeValue[7] {
        DataElementSequence[7] {
            0b00110 0b101 0x05
            UUID[5] {
                /* SynchronisationServiceClassID */
                0b00011 0b010 0xssssssss
            }
        }
    }
}
Attribute[35] {
    AttributeID[3] {
        0b00001 0b001 0x000C
    }
    AttributeValue[32] {
        /* IconURL; '*' replaced by client application */
        0b01000 0b101 0x1E
        "http://Synchronisation/icons/*"
    }
}
Attribute[22] {
    AttributeID[3] {
        0b00001 0b001 0x0100
    }
    AttributeValue[19] {
        /* service name */
        0b00100 0b101 0x11
        "Address Book Sync"
    }
}
Attribute[59] {
    AttributeID[3] {
        0b00001 0b001 0x0101
    }
    AttributeValue[56] {
        /* service description */
        0b00100 0b101 0x36
        "Synchronisation Service for"
        " vCard Address Book Entries"
    }
}
Attribute[37] {
    AttributeID[3] {
        0b00001 0b001 0x0102
    }
    AttributeValue[34] {
        /* service provider */
    }
}

```

Service Discovery Protocol (SDP) Specification

```

        0b00100 0b101 0x20
        "Synchronisation Specialists Inc."
    }
}
Attribute[5] {
    AttributeID[3] {
        0b00001 0b001 0x0301
    }
    AttributeValue[2] {
        /* Supported Data Store 'phonebook' */
        0b00001 0b000 0x01
    }
}
DataElementSequence[175] {
    0b00110 0b101 0xAD
    Attribute[8] {
        AttributeID[3] {
            0b00001 0b001 0x0000
        }
        AttributeValue[5] {
            /* service record handle */
            0b00001 0b010 0xmxxxxxxxx
        }
    }
    Attribute[10] {
        AttributeID[3] {
            0b00001 0b001 0x0001
        }
        AttributeValue[7] {
            DataElementSequence[7] {
                0b00110 0b101 0x05
                UUID[5] {
                    /* SynchronisationServiceClassID */
                    0b00011 0b010 0xsssssss
                }
            }
        }
    }
    Attribute[35] {
        AttributeID[3] {
            0b00001 0b001 0x000C
        }
        AttributeValue[32] {
            /* IconURL; '*' replaced by client application */
            0b01000 0b101 0x1E
            "http://Synchronisation/icons/*"
        }
    }
    Attribute[21] {
        AttributeID[3] {
            0b00001 0b001 0x0100
        }
        AttributeValue[18] {
            /* service name */
            0b00100 0b101 0x10
            "Appointment Sync"
        }
    }
}

```

Service Discovery Protocol (SDP) Specification

```

        }
    }

Attribute[57] {
    AttributeID[3] {
        0b00001 0b001 0x0101
    }
   AttributeValue[54] {
        /* service description */
        0b00100 0b101 0x34
        "Synchronisation Service for"
        " vCal Appointment Entries"
    }
}

Attribute[37] {
    AttributeID[3] {
        0b00001 0b001 0x0102
    }
   AttributeValue[34] {
        /* service provider */
        0b00100 0b101 0x20
        "Synchronisation Specialists Inc."
    }
}

Attribute[5] {
    AttributeID[3] {
        0b00001 0b001 0x0301
    }
   AttributeValue[2] {
        /* Supported Data Store 'calendar' */
        0b00001 0b000 0x03
    }
}
}

/* } Data element sequence of attribute lists */
/* is not completed in this PDU. */
}

ContinuationState[9] {
    /* 8 bytes of continuation state */
    0x08 0xzzzzzzzzzzzzzzz
}
}

/* Part 3 -- Sent from SDP Client to SDP server */
SdpSDP_ServiceSearchAttributeRequest[41] {
    PDUID[1] {
        0x06
    }
    TransactionID[2] {
        0xwwww
    }
    ParameterLength[2] {
        0x0024
    }
    ServiceSearchPattern[7] {
        DataElementSequence[7] {
            0b00110 0b101 0x05
            UUID[5] {

```

Service Discovery Protocol (SDP) Specification

```

        /* SynchronisationServiceClassID */
        0b00011 0b010 0xssssssss
    }
}
MaximumAttributeByteCount[2] {
    0x0180
}
AttributeIDList[18] {
    DataElementSequence[18] {
        0b00110 0b101 0x10
        AttributeIDRange[5] {
            0b00001 0b010 0x00000001
        }
        AttributeID[3] {
            0b00001 0b001 0x000C
        }
        AttributeIDRange[5] {
            0b00001 0b010 0x01000102
        }
        AttributeID[3] {
            0b00001 0b001 0x0301
        }
    }
}
ContinuationState[9] {
    /* same 8 bytes of continuation state */
    /* received in part 2 */
    0x08 0xzzzzzzzzzzzzzzz
}
}

```

Part 4 -- Sent from SDP server to SDP client

```

SdpSDP_ServiceSearchAttributeResponse[115] {
    PDUID[1] {
        0x07
    }
    TransactionID[2] {
        0xwwww
    }
    ParameterLength[2] {
        0x006E
    }
    AttributeListByteCount[2] {
        0x006B
    }
    AttributeLists[107] {
        /* Continuing the data element sequence of */
        /* attribute lists begun in Part 2. */
        DataElementSequence[107] {
            0b00110 0b101 0x69
            Attribute[8] {
                AttributeID[3] {
                    0b00001 0b001 0x0000
                }
               AttributeValue[5] {

```

Service Discovery Protocol (SDP) Specification

```

        /* service record handle */
        0b00001 0b010 0xffffffff
    }
}

Attribute[10] {
    AttributeID[3] {
        0b00001 0b001 0x0001
    }
    AttributeValue[7] {
        DataElementSequence[7] {
            0b00110 0b101 0x05
            UUID[5] {
                /* SynchronisationServiceClassID */
                0b00011 0b010 0xsssssss
            }
        }
    }
}

Attribute[35] {
    AttributeID[3] {
        0b00001 0b001 0x000C
    }
    AttributeValue[32] {
        /* IconURL; '*' replaced by client application */
        0b01000 0b101 0x1E
        "http://DevManufacturer/icons/*"
    }
}
}

Attribute[18] {
    AttributeID[3] {
        0b00001 0b001 0x0100
    }
    AttributeValue[15] {
        /* service name */
        0b00100 0b101 0x0D
        "Calendar Sync"
    }
}
}

Attribute[29] {
    AttributeID[3] {
        0b00001 0b001 0x0102
    }
    AttributeValue[26] {
        /* service provider */
        0b00100 0b101 0x18
        "Device Manufacturer Inc."
    }
}
}

Attribute[5] {
    AttributeID[3] {
        0b00001 0b001 0x0301
    }
    AttributeValue[2] {
        /* Supported Data Store 'calendar' */
        0b00001 0b000 0x03
    }
}
}

```

Service Discovery Protocol (SDP) Specification

```
        }
        /* This completes the data element sequence */
        /* of attribute lists begun in Part 2. */
    }
ContinuationState[1] {
    /* no continuation state */
    0x00
}
}
```

Core System Package [Host volume]

Part C

GENERIC ACCESS PROFILE

This profile defines the generic procedures related to discovery of Bluetooth devices (idle mode procedures) and link management aspects of connecting to Bluetooth devices (connecting mode procedures). It also defines procedures related to use of different security levels. In addition, this profile includes common format requirements for parameters accessible on the user interface level.

CONTENTS

Foreword	285
1 Introduction	286
1.1 Scope	286
1.2 Symbols and Conventions	287
1.2.1 Requirement Status Symbols.....	287
1.2.2 Signaling diagram conventions	288
1.2.3 Notation for Timers and Counters	288
2 Profile Overview.....	289
2.1 Profile Stack.....	289
2.2 Profile Roles	289
2.2.1 Roles when Operating over BR/EDR Physical Transport.....	289
2.2.2 Roles when Operating over an LE Physical Transport	290
2.2.2.1 Broadcaster Role.....	290
2.2.2.2 Observer Role	290
2.2.2.3 Peripheral Role	291
2.2.2.4 Central Role	291
2.2.2.5 Concurrent Operation in Multiple GAP Roles	291
2.3 User Requirements and Scenarios	293
2.4 Profile Fundamentals.....	293
2.5 Conformance	293
3 User Interface Aspects	294
3.1 The User Interface Level	294
3.2 Representation of Bluetooth Parameters.....	294
3.2.1 Bluetooth Device Address (BD_ADDR)	294
3.2.1.1 Definition	294
3.2.1.2 Term on user interface level	294
3.2.1.3 Representation	294
3.2.2 Bluetooth Device Name (the user-friendly name)	295
3.2.2.1 Definition	295
3.2.2.2 Term on user interface level	295
3.2.2.3 Representation	295
3.2.3 Bluetooth Passkey (Bluetooth PIN).....	296
3.2.3.1 Definition	296
3.2.3.2 Terms at user interface level	296
3.2.3.3 Representation	296
3.2.4 Class of Device	297
3.2.4.1 Definition	297
3.2.4.2 Term on user interface level	297

3.2.4.3	Representation	298
3.2.4.4	Usage	298
3.2.5	Appearance Characteristic.....	298
3.2.5.1	Definition	298
3.2.5.2	Usage at user interface level.....	298
3.2.5.3	Representation	298
3.3	Pairing	299
4	Modes – BR/EDR Physical Transport.....	300
4.1	Discoverability Modes.....	300
4.1.1	Non-discoverable Mode	301
4.1.1.1	Definition	301
4.1.1.2	Term on UI-level	301
4.1.2	Limited Discoverable Mode.....	301
4.1.2.1	Definition	301
4.1.2.2	Conditions	302
4.1.2.3	Term on UI-level	302
4.1.3	General Discoverable Mode	303
4.1.3.1	Definition	303
4.1.3.2	Conditions	303
4.1.3.3	Term on UI-level	303
4.2	Connectability Modes	304
4.2.1	Non-connectable Mode.....	304
4.2.1.1	Definition	304
4.2.1.2	Term on UI-level	304
4.2.2	Connectable Mode.....	304
4.2.2.1	Definition	304
4.2.2.2	Term on UI-level	305
4.3	Bondable Modes.....	306
4.3.1	Non-bondable Mode	306
4.3.1.1	Definition	306
4.3.1.2	Term on UI-level	306
4.3.2	Bondable Mode.....	306
4.3.2.1	Definition	306
4.3.2.2	Term on UI-level	307
4.4	Synchronizability Modes	307
4.4.1	Non-synchronizable Mode	307
4.4.1.1	Definition	307
4.4.1.2	Term on UI-level	307
4.4.2	Synchronizable Mode	307
4.4.2.1	Definition	307
5	Security Aspects – BR/EDR Physical Transport.....	308
5.1	Authentication.....	308

Generic Access Profile

5.1.1	Purpose.....	308
5.1.2	Term on UI level.....	308
5.1.3	Procedure	309
5.1.4	Conditions	309
5.2	Security Modes.....	310
5.2.1	Legacy Security Modes.....	311
5.2.1.1	Security mode 1 (non-secure).....	311
5.2.1.2	Security mode 2 (service level enforced security).....	311
5.2.1.3	Security mode 3 (link level enforced security).....	312
5.2.2	Security Mode 4 (service level enforced security)	312
5.2.2.1	Security for Access to Remote Service (Initiating Side)	314
5.2.2.2	Security for Access to Local Service by Remote Device (Responding Side).....	317
5.2.2.3	Simple Pairing after Authentication Failure ..	320
5.2.2.4	IO Capabilities	321
5.2.2.5	Mapping of Input / Output Capabilities to IO Capability.....	321
5.2.2.6	IO and OOB Capability Mapping to Authentication Stage 1 Method	322
5.2.2.7	Out of Band (OOB).....	323
5.2.2.8	Security Database	325
6	Idle Mode Procedures – BR/EDR Physical Transport.....	328
6.1	General Inquiry	328
6.1.1	Purpose.....	328
6.1.2	Term on UI level.....	328
6.1.3	Description	329
6.1.4	Conditions	329
6.2	Limited Inquiry	329
6.2.1	Purpose.....	329
6.2.2	Term on UI level.....	330
6.2.3	Description	330
6.2.4	Conditions	330
6.3	Name Discovery	331
6.3.1	Purpose.....	331
6.3.2	Term on UI level.....	331
6.3.3	Description	331
6.3.3.1	Name request.....	331
6.3.3.2	Name discovery.....	331
6.3.4	Conditions	332

6.4	Device Discovery	332
6.4.1	Purpose.....	332
6.4.2	Term on UI Level.....	332
6.4.3	Description	333
6.4.4	Conditions	333
6.5	Bonding	334
6.5.1	Purpose.....	334
6.5.2	Term on UI level.....	334
6.5.3	Description	334
6.5.3.1	General Bonding	334
6.5.3.2	Dedicated Bonding	335
6.5.4	Conditions	336
7	Establishment Procedures – BR/EDR Physical Transport.....	337
7.1	Link Establishment.....	337
7.1.1	Purpose.....	337
7.1.2	Term on UI Level.....	337
7.1.3	Description	338
7.1.3.1	B in security mode 1, 2, or 4.....	338
7.1.3.2	B in security mode 3	339
7.1.4	Conditions	339
7.2	Channel Establishment.....	340
7.2.1	Purpose.....	340
7.2.2	Term on UI level.....	340
7.2.3	Description	340
7.2.3.1	B in security mode 2 or 4.....	341
7.2.3.2	B in security mode 1 or 3.....	341
7.2.4	Conditions	341
7.3	Connection Establishment	342
7.3.1	Purpose.....	342
7.3.2	Term on UI level.....	342
7.3.3	Description	342
7.3.3.1	B in security mode 2 or 4.....	342
7.3.3.2	B in security mode 1 or 3.....	343
7.3.4	Conditions	343
7.4	Establishment of Additional Connection	343
7.5	Synchronization Establishment.....	344
7.5.1	Purpose.....	344
7.5.2	Term on UI Level.....	344
7.5.3	Description	344
7.5.4	Conditions	344

8	Extended Inquiry Response Data Format.....	346
9	Operational Modes and Procedures – LE Physical Transport....	348
9.1	Broadcast Mode and Observation Procedure	348
9.1.1	Broadcast Mode.....	348
9.1.1.1	Definition	348
9.1.1.2	Conditions	349
9.1.2	Observation Procedure.....	349
9.1.2.1	Definition	349
9.1.2.2	Conditions	349
9.2	Discovery Modes and Procedures.....	349
9.2.1	Requirements.....	350
9.2.2	Non-Discoverable Mode	350
9.2.2.1	Description	350
9.2.2.2	Conditions	350
9.2.3	Limited Discoverable Mode.....	351
9.2.3.1	Description	351
9.2.3.2	Conditions	351
9.2.4	General Discoverable Mode	352
9.2.4.1	Description	352
9.2.4.2	Conditions	352
9.2.5	Limited Discovery Procedure	354
9.2.5.1	Description	354
9.2.5.2	Conditions	354
9.2.6	General Discovery Procedure.....	355
9.2.6.1	Description	355
9.2.6.2	Conditions	356
9.2.7	Name Discovery Procedure	356
9.2.7.1	Description	356
9.2.7.2	Conditions	356
9.3	Connection Modes and Procedures	357
9.3.1	Requirements.....	357
9.3.2	Non-Connectable Mode	358
9.3.2.1	Description	358
9.3.2.2	Conditions	358
9.3.3	Directed Connectable Mode	358
9.3.3.1	Description	358
9.3.3.2	Conditions	358
9.3.4	Undirected Connectable Mode	359
9.3.4.1	Description	359
9.3.4.2	Conditions	359
9.3.5	Auto Connection Establishment Procedure	359
9.3.5.1	Description	359

Generic Access Profile

9.3.5.2	Conditions	359
9.3.6	General Connection Establishment Procedure.....	360
9.3.6.1	Description	360
9.3.6.2	Conditions	361
9.3.7	Selective Connection Establishment Procedure.....	362
9.3.7.1	Description	362
9.3.7.2	Conditions	362
9.3.8	Direct Connection Establishment Procedure	364
9.3.8.1	Description	364
9.3.8.2	Conditions	364
9.3.9	Connection Parameter Update Procedure.....	365
9.3.9.1	Description	365
9.3.9.2	Conditions	365
9.3.10	Terminate Connection Procedure	366
9.3.10.1	Description	366
9.3.10.2	Conditions	366
9.3.11	Connection Establishment Timing Parameters	366
9.3.11.1	Description	366
9.3.11.2	Conditions	366
9.3.12	Connection Interval Timing Parameters.....	367
9.3.12.1	Description	367
9.3.12.2	Conditions	367
9.4	Bonding Modes and Procedures	368
9.4.1	Requirements.....	368
9.4.2	Non-Bondable Mode	369
9.4.2.1	Description	369
9.4.2.2	Conditions	369
9.4.3	Bondable Mode.....	369
9.4.3.1	Description	369
9.4.3.2	Conditions	369
9.4.4	Bonding Procedure	370
9.4.4.1	Description	370
9.4.4.2	Conditions	370
10	Security Aspects – LE Physical Transport.....	371
10.1	Requirements	371
10.2	LE Security Modes	371
10.2.1	LE Security Mode 1.....	372
10.2.2	LE Security Mode 2.....	372
10.2.3	Mixed Security Modes Requirements	373
10.2.4	Secure Connections Only Mode	373
10.3	Authentication Procedure	374

10.3.1	Responding to a Service Request	374
10.3.1.1	Handling of GATT Indications and Notifications	378
10.3.1.2	Cross-transport Key Generation.....	378
10.3.2	Initiating a Service Request	378
10.3.2.1	Cross-transport Key Generation.....	381
10.4	Data Signing	381
10.4.1	Connection Data Signing Procedure.....	381
10.4.2	Authenticate Signed Data Procedure.....	382
10.5	Authorization Procedure	383
10.6	Encryption Procedure	383
10.7	Privacy Feature	384
10.7.1	Privacy Feature in a Peripheral.....	385
10.7.1.1	Privacy Feature in a Peripheral with Controller-based privacy	385
10.7.1.2	Privacy Feature in a Peripheral with Host-based privacy	385
10.7.2	Privacy Feature in a Central	386
10.7.2.1	Privacy Feature in a Central with Controller-based privacy	386
10.7.2.2	Privacy Feature in a Central with Host-based privacy	386
10.7.3	Privacy Feature in a Broadcaster.....	386
10.7.4	Privacy Feature in an Observer	387
10.8	Random Device Address	387
10.8.1	Static Address	388
10.8.2	Private address	388
10.8.2.1	Non-Resolvable Private Address Generation Procedure	388
10.8.2.2	Resolvable Private Address Generation Procedure	388
10.8.2.3	Resolvable Private Address Resolution Procedure	388
11	Advertising and Scan Response Data Format	389
12	GAP Service and Characteristics for GATT Server	390
12.1	Device Name Characteristic	391
12.2	Appearance Characteristic	391
12.3	Peripheral Preferred Connection Parameters Characteristic...	392
12.4	Central Address Resolution	393
13	BR/EDR/LE Operation.....	394
13.1	Modes, Procedures and Security Aspects.....	394
13.1.1	Discoverable Mode Requirements.....	395

13.2	Bonding for BR/EDR/LE Device Type	395
13.3	Relationship between Physical Transports.....	395
14	BR/EDR/LE Security Aspects	396
14.1	Cross-transport Key Derivation.....	396
14.2	Collision Handling.....	396
15	Bluetooth Device Requirements.....	397
15.1	Bluetooth Device Address	397
15.1.1	Bluetooth Device Address Types	397
15.1.1.1	Public Bluetooth Address	397
15.1.1.2	Random Bluetooth Address	397
15.2	GATT Profile Requirements	397
15.3	SDP Requirements	398
15.4	SDP Service Record Requirement	398
16	Definitions	399
16.1	General Definitions	399
16.2	Connection-related Definitions	399
16.3	Device-related Definitions	400
16.4	Procedure-related Definitions	401
16.5	Security-related Definitions.....	401
17	References	403
Appendix A (Normative): Timers and Constants		404
Appendix B (Informative): Information Flows of Related Procedures		408
B.1	LMP – Authentication	408
B.2	LMP – Pairing	409
B.3	Service Discovery	410
B.4	Generating a Resolvable Private Address	410
B.5	Resolving a Resolvable Private Address	410

FOREWORD

Interoperability between devices from different manufacturers is provided for a specific service and use case, if the devices conform to a Bluetooth SIG-defined profile specification. A profile defines a selection of messages and procedures (generally termed *capabilities*) from the Bluetooth SIG specifications and gives a description of the air interface for specified service(s) and use case(s).

All defined features are process-mandatory. This means that, if a feature is used, it is used in a specified manner. Whether the provision of a feature is mandatory or optional is stated separately for both sides of the Bluetooth air interface.

1 INTRODUCTION

1.1 SCOPE

The purpose of the Generic Access Profile is:

To introduce definitions, recommendations and common requirements related to modes and access procedures that are to be used by transport and application profiles.

To describe how devices are to behave in standby and connecting states in order to guarantee that links and channels always can be established between Bluetooth devices, and that multi-profile operation is possible. Special focus is put on discovery, link establishment and security procedures.

To state requirements on user interface aspects, mainly coding schemes and names of procedures and parameters, that are needed to guarantee a satisfactory user experience.

This profile defines three device types based on the supported Core Configurations as defined in [\[Vol. 0\], Part B Section 3.1](#). The device types are shown in [Table 1.1](#):

Device Type	Description
BR/EDR	Devices that support the “Basic Rate” Core Configuration (see [Vol. 0], Part B Section 4.1)
LE only	Devices that support the “Low Energy” Core Configuration (see [Vol. 0], Part B Section 4.4)
BR/EDR/LE	Devices that support the “Basic Rate and Low Energy Combined” Core Configuration (see [Vol. 0], Part B Section 4.5)

Table 1.1: Device Types

The terms physical transport, physical link and physical channel as defined in [\[Vol 1\] Part A Section 3](#) are used in this specification.

The requirements for device types are shown in [Table 1.2](#).

Device Types	Sections	Support
BR/EDR	1-8, 12, 15-18	C1
LE-only	1-3, 9-12, 15-18	C1
BR/EDR/LE	1-18	C1
C1: Mandatory to support only one device type		

Table 1.2: Requirements for device types

1.2 SYMBOLS AND CONVENTIONS

1.2.1 Requirement Status Symbols

In this document (especially in the profile requirements tables), the following symbols are used:

'M' for mandatory to support (used for capabilities that shall be used in the profile);

'O' for optional to support (used for capabilities that can be used in the profile);

'C' for conditional support (used for capabilities that shall be used in case a certain other capability is supported);

'E' for excluded within profile role (used for capabilities that may be supported by the unit but shall never be used in the profile role);

'N/A' for not applicable (in the given context it is impossible to use this capability).

Some excluded capabilities are capabilities that, according to the relevant Bluetooth specification, are mandatory. These are features that may degrade operation of devices following this profile. Therefore, these features shall never be activated while a unit is operating as a unit within this profile.

In this specification, the word *shall* is used for mandatory requirements, the word *should* is used to express recommendations and the word *may* is used for options.

1.2.2 Signaling diagram conventions

The following arrows are used in diagrams describing procedures:

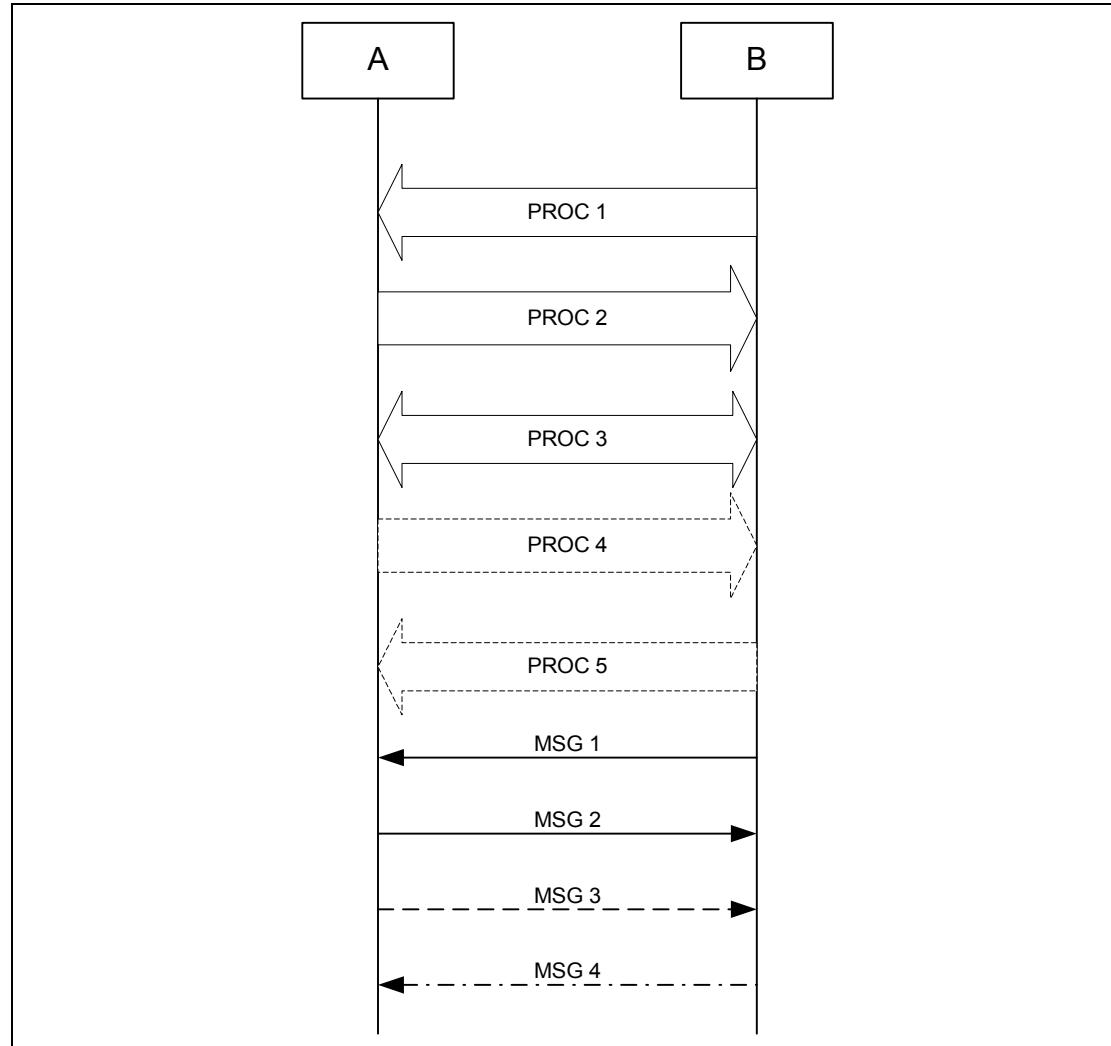


Figure 1.1: Arrows used in signaling diagrams

In Figure 1.1, the following cases are shown: PROC1 is a sub-procedure initiated by B. PROC2 is a sub-procedure initiated by A. PROC3 is a sub-procedure where the initiating side is undefined (may be both A or B). Dashed arrows denote optional steps. PROC4 indicates an optional sub-procedure initiated by A, and PROC5 indicates an optional sub-procedure initiated by B.

MSG1 is a message sent from B to A. MSG2 is a message sent from A to B. MSG3 indicates an optional message from A to B, and MSG4 indicates a conditional message from B to A.

1.2.3 Notation for Timers and Counters

Timers are introduced specific to this profile. To distinguish them from timers used in the Bluetooth protocol specifications and other profiles, these timers are named in the following format: ' $T_{GAP}(nnn)$ '.

2 PROFILE OVERVIEW

2.1 PROFILE STACK

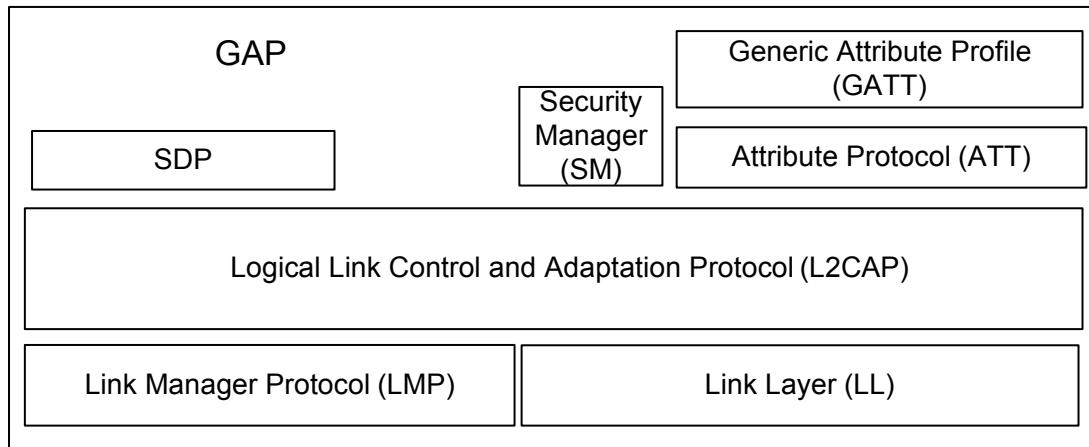


Figure 2.1: Relationship of GAP with lower layers of the Bluetooth architecture

The purpose of this profile is to describe:

- Profile roles
- Discoverability modes and procedures
- Connection modes and procedures
- Security modes and procedures

2.2 PROFILE ROLES

2.2.1 Roles when Operating over BR/EDR Physical Transport

In GAP, for describing the Bluetooth communication that occurs between two devices in the BR/EDR GAP role, the generic notation of the A-party (the *paging device* in case of link establishment, or *initiator* in case of another procedure on an established link) and the B-party (*paged device* or *acceptor*) is used. The A-party is the one that, for a given procedure, initiates device discovery, initiates the establishment of a physical link or initiates a transaction on an existing link.

This profile handles the procedures between two devices related to discovery and connecting (link and connection establishment) for the case where none of the two devices has any link established as well as the case where (at least) one device has a link established (possibly to a third device) before starting the described procedure.

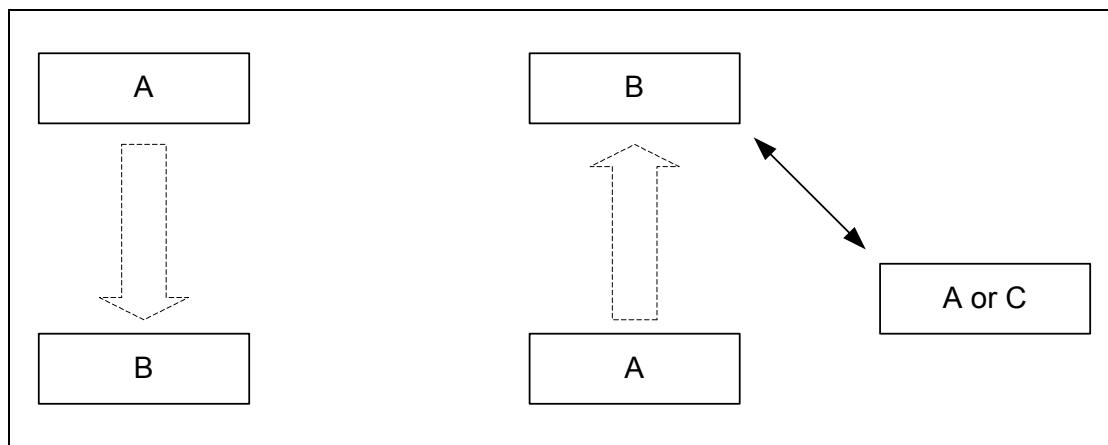


Figure 2.2: This profile covers procedures initiated by one device (A) towards another device (B) that may or may not have an existing Bluetooth link active

The initiator and the acceptor generally operate the generic procedures according to this profile or another profile referring to this profile. If the acceptor operates according to several profiles simultaneously, this profile describes generic mechanisms for how to handle this.

2.2.2 Roles when Operating over an LE Physical Transport

There are four GAP roles defined for devices operating over an LE physical transport:

- Broadcaster
- Observer
- Peripheral
- Central

2.2.2.1 Broadcaster Role

A device operating in the Broadcaster role is a device that sends advertising events as described in [\[Vol. 6\], Part B Section 4.4.2](#). A device operating in the Broadcaster role is referred to as a Broadcaster and shall have a transmitter and may have a receiver.

2.2.2.2 Observer Role

A device operating in the Observer role is a device that receives advertising events as described in [\[Vol. 6\], Part B Section 4.4.3](#). A device operating in the Observer role is referred to as an Observer and shall have a receiver and may have a transmitter.

2.2.2.3 Peripheral Role

Any device that accepts the establishment of an LE physical link using any of the connection establishment procedures as defined in [Section 9](#) is referred to as being in the Peripheral role. A device operating in the Peripheral role will be in the Slave role in the Link Layer Connection State as described in [\[Vol. 6\], Part B Section 4.5](#). A device operating in the Peripheral role is referred to as a Peripheral. A Peripheral shall have both a transmitter and a receiver.

2.2.2.4 Central Role

A device that supports the Central role initiates the establishment of a physical connection. A device operating in the Central role will be in the Master role in the Link Layer Connection State as described in [\[Vol. 6\], Part B Section 4.5](#). A device operating in the Central role is referred to as a Central. A Central shall have both a transmitter and a receiver.

2.2.2.5 Concurrent Operation in Multiple GAP Roles

A device may operate in multiple GAP roles concurrently if supported by the Controller. The Host should read the supported Link Layer States and State combinations from the Controller before any procedures or modes are used.

Only supported and allowed Link Layer States and State combinations may be used as described in [\[Vol. 6\], Part B Section 1.1.1](#). The Physical Layer and Link Layer functionalities of each GAP role are shown in [Table 2.1](#).

	GAP Roles When Operating Over an LE Physical Transport			
	Broadcaster	Observer	Peripheral	Central
Physical Layer functionality:				
• Transmitter Characteristics	M	O	M	M
• Receiver Characteristics	O	M	M	M
Link Layer functionality:				
States:				
• Standby State	M	M	M	M
• Advertising State	M	E	M	E
• Scanning State	E	M	E	M
• Initiating State	E	E	E	M

Generic Access Profile



		GAP Roles When Operating Over an LE Physical Transport			
		Broadcaster	Observer	Peripheral	Central
• Connection State	Slave Role	E	E	M	E
	Master Role	E	E	E	M
<i>Advertising event types:</i>					
• Connectable undirected event		E	E	M	E
• Connectable directed event		E	E	O	E
• Non-connectable undirected event		M	E	O	E
• Scannable undirected event		O	E	O	E
<i>Scanning type:</i>					
• Passive scanning		E	M	E	O
• Active scanning		E	O	E	C1
<i>Link Layer control procedures:</i>					
• Connection Update procedure		E	E	M	M
• Channel Map Update procedure		E	E	M	M
• Encryption procedure		E	E	O	O
• Master-initiated Feature Exchange procedure		E	E	M	M
• Slave-initiated Feature Exchange procedure		E	E	C2	C2
• Connection Parameters Request procedure		E	E	O	O
• Version Exchange procedure		E	E	M	M
• Termination procedure		E	E	M	M
C1: If passive scanning is supported then active scanning is optional, otherwise active scanning is mandatory.					
C2: Mandatory if Connection Parameters Request procedure is supported, otherwise optional					

Table 2.1: Physical Layer and Link Layer functionality for GAP roles when operating over an LE physical transport

2.3 USER REQUIREMENTS AND SCENARIOS

The Bluetooth user should, where expected, be able to connect a Bluetooth device to any other Bluetooth device. Even if the two connected devices don't share any common application, it should be possible for the user to find this out using basic Bluetooth capabilities. When the two devices do share the same application but are from different manufacturers, the ability to connect them should not be blocked just because manufacturers choose to call basic Bluetooth capabilities by different names on the user interface level or implement basic procedures to be executed in different orders.

2.4 PROFILE FUNDAMENTALS

This profile states the requirements on names, values and coding schemes used for names of parameters and procedures experienced on the user interface level.

This profile defines modes of operation that are not service- or profile-specific, but that are generic and can be used by profiles referring to this profile, and by devices implementing multiple profiles.

This profile defines the general procedures that can be used for discovering identities, names and basic capabilities of other Bluetooth devices that are in a mode where they can be discovered. Only procedures where no channel or connection establishment is used are specified.

This profile defines the general procedure for how to create bonds (i.e., dedicated exchange of link keys) between Bluetooth devices.

This profile describes the general procedures that can be used for establishing connections to other Bluetooth devices that are in a mode that allows them to accept connections and service requests.

2.5 CONFORMANCE

Bluetooth devices shall conform to this profile to ensure basic interoperability.

Bluetooth devices that conform to another Bluetooth profile may use adaptations of the generic procedures as specified by that other profile. They shall, however, be compatible with devices compliant to this profile at least on the level of the supported generic procedures.

All capabilities indicated as mandatory for this profile shall be supported in the specified manner (process-mandatory). This also applies for all optional and conditional capabilities for which support is indicated. All mandatory capabilities, and optional and conditional capabilities for which support is indicated, are subject to verification as part of the Bluetooth certification program.

3 USER INTERFACE ASPECTS

3.1 THE USER INTERFACE LEVEL

In the context of this specification, the user interface level refers to places (such as displays, dialog boxes, manuals, packaging, advertising, etc.) where users of Bluetooth devices encounter names, values and numerical representation of Bluetooth terminology and parameters.

This profile specifies the generic terms that should be used on the user interface level.

3.2 REPRESENTATION OF BLUETOOTH PARAMETERS

3.2.1 Bluetooth Device Address (BD_ADDR)

3.2.1.1 *Definition*

A BD_ADDR is the address used by a Bluetooth device as defined in [Section 15.1](#). It is received from a remote device during the device discovery procedure.

3.2.1.2 *Term on user interface level*

When the Bluetooth address is referred to on UI level, the term ‘Bluetooth Device Address’ should be used.

3.2.1.3 *Representation*

On the baseband level the BD_ADDR is represented as 48 bits (see [\[Vol. 2\], Part B Section 1.2](#)). On the Link Layer the public and random device address are represented as 48-bit addresses.

On the UI level the Bluetooth address shall be represented as 12 hexadecimal characters, possibly divided into sub-parts separated by ‘:’ (e.g., ‘000C3E3A4B69’ or ‘00:0C:3E:3A:4B:69’). On the UI level any number shall have the MSB -> LSB (from left to right) ‘natural’ ordering.

3.2.2 Bluetooth Device Name (the user-friendly name)

3.2.2.1 Definition

The Bluetooth device name is the user-friendly name that a Bluetooth device exposes to remote devices. For a device supporting the BR/EDR device type, the name is a character string returned in the LMP_name_res in response to an LMP_name_req. For a device supporting the LE-only device type, the name is a character string held in the Device Name characteristic as defined in [Section 12.1](#).

3.2.2.1.1 Bluetooth Device Name in a Device with BR/EDR/LE Device Type

A BR/EDR/LE device type shall have a single Bluetooth device name which shall be identical irrespective of the physical channel used to perform the name discovery procedure.

For the BR/EDR physical channel the name is received in the LMP_name_res. For the LE physical channel the name can be read from the Device Name characteristic as defined in [Section 12.1](#).

Note: The Device Name Characteristic of the local device can be read by a remote device using ATT over BR/EDR if the local device supports ATT over BR/EDR.

3.2.2.2 Term on user interface level

When the Bluetooth device name is referred to on UI level, the term ‘Bluetooth Device Name’ should be used.

3.2.2.3 Representation

The Bluetooth device name can be up to 248 bytes (see [\[Vol. 2\], Part C Section 4.3.5](#)). It shall be encoded according to UTF-8 (therefore the name entered on the UI level may be restricted to as few as 62 characters if codepoints outside the range U+0000 to U+007F are used).

A device cannot expect that a general remote device is able to handle more than the first 40 characters of the Bluetooth device name. If a remote device has limited display capabilities, it may use only the first 20 characters.

3.2.3 Bluetooth Passkey (Bluetooth PIN)

3.2.3.1 Definition

The Bluetooth passkey may be used to authenticate two Bluetooth devices with each other during the creation of a mutual link key via the pairing procedure. The passkey may be used in the pairing procedures to generate the initial link key.

The PIN may be entered on the UI level but may also be stored in the device; e.g., in the case of a device without sufficient MMI for entering and displaying digits.

3.2.3.2 Terms at user interface level

When the Bluetooth PIN is referred to on UI level, the term ‘Bluetooth Passkey’ should be used.

3.2.3.3 Representation

There are a number of different representations of the Bluetooth passkey. At a high level there are two distinct representations: one used with the Secure Simple Pairing and Security Manager, and another used with legacy pairing (where it is generally referred to as the Bluetooth PIN).

For Secure Simple Pairing and Security Manager, the Bluetooth passkey is a 6-digit numerical value. It is represented as integer value in the range 0x00000000 – 0x000F423F (000000 to 999999). The numeric value may be used as the input to the Authentication Stage 1 for Secure Simple Pairing Passkey Entry (see [\[Vol. 2\], Part H Section 7.2.3](#)), or as the TK value in the Security Manager for the process defined in [\[Vol. 3\], Part H Section 2.3.5](#).

For legacy pairing (see [Section B.2 on page 409](#)), the Bluetooth PIN has different representations on different levels. PINBB is used on baseband level, and PINUI is used on user interface level. PINBB is the PIN used by [1] for calculating the initialization key during the Pairing Procedure.

PINUI is the character representation of the PIN that is entered on the UI level. The transformation from PINUI to PINBB shall be according to UTF-8. PINBB may be 128 bits (16 bytes).

PIN codes may be up to 16 characters. In order to take advantage of the full level of security all PINs should be 16 characters long. Variable PINs should be composed of alphanumeric characters chosen from within the Unicode range U+0000 to U+007F. If the PIN contains any decimal digits these shall be encoded using the Unicode Basic Latin characters (i.e., U+0030 to U+0039) (Note 1).

For compatibility with devices with numeric keypads, fixed PINs shall be composed of only decimal digits, and variable PINs should be composed of only decimal digits.

If a device supports entry of characters outside the Unicode range U+0000 to U+007F, other Unicode code points may be used (Note 2), except the Halfwidth and Fullwidth Forms (i.e., U+FF00 to U+FFEF) shall not be used (Note 3).

Examples:

User-entered code	Corresponding PIN _{BB} [0..length-1] (value as a sequence of octets in hexadecimal notation)
'0196554200906493'	length = 16, value = 0x30 0x31 0x39 0x36 0x35 0x35 0x34 0x32 0x30 0x30 0x39 0x30 0x36 0x34 0x39 0x33
'Børnelitteratur'	length = 16, value = 0x42 0xC3 0xB8 0x72 0x6e 0x65 0x6c 0x69 0x74 0x74 0x65 0x72 0x61 0x74 0x75 0x72

Note 1: This is to prevent interoperability problems since there are decimal digits at other code points (e.g., the Fullwidth digits at code points U+FF10 to U+FF19).

Note 2: Unicode characters outside the Basic Latin range (U+0000 to U+007F) encode to multiple bytes; therefore, when characters outside the Basic Latin range are used the maximum number of characters in the PINUI will be less than 16. The second example illustrates a case where a 15 character string encodes to 16 bytes because the character ø is outside the Basic Latin range and encodes to two bytes (0xC3 0xB8).

Note 3: This is to prevent interoperability problems since the Halfwidth and Fullwidth forms contain alternative variants of ASCII, Katakana, Hangul, punctuation and symbols. All of the characters in the Halfwidth and Fullwidth forms have other related Unicode characters; for example, U+3150 (Hangul Letter AE) can be used instead of U+FFC3 (Halfwidth Hangul Letter AE).

3.2.4 Class of Device

3.2.4.1 Definition

Class of device is a parameter received during the device discovery procedure on the BR/EDR physical transport, indicating the type of device. The Class of Device parameter is only used on BR/EDR and BR/EDR/LE devices using the BR/EDR physical transport.

3.2.4.2 Term on user interface level

The information within the Class of Device parameter should be referred to as 'Bluetooth Device Class' (i.e., the major and minor device class fields) and 'Bluetooth Service Type' (i.e., the service class field). The terms for the defined Bluetooth Device Types and Bluetooth Service Types are defined in [7].

When using a mix of information found in the Bluetooth Device Class and the Bluetooth Service Type, the term 'Bluetooth Device Type' should be used.

3.2.4.3 Representation

The Class of device is a bit field and is defined in [7]. The UI-level representation of the information in the Class of Device is implementation specific.

3.2.4.4 Usage

Some devices provide more than one service and a given service may be provided by different device types. Therefore, the device type does not have a one-to-one relationship with services supported. The major and minor device class field should not be used to determine whether a device supports any specific service(s). It may be used as an indication of devices that are most likely to support desired services before service discovery requests are made, and it may be used to guide the user when selecting among several devices that support the same service.

3.2.5 Appearance Characteristic

3.2.5.1 Definition

The Appearance characteristic contains a UUID that can be mapped to an icon or string that describes the physical representation of the device during the device discovery procedure (See [Section 12.2](#)). It is a characteristic of the GAP service located on the device's GATT server.

3.2.5.2 Usage at user interface level

The Appearance characteristic UUID should be mapped to an icon or string or something similar that conveys to the user a visual description of the device. This allows the user to determine which device is being discovered purely by visual appearance. If a string is displayed, this string should be translated into the language selected by the user for the device.

3.2.5.3 Representation

The Appearance characteristic is a UUID assigned by the Bluetooth SIG and defined in [\[Core Specification Supplement\], Part A, Section 1.12](#). The UI-level representation of the Appearance characteristic value is implementation specific.

3.3 PAIRING

Pairing over a BR/EDR physical link is defined on LMP level (LMP pairing, see [B.2](#)). Pairing over an LE physical link is defined by the Security Manager specification ([\[Vol. 3\], Part H Section 2.3](#)). Either the user initiates the bonding procedure and enters the passkey with the explicit purpose of creating a bond (and maybe also a secure relationship) between two Bluetooth devices, or the user is requested to enter the passkey during the establishment procedure since the devices did not share a common link key beforehand. In the first case, the user is said to perform “bonding (with entering of passkey)” and in the second case the user is said to “authenticate using the passkey.”

4 MODES – BR/EDR PHYSICAL TRANSPORT

Procedure	Ref.	Support
Discoverability modes:	4.1	
Non-discoverable mode		C1
Discoverable mode		O
Limited discoverable mode		C3
General discoverable mode		C4
Connectability modes:	4.2	
Non-connectable mode		O
Connectable mode		M
Bondable modes:	4.3	
Non-bondable mode		O
Bondable mode		C2
Synchronizability modes:	4.4	
Non-synchronizable mode		M
Synchronizable mode		O
C1: If limited discoverable mode is supported, non-discoverable mode is mandatory, otherwise optional.		
C2: If the bonding procedure is supported, support for bondable mode is mandatory, otherwise optional.		
C3: If Discoverable mode is supported, support for Limited Discoverable mode is mandatory, otherwise optional.		
C4: If Discoverable mode is supported, support for General Discoverable mode is mandatory, otherwise optional		

Table 4.1: Conformance requirements related to modes defined in this section

4.1 DISCOVERABILITY MODES

With respect to inquiry, a Bluetooth device shall be either in non-discoverable mode or in a discoverable mode. (The device shall be in one, and only one, discoverability mode at a time.) The two discoverable modes defined here are called limited discoverable mode and general discoverable mode. Inquiry is defined in [Vol. 2], Part B Section 8.4.

When a Bluetooth device is in non-discoverable mode it does not respond to inquiry.

A Bluetooth device is said to be made discoverable, or set into a discoverable mode, when it is in limited discoverable mode or in general discoverable mode. Even when a Bluetooth device is made discoverable, it may be unable to respond to inquiry due to other baseband activity (for example, reserved synchronous slots should have priority over response packets, so that synchronous links may prevent a response from being returned). A Bluetooth device that does not respond to inquiry is called a silent device.

After being made discoverable, the Bluetooth device shall be discoverable for at least $T_{GAP}(103)$.

The speed of discovery is dependent on the configuration of the inquiry scan interval and inquiry scan type of the Bluetooth device. The Host is able to configure these parameters based on trade-offs between power consumption, bandwidth and the desired speed of discovery.

4.1.1 Non-discoverable Mode

4.1.1.1 *Definition*

When a Bluetooth device is in non-discoverable mode, it shall never enter the INQUIRY_SCAN state.

4.1.1.2 *Term on UI-level*

Bluetooth device is ‘non-discoverable’ or in ‘non-discoverable mode’.

4.1.2 Limited Discoverable Mode

4.1.2.1 *Definition*

The limited discoverable mode should be used by devices that need to be discoverable only for a limited period of time, during temporary conditions, or for a specific event. The purpose is to respond to a device that makes a limited inquiry (inquiry using the LIAC).

A Bluetooth device should not be in limited discoverable mode for more than $T_{GAP}(104)$. The scanning for the limited inquiry access code can be done either in parallel or in sequence with the scanning of the general inquiry access code. When in limited discoverable mode, one of the following options shall be used.

- *Parallel scanning*

When a Bluetooth device is in limited discoverable mode and when discovery speed is more important than power consumption or bandwidth, it is recommended that the Bluetooth device enter the INQUIRY_SCAN state at least every $T_{GAP}(105)$ and that Interlaced Inquiry scan is used.

If, however, power consumption or bandwidth is important, but not critical, it is recommended that the Bluetooth device enter the INQUIRY_SCAN state at least every $T_{GAP}(102)$ and Interlaced Inquiry scan is used.

When power consumption or bandwidth is critical it is recommended that the Bluetooth device enter the INQUIRY_SCAN state at least every $T_{GAP}(102)$ and Non-interlaced Inquiry scan is used.

In all cases the Bluetooth device shall enter the INQUIRY_SCAN state at least once in $T_{GAP}(102)$ and scan for the GIAC and the LIAC for at least $T_{GAP}(101)$.

When either a SCO or eSCO link is in operation, it is recommended to use interlaced scan to significantly decrease the discoverability time.

- *Sequential scanning*

When a Bluetooth device is in limited discoverable mode, it shall enter the INQUIRY_SCAN state at least once in $T_{GAP}(102)$ and scan for the GIAC for at least $T_{GAP}(101)$ and enter the INQUIRY_SCAN state more often than once in $T_{GAP}(102)$ and scan for the LIAC for at least $T_{GAP}(101)$.

If an inquiry message is received when in limited discoverable mode, the entry into the INQUIRY_RESPONSE state takes precedence over the next entries into INQUIRY_SCAN state until the inquiry response is completed.

4.1.2.2 Conditions

When a device is in limited discoverable mode it shall set bit no 13 in the Major Service Class part of the Class of Device/Service field [\[7\]](#).

4.1.2.3 Term on UI-level

Bluetooth device is ‘discoverable’ or in ‘discoverable mode’.

4.1.3 General Discoverable Mode

4.1.3.1 Definition

The general discoverable mode shall be used by devices that need to be discoverable continuously or for no specific condition. The purpose is to respond to a device that makes a general inquiry (inquiry using the GIAC).

4.1.3.2 Conditions

When a Bluetooth device is in general discoverable mode and when discovery speed is more important than power consumption or bandwidth, it is recommended that the Bluetooth device enter the INQUIRY_SCAN state at least every $T_{GAP}(105)$ and that Interlaced Inquiry scan is used.

If, however, power consumption or bandwidth is important, but not critical, it is recommended that the Bluetooth device enter the INQUIRY_SCAN state at least every $T_{GAP}(102)$ and Interlaced Inquiry scan is used.

When power consumption or bandwidth is critical it is recommended that the Bluetooth device enter the INQUIRY_SCAN state at least every $T_{GAP}(102)$ and Non-interlaced Inquiry scan is used.

In all cases the Bluetooth device shall enter the INQUIRY_SCAN state at least once in $T_{GAP}(102)$ and scan for the GIAC for at least $T_{GAP}(101)$.

When either a SCO or eSCO link is in operation, it is recommended to use interlaced scan to significantly decrease the discoverability time.

A device in general discoverable mode shall not respond to a LIAC inquiry.

4.1.3.3 Term on UI-level

Bluetooth device is ‘discoverable’ or in ‘discoverable mode’.

4.2 CONNECTABILITY MODES

With respect to paging, a Bluetooth device shall be either in non-connectable mode or connectable mode. Paging is defined in [\[Vol. 2\], Part B Section 8.3](#).

When a Bluetooth device is in non-connectable mode it does not respond to paging. When a Bluetooth device is in connectable mode it responds to paging.

The speed of connections is dependent on the configuration of the page scan interval and page scan type of the Bluetooth device. The Host is able to configure these parameters based on trade-offs between power consumption, bandwidth and the desired speed of connection.

4.2.1 Non-connectable Mode

4.2.1.1 *Definition*

When a Bluetooth device is in non-connectable mode it shall never enter the PAGE_SCAN state.

4.2.1.2 *Term on UI-level*

Bluetooth device is ‘non-connectable’ or in ‘non-connectable mode’.

4.2.2 Connectable Mode

4.2.2.1 *Definition*

When a Bluetooth device is in connectable mode it shall periodically enter the PAGE_SCAN state. The device makes page scan using the Bluetooth device address, BD_ADDR. Connection speed is a trade-off between power consumption / available bandwidth and speed. The Bluetooth Host is able to make these trade-offs using the Page Scan interval, Page Scan window, and Interlaced Scan parameters.

R0 page scanning should be used when connection speeds are critically important and when the paging device has a very good estimate of the Bluetooth clock. Under these conditions it is possible for paging to complete within two times the page scan window. Because the page scan interval is equal to the page scan window it is not possible for any other traffic to go over the Bluetooth link when using R0 page scanning. In R0 page scanning it is not possible to use interlaced scan. R0 page scanning is the highest power consumption mode of operation.

When connection times are critical but the other device either does not have an estimate of the Bluetooth clock or when the estimate is possibly out of date, it is better to use R1 page scanning with a very short page scan interval, $T_{GAP}(106)$, and Interlaced scan. This configuration is also useful to achieve

nearly the same connection speeds as R0 page scanning but using less power and leaving bandwidth available for other connections. Under these circumstances it is possible for paging to complete within $T_{GAP}(106)$. In this case the Bluetooth device shall page scan for at least $T_{GAP}(101)$.

When connection times are important but not critical enough to sacrifice significant bandwidth and/or power consumption it is recommended to use either $T_{GAP}(107)$ or $T_{GAP}(108)$ for the scanning interval. Using Interlaced scan will reduce the connection time by half but may use twice the power consumption. Under these circumstances it is possible for paging to complete within one or two times the page scanning interval depending on whether Interlaced Scan is used. In this case the Bluetooth device shall page scan for at least $T_{GAP}(101)$.

In all cases the Bluetooth device shall enter the PAGE_SCAN state at least once in $T_{GAP}(102)$ and scan for at least $T_{GAP}(101)$.

The page scan interval, page scan window size, and scan type for the six scenarios are described in [Table 4.2](#):

Scenario	Page Scan Interval	Page Scan Window	Scan Type
R0 (1.28s)	$T_{GAP}(107)$	$T_{GAP}(107)$	Normal scan
Fast R1 (100ms)	$T_{GAP}(106)$	$T_{GAP}(101)$	Interlaced scan
Medium R1 (1.28s)	$T_{GAP}(107)$	$T_{GAP}(101)$	Interlaced scan
Slow R1 (1.28s)	$T_{GAP}(107)$	$T_{GAP}(101)$	Normal scan
Fast R2 (2.56s)	$T_{GAP}(108)$	$T_{GAP}(101)$	Interlaced scan
Slow R2 (2.56s)	$T_{GAP}(108)$	$T_{GAP}(101)$	Normal scan

Table 4.2: Page scan parameters for connection speed scenarios

When either a SCO or eSCO link is in operation, it is recommended to use interlaced scan to significantly decrease the connection time.

4.2.2.2 Term on UI-level

Bluetooth device is ‘connectable’ or in ‘connectable mode’.

4.3 BONDABLE MODES

With respect to bonding, a Bluetooth device shall be either in non-bondable mode or in bondable mode. In bondable mode the Bluetooth device accepts bonding initiated by the remote device, and in non-bondable mode it does not.

4.3.1 Non-bondable Mode

4.3.1.1 Definition

When a Bluetooth device is in non-bondable mode it shall not accept a pairing request that results in bonding. Devices in non-bondable mode may accept connections that do not request or require bonding.

A device in non-bondable mode shall respond to a received LMP_in_rand with LMP_not_accepted with the reason *pairing not allowed*. An HCI-compliant host stack shall respond to an HCI_PIN_Code_Request event with the HCI_PIN_Code_Request_Negative_Reply command.

When both devices support Secure Simple Pairing and are in non-bondable mode, the local host shall respond to an IO capability request with the Authentication_Requirements parameter requesting dedicated bonding or general bonding with a negative response. An HCI-compliant host stack shall respond to an HCI_IO_Capabilities_Request event with an HCI_IO_Capabilities_Request_Negative_Reply command.

4.3.1.2 Term on UI-level

Bluetooth device is ‘non-bondable’ or in ‘non-bondable mode’ or “does not accept bonding”.

4.3.2 Bondable Mode

4.3.2.1 Definition

When a Bluetooth device is in bondable mode, and Secure Simple Pairing is not supported by either the local or remote device, the local device shall respond to a received LMP_in_rand with LMP_accepted (or with LMP_in_rand if it has a fixed PIN). An HCI-compliant host stack shall respond to an HCI_PIN_Code_Request event with the HCI_PIN_Code_Request_Reply command.

When both devices support Secure Simple Pairing, the local host shall respond to a user confirmation request with a positive response. An HCI-compliant host stack shall respond to an HCI_User_Confirmation_Request event with an HCI_User_Confirmation_Request_Reply command or an HCI_User_Passkey_Request event with an HCI_User_Passkey_Request_Reply command.

4.3.2.2 Term on UI-level

Bluetooth device is ‘bondable’ or in ‘bondable mode’ or “accepts bonding”.

4.4 SYNCHRONIZABILITY MODES

A Bluetooth device shall be either in non-synchronizable mode or synchronizable mode. The synchronization train procedure is defined in [\[Vol. 2, Part B\] Section 2.7.2 on page 98](#).

When a Bluetooth device is in synchronizable mode, it transmits timing and frequency information for its active Connectionless Slave Broadcast packets. When a Bluetooth device is non-synchronizable, timing and frequency information is not transmitted.

The host is able to configure the Synchronization Train interval based on trade-offs between bandwidth, potential interference to other devices, power consumption, and the desired time for a slave to receive a synchronization train packet.

4.4.1 Non-synchronizable Mode

4.4.1.1 Definition

When a Bluetooth device is in non-synchronizable mode it shall never enter the **synchronization train** substate.

4.4.1.2 Term on UI-level

Bluetooth device is ‘non-synchronizable’ or in ‘non-synchronizable mode’.

4.4.2 Synchronizable Mode

4.4.2.1 Definition

When a Bluetooth device is in synchronizable mode, it shall enter the **synchronization train** substate using a synchronization train interval of $T_{GAP}(\text{Sync_Train_Interval})$.

After being made synchronizable, the Bluetooth device shall be synchronizable for at least $T_{GAP}(\text{Sync_Train_Duration})$.

5 SECURITY ASPECTS – BR/EDR PHYSICAL TRANSPORT

	Procedure	Ref.	Support
1	Authentication	5.1	M
2	Security modes	5.2	
	Security mode 1		E
	Security mode 2		O.1
	Security mode 3		E
	Security mode 4		M
O.1: Security Mode 2 may only be used for backwards compatibility when the remote device does not support Secure Simple Pairing.			

Table 5.1: Conformance requirements related to the generic authentication procedure and the security modes defined in this section

5.1 AUTHENTICATION

5.1.1 Purpose

The generic authentication procedure describes how the LMP-authentication and LMP-pairing procedures are used when authentication is initiated by one Bluetooth device towards another, depending on if a link key exists or not and if pairing is allowed or not.

5.1.2 Term on UI level

‘Bluetooth authentication’.

5.1.3 Procedure

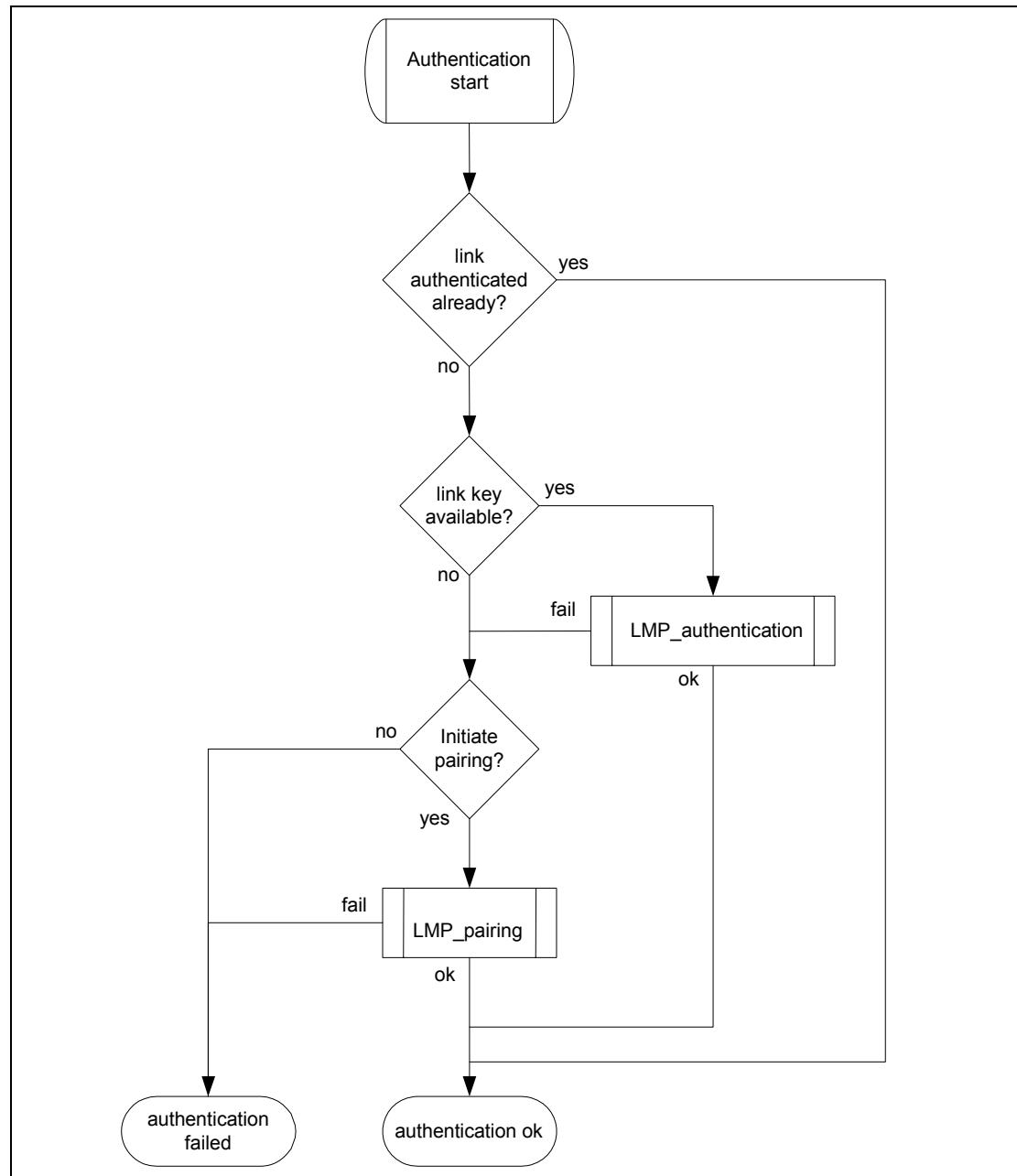


Figure 5.1: Definition of the generic authentication procedure

5.1.4 Conditions

The local device shall initiate authentication after link establishment. The remote device may initiate security during or after link establishment.

5.2 SECURITY MODES

The following flow chart provides an overview of the channel establishment procedures.

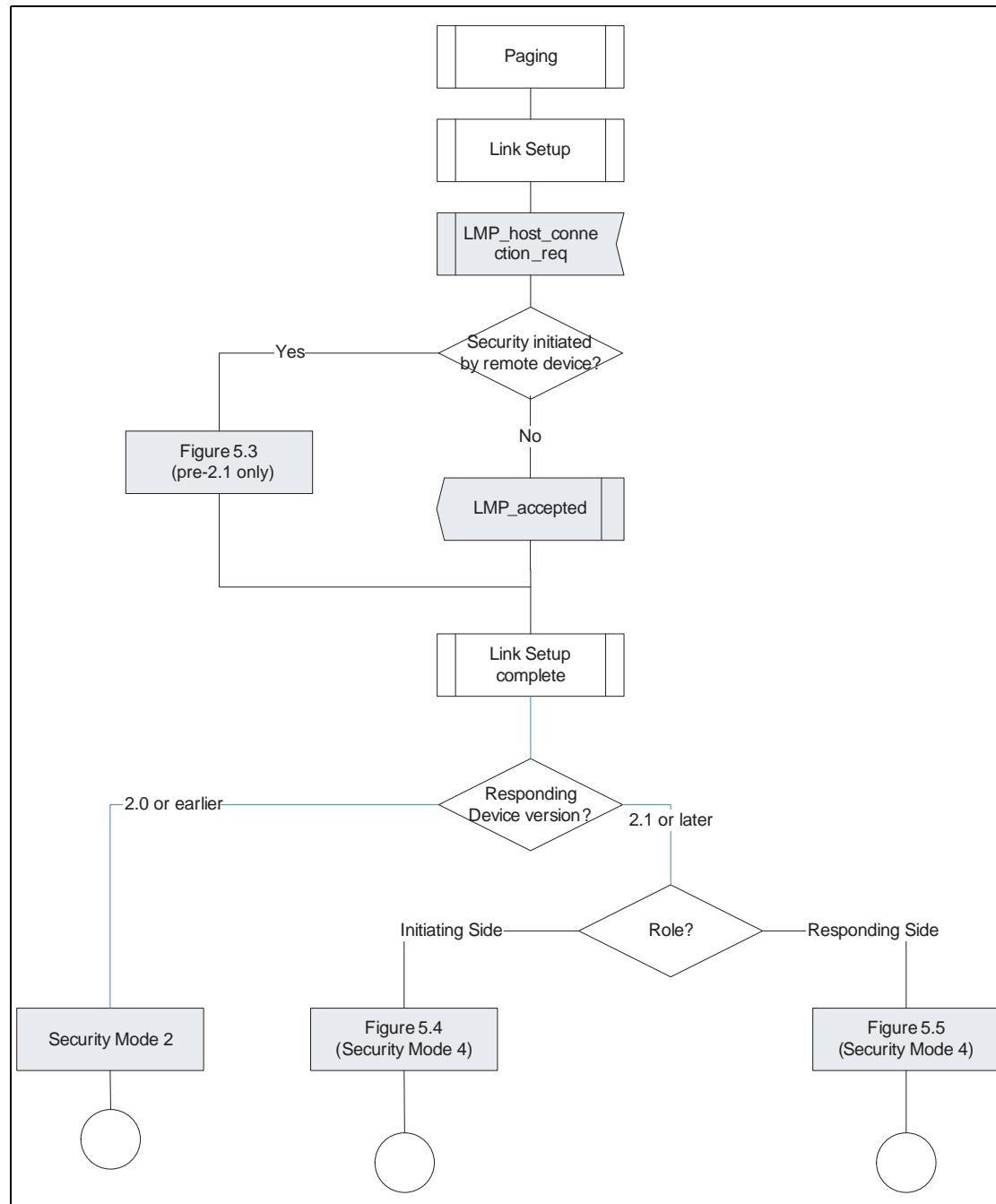


Figure 5.2: Channel establishment with security

A device may support two security modes simultaneously: security mode 2 for backwards compatibility with remote devices that do not support Secure Simple Pairing and security mode 4 for devices that support Secure Simple Pairing.

5.2.1 Legacy Security Modes

Legacy security modes apply to those devices with a Controller or a Host that does not support SSP.

5.2.1.1 Security mode 1 (non-secure)

When a remote Bluetooth device is in security mode 1 it will never initiate any security procedure (i.e., it will never send LMP_au_rand, LMP_in_rand or LMP_encryption_mode_req).

5.2.1.2 Security mode 2 (service level enforced security)

When a remote Bluetooth device is in security mode 2 it will not initiate any security procedure before a channel establishment request (L2CAP_ConnectReq) has been received or a channel establishment procedure has been initiated by itself. (The behavior of a device in security mode 2 is further described in [6].) Whether a security procedure is initiated or not depends on the security requirements of the requested channel or service.

A Bluetooth device in security mode 2 should classify the security requirements of its services using at least the following attributes:

- Authorization required
- Authentication required
- Encryption required

Note: Security mode 1 can be considered (at least from a remote device point of view) as a special case of security mode 2 where no service has registered any security requirements.

5.2.1.3 Security mode 3 (*link level enforced security*)

When a remote Bluetooth device is in security mode 3 it will initiate security procedures before it sends LMP_setup_complete.

A Bluetooth device in security mode 3 may reject the host connection request (respond with LMP_not_accepted to the LMP_host_connection_req) based on settings in the host (e.g., only communication with pre-paired devices allowed).

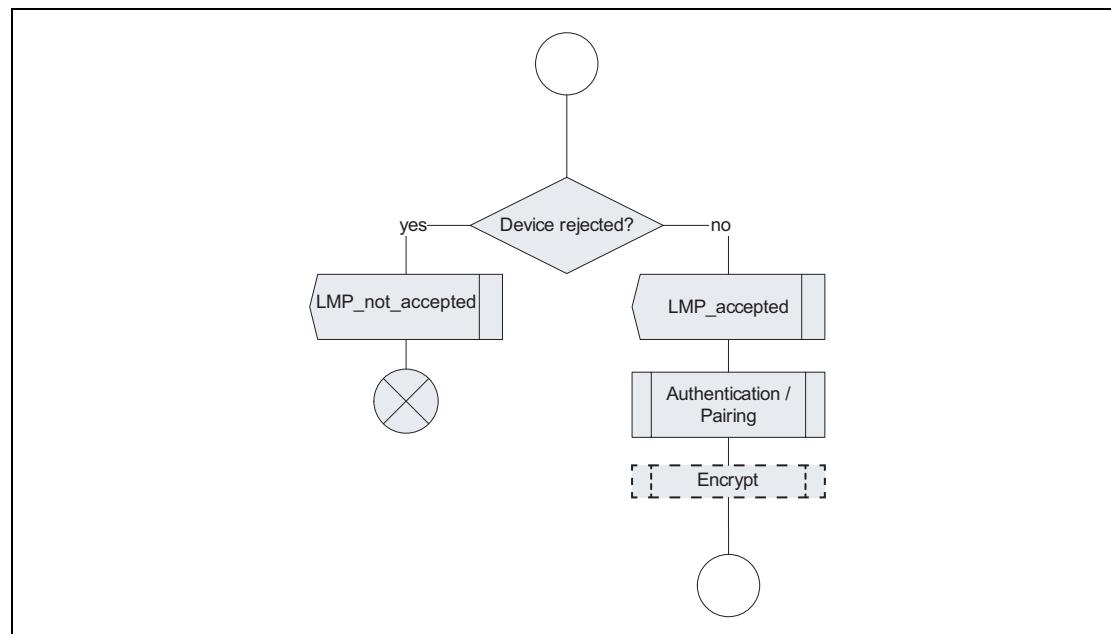


Figure 5.3: Security mode 3 with a legacy remote device

5.2.2 Security Mode 4 (service level enforced security)

A Bluetooth device in security mode 4 shall classify the security requirements of its services using at least the following attributes (in order of decreasing security):

- Authenticated link key required
- Unauthenticated link key required
- Security optional – SDP only. Limited to specific services (see [Section 5.2.2.8](#)).

An authenticated link key is a link key where either the numeric comparison, out-of-band, or passkey entry simple pairing association models were used. An authenticated link key has protection against man-in-the-middle (MITM) attacks. To ensure that an authenticated link key is created during the Simple Pairing procedure, the Authentication_Requirements parameter should be set to one of the MITM Protection Required options. An *unauthenticated link key* is a link key where the just works Secure Simple Pairing association model was used. An unauthenticated link key does not have protection against MITM attacks.

When both devices support Secure Simple Pairing, GAP shall require at least an unauthenticated link key and enabling encryption for all connections except those allowed to have security level 0 (see [Section 5.2.2.8](#)). A profile or protocol may define services that require more security (e.g., an authenticated link key) or no security (although unencrypted connections are only allowed when connecting to a service allowed to have security level 0). To allow an unauthenticated link key to be created during the Simple Pairing procedure, the Authentication_Requirements parameter may be set to one of the MITM Protection Not Required options.

When the device is in Bondable Mode, it shall enable Secure Simple Pairing mode prior to entering Connectable Mode or establishing a link.

A Bluetooth device in security mode 4 shall respond to authentication requests during link establishment when the remote device is in security mode 3 for backwards compatibility reasons.

A Bluetooth device in security mode 4 enforces its security requirements before it attempts to access services offered by a remote device and before it grants access to services it offers to remote devices. Service access may occur via L2CAP channels or via channels established by protocols above L2CAP such as RFCOMM.

For services transmitting unicast data over the connectionless L2CAP channel, the transmitting device shall enforce its security requirements prior to sending data. There is no mechanism for a device receiving data via the L2CAP connectionless channel to prevent unencrypted data from being received. Hence, [Section 5.2.2.1](#) addresses unicast connectionless data transmission together with devices initiating connection-oriented channels while [Section 5.2.2.2](#) covers only devices responding to requests for connection-oriented channel establishment but does not cover unicast connectionless data reception.

A device may be in a Secure Connections Only mode. When in Secure Connections Only mode, all services (except those allowed to have Security Mode 4, Level 0) available on the BR/EDR physical transport require Security Mode 4, Level 4. The device shall reject both new outgoing and incoming service level connections when the physical transport does not support Secure Connections and the service requires Security Mode 4, Level 4.

A device operating with a physical transport operating in Secure Connections Only mode may disconnect the ACL connection using error code 0x05 (Authentication Failure) when the physical transport that does not support Secure Connections, tries to access a service that requires Security Mode 4, Level 4.

Note: a device may operate several physical transports simultaneously - in this case all physical transports are required to enable Security Connections Only Mode simultaneously.

5.2.2.1 Security for Access to Remote Service (Initiating Side)

When the responding device does not support Secure Simple Pairing, it may disconnect the link while the initiating device is requesting the PIN to be entered by the user. This may occur due to the lack of an L2CAP channel being present for longer than an implementation-specific amount of time (e.g., a few seconds). When this occurs, the initiating device shall allow the user to complete entering the PIN and shall then re-page the remote device and restart the pairing procedure (see [\[Vol. 2, Part C\] Section 4.2.2 on page 277](#)) using the PIN entered.

5.2.2.1.1 Pairing Required for Access to Remote Service

When a Bluetooth device in security mode 4 initiates access to a remote service via a connection-oriented L2CAP channel and a sufficient link-key is not available, the local device shall perform pairing procedures and enable encryption before sending a channel establishment request (L2CAP_ConnectReq or a higher-layer channel establishment request such as that of RFCOMM).

When a Bluetooth device in security mode 4 transmits data to a remote service via the unicast connectionless L2CAP channel and a sufficient link-key is not available, the local device shall perform pairing procedures and enable encryption before transmitting unicast data on the connectionless L2CAP channel.

See [Section 5.2.2.8 on page 325](#) for details on determining whether or not a link key is sufficient.

If pairing does not succeed, the local device shall not send a channel establishment request. The local device may re-try pairing up to three (3) times. If pairing fails three consecutive times, the local device shall disconnect the ACL link with error code 0x05 - Authentication Failure.

If the link key generated is not at least as good as the expected or required type, the local device shall fail the establishment and may disconnect the ACL link with error code 0x05 - Authentication Failure.

5.2.2.1.2 Authentication Required for Access to Remote Service

When a Bluetooth device in security mode 4 initiates access to a remote service via a connection-oriented L2CAP channel and a sufficient link key is available for the remote device, it shall authenticate the remote device and enable encryption before sending a channel establishment request (L2CAP_ConnectReq or higher a layer-channel establishment request such as that of RFCOMM).

When a Bluetooth device in security mode 4 transmits unicast data to a remote service via the connectionless L2CAP channel and security is required for the

application and a sufficient link-key is available then the local device shall authenticate the remote device and enable encryption before transmitting unicast data on the L2CAP connectionless channel.

See [Section 5.2.2.8 on page 325](#) for details on determining whether or not a link key is sufficient.

If authentication is required by the service but does not succeed, or if a sufficient link-key is not available, then the local device shall not enable encryption and shall not send a channel establishment request and shall not send any unicast data via the L2CAP connectionless channel for that application. The host may then notify the user and offer to perform pairing.

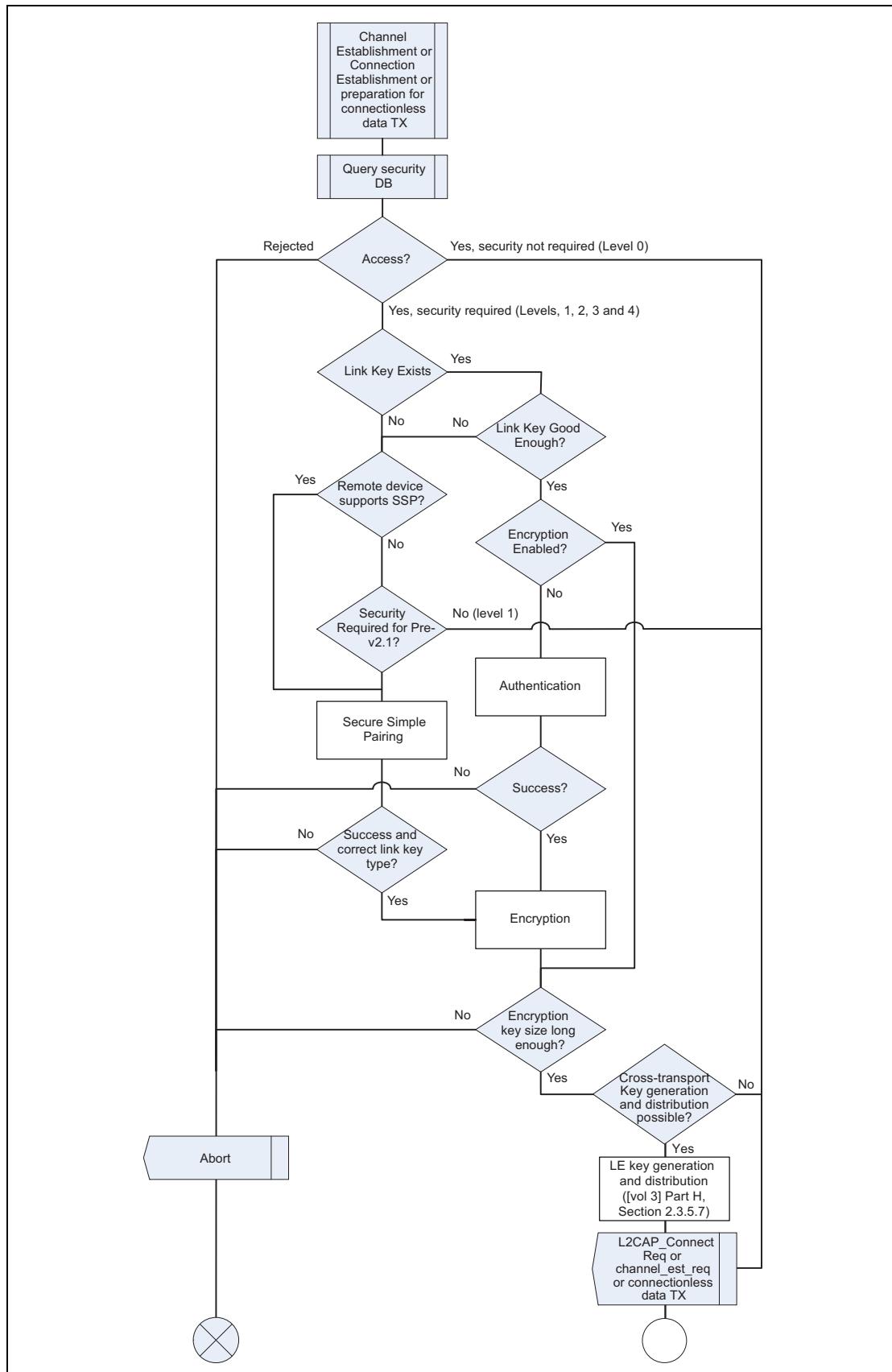


Figure 5.4: Channel establishment using security mode 4 for initiating side

5.2.2.1.3 Cross-transport Key Generation and Distribution

After encryption is enabled and both devices support cross-transport key generation, the master of the BR/EDR transport may perform LE key generation and distribution ([\[Vol 3\] Part H, Section 2.3.5.7](#)).

5.2.2.2 Security for Access to Local Service by Remote Device (Responding Side)

When a remote device attempts to access a service offered by a Bluetooth device that is in security mode 4 and a sufficient link key exists and authentication has not been performed the local device shall authenticate the remote device and enable encryption after the channel establishment request is received but before a channel establishment confirmation (L2CAP_ConnectRsp with result code of 0x0000 or a higher-level channel establishment confirmation such as that of RFCOMM) is sent.

When L2CAP is the channel establishment protocol being used for the requested service, an L2CAP_ConnectRsp signaling packet shall be sent by the responding device containing the result code 0x0001 (connection pending) following receipt of an L2CAP_ConnectReq and prior to initiating security procedures which can result in prompting the local user for input (e.g., pairing using a PIN or Secure Simple Pairing using either the Passkey entry or Numerical Comparison association models). This will stop the L2CAP RTX timer on the remote device (which may be as short as 1 second) and will invoke the ERTX timer on the remote device, which is a minimum duration of 60 seconds.

See [\[Vol. 3, Part A\] Section 6.1.7 on page 122](#), for additional information on L2CAP RTX and ERTX timers. See also [\[Vol. 3, Part A\] Section 4.3 on page 63](#) for additional information on the L2CAP_ConnectRsp signalling packet, and the defined result codes.

Higher layer channel establishment protocols should be designed to restrict timeouts to be 30 seconds or longer to allow for user input, or provide mechanisms to dynamically extend timeouts when user input may be required.

If authentication or pairing fails when a remote device is attempting to access a local service, the local device shall send a negative response to the channel establishment request (L2CAP_ConnectReq or channel_est_req) indicating a security issue if possible. If the channel establishment protocol is L2CAP, then the result code 0x0003 (connection refused - security block) shall be sent in the L2CAP_ConnectRsp signal.

If the remote device has indicated support for Secure Simple Pairing, a channel establishment request is received for a service other than SDP, and encryption has not yet been enabled, then the local device shall disconnect the ACL link with error code 0x05 - Authentication Failure.

5.2.2.2.1 Pairing Required for Access to Local Service by Remote Device

When a remote device attempts to access a service offered by a Bluetooth device that is in security mode 4 and pairing is required due to the link key being absent or insufficient, the local device shall perform pairing procedures and enable encryption after the channel establishment request is received and before a channel establishment confirmation (L2CAP_ConnectRsp with result code of 0x0000 or a higher-level channel establishment response such as that of RFCOMM) is sent.

See [Section 5.2.2.6](#) for details on determining whether or not a link key is sufficient.

If pairing does not succeed, then the local device shall not send a channel establishment confirmation. The local device may retry pairing up to three (3) times. If pairing fails three consecutive times, then the local device shall disconnect the ACL link with error code 0x05 - Authentication Failure.

If the link-key generated is not at least as good as the expected or required type, then the local device shall fail the channel establishment and may disconnect the ACL link with error code 0x05 - Authentication Failure.

5.2.2.2.2 Authentication Required for Access to Local Service by Remote Device

See [Section 5.2.2.6](#) for details on determining whether or not a link key is sufficient.

If authentication does not succeed, then the local device shall not send a channel establishment confirmation. The host may at this point notify the user and offer to perform pairing.

A Bluetooth device in security mode 4 shall respond to authentication and pairing requests during link establishment when the remote device is in security mode 3 for backwards compatibility reasons. However, authentication of the remote device shall be performed after the receipt of the channel establishment request is received, and before the channel establishment response is sent.

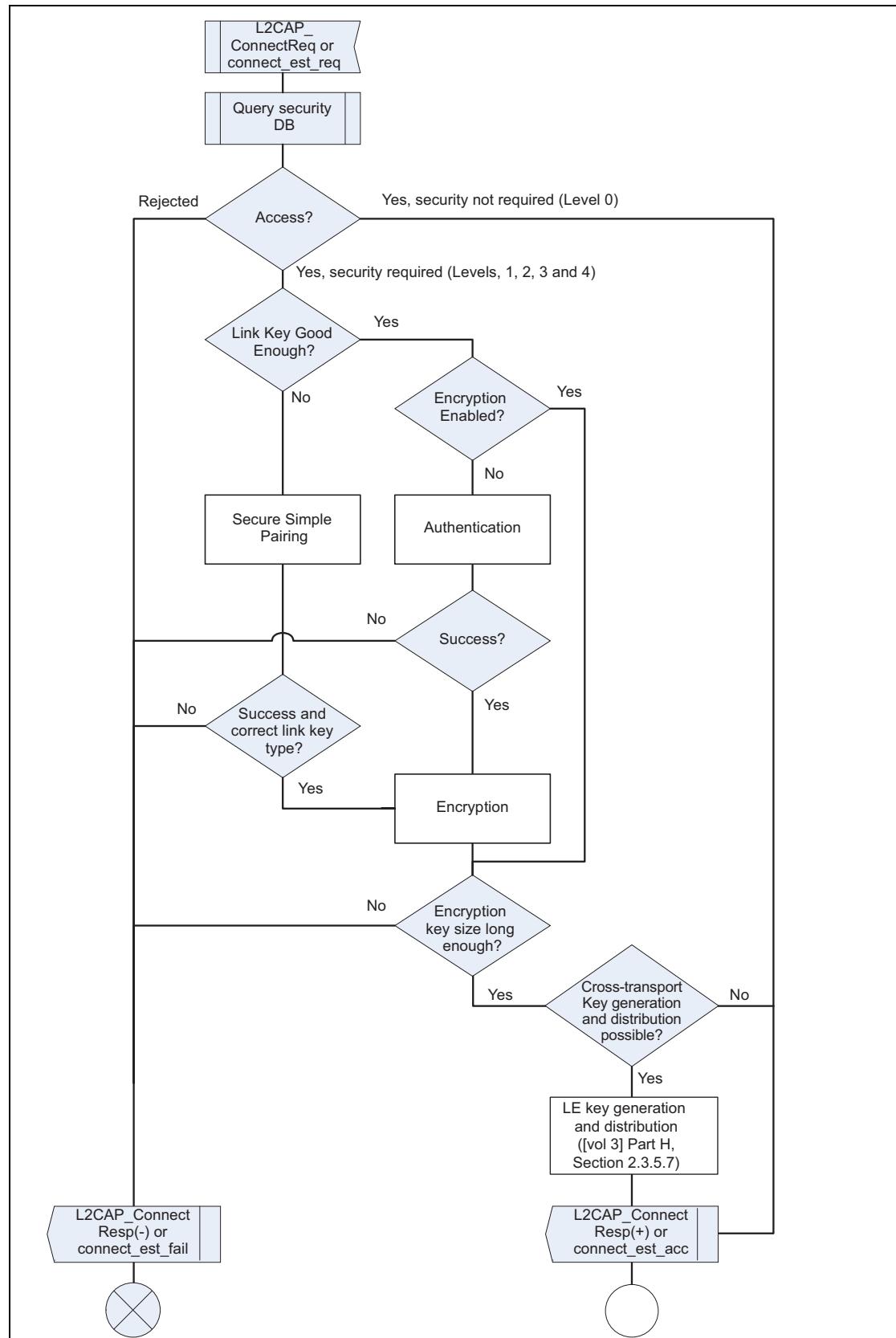


Figure 5.5: Channel establishment using security mode 4 for the responding side

5.2.2.2.3 Cross-transport Key Generation and Distribution

After encryption is enabled and both devices support cross-transport key generation, the master of the BR/EDR transport may perform LE key generation and distribution ([\[Vol 3\] Part H, Section 2.3.5.7](#)).

5.2.2.3 Simple Pairing after Authentication Failure

When both devices support Secure Simple Pairing all non-SDP connections are encrypted regardless of whether security was required or whether the devices are bonded or not. The initial connection between the two devices will result in a link key through Secure Simple Pairing. Depending on whether or not bonding was performed and the security policy of the initiating device, the link key may or may not be stored. When the link key is stored, subsequent connections to the same device will use authentication but this may fail if the remote device has deleted the link key. [Table 5.1](#) defines what shall be done depending on the type of the link key and whether bonding was performed or not.

Link Key Type	Devices Bonded?	Action to take when Authentication Fails
Combination	No	Depends on security policy of the device: <ul style="list-style-type: none"> • Option 1: Automatically initiate pairing • Option 2: Notify user and ask if pairing is ok Option 2 is recommended.
Combination	Yes	Notify user of security failure
Unauthenticated	No	Depends on security policy of the device: <ul style="list-style-type: none"> • Option 1: Automatically initiate secure simple pairing • Option 2: Notify user and ask if secure simple pairing is ok. Option 1 is recommended.
Unauthenticated	Yes	Notify user of security failure
Authenticated	No	Depends on security policy of the device: <ul style="list-style-type: none"> • Option 1: Automatically initiate secure simple pairing • Option 2: Notify user and ask if secure simple pairing is ok Option 2 is recommended.
Authenticated	Yes	Notify user of security failure

Table 5.2: Simple Pairing after Authentication Failure

Note that non-bonded authenticated or unauthenticated link keys may be considered disposable by either device and may be deleted at any time.

5.2.2.4 IO Capabilities

Once a connection is established, if the Host determines that security is necessary and both devices support Secure Simple Pairing, the devices perform an IO capability exchange. The purpose of the IO capability exchange is to determine the authentication algorithm used in the Authentication Stage 1 phase of Simple Pairing.

The input and output capabilities are described in [Table 5.3](#):

Capability	Description
No input	Device does not have the ability to indicate 'yes' or 'no'
Yes / No	Device has at least two buttons that are mapped easily to 'yes' and 'no' or the device has a mechanism whereby the user can indicate either 'yes' or 'no' (see note below).
Keyboard	Device has a numeric keyboard that can input the numbers '0' through '9' and a confirmation. Device also has two buttons that can be easily mapped to 'yes' and 'no' or the device has a mechanism whereby the user can indicate either 'yes' or 'no' (see Note below).

Table 5.3: User Input Capabilities

Note: 'yes' could be indicated by pressing a button within a certain time limit otherwise 'no' would be assumed.

Capability	Description
No output	Device does not have the ability to display or communicate a 6 digit decimal number
Numeric output	Device has the ability to display or communicate a 6 digit decimal number

Table 5.4: User Output Capabilities

5.2.2.5 Mapping of Input / Output Capabilities to IO Capability

The individual input and output capabilities are mapped to a single IO capability which is later used to determine which authentication algorithm will be used.

		Local Output Capability	
		No Output	Numeric Output
Local Input Capability	No input	NoInputNoOutput	DisplayOnly
	Yes / No	NoInputNoOutput	DisplayYesNo
	Keyboard	KeyboardOnly	DisplayYesNo

Table 5.5: IO Capability mapping

When a device has OOB authentication information from the remote device, it will indicate it in the LMP_IO_Capability_Res PDU. When either device has OOB information, the OOB association model will be used.

The Host may allow the Link Manager to ignore the IO capabilities and use the Numeric Comparison protocol with automatic accept by setting the Authentication_Requirements parameter to one of the MITM Protection *Not Required* options.

5.2.2.6 IO and OOB Capability Mapping to Authentication Stage 1 Method

Determining which association model to use in Authentication Stage 1 is performed in three steps. First, the devices look at the OOB Authentication Data Present parameter received in the remote IO capabilities. If either device has received OOB authentication data then the OOB association model is used. The event of receiving the OOB information is indicated by a device to its peer in the IO Capability Exchange step of simple pairing.

		Device A	
		Has not received remote OOB authentication data	Has received remote OOB authentication table
Device B	Has not received remote OOB authentication data	Use the IO capability mapping table	Use OOB association with ra = 0 rb from OOB
	Has received remote OOB authentication data	Use OOB association with ra from OOB rb = 0	Use OOB association with ra from OOB rb from OOB

Table 5.6: IO and OOB capability mapping

Second, if neither device has received OOB authentication data and if both devices have set the Authentication_Requirements parameter to one of the MITM Protection Not Required options, authentication stage 1 shall function as if both devices set their IO capabilities to DisplayOnly (e.g., Numeric comparison with automatic confirmation on both devices).

Finally, if neither device has received OOB authentication data and if one or both devices have set the Authentication_Requirements parameter to one of the *MITM Protection Required* options, the IO and OOB capabilities are mapped to the authentication stage 1 method as defined in the following table. A Host that has set the Authentication_Requirements parameter to one of the *MITM Protection Required* options shall verify that the resulting Link Key is an Authenticated Combination Key (see [\[Vol. 2\], Part E Section 7.7.24](#)). The table also lists whether the combination key results in an authenticated or unauthenticated link key.

		Device A (Initiator)			
		Display Only	DisplayYesNo	KeyboardOnly	NoInputNoOutput
Device B (Responder)	DisplayOnly	Numeric Comparison with automatic confirmation on both devices. Unauthenticated	Numeric Comparison with automatic confirmation on device B only. Unauthenticated	Passkey Entry: Responder Display, Initiator Input. Authenticated	Numeric Comparison with automatic confirmation on both devices. Unauthenticated
	DisplayYesNo	Numeric Comparison with automatic confirmation on device A only. Unauthenticated	Numeric Comparison: Both Display, Both Confirm. Authenticated	Passkey Entry: Responder Display, Initiator Input. Authenticated	Numeric Comparison with automatic confirmation on device A only. Unauthenticated
	Keyboard Only	Passkey Entry: Initiator Display, Responder Input. Authenticated	Passkey Entry: Initiator Display, Responder Input. Authenticated	Passkey Entry: Initiator and Responder Input. Authenticated	Numeric Comparison with automatic confirmation on both devices. Unauthenticated
	NoInputNoOutput	Numeric Comparison with automatic confirmation on both devices. Unauthenticated	Numeric Comparison with automatic confirmation on device B only. Unauthenticated	Numeric Comparison with automatic confirmation on both devices. Unauthenticated	Numeric Comparison with automatic confirmation on both devices. Unauthenticated

Table 5.7: IO Capability Mapping to Authentication Stage 1

Note: The "DisplayOnly" IO capability only provides unidirectional authentication.

5.2.2.7 Out of Band (OOB)

An out of band mechanism may also be used to communicate discovery information as well as other information related to the pairing process.

The contents of the OOB data block are:

Generic Access Profile



Mandatory contents:

- Length (2 bytes)
- BD_ADDR (6 bytes)

Optional contents:

- Class of Device (3 bytes)
- Simple Pairing Hash C (16 bytes)
- Simple Pairing Randomizer R (16 bytes)
- Local name (variable length)
- Other information

The length field includes all bytes in the OOB data block including the length field itself. The BD_ADDR will be a fixed field in the beginning of the OOB data block. Following the BD_ADDR will be zero or more EIR tag fields containing optional contents. The EIR tag format will be used for the optional contents.

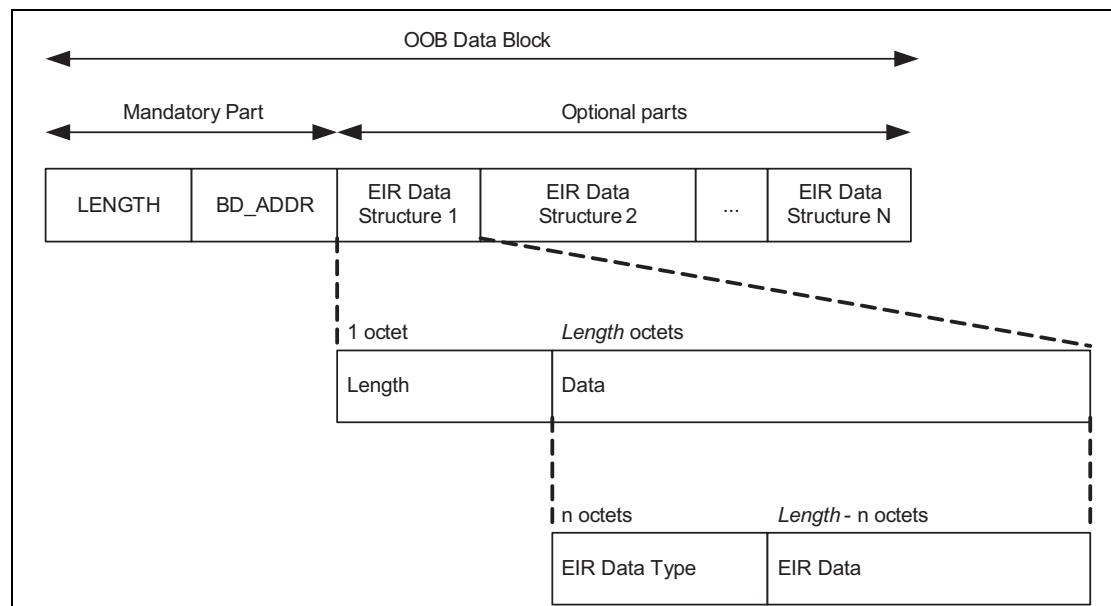


Figure 5.6: OOB Data Block Format

If Simple Pairing fails when one or both devices have OOB Authentication Data present, both devices shall discard the OOB Authentication Data and the device that originally initiated authentication shall re-initiate authentication. Note that although the user may get involved in authentication as defined by the IO capabilities of the two devices, falling back to the in-band association model will prevent deadlock conditions when one or both devices has stale OOB Authentication Data.

There is a MIME type defined for use with the OOB data format. The MIME type can be found from the following link: <http://www.iana.org/assignments/media-types/application/vnd.bluetooth.ep.oob>

5.2.2.8 Security Database

A Bluetooth device in security mode 4 shall classify the security requirements of its services using at least the following levels attributes (in order of decreasing security) for use when pairing with remote devices supporting Secure Simple Pairing:

- Level 4, for services with the following attributes or devices in Secure Connections Only Mode:
 - MITM protection required
 - 128-bit equivalent strength for link and encryption keys required using FIPS approved algorithms (E0 not allowed, SAFER+ not allowed, and P-192 not allowed)
 - User interaction acceptable
 - Level 3, for services with the following attributes:
 - MITM protection required
 - Encryption required
 - User interaction acceptable
 - Level 2, for services with the following attributes:
 - MITM protection not required
 - Encryption required
 - Level 1, for services with the following attributes:
 - MITM protection not required
 - Minimal user interaction desired
 - Level 0: Service requires the following:
 - MITM protection not required
 - No encryption required
 - No user interaction required
- Security Mode 4 Level 0 shall only be used for:
- a) L2CAP fixed signaling channels with CIDs 0x0001, 0x0003, and 0x003F
 - b) SDP
 - c) broadcast data sent on the connectionless L2CAP channel (CID 0x0002)
 - d) services with the combinations of Service Class UUIDs and L2CAP traffic types listed in [\[Core Specification Supplement\]](#), Part C.

The security level required for each service offered should be stored in a security database that is accessed to determine the type of link key that is required for access to the respective service. The security level required for service data transmitted on an L2CAP connection-oriented channel may differ from the security level required for service data transmitted on another L2CAP connec-

Generic Access Profile



tion-oriented channel or on the connectionless L2CAP channel. [Table 5.8 on page 326](#) shows the type of link key required for each security level for both remote devices that support Secure Simple Pairing (v2.1 + EDR remote devices) and for those that do not (pre-v2.1 + EDR remote devices).

Security Level Required for Service	Link Key type required for remote devices	Link Key type required for pre-v2.1 remote devices	Comments
Level 4 • MITM protection required • Encryption required • User interaction acceptable	Authenticated (P-256 based Secure Simple Pairing and Secure Authentication)	NA	Highest Security Only possible when both devices support Secure Connections
Level 3 • MITM protection required • Encryption required • User interaction acceptable	Authenticated	Combination (16 digit PIN recommended)	High Security
Level 2 • MITM protection not necessary • Encryption desired	Unauthenticated	Combination	Medium Security
Level 1 • MITM protection not necessary • Encryption not necessary ¹ • Minimal user interaction desired	Unauthenticated	None	Low Security
Level 0 • MITM protection not necessary • Encryption not necessary • No user interaction desired	None	None	Permitted only for SDP and service data sent via either L2CAP fixed signaling channels or the L2CAP connectionless channel to PSMs that correspond to service class UUIDs which are allowed to utilize Level 0.

Table 5.8: Security Level mapping to link key requirements

1. Though encryption is not necessary for the service for Level 1, this specification mandates the use of encryption when the remote device is v2.1+EDR for all services other than SDP.

An *authenticated* link key is a link key where either the numeric comparison, out-of-band, or passkey entry simple pairing association models were used. An authenticated link key has protection against MITM attacks. To ensure that an

authenticated link key is created during the Simple Pairing procedure, the Authentication_Requirements parameter should be set to one of the *MITM Protection Required* options.

An *unauthenticated* link key is a link key where the “Just Works” simple pairing association model was used (see [\[Vol. 1, Part A\] Section 5.2.4 on page 89](#)). An unauthenticated link key does not have protection against MITM attacks. To allow an unauthenticated link key to be created during the Simple Pairing procedure, the Authentication_Requirements parameter may be set to one of the *MITM Protection Not Required* options.

When both devices support Secure Simple Pairing and at least one device does not support Secure Connections, the strength of the link key is 96 effective bits. When both devices support Secure Connections, the strength of the link key is 128 effective bits. Secure Connections does not change the protection against MITM attacks.

A combination link key is a link key where the v2.0 pairing mechanism was used to generate the link-key (see [\[Vol. 2, Part C\] Section 4.2.2.4 on page 279](#)).

When both devices support Secure Simple Pairing, GAP shall require at least an unauthenticated link key and enable encryption for service traffic sent or received via connection-oriented L2CAP channels. A profile or protocol may define services that require more security (for example, an authenticated link key) or no security in the case of SDP or service traffic sent via the L2CAP connectionless channel for services that do not require security.

When the device is in Bondable Mode, it shall enable Secure Simple Pairing mode prior to entering Connectable Mode or establishing a link.

A Bluetooth device in security mode 4 shall respond to authentication and pairing requests during link establishment when the remote device is in security mode 3 for backwards compatibility reasons. See [Section 5.2.1.3](#) for more information.

The remote Controller's and remote Host's support for Secure Simple Pairing shall be determined by the Link Manager Secure Simple Pairing (Host Support) feature bit.

A previously generated link key is considered “sufficient” if the link key type is of the type required for the service, or of a higher strength. Authenticated link keys are considered higher strength than Unauthenticated or Combination keys. Unauthenticated link keys are considered higher strength than Combination keys.

6 IDLE MODE PROCEDURES – BR/EDR PHYSICAL TRANSPORT

The inquiry and discovery procedures described here are applicable only to the device that initiates them (A). The requirements on the behavior of B is according to the modes specified in [4](#) and to [\[2\]](#).

	Procedure	Ref.	Support
1	General inquiry	6.1	C1
2	Limited inquiry	6.2	C1
3	Name discovery	6.3	O
4	Device discovery	6.4	O
5	Bonding	6.5	O

C1: If initiation of bonding is supported, support for at least one inquiry procedure is mandatory, otherwise optional.
 (Note: support for LMP-pairing is mandatory [\[2\]](#).)

6.1 GENERAL INQUIRY

6.1.1 Purpose

The purpose of the general inquiry procedure is to provide the initiator with the Bluetooth device address, clock, Class of Device, page scan mode, and extended inquiry response information of general discoverable devices (i.e., devices that are in range with regard to the initiator and are set to scan for inquiry messages with the General Inquiry Access Code). Also devices in limited discoverable mode will be discovered using general inquiry.

The general inquiry should be used by devices that need to discover devices that are made discoverable continuously or for no specific condition.

6.1.2 Term on UI level

‘Bluetooth Device Inquiry’.

6.1.3 Description

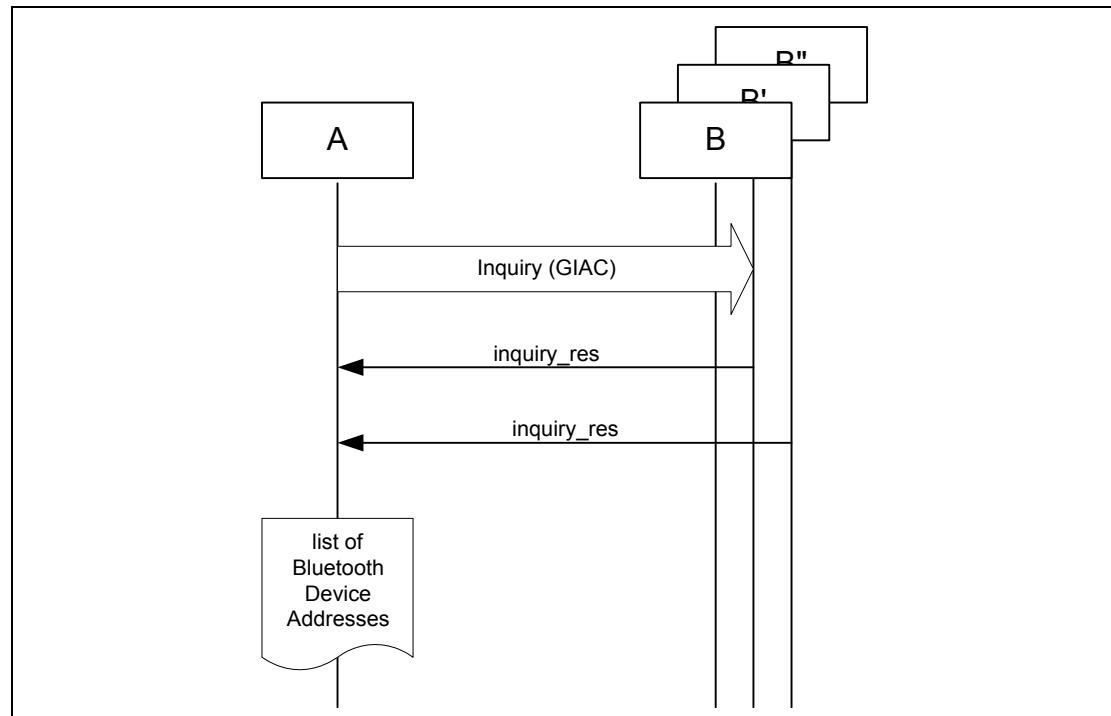


Figure 6.1: General inquiry, where B is a device in non-discoverable mode, B' is a device in limited discoverable mode and B'' is a device in general discoverable mode. (Note that all discoverable devices are discovered using general inquiry, independent of which discoverable mode they are in.)

6.1.4 Conditions

When general inquiry is initiated by a Bluetooth device, the INQUIRY state shall last $T_{GAP}(100)$ or longer, unless the inquirer collects enough responses and determines to abort the INQUIRY state earlier. The Bluetooth device shall perform inquiry using the GIAC.

In order for Device A to receive inquiry responses, the remote devices in range have to be made discoverable (limited or general).

6.2 LIMITED INQUIRY

6.2.1 Purpose

The purpose of the limited inquiry procedure is to provide the initiator with the Bluetooth device address, clock, Class of Device, page scan mode, and extended inquiry response information of limited discoverable devices. The latter devices are devices that are in range with regard to the initiator, and may be set to scan for inquiry messages with the Limited Inquiry Access Code, in addition to scanning for inquiry messages with the General Inquiry Access Code.

The limited inquiry should be used by devices that need to discover devices that are made discoverable only for a limited period of time, during temporary conditions or for a specific event. Since it is not guaranteed that the discoverable device scans for the LIAC, the initiating device may choose any inquiry procedure (general or limited). Even if the remote device that is to be discovered is expected to be made limited discoverable (e.g., when a dedicated bonding is to be performed), the limited inquiry should be done in sequence with a general inquiry in such a way that both inquiries are completed within the time the remote device is limited discoverable; i.e., at least $T_{GAP}(103)$.

6.2.2 Term on UI level

'Bluetooth Device Inquiry'.

6.2.3 Description

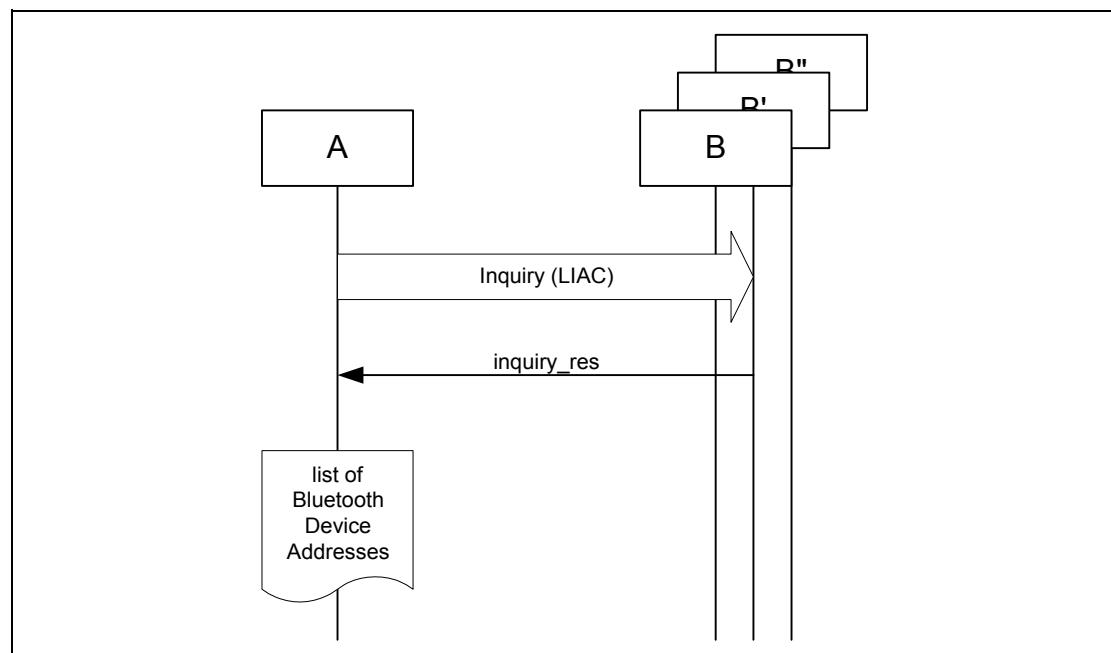


Figure 6.2: Limited inquiry where B is a device in non-discoverable mode, B' is a device in limited discoverable mode and B'' is a device in general discoverable mode. (Note that only limited discoverable devices can be discovered using limited inquiry.)

6.2.4 Conditions

When limited inquiry is initiated by a Bluetooth device, the INQUIRY state shall last $T_{GAP}(100)$ or longer, unless the inquirer collects enough responses and determines to abort the INQUIRY state earlier. The Bluetooth device shall perform inquiry using the LIAC.

In order for Device A to receive inquiry responses, the remote devices in range has to be made limited discoverable.

6.3 NAME DISCOVERY

6.3.1 Purpose

The purpose of name discovery is to provide the initiator with the Bluetooth Device Name of connectable devices (i.e., devices in range that will respond to paging).

6.3.2 Term on UI level

'Bluetooth Device Name Discovery'.

6.3.3 Description

6.3.3.1 Name request

Name request is the procedure for retrieving the Bluetooth Device Name from a connectable Bluetooth device. It is not necessary to perform the full link establishment procedure (see 7.1) in order to just to get the name of another device.

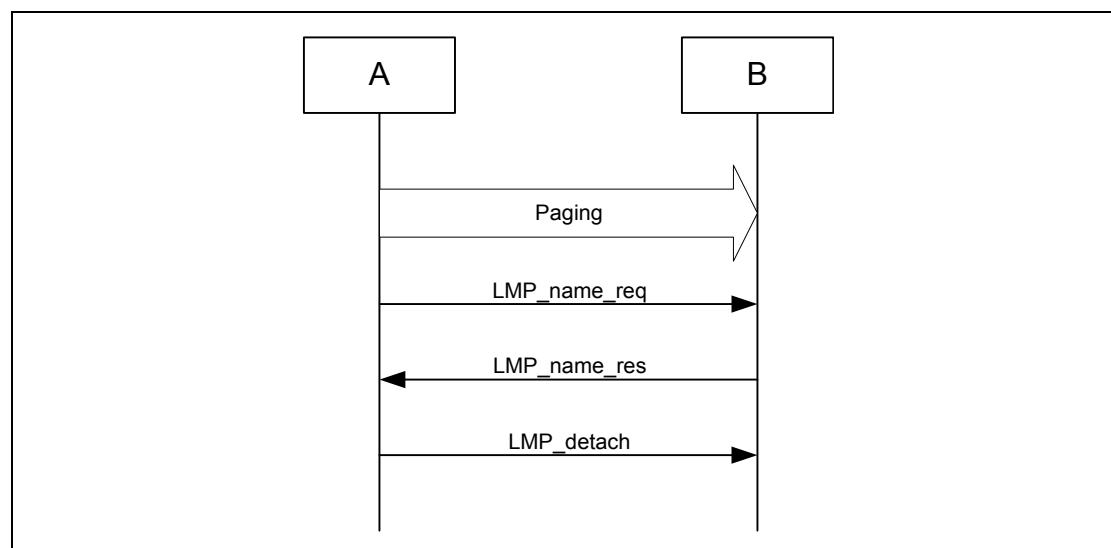


Figure 6.3: Name request procedure

6.3.3.2 Name discovery

Name discovery is the procedure for retrieving the Bluetooth Device Name from connectable Bluetooth devices by performing name request towards known devices (i.e., Bluetooth devices for which the Bluetooth Device Addresses are available).

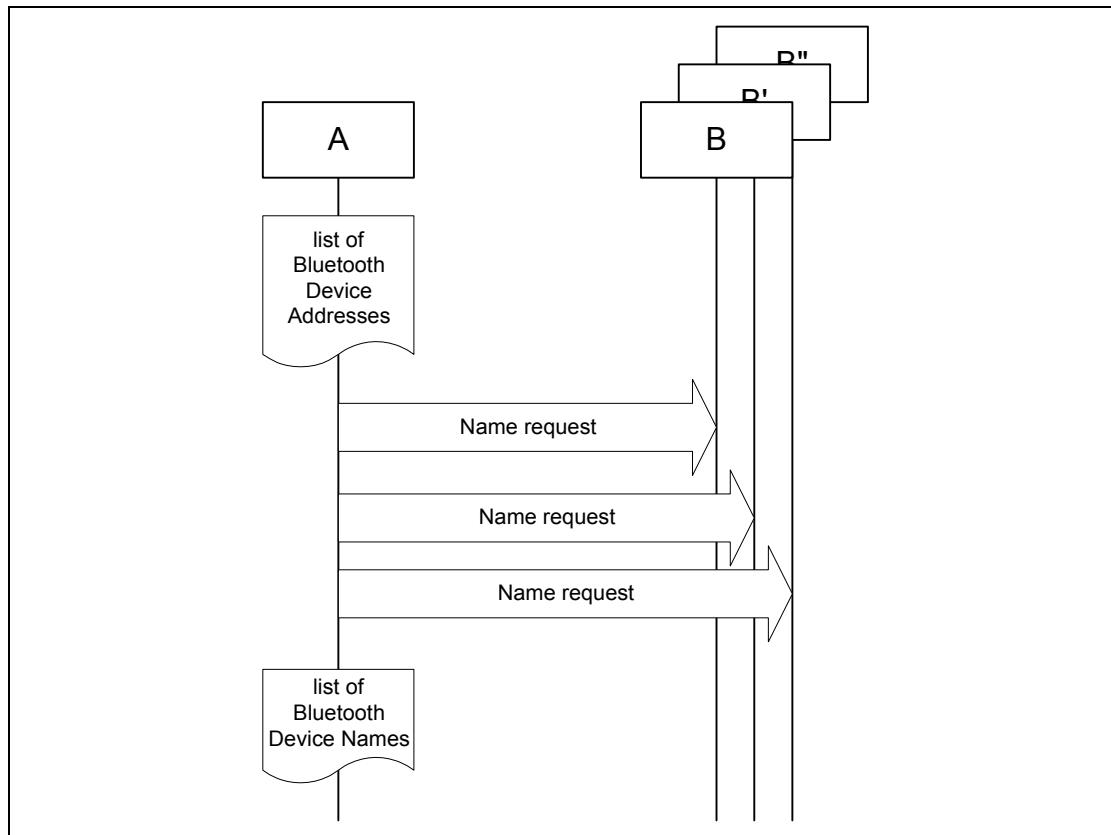


Figure 6.4: Name discovery procedure

6.3.4 Conditions

In the name request procedure, the initiator will use the Device Access Code of the remote device as retrieved immediately beforehand – normally through an inquiry procedure.

6.4 DEVICE DISCOVERY

This section only applies to a device of the BR/EDR and BR/EDR/LE device type.

6.4.1 Purpose

The purpose of device discovery is to provide the initiator with the Bluetooth Device Address, clock, Class of Device, used page scan mode, Bluetooth Device Name, and extended inquiry response information of discoverable devices.

6.4.2 Term on UI Level

'Bluetooth Device Discovery'.

6.4.3 Description

During the device discovery procedure, first an inquiry (either general or limited) is performed, and then name discovery is done towards some or all of the devices that responded to the inquiry. If the initiator of the device discovery receives a complete local name or a shortened local name that is considered long enough, via an extended inquiry response from a remote device, the initiator should not do a separate name discovery for that device.

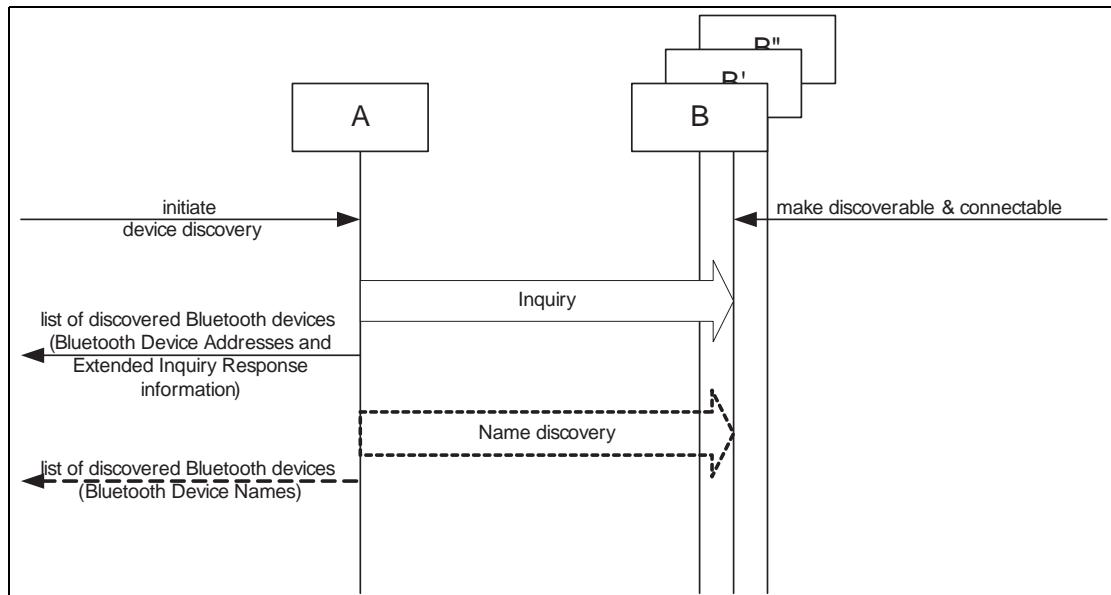


Figure 6.5: Device discovery procedure

6.4.4 Conditions

Conditions for both inquiry (general or limited) and name discovery must be fulfilled (i.e., devices discovered during device discovery must be both discoverable and connectable).

6.5 BONDING

6.5.1 Purpose

The purpose of bonding is to create a relation between two Bluetooth devices based on a common link key (a bond). The link key is created and exchanged (pairing) during the bonding procedure and is expected to be stored by both Bluetooth devices, to be used for future authentication. In addition to pairing, the bonding procedure can involve higher-layer initialization procedures.

6.5.2 Term on UI level

'Bluetooth Bonding'.

6.5.3 Description

Two forms of bonding are described in the following sections: General Bonding and Dedicated Bonding.

6.5.3.1 *General Bonding*

General Bonding refers to the process of performing bonding during connection setup or channel establishment procedures as a precursor to accessing a service.

When the devices that are performing General Bonding both support Secure Simple Pairing, the Authentication_Requirements parameter should be set to MITM Protection Not Required – General Bonding unless the security policy of an available local service requires MITM Protection in which case the Authentication_Requirements parameter shall be set to MITM Protection Required – General Bonding. 'No bonding' is used when the device is performing a Secure Simple Pairing procedure, but does not intend to retain the link key after the physical link is disconnected.

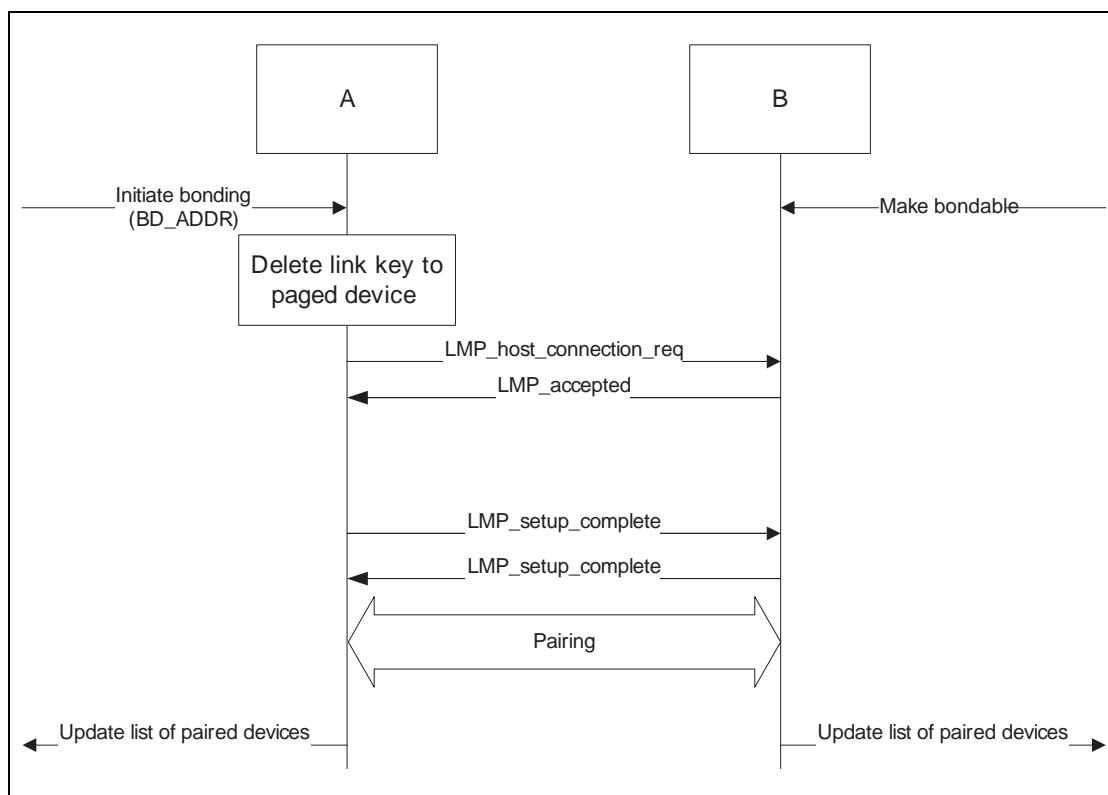


Figure 6.6: General description of general bonding as being the link establishment procedure executed under specific conditions on both devices, followed by authentication and an optional higher layer initialization process.

6.5.3.2 Dedicated Bonding

Dedicated Bonding refers to a procedure wherein one device connects to another only for the purpose of pairing without accessing a particular service. The main difference with dedicated bonding, as compared to a pairing done during link or channel establishment, is that for bonding it is the paging device (A) that must initiate the authentication.

When the devices that are performing Dedicated Bonding both support Secure Simple Pairing, the Authentication_Requirements parameter should be set to *MITM Protection Not Required – Dedicated Bonding* unless the security policy of an available local service requires MITM Protection in which case the Authentication Required parameter shall be set to *MITM Protection Required – Dedicated Bonding*. 'No bonding' is used when the device is performing a Secure Simple Pairing procedure, but does not intend to retain the link key after the physical link is disconnected.

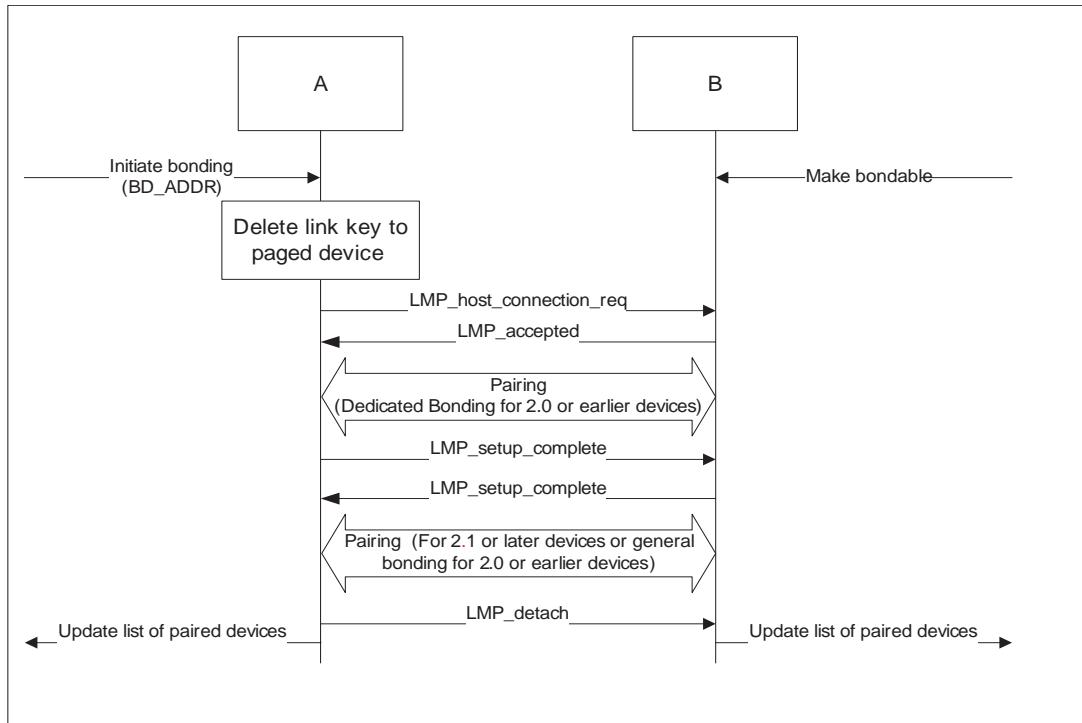


Figure 6.7: Dedicated Bonding as performed when the purpose of the procedure is only to create and exchange a link key between two Bluetooth devices

6.5.4 Conditions

Before bonding can be initiated, the initiating device (A) must know the Device Access Code of the device to pair with. This is normally done by first performing device discovery. A Bluetooth Device that can initiate bonding (A) should use limited inquiry, and a Bluetooth Device that accepts bonding (B) should support the limited discoverable mode.

Bonding is in principle the same as link establishment with the conditions:

- The paged device (B) shall be set into bondable mode. The paging device (A) is assumed to allow pairing since it has initiated the bonding procedure.
- The paging device (the initiator of the bonding procedure, A) shall initiate authentication.
- Before initiating the authentication part of the bonding procedure, the paging device should delete any link key corresponding to a previous bonding with the paged device.

7 ESTABLISHMENT PROCEDURES – BR/EDR PHYSICAL TRANSPORT

	Procedure	Ref.	Support in A	Support in B
1	Link establishment	7.1	M	M
2	Channel establishment	7.2	O	M
3	Connection establishment	7.3	O	O
4	Synchronization establishment	7.5	O	O

Table 7.1: Establishment procedures

The establishment procedures defined here do not include any discovery part. Before establishment procedures are initiated, the information provided during device discovery (in the FHS packet or the extended inquiry response packet of the inquiry response or in the response to a name request or in the synchronization train packet) must be available in the initiating device. This information is:

- The Bluetooth Device Address (BD_ADDR) from which the Device Access Code is generated
- The system clock of the remote device
- The page scan mode used by the remote device for Link establishment.

Additional information provided during device discovery that may be useful for making the decision to initiate an establishment procedure is:

- The Class of device
- The Device name
- The supported Service Classes.

7.1 LINK ESTABLISHMENT

7.1.1 Purpose

The purpose of the link establishment procedure is to establish a logical transport (of ACL type) between two Bluetooth devices using procedures from [\[1\]](#) and [\[2\]](#).

7.1.2 Term on UI Level

'Bluetooth link establishment'

7.1.3 Description

In this sub-section, the paging device (A) is in security mode 3. During link establishment, the paging device cannot distinguish if the paged device (B) is in security mode 1, 2 or 4.

7.1.3.1 *B in security mode 1, 2, or 4*

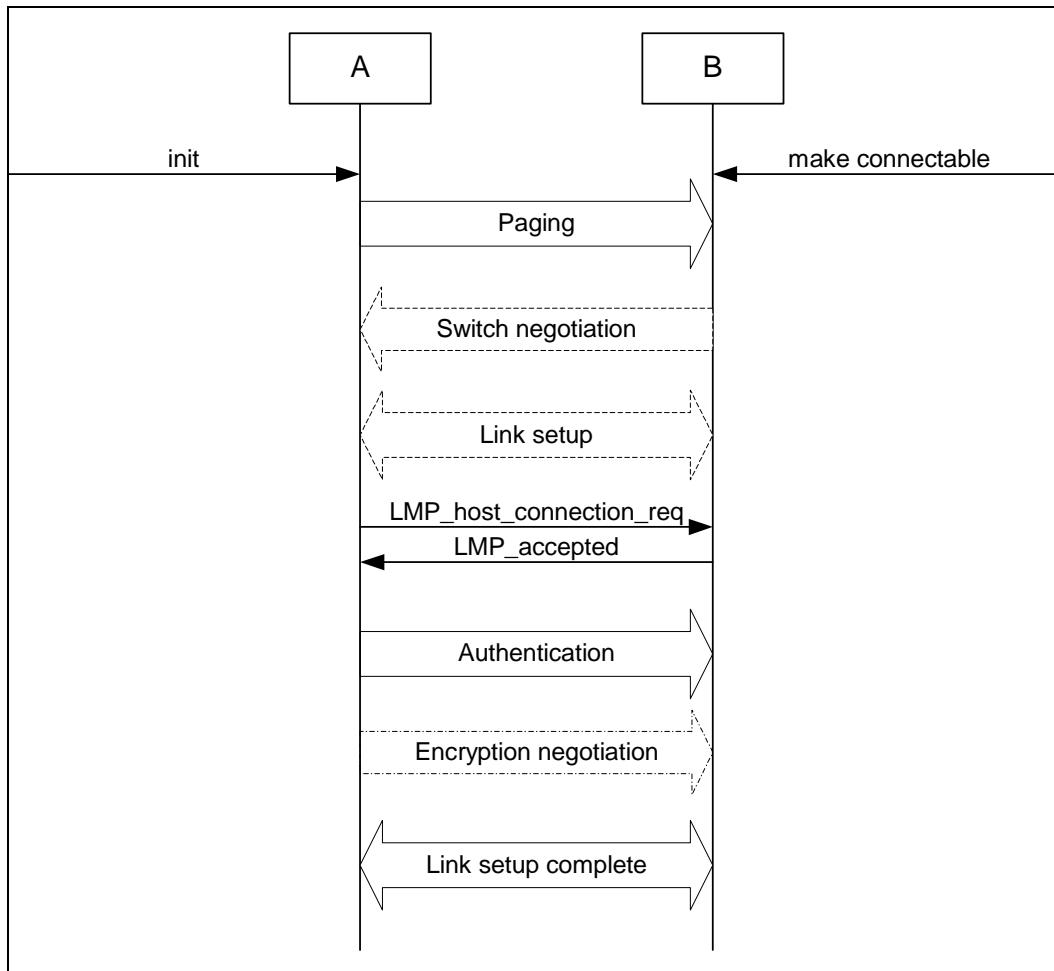


Figure 7.1: Link establishment procedure when the paging device (A) is in security mode 3 and the paged device (B) is in security mode 1, 2, or 4

7.1.3.2 B in security mode 3

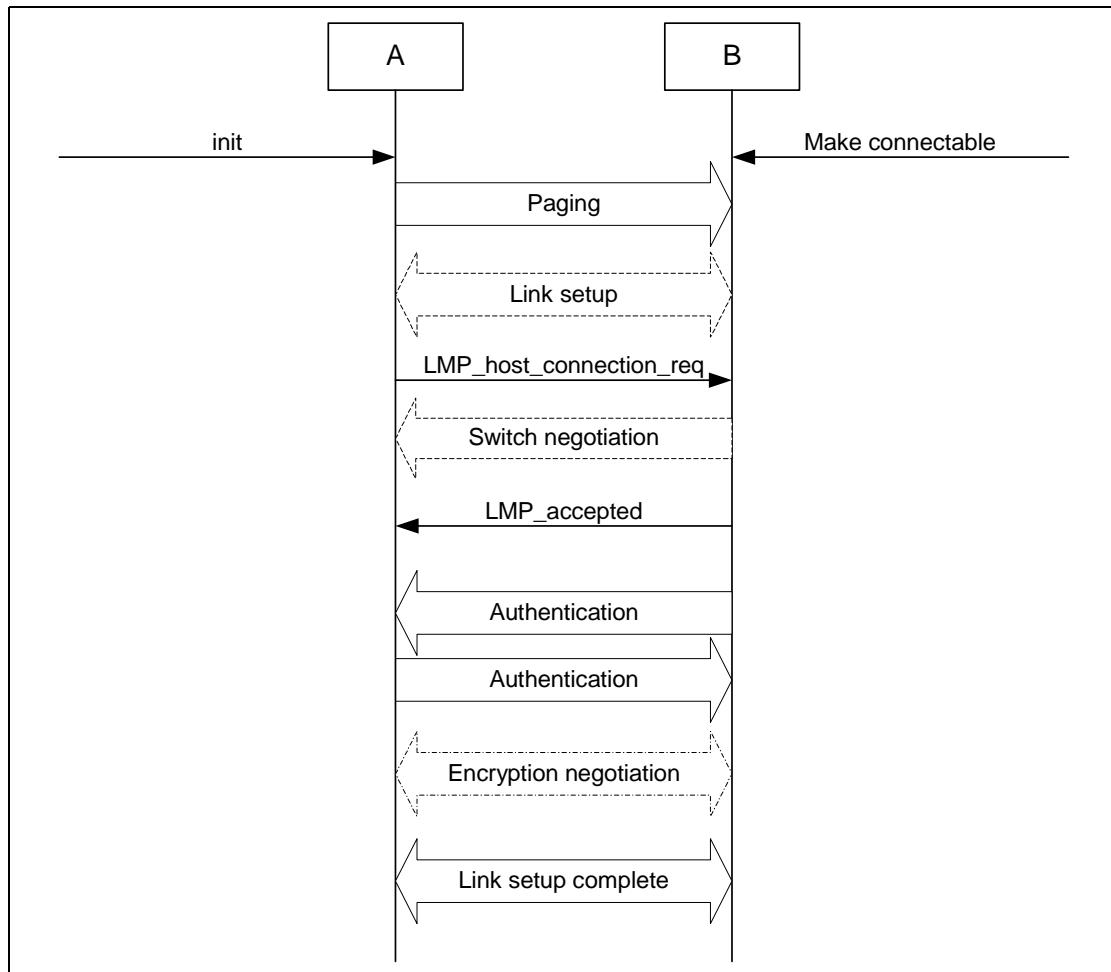


Figure 7.2: Link establishment procedure when both the paging device (A) and the paged device (B) are in security mode 3

7.1.4 Conditions

The paging procedure shall be according to [Vol. 2], Part B Section 8.3 and the paging device should use the Device Access Code and page mode received through a previous inquiry. When paging is completed, a physical link between the two Bluetooth devices is established.

If role switching is needed (normally it is the paged device that has an interest in changing the master/slave roles) it should be done as early as possible after the physical link is established. If the paging device does not accept the switch, the paged device has to consider whether to keep the physical link or not.

Both devices may perform link setup (using LMP procedures that require no interaction with the host on the remote side). Optional LMP features can be used after having confirmed (using LMP_features_req) that the other device supports the feature.

When the paging device needs to go beyond the link setup phase, it issues a request to be connected to the host of the remote device. If the paged device is in security mode 3, this is the trigger for initiating authentication.

The paging device shall send LMP_host_connection_req during link establishment (i.e., before channel establishment) and may initiate authentication only after having sent LMP_host_connection_req.

After an authentication has been performed, any of the devices can initiate encryption.

Further link configuration may take place after the LMP_host_connection_req. When both devices are satisfied, they send LMP_setup_complete.

Link establishment is completed when both devices have sent LMP_setup_complete.

7.2 CHANNEL ESTABLISHMENT

7.2.1 Purpose

The purpose of the channel establishment procedure is to establish a Bluetooth channel (L2CAP channel) between two Bluetooth devices using [1].

7.2.2 Term on UI level

'Bluetooth channel establishment'.

7.2.3 Description

In this sub-section, the initiator (A) is in security mode 3. During channel establishment, the initiator cannot distinguish if the acceptor (B) is in security mode 1 or 3.

7.2.3.1 B in security mode 2 or 4

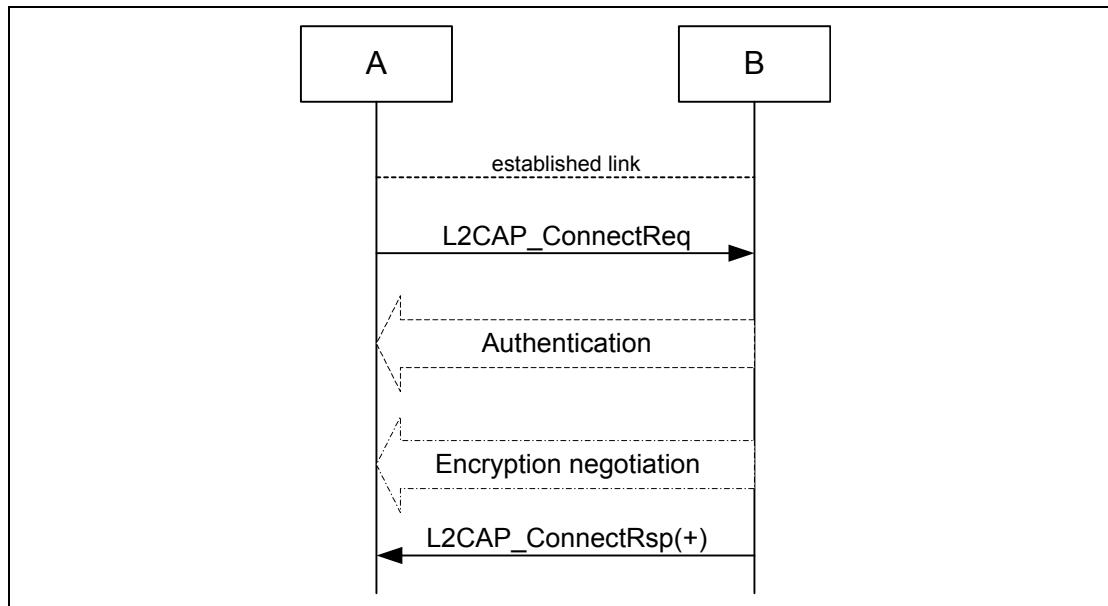


Figure 7.3: Channel establishment procedure when the initiator (A) is in security mode 3 and the acceptor (B) is in security mode 2 or 4

7.2.3.2 B in security mode 1 or 3

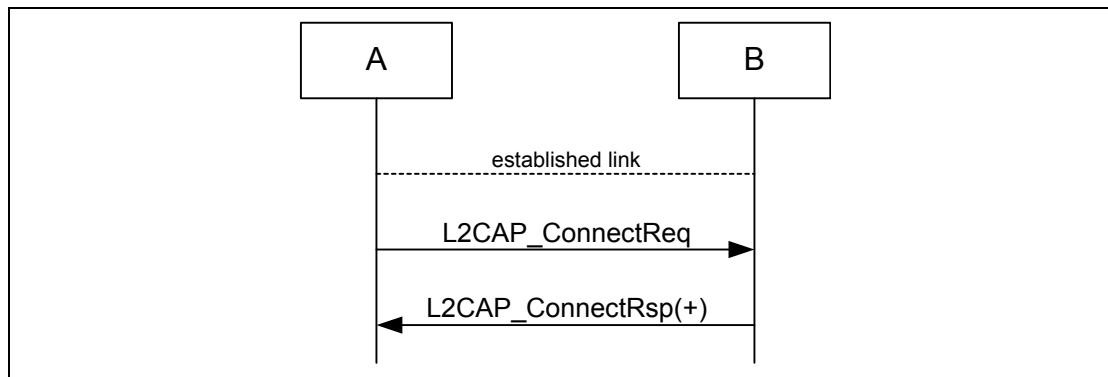


Figure 7.4: Channel establishment procedure when the initiator (A) is in security mode 3 and the acceptor (B) is in security mode 1 or 3

7.2.4 Conditions

Channel establishment starts after link establishment is completed when the initiator sends a channel establishment request (L2CAP_ConnectReq).

Depending on security mode, security procedures may take place after the channel establishment has been initiated.

Channel establishment is completed when the acceptor responds to the channel establishment request (with a positive L2CAP_ConnectRsp).

7.3 CONNECTION ESTABLISHMENT

7.3.1 Purpose

The purpose of the connection establishment procedure is to establish a connection between applications on two Bluetooth devices.

7.3.2 Term on UI level

'Bluetooth connection establishment'

7.3.3 Description

In this sub-section, the initiator (A) is in security mode 3. During connection establishment, the initiator cannot distinguish if the acceptor (B) is in security mode 1 or 3.

7.3.3.1 *B* in security mode 2 or 4

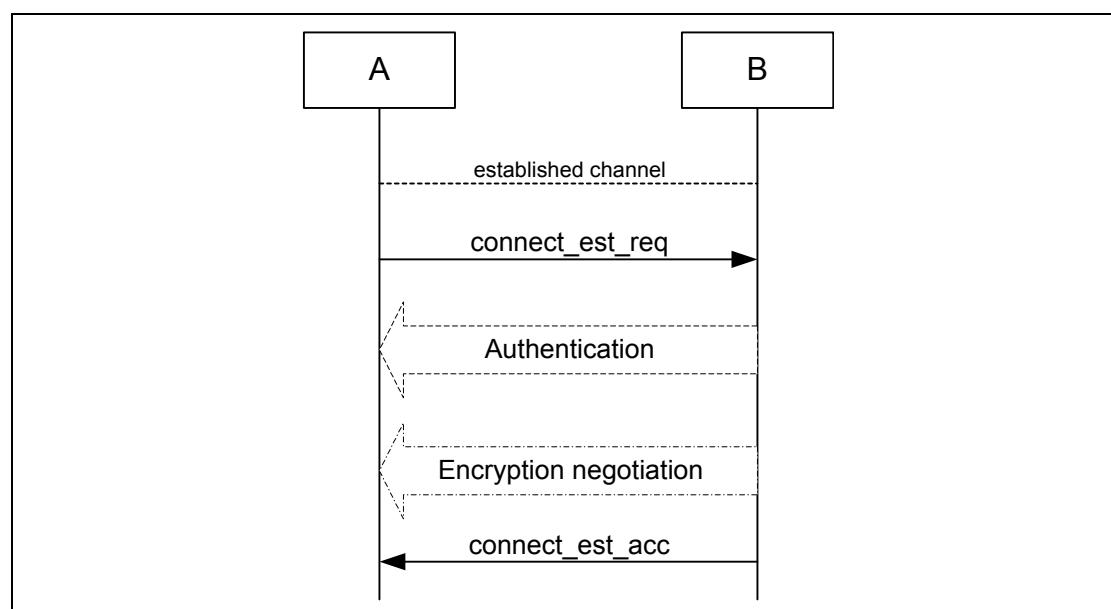


Figure 7.5: Connection establishment procedure when the initiator (A) is in security mode 3 and the acceptor (B) is in security mode 2 or 4

7.3.3.2 B in security mode 1 or 3

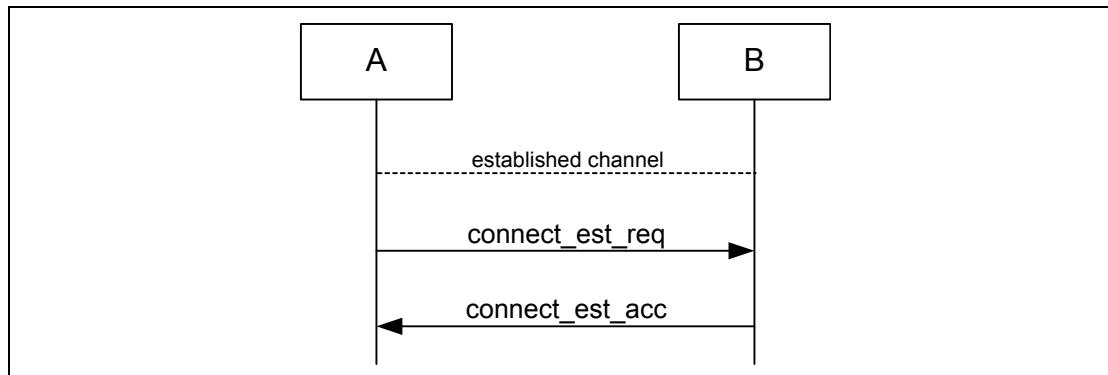


Figure 7.6: Connection establishment procedure when the initiator (A) is in security mode 3 and the acceptor (B) is in security mode 1 or 3

7.3.4 Conditions

Connection establishment starts after channel establishment is completed, when the initiator sends a connection establishment request ('connect_est_req' is application protocol-dependent). This request may be a TCS SETUP message [4] in the case of a Bluetooth telephony application [Cordless Telephony Profile](#), or initialization of RFCOMM and establishment of DLC [3] in the case of a serial port-based application [Serial Port Profile](#) (although neither TCS or RFCOMM use the term 'connection' for this).

Connection establishment is completed when the acceptor accepts the connection establishment request ('connect_est_acc' is application protocol dependent).

7.4 ESTABLISHMENT OF ADDITIONAL CONNECTION

When a Bluetooth device has established one connection with another Bluetooth device, it may be available for establishment of:

- A second connection on the same channel, and/or
- A second channel on the same logical link, and/or
- A second physical link.

If the new establishment procedure is to be towards the same device, the security part of the establishment depends on the security modes used. If the new establishment procedure is to be towards a new remote device, the device should behave according to active modes independent of the fact that it already has another physical link established (unless allowed co-incident radio and baseband events have to be handled).

7.5 SYNCHRONIZATION ESTABLISHMENT

7.5.1 Purpose

The purpose of the Synchronization Establishment procedure is for a device to receive synchronization train packets using the procedures in [Vol. 2], Part B Section 2.7.3

7.5.2 Term on UI Level

'Bluetooth synchronization establishment'

7.5.3 Description

In [Figure 7.7](#) the receiving device (B) is attempting to receive synchronization train packets from device (A).

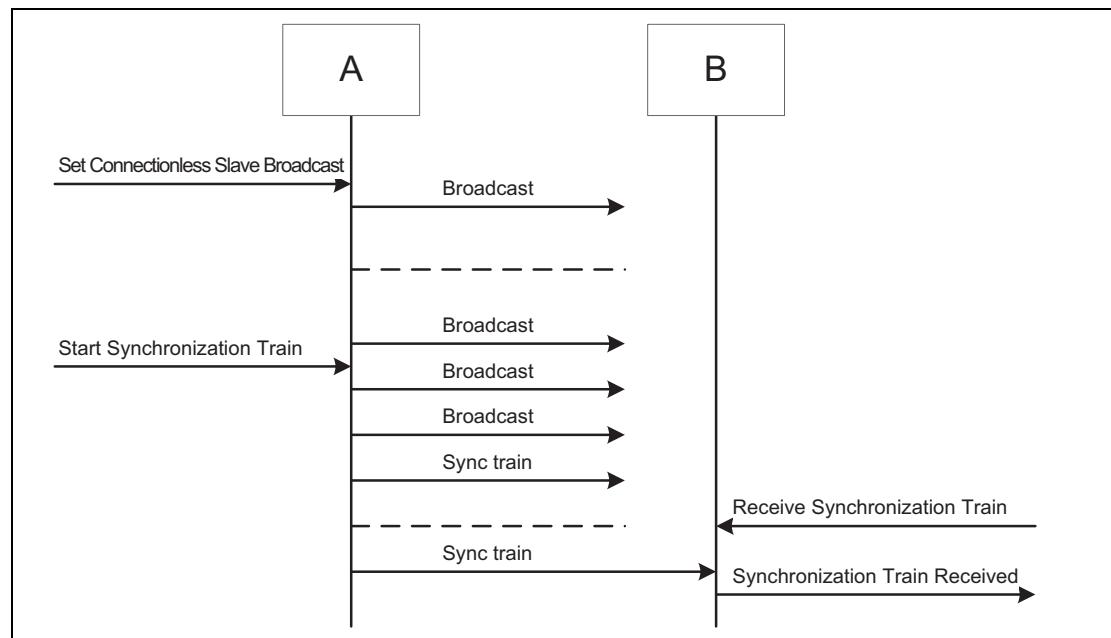


Figure 7.7: Synchronization establishment procedure

7.5.4 Conditions

After receiving a synchronization train packet, the receiving device can listen to and receive profile data sent via Connectionless Slave Broadcast by device A.

Note that devices A and B may go through a separate Link Establishment procedure any time they desire to establish an ACL logical transport between each other. They may also use Connectionless Slave Broadcast procedures with an ACL logical transport already established.

Generic Access Profile

The receiving device shall enter the **synchronization scan** substate using a scan interval of $T_{GAP}(\text{Sync_Scan_Interval})$ and a scan window of $T_{GAP}(\text{Sync_Scan_Window})$.

8 EXTENDED INQUIRY RESPONSE DATA FORMAT

The extended inquiry response data format is shown in [Figure 8.1](#). The data is 240 octets and consists of a significant part and a non-significant part. The significant part contains a sequence of data structures. Each data structure shall have a length field of one octet, which contains the Length value, and a data field of Length octets. The first n octets of the data field contain the extended inquiry response (EIR) data type. The content of the remaining $\text{Length} - n$ octets in the data field depends on the value of the EIR data type and is called the EIR data. The non-significant part extends the extended inquiry response to 240 octets and shall contain all-zero octets.

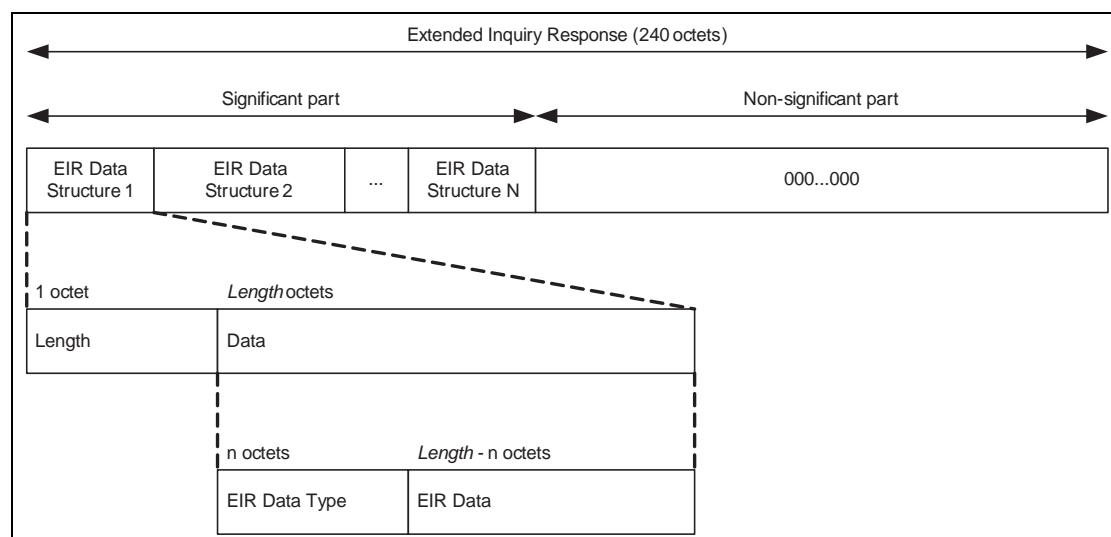


Figure 8.1: Extended Inquiry Response data format

The extended inquiry response data formats and meanings are defined in [\[Core Specification Supplement\]](#), Part A. The extended inquiry response data type values are defined in the [Assigned Numbers](#) document.

If the length field is set to zero, then the data field has zero octets. This shall only occur to allow an early termination of the tagged data.

To reduce interference, the host should try to minimize the amount of EIR data such that the baseband can use a 1-slot or 3-slot EIR packet. This is advantageous because it reduces interference and maximizes the probability that the EIR packet will be received. If applications on a host provide more than 240 bytes of extended inquiry response data, it is up to the host to limit it to 240 octets.

The EIR data shall be sent during the inquiry response state. EIR data can contain device name, Tx power level, service class UUIDs, as well as manufacturer's data, as defined in [\[Core Specification Supplement\]](#), Part A. In selecting the packet type to be used, FEC (DM1 or DM3) should be considered to maximize the range.

The Host shall include the device name in the EIR data according to the following rules:

1. If the device does not have a device name (i.e., 0 octet) and
 - a) If there is no other data to be sent in the EIR packet, the Host shall send a name tag with zero length and the type field set to indicate that this is the complete name (i.e., total of 2 octets with length = 1).
 - b) If there is other important data to be sent in the EIR packet and a zero octet name tag will not fit, the Host may avoid sending the name tag.
2. If the device has a device name (greater than 0 octet) and
 - a) If it is too long be included in the EIR packet (given the choice of packet type and any other data that is being sent), the Host may send a shortened version of the name (even 0 octet) and shall mark the name as 'shortened' to inform the receiver that a remote name request is required obtain the full name if the name is needed.
 - b) If there are no other data to be sent in the EIR packet (given the choice of packet type selected), the Host shall maximize the length of the device name to be sent, this may be complete or shortened name (e.g., if DM1 packet is chosen and device name characters equates to greater than 15 octets, then host sends first few characters that equates to 15 octets or less with shortened flag).

Note: It is not necessary to understand each and every EIR data type. If the Host does not understand a given EIR data type value it should just skip over Length octets and look for the next EIR data structure.

9 OPERATIONAL MODES AND PROCEDURES – LE PHYSICAL TRANSPORT

Several different modes and procedures may be performed simultaneously over an LE physical transport. The following modes and procedures are defined for use over an LE physical transport:

- Broadcast mode and observation procedure
- Discovery modes and procedures
- Connection modes and procedures
- Bonding modes and procedures

Each of the above modes and procedures are independent from each other but are closely related since a combination of the modes and procedures are necessary for most devices to communicate with each other. Both the modes and procedures may be entered or executed respectively as a result of direct user action or autonomously by a device.

The Host shall configure the Controller with its local Link Layer feature information as defined in [\[Vol. 6\], Part B Section 4.6](#) before performing any of the above modes and procedures.

9.1 BROADCAST MODE AND OBSERVATION PROCEDURE

The broadcast mode and observation procedure allow two devices to communicate in a unidirectional connectionless manner using the advertising events. The requirements for a device operating in a specific GAP role to support the broadcast mode and observation procedure are defined in [Table 9.1](#).

Broadcast Mode and Observation procedure	Ref.	Broadcaster	Observer
Broadcast mode	9.1.1	M	E
Observation procedure	9.1.2	E	M

Table 9.1: Broadcast mode and Observation procedure Requirements

9.1.1 Broadcast Mode

9.1.1.1 Definition

The broadcast mode provides a method for a device to send connectionless data in advertising events.

9.1.1.2 Conditions

A device in the broadcast mode shall send data in either non-connectable undirected or scannable undirected advertising events.

The advertising data shall be formatted using the Advertising Data (AD) type format as defined in [[Core Specification Supplement]], Part A, Section 1.3. A device in the broadcast mode shall not set the ‘LE General Discoverable Mode’ flag or the ‘LE Limited Discoverable Mode’ flag in the Flags AD Type as defined in [Core Specification Supplement], Part A, Section 1.3.

Note: All data sent by a device in the broadcast mode is considered unreliable since there is no acknowledgement from any device that may have received the data.

9.1.2 Observation Procedure

9.1.2.1 Definition

The observation procedure provides a method for a device to receive connectionless data from a device that is sending advertising events.

9.1.2.2 Conditions

A device performing the observation procedure may use passive scanning or active scanning to receive advertising events.

A device performing the observation procedure may use active scanning to also receive scan response data sent by any device in the broadcast mode that advertises using scannable undirected advertising events. When a device performing the observation procedure receives a resolvable private address in the advertising event, the device may resolve the private address by using the resolvable private address resolution procedure as defined in [Section 10.8.2.3](#).

Note: In use cases where a device in the broadcast mode sends dynamic data, the receiving device should disable duplicate filtering capability in the Controller so that the Host receives all advertising packets received by the Controller.

9.2 DISCOVERY MODES AND PROCEDURES

All devices shall be in either non-discoverable mode or one of the discoverable modes. A device in the discoverable mode shall be in either the general discoverable mode or the limited discoverable mode. A device in the non-discoverable mode is not discoverable. Devices operating in either the general discoverable mode or the limited discoverable mode can be found by the discovering device. A device that is discovering other devices performs either the limited discovery procedure as defined in [Section 9.2.5](#) or the general discovery procedure as defined in [Section 9.2.6](#).

9.2.1 Requirements

Discovery modes and procedures	Ref.	Peripheral	Central
Non-Discoverable mode	9.2.2	M	E
Limited Discoverable mode	9.2.3	O	E
General Discoverable mode	9.2.4	C1	E
Limited Discovery procedure	9.2.5	E	O
General Discovery procedure	9.2.6	E	M
Name Discovery procedure	9.2.7	O	O

C1: if limited discoverable mode is not supported then general discoverable mode is mandatory, else optional.

Table 9.2: Device discovery requirements

9.2.2 Non-Discoverable Mode

9.2.2.1 Description

A device configured in non-discoverable mode will not be discovered by any device that is performing either the general discovery procedure or the limited discovery procedure.

9.2.2.2 Conditions

A device in the non-discoverable mode that sends advertising events shall not set the 'LE General Discoverable Mode' flag or 'LE Limited Discoverable Mode' flag in the Flags AD type (see [[Core Specification Supplement](#)], Part A, Section 1.3). A Peripheral device in the non-connectable mode may send non-connectable undirected advertising events or scannable undirected advertising events or may not send advertising packets.

If the Peripheral device in the non-discoverable mode sends non-connectable advertising events or scannable undirected advertising events then it is recommended that the Host configures the Controller as follows:

- The Host should set the advertising filter policy to either 'process scan and connection requests only from devices in the White List' or 'process scan and connection requests from all devices'.
- The Host should set the advertising intervals as defined in [Section 9.3.11](#).

9.2.3 Limited Discoverable Mode

9.2.3.1 Description

Devices configured in the limited discoverable mode are discoverable for a limited period of time by other devices performing the limited or general device discovery procedure. The limited discoverable mode is typically used when a user performs a specific action to make the device discoverable for a limited period of time.

9.2.3.2 Conditions

While a device is in the Peripheral role the device may support the limited discoverable mode. While a device is in the Broadcaster, Observer or Central role the device shall not support the limited discoverable mode.

A device in the limited discoverable mode sends either non-connectable advertising events, scannable undirected advertising events or connectable undirected advertising events.

While in the limited discoverable mode the device shall send advertising event types with the advertising data including the Flags AD type as defined in [[Core Specification Supplement](#)], Part A, Section 1.3 with all the following flags set as described:

- The LE Limited Discoverable Mode flag set to one.
- The LE General Discoverable Mode flag set to zero.
- For a device of the LE-only device type with all the following flags set as described:
 - a) The ‘BR/EDR Not Supported’ flag to set one.
 - b) The ‘Simultaneous LE and BR/EDR to Same Device Capable (Controller)’ flag set to zero.
 - c) The ‘Simultaneous LE and BR/EDR to Same Device Capable (Host)’ flag set to zero.

The advertising data should also include the following AD types to enable a faster connectivity experience:

- TX Power Level AD type defined in [[Core Specification Supplement](#)], Part A, Section 1.5.
- Local Name AD type defined in [[Core Specification Supplement](#)], Part A, Section 1.2.
- Service UUIDs AD type defined in [[Core Specification Supplement](#)], Part A, Section 1.1.
- Slave Connection Interval Range AD type as defined in [[Core Specification Supplement](#)], Part A, Section 1.9.

Devices shall remain in the limited discoverable mode no longer than $T_{GAP}(\text{lim_adv_timeout})$.

While a device is in limited discoverable mode the Host configures the Controller as follows:

- The Host shall set the advertising filter policy to ‘process scan and connection requests from all devices’.
- The Host should set the advertising intervals as defined in [Section 9.3.11](#).

The device shall remain in limited discoverable mode until a connection is established or the Host terminates the mode.

If a device is in the limited discoverable mode and in the undirected connectable mode the advertising interval values should be set as defined in [Section 9.3.4](#)

A device in the limited discoverable mode shall not set both the LE Limited Discoverable Flag and the LE General Discoverable Flag to one.

Note: Data that change frequently should be placed in the advertising data and static data should be placed in the scan response data.

Note: The choice of advertising interval is a trade-off between power consumption and device discovery time.

9.2.4 General Discoverable Mode

9.2.4.1 Description

Devices configured in general discoverable mode are intended to be discoverable by devices performing the general discovery procedure. The general discoverable mode is typically used when the device is intending to be discoverable for a long period of time.

9.2.4.2 Conditions

While a device is in the Peripheral role the device may support the general discoverable mode. While a device is in the Broadcaster, Observer or Central role the device shall not support the general discoverable mode.

A device in the general discoverable mode sends either non-connectable advertising events, scannable undirected advertising events, or connectable undirected advertising events.

While in general discoverable mode the device shall send advertising events with the advertising data including the Flags AD data type as defined in [[Core Specification Supplement](#)], Part A, Section 1.3 with all the following flags set as described:

- The LE Limited Discoverable Mode flag set to zero.
- The LE General Discoverable Mode flag set to one.
- For a device of the LE-only device type with all the following flags set as described:
 - a) The ‘BR/EDR Not Supported’ flag set to one.
 - b) The ‘Simultaneous LE and BR/EDR to Same Device Capable (Controller)’ flag set to zero.
 - c) The ‘Simultaneous LE and BR/EDR to Same Device Capable (Host)’ flag set to zero.

The advertising data should also include the following AD types to enable a faster connectivity experience:

- TX Power Level AD type as defined in [[Core Specification Supplement](#)], Part A, Section 1.5.
- Local Name AD type as defined in [[Core Specification Supplement](#)], Part A, Section 1.2.
- Service UUIDs AD type as defined in [[Core Specification Supplement](#)], Part A, Section 1.1.
- Slave Connection Interval Range AD type as defined in [[Core Specification Supplement](#)], Part A, Section 1.9.

While a device is in general discoverable mode the Host configures the Controller as follows:

- The Host shall set the advertising filter policy to ‘process scan and connection requests from all devices’.
- The Host should set the advertising intervals as defined in [Section 9.3.11](#).

The device shall remain in general discoverable mode until a connection is established or the Host terminates the mode.

If a device is in the general discoverable mode and in the directed connectable mode or the non-connectable mode, the advertising interval values should be set as defined in [Section 9.3.3](#).

A device in the general discoverable mode shall not set both the LE Limited Discoverable Flag and the LE General Discoverable Flag to one.

Note: Data that change frequently should be placed in the advertising data and static data should be placed in the scan response data.

Note: The choice of advertising interval is a trade-off between power consumption and device discovery time.

9.2.5 Limited Discovery Procedure

9.2.5.1 Description

A device performing the limited discovery procedure receives the device address, advertising data and scan response data from devices in the limited discoverable mode only.

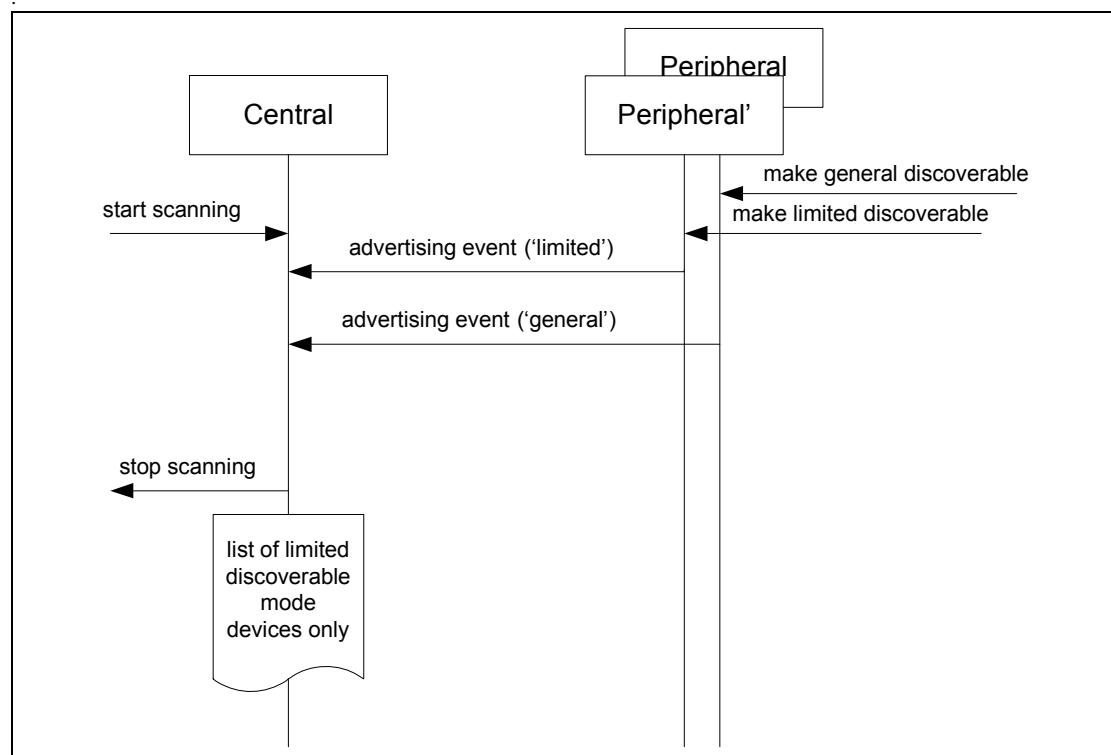


Figure 9.1: A Central performing limited discovery procedure discovering Peripherals in the limited discoverable mode

9.2.5.2 Conditions

While a device is in the Central role the device may support the limited discovery procedure. While a device is in the Broadcaster, Observer or Peripheral role the device shall not support the limited discovery procedure.

When a Host performs the limited discovery procedure, the Host configures the Controller as follows:

1. The Host shall set the scanner filter policy to ‘process all advertising packets’.
2. The Host should set the scan interval and scan window as defined in [Section 9.3.11](#).
3. The Host should configure the Controller to use active scanning.

The Host shall begin scanning for advertising packets and should continue for a minimum of $T_{GAP}(\text{lim_disc_scan_min})$, unless the host ends the limited discovery procedure.

The Host shall check for the Flags AD type in the advertising data. If the Flags AD type is present and the LE Limited Discoverable Flag is set to one then the Host shall consider the device as a discovered device, otherwise the advertising data shall be ignored. The Flag AD type is defined in [[Core Specification Supplement](#)], Part A, Section 1.3. The advertising data of the discovered device may contain data with other AD types, e.g. Service UUIDs AD type, TX Power Level AD type, Local Name AD type, Slave Connection Interval Range AD type. The Host may use the data in performing any of the connection establishment procedures.

The host shall ignore the 'Simultaneous LE and BR/EDR to Same Device Capable (Controller)' and 'Simultaneous LE and BR/EDR to Same Device Capable (Host)' bits in the Flags AD type.

9.2.6 General Discovery Procedure

9.2.6.1 Description

A device performing the general discovery procedure receives the device address, advertising data and scan response data from devices in the limited discoverable mode or the general discoverable mode.

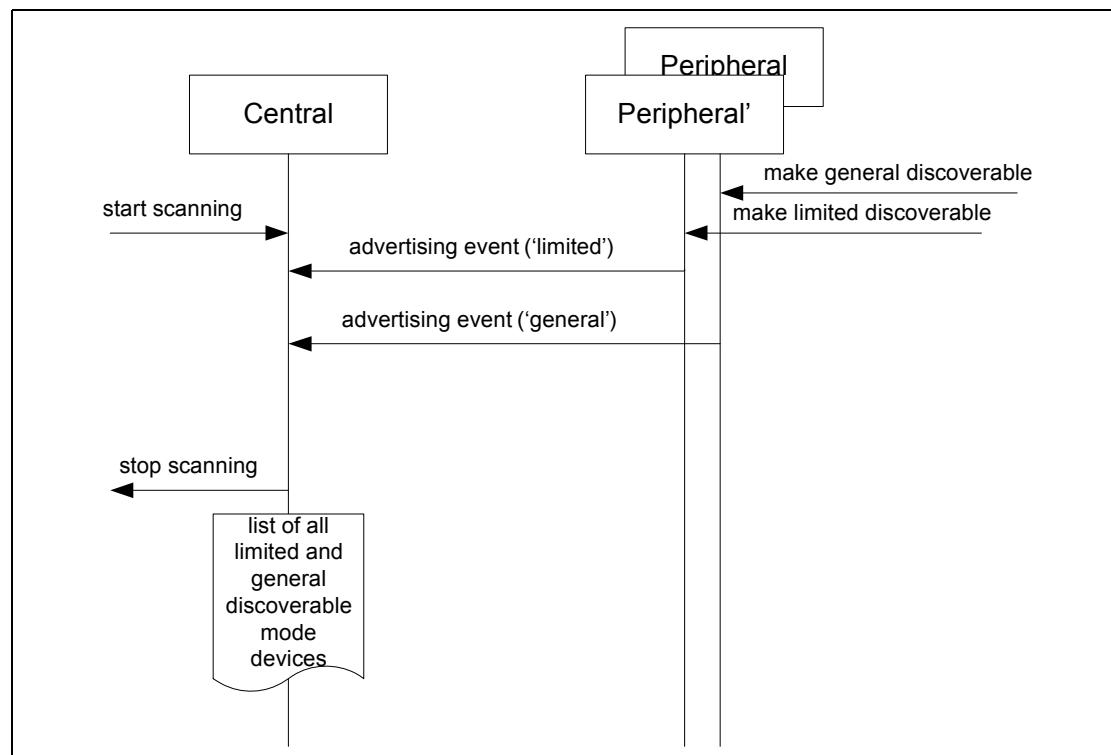


Figure 9.2: A Central performing general discovery procedure discovering Peripherals in the limited discoverable mode and general discoverable mode

9.2.6.2 Conditions

While a device is in the Central role the device shall support the general discovery procedure. While a device is in the Broadcaster, Observer or Peripheral role the device shall not support the general discovery procedure.

When a Host performs the general discovery procedure, the Host configures the Controller as follows:

1. The Host shall set the scanner filter policy to 'process all advertising packets'.
2. The Host should set the scan interval and scan window as defined in [Section 9.3.11](#).
3. The Host should configure the Controller to use active scanning.

The Host shall begin scanning for advertising packets and should continue for a minimum of $T_{GAP}(\text{gen_disc_scan_min})$. The procedure may be terminated early by the Host.

The Host shall check for the Flags AD type in the advertising data. If the Flags AD type (see [[Core Specification Supplement](#)], Part A, Section 1.3) is present and either the LE General Discoverable Mode flag is set to one or the LE Limited Discoverable Mode flag is set to one then the Host shall consider the device as a discovered device, otherwise the advertising data shall be ignored. The advertising data of the discovered device may contain data with other AD types, e.g. Service UUIDs AD type, TX Power Level AD type, Local Name AD type, Slave Connection Interval Range AD type. The Host may use the data in performing any of the connection establishment procedures as defined in [Section 9.3](#).

The host shall ignore the 'Simultaneous LE and BR/EDR to Same Device Capable (Controller)' and 'Simultaneous LE and BR/EDR to Same Device Capable (Host)' bits in the Flags AD type.

9.2.7 Name Discovery Procedure

9.2.7.1 Description

The name discovery procedure is used to obtain the Bluetooth Device Name of a remote connectable device.

9.2.7.2 Conditions

If the complete device name is not acquired while performing either the limited discovery procedure or the general discovery procedure, then the name discovery procedure may be performed.

The name discovery procedure shall be performed as follows:

1. The Host shall establish a connection using one of the connection establishment procedures as defined in [Section 9.3](#).
2. The Host shall read the device name characteristic using the GATT procedure Read Using Characteristic UUID [\[Vol. 3\], Part G Section 4.8.2](#)
3. The connection may be terminated after the GATT procedure has completed.

9.3 CONNECTION MODES AND PROCEDURES

The connection modes and procedures allow a device to establish a connection to another device.

When devices are connected, the parameters of the connection can be updated with the Connection Parameter Update procedure. The connected device may terminate the connection using the Terminate Connection procedure. The requirements for a device to support the connection modes and procedures are defined in [Table 9.3](#).

9.3.1 Requirements

Connection Modes and Procedures	Ref.	Peripheral	Central	Broadcaster	Observer
Non-connectable mode	9.3.2	M	E	M	M
Directed connectable mode	9.3.3	O	E	E	E
Undirected connectable mode	9.3.4	M	E	E	E
Auto connection establishment procedure	9.3.5	E	O	E	E
General connection establishment procedure	9.3.6	E	C1	E	E
Selective connection establishment procedure	9.3.7	E	O	E	E
Direct connection establishment procedure	9.3.8	E	M	E	E
Connection parameter update procedure	9.3.9	O	M	E	E
Terminate connection procedure	9.3.10	M	M	E	E

C1: Mandatory if privacy feature is supported, else optional

Table 9.3: Connection modes and procedures requirements

9.3.2 Non-Connectable Mode

9.3.2.1 Description

A device in the non-connectable mode shall not allow a connection to be established.

9.3.2.2 Conditions

While a device is in the Peripheral role the device shall support the non-connectable mode. A device in the Broadcaster or Observer role cannot establish a connection, therefore the device is considered to support the non-connectable mode.

A Peripheral device in the non-connectable mode may send non-connectable undirected advertising events or scannable undirected advertising events. In this case it is recommended that the Host configures the Controller as follows:

- The Host should set the advertising filter policy to either ‘process scan and connection requests only from devices in the White List’ or ‘process scan and connection requests from all devices’.
- The Host should set the advertising intervals as defined in [Section 9.3.11](#).

9.3.3 Directed Connectable Mode

9.3.3.1 Description

A device in the directed connectable mode shall accept a connection request from a known peer device performing the auto connection establishment procedure or the general connection establishment procedure.

9.3.3.2 Conditions

While a device is in the Peripheral role the device may support the directed connectable mode. This mode shall only be used if the device has a known peer device address. While a device is in the Broadcaster, Observer, or the Central role the device shall not support the direct connectable mode.

The Host shall configure the Controller to send directed connectable advertising events.

9.3.4 Undirected Connectable Mode

9.3.4.1 Description

A device in the undirected connectable mode shall accept a connection request from a device performing the auto connection establishment procedure or the general connection establishment procedure.

9.3.4.2 Conditions

While a device is in the Peripheral role the device shall support the undirected connectable mode. While a device is in the Broadcaster, Observer, or the Central role the device shall not support the undirected connectable mode.

The Host should configure the Controller as defined in [Section 9.3.11](#). The Host shall configure the Controller to send undirected connectable advertising events.

9.3.5 Auto Connection Establishment Procedure

9.3.5.1 Description

The auto connection establishment procedure allows the Host to configure the Controller to autonomously establish a connection with one or more devices in the directed connectable mode or the undirected connectable mode. White Lists in the Controller are used to store device addresses. This procedure uses the initiator White List in the Controller. The Controller autonomously establishes a connection with a device with the device address that matches the address stored in the White List.

9.3.5.2 Conditions

While a device is in the Central role the device may support the auto connection establishment procedure. While a device is in the Broadcaster, Observer, or Peripheral role the device shall not support the auto connection establishment procedure.

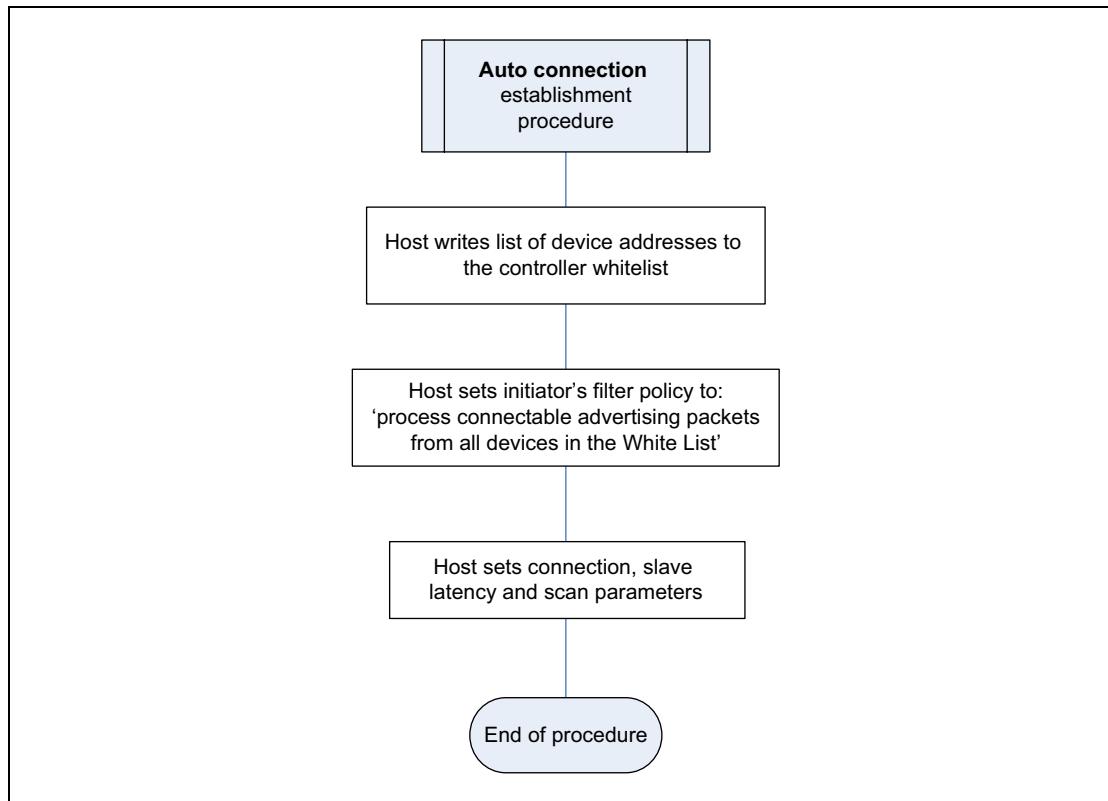


Figure 9.3: Flow chart for a device performing the auto connection establishment procedure

When a Host performs the auto connection establishment procedure, the Host configures the Controller as follows:

1. The Host shall write the list of device addresses that are to be auto connected to into the White List.
2. The Host shall set the initiator filter policy to ‘process connectable advertising packets from all devices in the White List’.
3. The Host should set the scan interval and scan window as defined in [Section 9.3.11](#).
4. The Host should set connection parameters as defined in [Section 9.3.12](#).

This procedure is terminated when a connection is established or when the Host terminates the procedure.

9.3.6 General Connection Establishment Procedure

9.3.6.1 Description

The general connection establishment procedure allows the Host to establish a connection with a set of known peer devices in the directed connectable mode or the undirected connectable mode.

9.3.6.2 Conditions

While a device is in the Central role the device may support the general connection establishment procedure. While a device is in the Broadcaster, Observer, or Peripheral role the device shall not support the general connection establishment procedure.

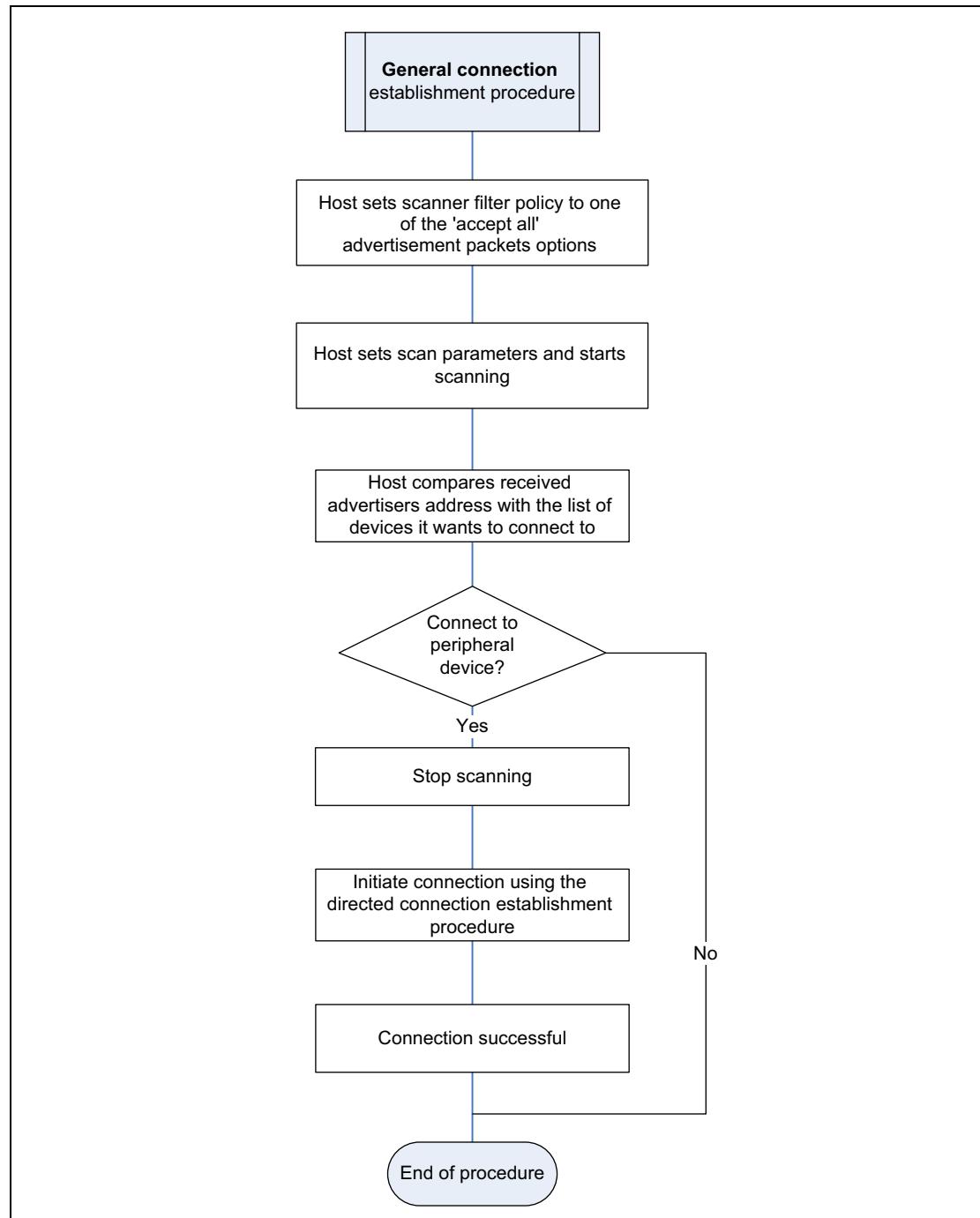


Figure 9.4: Flow chart for a device performing the general connection establishment procedure

When a Host performs the general connection establishment procedure, the Host configures the Controller as follows:

1. The Host shall set the scanner filter policy to one of the 'accept all' options as defined in [\[Vol 2\] Part E, Section 7.8.10](#).
2. The Host should set the scan interval as defined in [Section 9.3.11](#).
3. The Host should set the scan window as defined in [Section 9.3.11](#).
4. The Host shall start scanning.
5. The Host should set connection parameters as defined in [Section 9.3.12](#).

When the Host discovers a device to which the Host may attempt to connect, the Host shall stop the scanning, and initiate a connection using the direct connection establishment procedure.

This procedure is terminated when a connection is established or when the Host terminates the procedure.

9.3.7 Selective Connection Establishment Procedure

9.3.7.1 Description

The selective connection establishment procedure allows the Host to establish a connection with the Host selected connection configuration parameters with a set of devices addresses in the White List.

9.3.7.2 Conditions

While a device is in the Central role the device may support the selective connection establishment procedure. While a device is in the Broadcaster, Observer, or the Peripheral role the device shall not support the selective connection establishment procedure. [Figure 9.5](#) shows the flow chart for a device performing the selective connection establishment procedure.

When a Host performs the selective connection establishment procedure, the Host configures the Controller as follows:

1. The Host shall write the list of device addresses that are to be selectively connected into the White List.
2. The Host shall set the scanner filter policy to the 'accept only' option as defined in [\[Vol 2\] Part E, Section 7.8.10](#).
3. The Host should set the scan interval as defined in [Section 9.3.11](#).
4. The Host should set the scan window as defined in [Section 9.3.11](#).
5. The Host shall start active scanning or passive scanning.

When the Host discovers one of the peer devices it is connecting to, the Host shall stop scanning, and initiate a connection using the direct connection establishment procedure with the connection configuration parameters for that peer device.

This procedure is terminated when a connection is established or when the Host terminates the procedure.

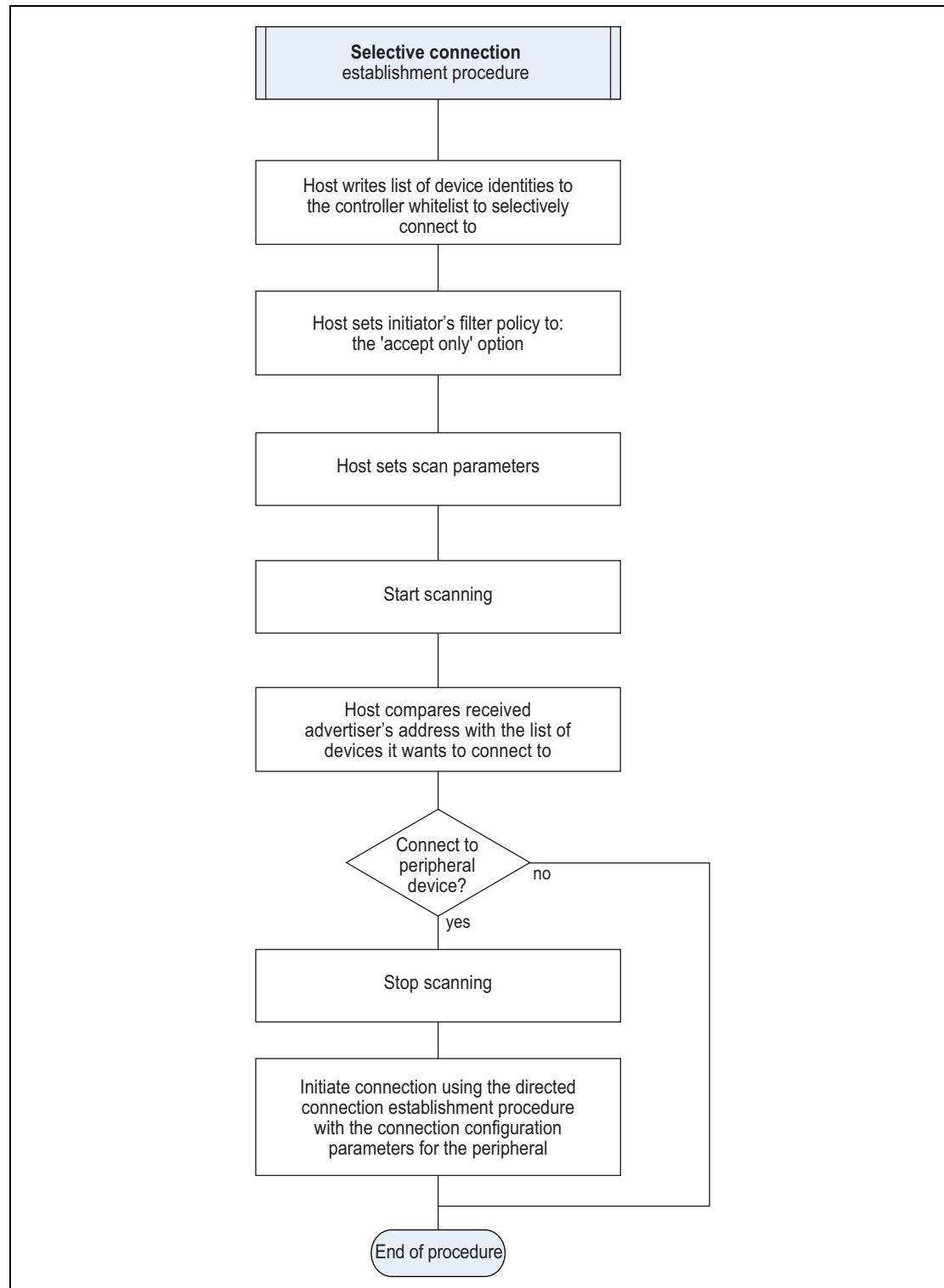


Figure 9.5: Flow chart for a device performing the selective connection establishment procedure

9.3.8 Direct Connection Establishment Procedure

9.3.8.1 Description

The direct connection establishment procedure allows the Host to establish a connection with the Host selected connection configuration parameters with a single peer device.

9.3.8.2 Conditions

While a device is in the Central role the device shall support the direct connection establishment procedure. While a device is in the Broadcaster, Observer, or the Peripheral role the device shall not support the direct connection establishment procedure.

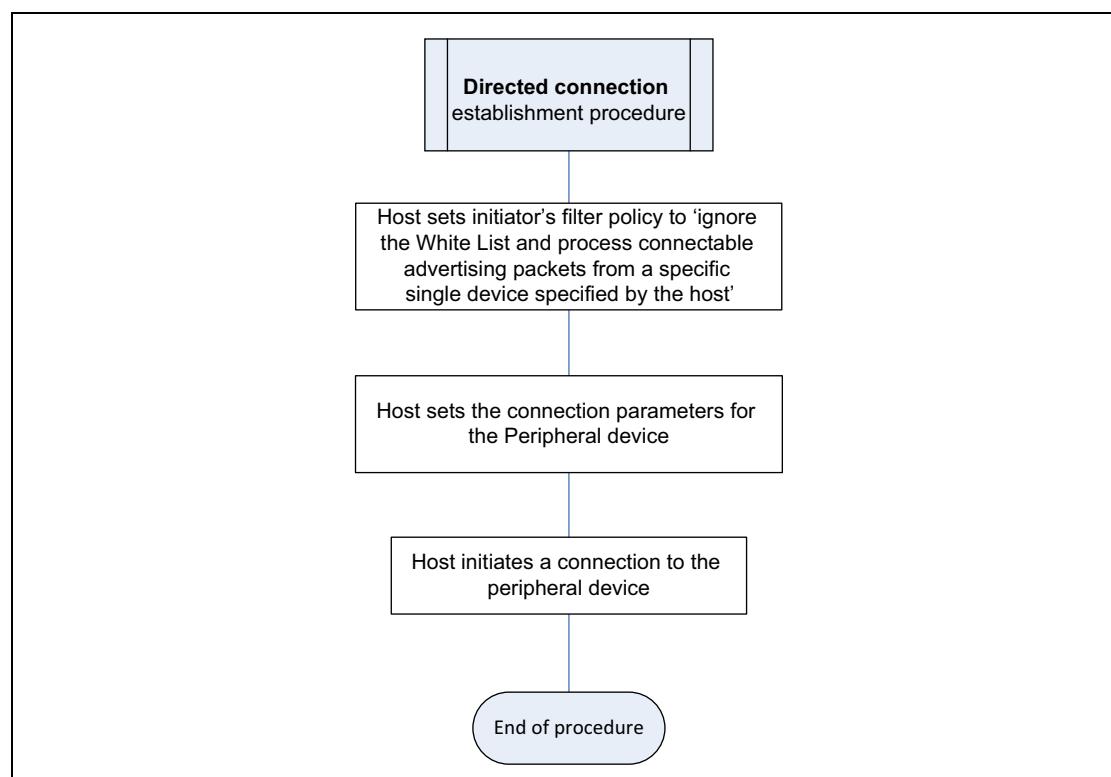


Figure 9.6: Flow chart for a device performing the directed connection establishment procedure

When a Host performs the direct connection establishment procedure, the Host configures the Controller as follows:

1. The Host shall set the initiator filter policy to "ignore the White List and process connectable advertising packets from a specific single device specified by the Host".
2. The Host shall set the peer address to the device address of the specific single device.
3. The Host should set connection parameters as defined in [Section 9.3.12](#).

4. The Host shall initiate a connection.

This procedure is terminated when a connection is established or when the Host terminates the procedure.

9.3.9 Connection Parameter Update Procedure

9.3.9.1 Description

The connection parameter update procedure allows a Peripheral or Central to update the Link Layer connection parameters of an established connection.

9.3.9.2 Conditions

While a device is in the Central role the device shall support the connection parameter update procedure. While a device is in the Peripheral role the device may support the connection parameter update procedure. While a device is in the Broadcaster or the Observer role the device shall not support the connection parameter update procedure.

A Central initiating the connection parameter update procedure shall use the Link Layer Connection Update procedure defined in [\[Vol. 6\], Part B Section 5.1.1](#) with the required connection parameters if either the Central or the Peripheral does not support the Connection Parameters Request Link Layer Control procedure.

If both the Central and Peripheral support the Connection Parameters Request Link Layer control procedure, then the Central or Peripheral initiating the connection parameter update procedure shall use the Connection Parameters Request Link Layer Control procedure defined in [\[Vol. 6\], Part B Section 5.1.7](#) with the required connection parameters.

If either the Central or the Peripheral does not support the Connection Parameters Request Link Layer Control procedure, then the Peripheral initiating the connection parameter update procedure shall use the L2CAP Connection Parameter Update Request command defined in [\[Vol. 1\], Part A Section 4.2](#) with the required connection parameters. The Peripheral shall not send an L2CAP Connection Parameter Update Request command within $T_{GAP}(\text{conn_param_timeout})$ of an L2CAP Connection Parameter Update Response being received. When the Central accepts the Peripheral initiated Connection Parameter Update, the Central should initiate the Link Layer Connection Update procedure defined in [\[Vol. 6\], Part B Section 5.1.1](#) with the required connection parameters within $T_{GAP}(\text{conn_param_timeout})$.

If the requested or updated connection parameters are unacceptable to the Central or Peripheral then it may disconnect the connection with the error code 0x3B (Unacceptable Connection Parameters). Devices should be tolerant of connection parameters given to them by the remote device.

9.3.10 Terminate Connection Procedure

9.3.10.1 Description

The terminate connection procedure allows a Host to terminate the connection with a peer device.

9.3.10.2 Conditions

Centrals and Peripherals shall support the terminate connection procedure. Broadcasters and Observers shall not support the terminate connection procedure.

The Host initiating the terminate connection procedure shall use the Link Layer Termination Procedure defined in [\[Vol. 6\], Part B Section 5.1.6](#).

9.3.11 Connection Establishment Timing Parameters

9.3.11.1 Description

The connection establishment timing parameters are used during initial connection establishment between a Central and a Peripheral device.

A Central device should use one of the GAP connection establishment procedures to initiate a connection to a Peripheral device in a connectable mode. The procedures and modes that may use these timing parameters are defined in [Section 9.3.4](#) through [Section 9.3.8](#).

9.3.11.2 Conditions

A Central device starting a GAP connection establishment procedure should use the recommended scan interval $T_{GAP}(\text{scan_fast_interval})$ and scan window $T_{GAP}(\text{scan_fast_window})$ for $T_{GAP}(\text{scan_fast_period})$.

A Central device that is background scanning should use the recommended scan interval $T_{GAP}(\text{scan_slow_interval1})$ and scan window $T_{GAP}(\text{scan_slow_window1})$. Alternatively the central may use $T_{GAP}(\text{scan_slow_interval2})$ and scan window $T_{GAP}(\text{scan_slow_window2})$.

A Peripheral device entering any of the following GAP Modes should use the recommended advertising interval $T_{GAP}(\text{adv_fast_interval1})$ for $T_{GAP}(\text{adv_fast_period})$:

- Undirected Connectable Mode
- Limited Discoverable Mode and sending connectable undirected advertising events
- General Discoverable Mode and sending connectable undirected advertising events

- Directed Connectable Mode and sending low duty cycle directed advertising events

A Peripheral device when entering any of the following GAP Modes and sending either non-connectable advertising events or scannable undirected advertising events should use the recommended advertising interval $T_{GAP}(\text{adv_fast_interval2})$ for $T_{GAP}(\text{adv_fast_period})$:

- Non-Discoverable Mode
- Non-Connectable Mode
- Limited Discoverable Mode
- General Discoverable Mode

A Peripheral device that is background advertising in any GAP Mode other than GAP Directed Connectable Mode with high duty cycle directed advertising events should use the recommended advertising interval $T_{GAP}(\text{adv_slow_interval})$.

9.3.12 Connection Interval Timing Parameters

9.3.12.1 Description

The connection interval timing parameters are used within a connection. Initial connection interval is used to ensure procedures such as bonding, encryption setup and service discovery are completed quickly. The connection interval should be changed to the Peripheral Preferred Connection Parameters after initiating procedures are complete.

9.3.12.2 Conditions

The Central device should either read the Peripheral Preferred Connection Parameters characteristic (see [Section 12.3](#)) or retrieve the parameters from advertising data (see [Section 12.3](#)).

The connection interval should be set to $T_{GAP}(\text{initial_conn_interval})$ and slave latency should be set to zero. These parameters should be used until the Central device has no further pending actions to perform or until the Peripheral device performs a Connection Parameter Update procedure (see [Section 9.3.9](#)).

After the Central device has no further pending actions to perform and the Peripheral device has not initiated any other actions within $T_{GAP}(\text{conn_pause_central})$, then the Central device should invoke the Connection Parameter Update procedure (see [Section 9.3.9](#)) and change the connection intervals as defined in the Peripheral Preferred Connection Parameters.

If the Central device does not have the Peripheral Preferred Connection Parameters, then the Central device may choose the connection parameters to be used.

After the Peripheral device has no further pending actions to perform and the Central device has not initiated any other actions within $T_{GAP}(\text{conn_pause_central})$, then the Peripheral device may perform a Connection Parameter Update procedure (see [Section 9.3.9](#)). The Peripheral device should not perform a Connection Parameter Update procedure within $T_{GAP}(\text{conn_pause_peripheral})$ after establishing a connection.

At any time a key refresh or encryption setup procedure is required and the current connection interval is greater than $T_{GAP}(\text{initial_conn_interval})$, the key refresh or encryption setup procedure should be preceded with a Connection Parameter Update procedure (see [Section 9.3.9](#)). The connection interval should be set to $T_{GAP}(\text{initial_conn_interval})$ and slave latency should be set to zero. This fast connection interval should be maintained until the key refresh or encryption setup procedure is complete. It should then switch to the Peripheral Preferred Connection Parameters.

9.4 BONDING MODES AND PROCEDURES

Bonding allows two connected devices to exchange and store security and identity information to create a trusted relationship. The security and identity information as defined in [\[Vol. 3\], Part H Section 2.4.1](#) is also known as the bonding information. When the devices store the bonding information, it is known as the phrases ‘devices have bonded’ or ‘a bond is created’.

There are two modes for bonding, non-bondable mode and bondable mode. Bonding may only occur between two devices in bondable mode. The requirements for a device to support the bonding modes and procedure are defined in [Table 9.4](#).

9.4.1 Requirements

Bonding	Ref.	Peripheral	Central
Non-Bondable mode	9.4.2	M	M
Bondable mode	9.4.3	O	O
Bonding procedure	9.4.4	O	O

Table 9.4: Bonding Compliance Requirements

9.4.2 Non-Bondable Mode

9.4.2.1 Description

A device in the non-bondable mode does not allow a bond to be created with a peer device.

9.4.2.2 Conditions

While a device is in the Peripheral or the Central role the device shall support the non-bondable mode. If a device does not support pairing as defined in the Security Manager section then it is considered to be in non-bondable mode.

If Security Manager pairing is supported, the Host shall set the Bonding_Flags to 'No Bonding' as defined in [\[Vol. 3\], Part H Section 3.5.1](#) and bonding information shall not be exchanged or stored.

9.4.3 Bondable Mode

9.4.3.1 Description

A device in the bondable mode allows a bond to be created with a peer device in the bondable mode.

9.4.3.2 Conditions

The Host shall set the Bonding_Flags to 'Bonding' during the pairing procedure.

9.4.4 Bonding Procedure

9.4.4.1 Description

The bonding procedure may be performed when a non-bonded device tries to access a service that requires bonding. The bonding procedure may be performed for the purpose of creating a bond between two devices.

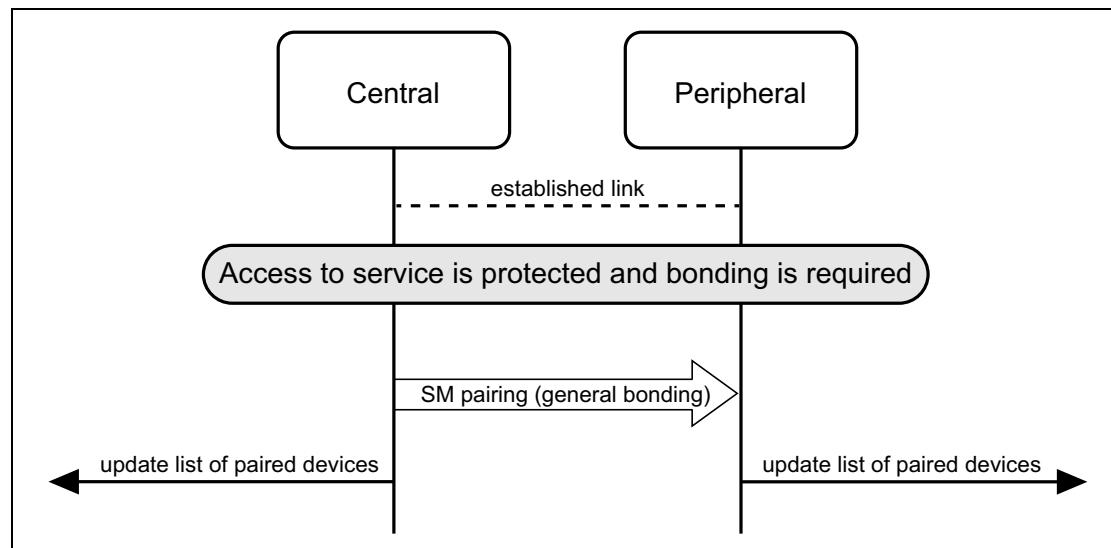


Figure 9.7: The Bonding procedure

9.4.4.2 Conditions

While a device is in the Peripheral or the Central role the device may support the Bonding procedure. While a device is in the Broadcaster or the Observer role the device shall not support the bonding procedure.

The Host of the Central initiates the pairing process as defined in [\[Vol. 3\], Part C Section 2.1](#) with the Bonding_Flags set to Bonding as defined in [\[Vol. 3\], Part H Section 3.5.1](#). If the peer device is in the bondable mode, the devices shall exchange and store the bonding information in the security database.

If a device supports the generation of resolvable private addresses as defined in [Section 10.8.2.2](#), the Host should send its IRK to the peer device and request the IRK of the peer device during the pairing procedure. The Host can abort the pairing procedure if the authentication requirements are not sufficient to distribute the IRK.

10 SECURITY ASPECTS – LE PHYSICAL TRANSPORT

This section defines the modes and procedures that relate to the security of a connection. The following modes and procedures are defined:

- LE security mode 1
- LE security mode 2
- Authentication procedure
- Authorization procedure
- Connection data signing procedure
- Authenticate signed data procedure

Requirements for a device to support the LE security modes and procedures is shown in [Table 10.1](#).

10.1 REQUIREMENTS

Security Modes and Procedures	Ref.	Broadcaster	Observer	Peripheral	Central
LE Security mode 1	10.2.1	E	E	O	O
LE Security mode 2	10.2.2	E	E	O	O
Authentication procedure	10.3	E	E	O	O
Authorization procedure	10.5	E	E	O	O
Connection data signing procedure	10.4.1	E	E	O	O
Authenticate signed data procedure	10.4.2	E	E	O	O

Table 10.1: Requirements related to security modes and procedures

10.2 LE SECURITY MODES

The security requirements of a device, a service or a service request are expressed in terms of a security mode and security level. Each service or service request may have its own security requirement. The device may also have a security requirement. A physical connection between two devices shall operate in only one security mode.

There are two LE security modes, LE security mode 1 and LE security mode 2.

10.2.1 LE Security Mode 1

LE security mode 1 has the following security levels:

1. No security (No authentication and no encryption)
2. Unauthenticated pairing with encryption
3. Authenticated pairing with encryption
4. Authenticated LE Secure Connections pairing with encryption

A connection operating in LE security mode 1 level 2 shall also satisfy the security requirements for LE security mode 1 level 1.

A connection operating in LE security mode 1 level 3 shall also satisfy the security requirements for LE security mode 1 level 2 or LE security mode 1 level 1.

A connection operating in LE security mode 1 level 3 shall also satisfy the security requirements for LE security mode 2.

A connection operating in LE security mode 1 level 4 shall also satisfy the security requirements for LE security mode 1 level 3 or LE security mode 1 level 2 or LE security mode 1 level 1.

A connection operating in LE security mode 1 level 4 shall also satisfy the security requirements for LE security mode 2.

10.2.2 LE Security Mode 2

LE security mode 2 has two security levels:

1. Unauthenticated pairing with data signing
2. Authenticated pairing with data signing

LE security mode 2 shall only be used for connection based data signing.

Data signing as defined in [Section 10.4](#) shall not be used when a connection is operating in LE security mode 1 level 2, LE security mode 1 level 3, or LE security mode 1 level 4.

10.2.3 Mixed Security Modes Requirements

If there are requirements for both LE security mode 1 and LE security mode 2 level 2 for a given physical link then LE security mode 1 level 3 shall be used.

If there are requirements for both LE security mode 1 level 3 and LE security mode 2 for a given physical link then LE security mode 1 level 3 shall be used.

If there are requirements for both LE security mode 1 level 2 and LE security mode 2 level 1 for a given physical link then LE security mode 1 level 2 shall be used.

If there are requirements for both LE security mode 1 level 4 and any other security mode or level for a given physical link then LE security mode 1 level 4 shall be used.

10.2.4 Secure Connections Only Mode

A device may be in a Secure Connections Only mode. When in Secure Connections Only mode only security mode 1 level 4 shall be used except for services that only require security mode 1 level 1.

The device shall only accept new outgoing and incoming service level connections for services that require Security Mode 1, Level 4 when the remote device supports LE Secure Connections and authenticated pairing is used.

10.3 AUTHENTICATION PROCEDURE

The authentication procedure describes how the required security is established when a device initiates a service request to a remote device and when a device receives a service request from a remote device. The authentication procedure covers LE security mode 1. The authentication procedure shall only be initiated after a connection has been established.

LE security mode 2 pertains to the use of data signing and is covered in [Section 10.4](#).

Authentication in LE security mode 1 is achieved by enabling encryption as defined in [Section 10.6](#). The security of the encryption is impacted by the type of pairing performed. There are two types of pairing: authenticated pairing or unauthenticated pairing. Authenticated pairing involves performing the pairing procedure defined in [\[Vol. 3\], Part H Section 2.1](#) with the authentication set to ‘MITM protection’. Unauthenticated pairing involves performing the pairing procedure with authentication set to ‘No MITM protection’.

Note: In this section, the terms “authenticated pairing” and “unauthenticated pairing” refer to the security method used to perform pairing and are not related to the authentication of previously bonded devices during a reconnection.

[Section 10.3.1](#) specifies the authentication procedure when a device responds to a service request. [Section 10.3.2](#) specifies the authentication procedure when a device initiates a service request.

10.3.1 Responding to a Service Request

In this section the local device is the device responding to a service request made by a remote device. In the L2CAP protocol the local device responds with a connection response to a remote device making a connection request. In GATT, the local device is the GATT server and the remote device is the GATT client.

When a local device receives a service request from a remote device, it shall respond with an error code if the service request is denied. The error code is dependent on whether the current connection is encrypted or not and on the type of pairing that was performed to create the LTK to be used.

When a local device receives a service request from a remote device it shall behave according to the following rules:

- The local device’s security database specifies the security settings required to accept a service request. If no encryption and no data signing are required, the service request shall be accepted. If encryption is required the local device must send an error code as defined in [Table 10.2](#). If no encryption is required, but data signing is required, then the local device behavior is as defined in [Section 10.4](#).

- If an LTK is not available, the service request shall be rejected with the error code “Insufficient Authentication”.

Note: When the link is not encrypted, the error code “Insufficient Authentication” does not indicate that MITM protection is required.

- If an LTK is available and encryption is required (LE security mode 1) but encryption is not enabled, the service request shall be rejected with the error code “Insufficient Encryption”. If the encryption is enabled with insufficient key size then the service request shall be rejected with the error code “Insufficient Encryption Key Size.”

- If an authenticated pairing is required but only an unauthenticated pairing has occurred and the link is currently encrypted, the service request shall be rejected with the error code “Insufficient Authentication.”

Note: When unauthenticated pairing has occurred and the link is currently encrypted, the error code “Insufficient Authentication” indicates that MITM protection is required.

- If LE Secure Connections authenticated pairing is required but LE legacy pairing has occurred and the link is currently encrypted, the service request shall be rejected with the error code “Insufficient Authentication”.

The local device will respond with the minimum security level required for access to its services. If the local device has no security requirement it should default to the minimum security level that the local device is capable of as defined in pairing phase 1, (see [\[Vol. 3\], Part H Section 2.1](#)).

A local device shall not require an authenticated pairing (MITM) if the local device does not support the required IO capabilities or OOB data¹.

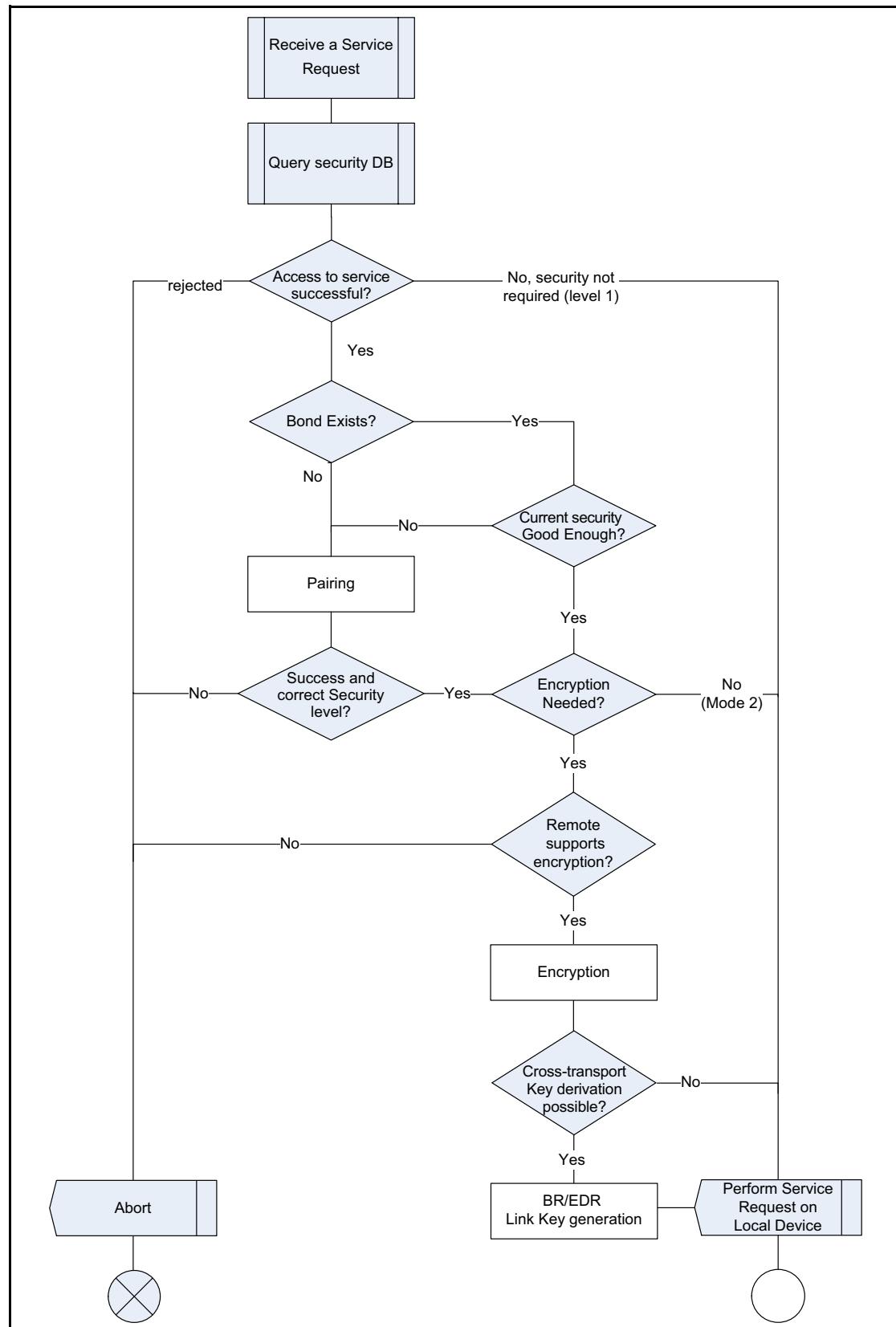
The local device responds to a service request from a remote device are summarized in [Table 10.2](#).

1. Note that if an OOB mechanism is used, the level of MITM protection is dependent upon the properties of the OOB communications channel. See [\[Vol. 3\], Part H Section 2.3.5.1](#) for more information

Link Encryption State	Local Device's Access Requirement for Service	Local Device Pairing Status			
		No LTK	Unauthenticated LTK	Authenticated LTK	Authenticated LTK with Secure Connections
Unencrypted	None	Request succeeds	Request succeeds	Request succeeds	Request succeeds
	Encryption, No MITM Protection	Error Resp.: Insufficient Authentication	Error Resp.: Insufficient Encryption	Error Resp.: Insufficient Encryption	Error Resp.: Insufficient Encryption
	Encryption, MITM Protection	Error Resp.: Insufficient Authentication	Error Resp.: Insufficient Encryption	Error Resp.: Insufficient Encryption	Error Resp.: Insufficient Encryption
	Encryption, MITM Protection, Secure Connections	Error Resp.: Insufficient Authentication	Error Resp.: Insufficient Encryption	Error Resp.: Insufficient Encryption	Error Resp.: Insufficient Encryption
Encrypted	None	N/A (Not possible to be encrypted without LTK)	Request succeeds	Request succeeds	Request succeeds
	Encryption, No MITM Protection		Request succeeds	Request succeeds	Request succeeds
	Encryption, MITM Protection		Error Resp.: Insufficient Authentication	Request succeeds	Request succeeds
	Encryption, MITM Protection, Secure Connections		Error Resp.: Insufficient Authentication	Error Resp.: Insufficient Authentication	Request succeeds

Table 10.2: Local device responds to a service request

[Figure 10.1](#) shows how a server handles a service request.



[Figure 10.1: Flow chart for a local device handling a service request from a remote device](#)

10.3.1.1 Handling of GATT Indications and Notifications

A client “requests” a server to send indications and notifications by appropriately configuring the server via a Client Characteristic Configuration Descriptor. Since the configuration is persistent across a disconnection and reconnection, security requirements must be checked upon a reconnection before sending indications or notifications. When a server reconnects to a client to send an indication or notification for which security is required, the server shall initiate or request encryption with the client prior to sending an indication or notification. If the client does not have an LTK indicating that the client has lost the bond, enabling encryption will fail.

10.3.1.2 Cross-transport Key Generation

After encryption is enabled, the correct security level has been achieved, and both devices support cross-transport key generation, the both devices may perform BR/EDR link key derivation.

10.3.2 Initiating a Service Request

In this section the local device is the device initiating a service request to a remote device. In the L2CAP protocol the local device sends the connection request and the remote device sends the connection response. In GATT, the local device is the GATT client and the remote device is the GATT server.

When a local device initiates a service request to a remote device it shall behave according to the following rules:

- The local device’s security database specifies the security required to initiate a service request. If no encryption is required by the local device then the service request may proceed without encryption or pairing.
- If an LTK is not available but encryption is required, the pairing procedure shall be initiated with the local device’s required authentication settings. If the pairing procedure fails then the service request shall be aborted.

Note: When encryption is not enabled, the error code “Insufficient Authentication” does not indicate to the local device that MITM protection is required.

Note: If the local device is a Peripheral then it may send a Slave Initiated Security Request as defined in [\[Vol. 3\], Part H Section 2.4.6](#).

- If pairing has occurred but the encryption key size is insufficient the pairing procedure shall be executed with the required encryption key size. If the pairing procedure fails then the service request shall be aborted.
- If an LTK is available and encryption is required (LE security mode 1) then encryption shall be enabled before the service request proceeds as defined in [Section 10.6](#). Once encryption is enabled the service request shall proceed. If encryption fails either the bond no longer exists on the remote device, or the wrong device has been connected. The local device must,

after user interaction to confirm the remote device, re-bond, perform service discovery and re-configure the remote device. If the local device had previously determined that the remote device did not have the «Service Changed» characteristic then service discovery may be skipped.

- If an authenticated pairing is required but only an unauthenticated pairing has occurred and the link is currently encrypted, the pairing procedure should be executed with the required authentication settings. If the pairing procedure fails, or an authenticated pairing cannot be performed with the IO capabilities of the local device and remote device, then the service request shall be aborted.

When a bond has been created between two devices, any reconnection should result in the local device enabling or requesting encryption with the remote device before initiating any service request.

If a local device does not enable encryption before initiating a service request and relies on the error codes to determine the security requirements, the local device shall not request pairing with MITM protection in response to receiving an “Insufficient Authentication” error code from the remote device while the link is unencrypted. The local device shall only set the MITM protection required flag if the local device itself requires MITM protection.

- If encryption is not enabled at the time of the service request, the error code “Insufficient Authentication” is received, and the local device currently has an LTK, then the encryption procedure should be started (see [Section 10.6](#)). If this fails (likely indicating that the remote device has lost the bond and no longer has the LTK) or the local device does not have the correct LTK, then the pairing procedure should be started. IO capabilities are exchanged in pairing phase 1, (see [\[Vol. 3\], Part H Section 2.1](#)) and the security level shall be determined by the device’s IO capabilities and MITM requirements.
- If encryption is not enabled at the time of the service request, the error code “Insufficient Encryption” is received, and the local device currently has an LTK, then the encryption procedure shall be started (see [Section 10.6](#)). If starting encryption fails (likely indicating that the remote device has lost the bond and no longer has the LTK) or the local device does not have the correct LTK, then the pairing procedure should be started.
- If LE Secure Connections authenticated pairing is required but the remote device does not support LE Secure Connections, then the service request shall be aborted.

Figure 10.2 shows how a client issues a service request.

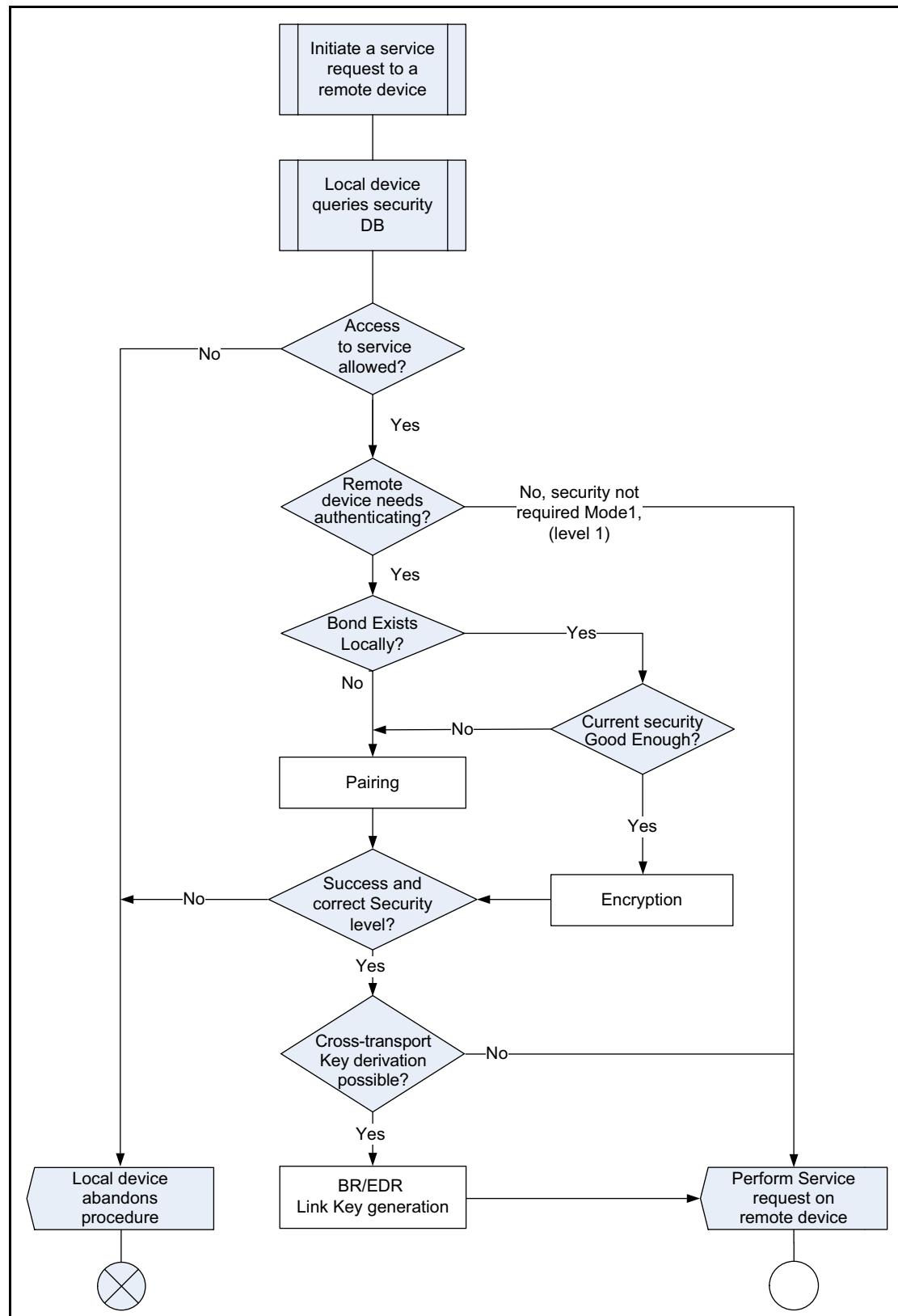


Figure 10.2: Flow chart for a local device issuing a service request to a remote device

10.3.2.1 Cross-transport Key Generation

After encryption is enabled, the correct security level has been achieved, and both devices support cross-transport key generation, the both devices may perform BR/EDR link key derivation.

10.4 DATA SIGNING

The data signing is used for transferring authenticated data between two devices in an unencrypted connection. The data signing method is used by services that require fast connection set up and fast data transfer.

If a service request specifies LE security mode 2, the connection data signing procedure shall be used.

10.4.1 Connection Data Signing Procedure

A device shall generate a new Connection Signature Resolving Key CSRK for each peer device to which it sends signed data in a connection. CSRK is defined in [\[Vol. 3\], Part H Section 2.4.2.2](#).

The data shall be formatted using the Signing Algorithm as defined in [\[Vol. 3\], Part H Section 2.4.5](#) where m is the Data PDU to be signed, k is the CSRK and the SignCounter is the counter value. A Signature is composed of the counter value and the Message Authentication Code (MAC) generated by the Signing Algorithm. The counter value shall be incremented by one for each new Data PDU sent.

The format of signed data is shown in [Figure 10.3](#).

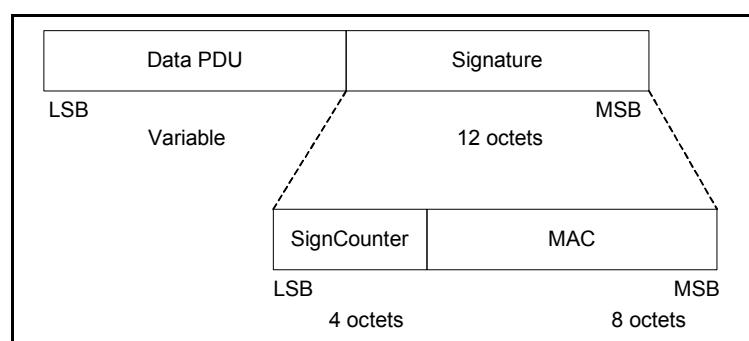


Figure 10.3: Generic format of signed data

10.4.2 Authenticate Signed Data Procedure

If encryption is not required and CSRK is available (LE security mode 2) then the data signing procedure shall be used when making a service request involving a write operation.

Note: The existence of the bond on the server can be determined by successfully enabling encryption with the server using the encryption procedure defined in Section 10.6. A higher layer profile may allow a client to not perform the authentication procedure. Alternatively, a higher layer protocol may signal the client that the signature check has failed due to a lost bond, and the client may then take action to notify the user or attempt to pair again to reestablish the bond.

A device receiving signed data shall authenticate it by performing the Signing Algorithm. The signed data shall be authenticated by performing the Signing Algorithm where m is the Data PDU to be authenticated, k is the stored CSRK and the SignCounter is the received counter value. If the MAC computed by the Signing Algorithm does not match the received MAC, the verification fails and the Host shall ignore the received Data PDU.

Note that since the server does not respond to a Signed Write Command, the higher layer application may not be notified of the ignored request. Hence, it is recommended that the server disconnect the link in case the client is a malicious device attempting to mount a security attack.

If the server has no stored CSRK upon receiving a signed write command, it shall ignore the received Data PDU. Note that since the server does not respond to a Signed Write Command, the higher layer application may not be notified of the ignored request. Although the disconnection may be an adequate indication to the user that the devices need to be paired, it is recommended that implementers consider providing a more informative indication to the user that the devices need to be paired to establish a CSRK.

If the server receives a request from a client for a write operation that requires a response (i.e. other than a Signed Write Command or Write Command), and encryption is not enabled, then the server shall respond with the error code “Insufficient Authentication”.

If the link is encrypted and the server receives a request from a client for which the server requires data signing but does not require encryption, then the server shall complete the request if it is otherwise valid as the encrypted state of the link is considered to satisfy the signing requirement.

As required by [Section 10.2.2](#), for a given link, signed data is not used at the same time as encryption. Therefore, if the client wishes to test that the server is still bonded, and thus enables encryption, further data transfer must occur without signing, assuming the server does not disconnect the link as recommended above.

If a higher layer determines the bond no longer exists on the server, the client must, after user interaction to confirm the remote device, re-bond, perform service discovery and re-configure the server. If the client had previously determined that the server did not have the «Service Changed» characteristic then service discovery may be skipped.

The receiving device shall protect against a replay attack by comparing the received SignCounter with previously received SignCounter from the same peer device. If the SignCounter was previously used then the receiving device shall ignore the Data PDU.

10.5 AUTHORIZATION PROCEDURE

A service may require authorization before allowing access. Authorization is a confirmation by the user to continue with the procedure. Authentication does not necessarily provide authorization. Authorization may be granted by user confirmation after successful authentication.

10.6 ENCRYPTION PROCEDURE

A Central may encrypt a connection using the Encryption Session Setup as defined in [\[Vol. 3\], Part H Section 2.4.4](#) to provide integrity and confidentiality.

A Peripheral may encrypt a connection using the Slave Initiated Security Request as defined in [\[Vol. 3\], Part H Section 2.4.6](#) to provide integrity and confidentiality.

If the encryption procedure fails and either the Central or Peripheral used a Resolvable Private Address for the connection establishment, then the current Resolvable Private Address(es) shall be immediately discarded and new Resolvable Private Address(es) shall be generated.

10.7 PRIVACY FEATURE

The privacy feature provides a level of privacy which makes it more difficult for an attacker to track a device over a period of time. The requirements for a device to support the privacy feature are defined in [Table 10.3](#).

Privacy Requirements	Ref.	Broadcaster	Observer	Peripheral	Central
Privacy feature	10.7	O	O	O	O
Non-resolvable private address generation procedure	10.8.2.1	C2	C4	O	O
Resolvable private address generation procedure	10.8.2.2	C3	C5	C1	C1
Resolvable private address resolution procedure	10.8.2.3	E	O	C1	C1
Bondable Mode	9.4.3	E	E	C1	C1
Bonding procedure	9.4.4	E	E	C1	C1
C1: Mandatory if privacy feature is supported, else optional C2: Mandatory if privacy feature is supported and resolvable private address generation procedure is not supported, else optional C3: Mandatory if privacy feature is supported and non-resolvable private address generation procedure is not supported, else optional C4: Mandatory if privacy feature and active scanning are supported and resolvable private address generation procedure is not supported, else optional C5: Mandatory if privacy feature and active scanning are supported and non-resolvable private address generation procedure is not supported, else optional					

Table 10.3: Requirements related to privacy feature

If a device, i.e. host and controller, claims support for the privacy feature, the requirements in this section shall be met.

The Host may maintain a resolving list by adding and removing device identities. A device identity consists of the peer's identity address, and a local and peer's IRK pair. The local or peer's IRK shall be an all-zero key, if not applicable for the particular device identity.

If a peer device provides an all-zero identity address during pairing, the Host shall choose a unique identifier to substitute the peer's device identity address. The Host shall ensure that all identities provided to the controller are unique.

When address resolution is enabled in the Controller, all references to peer devices that are included in the resolving list from Host to the Controller shall be done using the peer's device identity address. Likewise, all incoming events from the Controller to the Host will use the peer's device identity, if the peer's device address has been resolved.

10.7.1 Privacy Feature in a Peripheral

The privacy-enabled Peripheral shall use a resolvable private address as the advertiser's device address when in connectable mode.

A Peripheral shall use non-resolvable or resolvable private addresses when in non-connectable mode as defined in [Section 9.3.2](#).

If a privacy-enabled Peripheral, that has a stored bond, receives a resolvable private address, the Host may resolve the resolvable private address by performing the 'resolvable private address resolution procedure' as defined in [Section 10.8.2.3](#). If the resolution is successful, the Host may accept the connection. If the resolution procedure fails, then the Host shall disconnect with the error code "Authentication failure", or perform the pairing procedure, or perform the authentication procedure as defined in [Section 10.3](#).

The device should not send the device name or unique data in the advertising data that can be used to recognize the device.

10.7.1.1 Privacy Feature in a Peripheral with Controller-based privacy

A privacy-enabled Peripheral shall use either the undirected connectable mode as defined in [Section 9.3.4](#) or directed connectable mode as defined in [Section 9.3.3](#). The directed connectable mode shall only be used if the peer device supports Address Resolution in the Controller.

The Host shall enable resolvable private address generation by enabling it in the controller and populating the resolving list.

10.7.1.2 Privacy Feature in a Peripheral with Host-based privacy

A privacy-enabled Peripheral should use the undirected connectable mode as defined in Section 9.3.4, to create a connection.

The Host shall generate a resolvable private address using the 'resolvable private address generation procedure' as defined in [Section 10.8.2.2](#) or non-resolvable private address procedure as defined in [Section 10.8.2.1](#). The Host shall set a timer equal to $T_{GAP}(\text{private_addr_int})$. The Host shall generate a new resolvable private address or non-resolvable private address when the timer $T_{GAP}(\text{private_addr_int})$ expires

Note: $T_{GAP}(\text{private_addr_int})$ timer may not be run if a Peripheral is not advertising.

10.7.2 Privacy Feature in a Central

The Central shall use a resolvable private address as the initiator's device address.

During active scanning, a privacy enabled Central shall use a non-resolvable or resolvable private address.

If, a privacy-enabled Central, that has a stored bond, receives a resolvable private address, the Host may resolve the resolvable private address by performing the "resolvable private address resolution procedure" as defined in [Section 10.8.2.3](#). If the resolution is successful, the Host may accept this and future connection attempts from this device; otherwise, the Host will ignore the remote device.

10.7.2.1 Privacy Feature in a Central with Controller-based privacy

A privacy-enabled Central with Address Resolution enabled in the Controller can use any of the connection establishment procedures defined in [Section 9.3](#).

10.7.2.2 Privacy Feature in a Central with Host-based privacy

A privacy-enabled Central should use the general connection establishment procedure defined in [Section 9.3.6](#) to create a connection.

The Host shall generate a resolvable private address using the 'resolvable private address generation procedure' as defined in [Section 10.8.2.2](#) or non-resolvable private address procedure as defined in [Section 10.8.2.1](#). The Host shall set a timer equal to $T_{GAP}(\text{private_addr_int})$. The Host shall generate a new resolvable private address or non-resolvable private address when the timer $T_{GAP}(\text{private_addr_int})$ expires.

Note: $T_{GAP}(\text{private_addr_int})$ timer may not be run if a Central is not scanning or connected.

10.7.3 Privacy Feature in a Broadcaster

A privacy-enabled Broadcaster shall use the Broadcast mode defined in [Section 9.1.1](#). The Broadcaster shall use either a resolvable private address or non-resolvable private address.

If Address Resolution is not supported or disabled in the Controller, the following applies to the Host: The Host shall generate a resolvable private address using the 'resolvable private address generation procedure' as defined in [Section 10.8.2.2](#) or non-resolvable private address procedure as defined in [Section 10.8.2.1](#). The Host shall set a timer to $T_{GAP}(\text{private_addr_int})$. The Host shall generate a new resolvable private address or non-resolvable private address when the timer $T_{GAP}(\text{private_addr_int})$ expires.

Note: $T_{GAP}(\text{private_addr_int})$ timer may not be run if a Broadcaster is not advertising.

The device should not send the device name or unique data in the advertising data which can be used to recognize the device.

10.7.4 Privacy Feature in an Observer

A privacy-enabled Observer shall use the observation procedure defined in [Section 9.1.2](#). During active scanning, a privacy enabled Observer shall use either a resolvable private address or non-resolvable private address.

If Address Resolution is not supported or disabled in the Controller, the following applies to the Host: The Host shall generate a resolvable private address using the ‘resolvable private address generation procedure’ as defined in [Section 10.8.2.2](#) or non-resolvable private address procedure as defined in [Section 10.8.2.1](#). The Host shall set a timer equal to $T_{GAP}(\text{private_addr_int})$. The Host shall generate a new resolvable private address or non-resolvable private address when the timer $T_{GAP}(\text{private_addr_int})$ expires.

Note: $T_{GAP}(\text{private_addr_int})$ timer may not be run if an Observer is not scanning.

10.8 RANDOM DEVICE ADDRESS

For the purposes of this profile, the random device address may be of either of the following two sub-types:

- Static address
- Private address.

The term random device address refers to both static and private address types.

The transmission of a random device address is optional. A device shall accept the reception of a random device address from a remote device.

The private address may be of either of the following two sub-types:

- Non-resolvable private address
- Resolvable private address

A bonded device shall process a resolvable private address as defined in [Section 10.8.2.3](#) or by establishing a connection and then performing the authentication procedure as defined in [Section 10.3](#). A device that uses resolvable private address shall always request to distribute its IRK value as defined in [\[Vol. 3\], Part H Section 3.6.4](#) if both sides are bondable, unless keys have been pre-distributed. If the request to distribute the IRK fails then the peer

device may authenticate re-connections using the authentication procedure as defined in [Section 10.3](#) or terminate the current connection.

10.8.1 Static Address

The Host can generate a static address using the procedure described in [\[Vol 6\] Part B, Section 1.3.2.1](#).

10.8.2 Private address

The private address may be of either of the following two sub-types:

- Non-resolvable private address
- Resolvable private address

10.8.2.1 Non-Resolvable Private Address Generation Procedure

The Host can generate a non resolvable private address using the procedure described in [\[Vol 6\] Part B, Section 1.3.2.2](#).

10.8.2.2 Resolvable Private Address Generation Procedure

The Host can generate a resolvable private address where the Host has its IRK using the procedure described in [\[Vol 6\] Part B, Section 1.3.2.2](#).

10.8.2.3 Resolvable Private Address Resolution Procedure

The Host can resolve a resolvable private address where the Host has the peer device's IRK or the local device's IRK, using the procedure described in [\[Vol 6\] Part B, Section 1.3.2.3](#).

11 ADVERTISING AND SCAN RESPONSE DATA FORMAT

The format of Advertising data and Scan Response data is shown in [Figure 11.1](#). The data consists of a significant part and a non-significant part. The significant part contains a sequence of AD structures. Each AD structure shall have a Length field of one octet, which contains the Length value, and a Data field of Length octets. The first octet of the Data field contains the AD type field. The content of the remaining Length - 1 octet in the Data field depends on the value of the AD type field and is called the AD data. The non-significant part extends the Advertising and Scan Response data to 31 octets and shall contain all-zero octets.

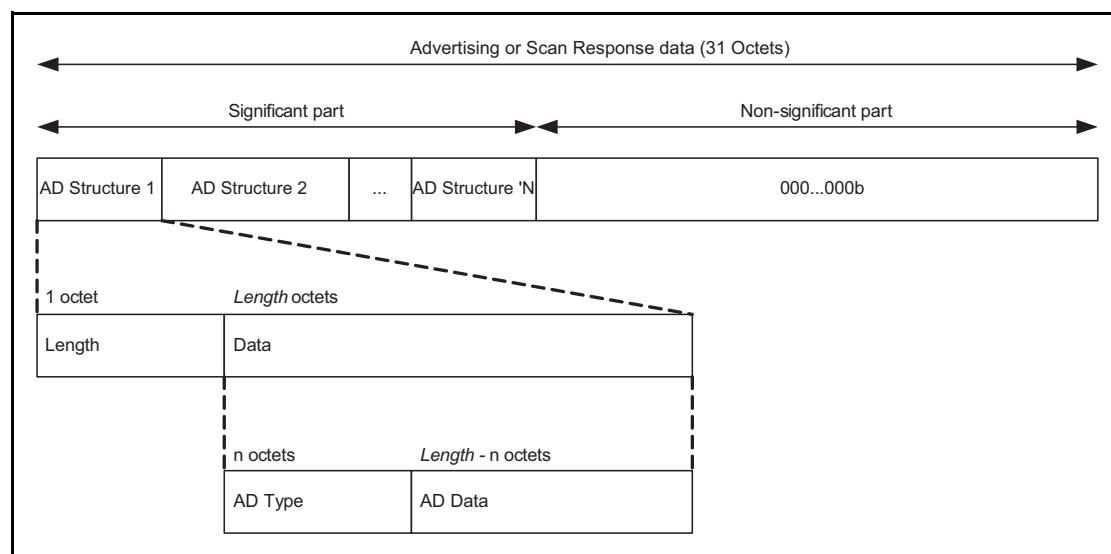


Figure 11.1: Advertising and Scan Response data format

If the Length field is set to zero, then the Data field has zero octets. This shall only occur to allow an early termination of the Advertising or Scan Response data.

Only the significant part of the Advertising or Scan Response data needs to be sent over the air.

The Advertising and Scan Response data is sent in advertising events. The Advertising Data is placed in the AdvData field of ADV_IND, ADV_NONCONN_IND, and ADV_SCAN_IND packets. The Scan Response data is sent in the ScanRspData field of SCAN_RSP packets.

The AD type data formats and meanings are defined in [[Core Specification Supplement](#)], Part A. The AD type identifier values are defined in the [Assigned Numbers](#) document.

12 GAP SERVICE AND CHARACTERISTICS FOR GATT SERVER

The GATT server shall contain the GAP service for devices supporting the Central and Peripheral GAP roles and BR/EDR GAP role for BR/EDR/LE type devices. The GATT Server may contain the GAP service for BR/EDR type devices supporting the BR/EDR GAP role. A device shall have only one instance of the GAP service in the GATT server. The GAP service is a GATT based service with the service UUID as «GAP Service» defined in the Bluetooth [Assigned Numbers](#) document.

	BR/EDR GAP Role	LE Broadcaster	LE Observer	LE Peripheral	LE Central
GAP Service	C1	E	E	M	M
C1: Mandatory for BR/EDR/LE type devices, else optional					

Table 12.1: GAP Service Requirements

The characteristics requirements for the GAP service in each of the GAP roles are shown in [Table 12.2](#).

Characteristics	Ref.	BR/EDR GAP Role	LE Broadcaster	LE Observer	LE Peripheral	LE Central
Device Name	12.1	C1	E	E	M	M
Appearance	12.2	C1	E	E	M	M
Peripheral Preferred Connection Parameters	12.3	O	E	E	O	E
Central Address Resolution	12.4	O	E	E	C2	C2
C1: Mandatory for BR/EDR/LE type devices, else optional						
C2: Mandatory if Link Layer Privacy is supported, otherwise excluded						

Table 12.2: Requirements related to GAP Service characteristics

A device that supports multiple GAP roles contains all the characteristics to meet the requirements for the supported roles. The device must continue to expose the characteristics when the device is operating in the role in which the characteristics are not valid.

12.1 DEVICE NAME CHARACTERISTIC

The Device Name characteristic shall contain the name of the device as an UTF-8 string as defined in [Section 3.2.2](#). When the device is discoverable, the Device Name characteristic value shall be readable without authentication or authorization. When the device is not discoverable, the Device Name Characteristic should not be readable without authentication or authorization. The Device Name characteristic value may be writable. If writable, authentication and authorization may be defined by a higher layer specification or be implementation specific.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xMMMM	0x2A00 – UUID for «Device Name»	Device Name	Readable without authentication or authorization when discoverable. Optionally writable, authentication and authorization may be defined by a higher layer specification or be implementation specific.

Table 12.3: Device Name characteristic

The Device Name characteristic value shall be 0 to 248 octets in length. A device shall have only one instance of the Device Name characteristic.

12.2 APPEARANCE CHARACTERISTIC

The Appearance characteristic defines the representation of the external appearance of the device. This enables the discovering device to represent the device to the user using an icon, or a string, or similar. The Appearance characteristic value shall be readable without authentication or authorization. The Appearance characteristic value may be writable. If writable, authentication and authorization may be defined by a higher layer specification or be implementation specific.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xMMMM	0x2A01 – UUID for «Appearance»	Appearance	Readable without authentication or authorization. Optionally writable, authentication and authorization may be defined by a higher layer specification or be implementation specific.

Table 12.4: Appearance characteristic

The Appearance characteristic value shall be the enumerated value as defined by Bluetooth [Assigned Numbers](#) document. The Appearance characteristic value shall be 2 octets in length. A device shall have only one instance of the Appearance characteristic.

12.3 PERIPHERAL PREFERRED CONNECTION PARAMETERS CHARACTERISTIC

The Peripheral Preferred Connection Parameters (PPCP) characteristic contains the preferred connection parameters of the Peripheral.

The Peripheral Preferred Connection Parameters characteristic value shall be readable. Authentication and authorization may be defined by a higher layer specification or be implementation specific.

The Peripheral Preferred Connection Parameters characteristic value shall not be writable.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xMMMM	0x2A04 – UUID for «Peripheral Preferred Connection Parameters»	Peripheral Preferred Connection Parameter	Readable, authentication and authorization may be defined by a higher layer specification or be implementation specific Not writable

Table 12.5: Peripheral Preferred Connection Parameters characteristic

The Peripheral Preferred Connection Parameters characteristic value shall be 8 octets in length. A device shall have only one instance of the Peripheral Preferred Connection Parameters characteristic.

The preferred connection parameters structured data is defined as follows:

Name	Size (Octet)	Description
Minimum connection interval	2	Defines minimum value for the connection interval in the following manner: $connInterval_{min} = Conn_Interval_Min * 1.25 \text{ ms}$ Conn_Interval_Min range: 0x0006 to 0x0C80 Value of 0xFFFF indicates no specific minimum. Values outside the range are reserved. (excluding 0xFFFF)
Maximum connection interval	2	Defines maximum value for the connection interval in the following manner: $connInterval_{max} = Conn_Interval_Max * 1.25 \text{ ms}$ Conn_Interval_Max range: 0x0006 to 0x0C80 Shall be equal to or greater than the Conn_Interval_Min. Value of 0xFFFF indicates no specific maximum. Values outside the range are reserved. (excluding 0xFFFF)

Table 12.6: Format of the preferred connection parameters structured data

Name	Size (Octet)	Description
Slave latency	2	Defines the slave latency for the connection in number of connection events. Slave latency range: 0x0000 to 0x01F3 Values outside the range are reserved.
Connection Supervision timeout multiplier	2	Defines the connection supervisor timeout multiplier as a multiple of 10ms. Range: 0xFFFF indicates no specific value requested. Range: 0x000A to 0x0C80 Time = N * 10 msec Time Range: 100 msec to 32 seconds Values outside the range are reserved. (excluding 0xFFFF)

Table 12.6: Format of the preferred connection parameters structured data

12.4 CENTRAL ADDRESS RESOLUTION

The Peripheral shall check if the peer device supports address resolution by reading the Central Address Resolution characteristic before using directed advertisement where the initiator address is set to a Resolvable Private Address (RPA).

The Central Address Resolution characteristic defines whether the device supports privacy with address resolution. See [Table 12.7](#).

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xMmmm	0x2AA6 - UUID of <> Central Address Resolution >>	Central Address Resolution Support	Readable without authentication or authorization. Not writable.

Table 12.7: Central Address Resolution Characteristic

The Central Address Resolution characteristic value shall be 1 octet in length:

- 0 = address resolution is not supported in this device
- 1 = address resolution is supported in this device
- 2 – 255 = Reserved

A device shall have only one instance of the Central Address Resolution characteristic. If the Central Address Resolution characteristic is not present, then it shall be assumed that Central Address Resolution is not supported.

13 BR/EDR/LE OPERATION

This section describes the requirements for devices that support the BR/EDR/LE device type. The device may support any LE GAP roles allowed by the Controller over the LE physical channel.

Feature	Ref.	Broadcaster	Observer	Peripheral	Central
BR/EDR/LE modes and procedures	13.1 and 13.2	O	O	O	O

Table 13.1: Requirements for the modes of a BR/EDR/LE device type

A BR/EDR/LE device type that supports the Broadcaster role shall meet the requirements for the Broadcaster role as defined in [Section 9](#).

A BR/EDR/LE device type that supports the Observer role shall meet the requirements for the Observer role as defined in [Section 9](#).

A BR/EDR/LE device type that supports the Peripheral role shall meet the requirements for the Peripheral role as defined in [Section 9](#).

A BR/EDR/LE device type that supports the Central role shall meet the requirements for the Central role as defined in [Section 9](#).

13.1 MODES, PROCEDURES AND SECURITY ASPECTS

All modes, procedures and security aspects shall follow the requirements as specified for the physical transport over which they operate. Sections [4](#), [5](#), [6](#) and [7](#) specify the requirements for the modes, procedures and security aspects for operations performed over the BR/EDR physical transport. Sections [9](#) and [10](#) specify the requirements for the modes, procedures and security aspects performed over the LE physical transport. It is optional to support both physical transports simultaneously to the same remote device.

13.1.1 Discoverable Mode Requirements

A device shall expose the capabilities of both physical transports for both Limited and General Discoverable Mode using the advertisement Flag AD Type as follows:

- a) The 'BR/EDR Not Supported' bit in the Flags AD type shall be set to 0 as defined in [\[Core Specification Supplement\]](#), Part A, Section 1.3.
- b) The 'Simultaneous LE and BR/EDR to Same Device Capable (Controller)' and 'Simultaneous LE and BR/EDR to Same Device Capable (Host)' bits in the Flags AD type shall be set to 0.

The 'LE Supported (Controller)' and 'LE Supported (Host)' bits in the LMP features shall be set as defined in [\[Vol. 2\], Part C Section 3.2](#).

13.2 BONDING FOR BR/EDR/LE DEVICE TYPE

The requirements for a device supporting BR/EDR/LE device type are shown in [Table 13.2](#).

Bonding Requirement	Ref.	Peripheral	Central
Bonding	6.5 / 9.4.4	M	M

Table 13.2: Requirements for the bonding of a BR/EDR/LE device type

If the remote device supports the BR/EDR physical transport, the bonding procedures for the BR/EDR physical transport as defined in [Section 6.5](#) shall be used.

If the remote device supports the LE physical transport, the bonding procedures for the LE physical transport as defined in [Section 9.4.4](#) shall be used.

13.3 RELATIONSHIP BETWEEN PHYSICAL TRANSPORTS

To determine if both the BR/EDR physical transport and the LE physical transport are established to the same peer device, the device shall use either the public address used on the LE advertising channel or the public address from the BD_ADDR field contained in the SMP Identity Address Information packet ([\[Vol. 3\], Part H Section 3.6.5](#)) if it has been received.

14 BR/EDR/LE SECURITY ASPECTS

The requirements for a device supporting BR/EDR/LE device type are shown in [Table 14.1](#).

Security aspects	Ref.	Peripheral	Central
Security aspects	14	M	M

Table 14.1: Requirements for the security aspects of a BR/EDR/LE device type

If the remote device supports the BR/EDR physical transport the security procedures for the BR/EDR physical transport as defined in [Section 5](#) shall be used.

If the remote device supports the LE physical transport the security procedures for the LE physical transport as defined in [Section 10](#) shall be used.

If the LE transport in a BR/EDR/LE device supports Secure Connections, the security procedures in [Section 14.1](#) may also be used.

14.1 CROSS-TRANSPORT KEY DERIVATION

If both the local and remote devices support Secure Connections over the BR/EDR and LE transports, devices may optionally generate keys of identical strength and the same MITM protection for both transports as part of a single pairing procedure.

If both the local and remote devices support Secure Connections over the LE transport but not over the BR/EDR transport, then the devices may optionally generate the BR/EDR keys of identical strength and the same MITM protection as the LE keys as part of the LE pairing procedure.

If Secure Connections pairing occurs first on the LE transport the procedures in [\[Vol 3\] Part H, Section 2.3.5.7](#) may be used.

If Secure Connections pairing occurs first on the BR/EDR transport the procedures in [\[Vol 3\] Part H, Section 2.3.5.7](#) may be used.

14.2 COLLISION HANDLING

If pairing has been initiated by the local device on the BR/EDR transport, and a pairing request is received from the same remote device on the LE transport, the LE pairing shall be rejected with SMP error code “BR/EDR pairing in progress” (0x0D) if both sides support LE Secure Connections.

If a BR/EDR/LE device supports LE Secure Connections, then it shall initiate pairing on only one transport at a time to the same remote device.

15 BLUETOOTH DEVICE REQUIREMENTS

15.1 BLUETOOTH DEVICE ADDRESS

All Bluetooth devices shall have a Bluetooth Device Address (BD_ADDR) that uniquely identifies the device to another Bluetooth device. The specific Bluetooth Device Address requirements depend on the type of Bluetooth device.

15.1.1 Bluetooth Device Address Types

15.1.1.1 Public Bluetooth Address

A Bluetooth public address used as the BD_ADDR for the BR/EDR physical channel is defined in [\[Vol. 2\], Part B Section 1.2](#). A Bluetooth public address used as the BD_ADDR for the LE physical channel is defined in [\[Vol. 6\], Part B Section 1.3](#).

15.1.1.2 Random Bluetooth Address

A random device address used as the BD_ADDR on the LE physical channel is defined in [Section 10.8](#).

15.2 GATT PROFILE REQUIREMENTS

A BR/EDR or BR/EDR/LE device that supports the GATT profile on the BR/EDR physical transport shall implement the GATT server. A BR/EDR/LE or LE-only device that supports the LE Central and/or Peripheral roles shall implement the GATT server.

	BR/EDR GAP Role	LE Broadcaster	LE Observer	LE Peripheral	LE Central
GATT Client	O	E	E	O	O
GATT Server	C1	E	E	M	M
C1: Mandatory if the GATT profile is supported on the BR/EDR physical transport; otherwise excluded					

Table 15.1: Requirements based on GAP Roles Supported

15.3 SDP REQUIREMENTS

BR/EDR and BR/EDR/LE devices shall implement either an SDP server or an SDP client. A BR/EDR or BR/EDR/LE device that supports a GATT server shall implement an SDP server. BR/EDR and BR/EDR/LE devices may implement both an SDP server and an SDP client. There shall be no more than one active SDP server per device. SDP client and server are excluded for the LE-only device type.

	BR/EDR and BR/EDR/LE Device Type	LE-Only Device Type
SDP Client	C1	E
SDP Server	C1 or C2	E
C1: Mandatory to support at least one; Optional to support both		
C2: Mandatory to support if GATT server is supported		

Table 15.2: Requirements based on GAP Roles Supported

15.4 SDP SERVICE RECORD REQUIREMENT

A BR/EDR or BR/EDR/LE device that supports a GATT server accessible over the BR/EDR physical transport shall publish the SDP record shown below in [Table 15.3](#). The GAP Start Handle shall be set to the attribute handle of the «Generic Attribute Profile» service declaration. The GAP End Handle shall be set to the attribute handle of the last attribute within the «Generic Attribute Profile» service definition group.

Item	Definition	Type	Value	Status
Service Class ID List				M
Service Class #0		UUID	«Generic Attribute Profile»	M
ProtocolDescriptorList				M
Protocol #0		UUID	«L2CAP»	M
Parameter #0 for Protocol #0	PSM	Unit16	PSM = ATT	M
Protocol #1		UUID	«ATT»	M
Parameter #0 for Protocol #1	GAP Start Handle	Uint16		M
Parameter #1 for Protocol #1	GAP End Handle	Uint16		M
BrowseGroupList			PublicBrowseRoot	M

Table 15.3: SDP Record for the Generic Access Profile

If a BR/EDR or BR/EDR/LE device supports a GATT-based service on the BR/EDR transport, the service shall exist in the SDP server and the GATT server.

16 DEFINITIONS

In the following, terms written with capital letters refer to states.

16.1 GENERAL DEFINITIONS

Mode. A set of directives that defines how a device will respond to certain events.

Idle. As seen from a remote device, a Bluetooth device is idle, or is in idle mode, when there is no link established between them.

Bond. A relation between two Bluetooth devices defined by creating, exchanging and storing a common link key. The bond is created through the bonding or LMP-pairing procedures.

16.2 CONNECTION-RELATED DEFINITIONS

Physical channel. A synchronized Bluetooth baseband-compliant RF hopping sequence.

Piconet. A set of Bluetooth devices sharing the same physical channel defined by the master parameters (clock and BD_ADDR).

Physical link. A Baseband-level connection¹ between two devices established using paging. A physical link comprises a sequence of transmission slots on a physical channel alternating between master and slave transmission slots.

ACL link. An asynchronous (packet-switched) connection¹ between two devices created on LMP level. Traffic on an ACL link uses ACL packets to be transmitted.

SCO link. A synchronous (circuit-switched) connection¹ for reserved bandwidth communications; e.g. voice between two devices, created on the LMP level by reserving slots periodically on a physical channel. Traffic on an SCO link uses SCO packets to be transmitted. SCO links can be established only after an ACL link has first been established.

Link. Shorthand for an ACL link.

PAGE. A baseband state where a device transmits page trains, and processes any eventual responses to the page trains.

PAGE_SCAN. A baseband state where a device listens for page trains.

1. The term 'connection' used here is not identical to the definition below. It is used in the absence of a more concise term.

Page . The transmission by a device of page trains containing the Device Access Code of the device to which the physical link is requested.

Page scan . The listening by a device for page trains containing its own Device Access Code.

Channel . A logical connection on L2CAP level between two devices serving a single application or higher layer protocol.

Connection . A connection between two peer applications or higher layer protocols mapped onto a channel.

Connecting . A phase in the communication between devices when a connection between them is being established. (Connecting phase follows after the link establishment phase is completed.)

Connect (to service) . The establishment of a connection to a service. If not already done, this includes establishment of a physical link, link and channel as well.

16.3 DEVICE-RELATED DEFINITIONS

Discoverable device . A Bluetooth device in range that will respond to an inquiry (normally in addition to responding to page).

Silent device . A Bluetooth device appears as silent to a remote device if it does not respond to inquiries made by the remote device. A device may be silent due to being non-discoverable or due to baseband congestion while being discoverable.

Connectable device . A Bluetooth device in range that will respond to a page.

Trusted device . A paired device that is explicitly marked as trusted.

Paired device . A Bluetooth device with which a link key has been exchanged (either before connection establishment was requested or during connecting phase).

Pre-paired device . A Bluetooth device with which a link key was exchanged, and the link key is stored, before link establishment.

Un-paired device . A Bluetooth device for which there was no exchanged link key available before connection establishment was requested.

Known device . A Bluetooth device for which at least the BD_ADDR is stored.

Un-known device . A Bluetooth device for which no information (BD_ADDR, link key or other) is stored.

Authenticated device . A Bluetooth device whose identity has been verified during the lifetime of the current link, based on the authentication procedure.

16.4 PROCEDURE-RELATED DEFINITIONS

Paging. A procedure for establishing a physical link of ACL type on baseband level, consisting of a page action of the initiator and a page scan action of the responding device.

Link establishment. A procedure for establishing a link on LMP level. A link is established when both devices have agreed that LMP setup is completed.

Channel establishment. A procedure for establishing a channel on L2CAP level.

Connection establishment. A procedure for creating a connection mapped onto a channel.

Creation of a trusted relationship. A procedure where the remote device is marked as a trusted device. This includes storing a common link key for future authentication and pairing (if the link key is not available).

Creation of a secure connection. A procedure of establishing a connection, including authentication and encryption.

Device discovery. A procedure for retrieving the Bluetooth device address, clock, class-of-device field and used page scan mode from discoverable devices.

Name discovery. A procedure for retrieving the user-friendly name (the Bluetooth device name) of a connectable device.

Service discovery. Procedures for querying and browsing for services offered by or through another Bluetooth device.

16.5 SECURITY-RELATED DEFINITIONS

Authentication. A generic procedure based on LMP-authentication if a link key exists or on LMP-pairing if no link key exists.

LMP-authentication. An LMP level procedure for verifying the identity of a remote device. The procedure is based on a challenge-response mechanism using a random number, a secret key and the BD_ADDR of the non-initiating device. The secret key used can be a previously exchanged link key.

Authorization. A procedure where a user of a Bluetooth device grants a specific (remote) Bluetooth device access to a specific service. Authorization implies that the identity of the remote device can be verified through authentication.

Authorize. The act of granting a specific Bluetooth device access to a specific service. It may be based upon user confirmation, or given the existence of a trusted relationship.

LMP-pairing . A procedure that authenticates two devices, based on a PIN, and subsequently creates a common link key that can be used as a basis for a trusted relationship or a (single) secure connection. The procedure consists of the steps: creation of an initialization key (based on a random number and a PIN), creation and exchange of a common link key and LMP-authentication based on the common link key.

Bonding . A dedicated procedure for performing the first authentication, where a common link key is created and stored for future use.

Trusting . The marking of a paired device as trusted. Trust marking can be done by the user, or automatically by the device (e.g. when in bondable mode) after a successful pairing.

17 REFERENCES

- [1] Baseband Specification
- [2] Link Manager Protocol
- [3] RFCOMM
- [4] Telephony Control Specification
- [5] Service Discovery Protocol
- [6] Security Architecture (white paper)
- [7] Bluetooth [Assigned Numbers](#)

APPENDIX A (NORMATIVE): TIMERS AND CONSTANTS

The following timers are required by this profile.

Timer name	Value	Description	Requirement or Recommendation
T _{GAP(100)}	10.24 s	Time span that a Bluetooth device performs device discovery.	Recommended value
T _{GAP(101)}	10.625 ms	A discoverable Bluetooth device enters INQUIRY_SCAN for at least TGAP(101) every TGAP(102).	Required value
T _{GAP(102)}	2.56 s	Maximum time between repeated INQUIRY_SCAN enterings.	Recommended value
T _{GAP(103)}	30.72 s	Minimum time span that a device is in discoverable mode.	Required value
T _{GAP(104)}	1 min.	Maximum time span that a device is in limited discoverable mode.	Recommended value
T _{GAP(105)}	100ms	Maximum time between INQUIRY_SCAN enterings	Recommended value
T _{GAP(106)}	100ms	Maximum time between PAGE_SCAN enterings	Recommended value
T _{GAP(107)}	1.28s	Maximum time between PAGE_SCAN enterings (R1 page scan)	Recommended value
T _{GAP(108)}	2.56s	Maximum time between PAGE_SCAN enterings (R2 page scan)	Recommended value
T _{GAP(lim_adv_timeout)}	180 s	Maximum time to remain advertising when in the limited discoverable mode	Required value
T _{GAP(gen_disc_scan_min)}	10.24 s	Minimum time to perform scanning when performing the general discovery procedure	Recommended value

Table A.1: Defined GAP timers (Sheet 1 of 4)

Generic Access Profile



Timer name	Value	Description	Requirement or Recommendation
T _{GAP} (lim_disc_scan_min)	10.24 s	Minimum time to perform scanning when performing the limited discovery procedure	Recommended value
T _{GAP} (lim_disc_scan_int)	11.25 ms	Scan interval used in the limited discovery procedure	Recommended value
T _{GAP} (conn_param_timeout)	30 s	Connection parameter update notification timer when performing the connection parameter update procedure	Recommended value
T _{GAP} (scan_fast_period)	30.72 s	Minimum time to perform scanning when user initiated	Recommended value
T _{GAP} (scan_fast_interval)	30 ms to 60 ms	Scan interval in any discovery or connection establishment procedure when user initiated	Recommended value
T _{GAP} (scan_fast_window)	30 ms	Scan window in any discovery or connection establishment procedure when user initiated	Recommended value
T _{GAP} (scan_slow_interval1)	1.28 s	Scan interval in any discovery or connection establishment procedure when background scanning	Recommended value
T _{GAP} (scan_slow_window1)	11.25 ms	Scan window in any discovery or connection establishment procedure when background scanning	Recommended value
T _{GAP} (scan_slow_interval2)	2.56 s	Scan interval in any discovery or connection establishment procedure when background scanning	Recommended value
T _{GAP} (scan_slow_window2)	22.5 ms	Scan window in any discovery or connection establishment procedure when background scanning	Recommended value

Table A.1: Defined GAP timers (Sheet 2 of 4)

Generic Access Profile



Timer name	Value	Description	Requirement or Recommendation
$T_{GAP}(\text{adv_fast_period})$	30 s	Minimum time to perform advertising when user initiated	Recommended value
$T_{GAP}(\text{adv_fast_interval1})$	30 ms to 60 ms	Minimum to maximum advertising interval in the following GAP Modes when user initiated: <ol style="list-style-type: none"> 1. Undirected Connectable Mode 2. Limited Discoverable Mode and sending connectable undirected advertising events 3. General Discoverable Mode and sending connectable undirected advertising events 4. Directed Connectable Mode and sending low duty cycle directed advertising events 	Recommended value
$T_{GAP}(\text{adv_fast_interval2})$	100 ms to 150 ms	Minimum to maximum advertising interval in the following GAP Modes when user initiated and sending either non-connectable advertising events or scannable undirected advertising events: <ol style="list-style-type: none"> 1. Non-Discoverable Mode 2. Non-Connectable Mode 3. Limited Discoverable Mode 4. General Discoverable Mode 	Recommended value
$T_{GAP}(\text{adv_slow_interval})$	1 s to 1.2 s	Minimum to maximum advertisement interval in any discoverable or connectable mode when background advertising	Recommended value
$T_{GAP}(\text{initial_conn_interval})$	30 ms to 50 ms	Minimum to maximum connection interval upon any connection establishment	Recommended value
$T_{GAP}(\text{conn_pause_central})$	1 s	Central idle timer	Recommended value

Table A.1: Defined GAP timers (Sheet 3 of 4)

Timer name	Value	Description	Requirement or Recommendation
T _{GAP} (conn_pause_peripheral)	5 s	Minimum time upon connection establishment before the peripheral starts a connection update procedure	Recommended value
T _{GAP} (private_addr_int)	15 mins	Minimum time interval between private address change	Recommended value
T _{GAP} (conn_param_timeout)	30 s	Timer used in connection parameter update procedure	Recommended value
T _{GAP} (Sync_Train_Interval)	80 ms	Interval between Synchronization Train events	Recommended value
T _{GAP} (Sync_Train_Duration)	30.72 s	Duration of synchronizability mode	Required value
T _{GAP} (Sync_Scan_Interval)	320 ms	Interval between the start of adjacent synchronization scan windows	Recommended value
T _{GAP} (Sync_Scan_Window)	91.25 ms	Duration of Synchronization scan window	Recommended value

Table A.1: Defined GAP timers (Sheet 4 of 4)

APPENDIX B (INFORMATIVE): INFORMATION FLOWS OF RELATED PROCEDURES

B.1 LMP – AUTHENTICATION

The specification of authentication on link level is found in [\[2\]](#).

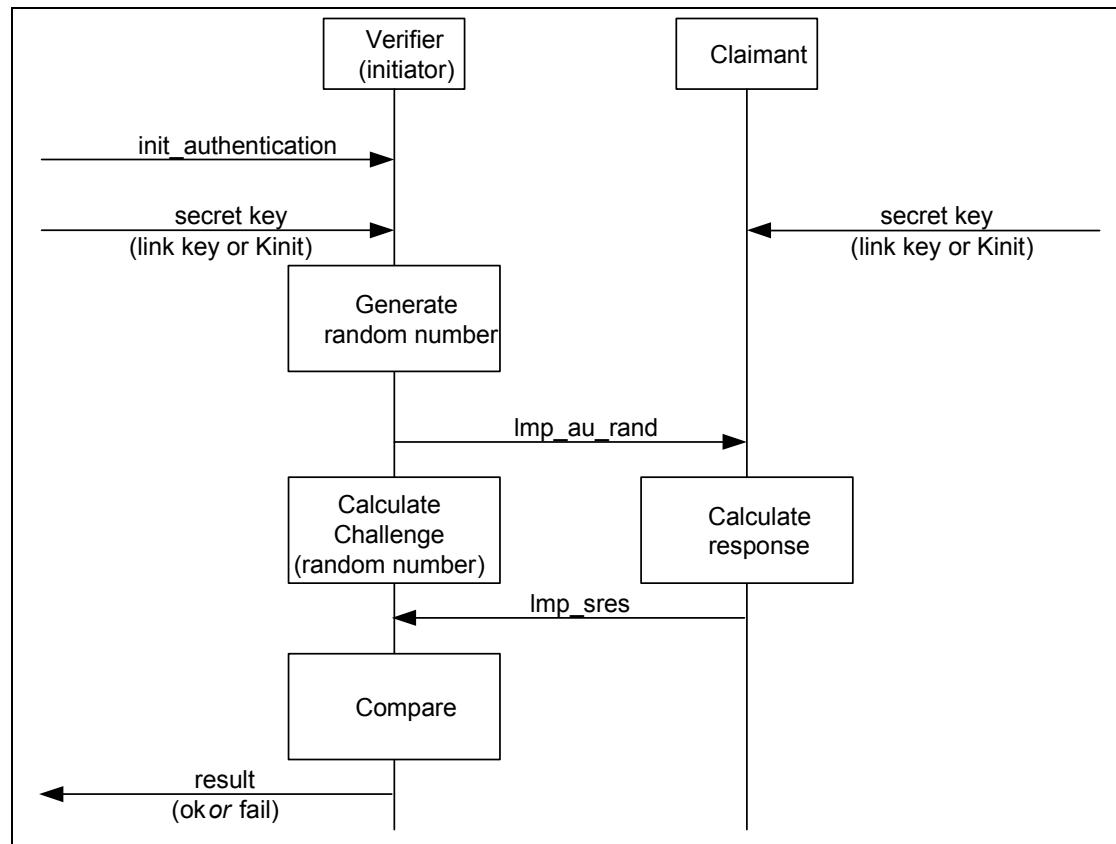


Figure B.1: LMP-authentication as defined by [\[2\]](#)

The secret key used here is an already exchanged link key.

B.2 LMP – PAIRING

The specification of pairing on link level is found in [\[2\]](#).

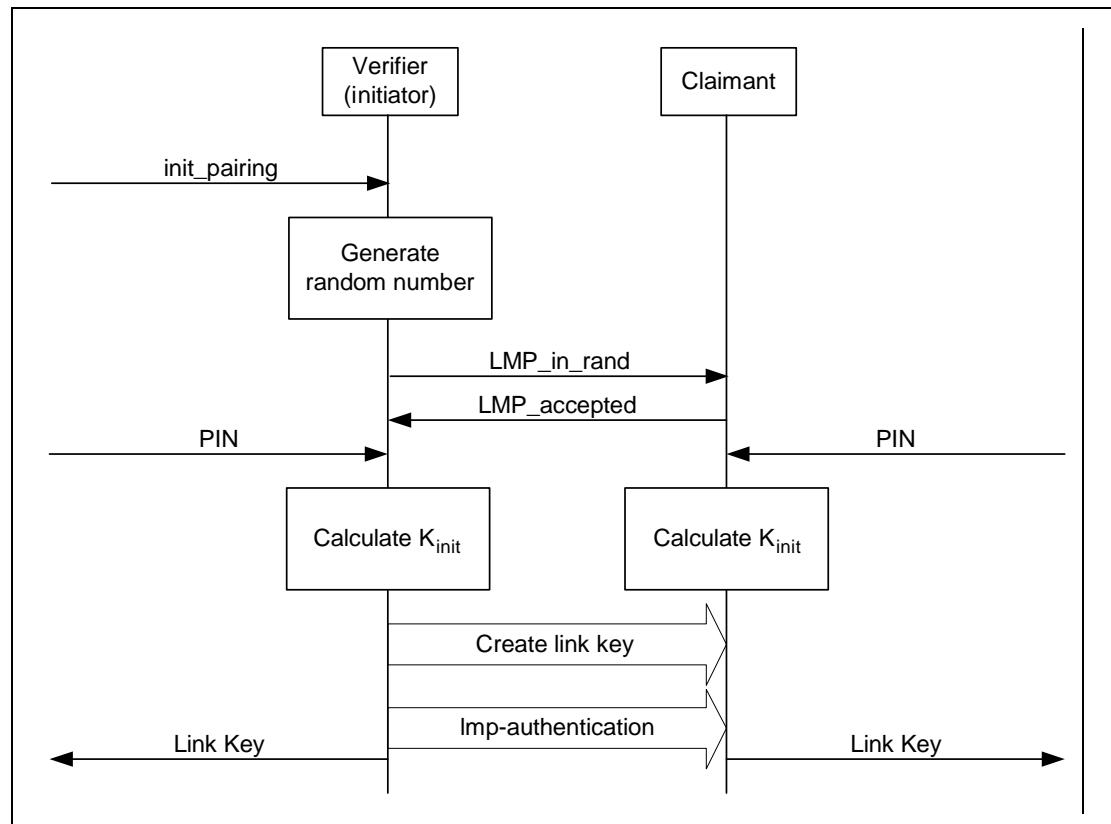


Figure B.2: LMP-pairing as defined in [\[2\]](#)

The PIN used here is PINBB.

The create link key procedure is described in [\[Vol. 2, Part C\] Section 4.2.2.4 on page 279](#) and [\[Vol. 2, Part H\] Section 3.2 on page 1312](#). In case the link key is based on a combination key, a mutual authentication takes place and shall be performed irrespective of current security mode.

B.3 SERVICE DISCOVERY

The Service Discovery Protocol [5] specifies what PDUs are used over-the-air to inquire about services and service attributes. The procedures for discovery of supported services and capabilities using the Service Discovery Protocol are described in higher layer specifications. This is just an example.

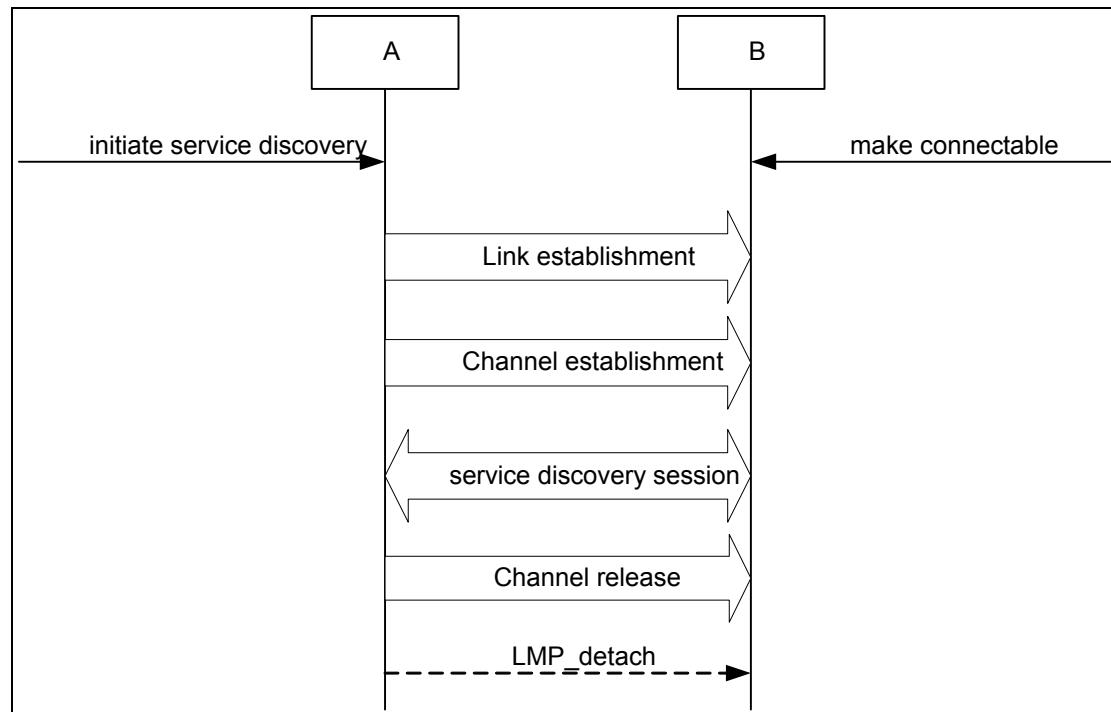


Figure B.3: Service discovery procedure

B.4 GENERATING A RESOLVABLE PRIVATE ADDRESS

Generating a resolvable private address is described in [Section 10.8.2.2](#).

B.5 RESOLVING A RESOLVABLE PRIVATE ADDRESS

Resolving a resolvable private address is described in [Section 10.8.2.3](#).

Core System Package [Host volume]
Part D

TEST SUPPORT

CONTENTS

1	Test Methodology	414
1.1	BR/EDR Test Scenarios	414
1.1.1	Test Setup	414
1.1.2	Transmitter Test	415
1.1.2.1	Packet Format	416
1.1.2.2	Pseudorandom Sequence	417
1.1.2.3	Control of Transmit Parameters	418
1.1.2.4	Power Control	418
1.1.2.5	Switch Between Different Frequency Settings	418
1.1.2.6	Adaptive Frequency Hopping	419
1.1.3	LoopBack Test	419
1.1.4	Pause Test	423
1.2	AMP Test Scenarios	424
1.2.1	Methodology Overview	424
1.2.1.1	Initiation Example Description	425
1.2.2	Control and Configuration	426
1.2.3	AMP Test Manager	426
1.2.4	Test Commands/Events Format	427
1.2.5	AMP Test Manager Commands/Events	429
1.2.5.1	AMP Command Rejected Event	429
1.2.5.2	AMP Discover Request	430
1.2.5.3	AMP Discover Response Event	430
1.2.5.4	AMP Read PHY Capability Bit Map Command	431
1.2.5.5	AMP Read PHY Capability Bit Map Response Event	432
1.3	References	432
2	Test Control Interface (TCI).....	433
2.1	Introduction	433
2.1.1	Terms Used	433
2.1.2	Usage of the Interface	433
2.2	TCI Configurations	434
2.2.1	Bluetooth RF Requirements	434
2.2.1.1	Required interfaces	434
2.2.2	Bluetooth Protocol Requirements	435
2.2.2.1	Required interfaces	435
2.2.3	Bluetooth Profile Requirements	436
2.2.3.1	Required interfaces	436

Test Support

2.3	TCI Configuration and Usage	437
2.3.1	Transport Layers.....	437
2.3.1.1	Physical bearer.....	437
2.3.1.2	Software bearer.....	437
2.3.2	Baseband and Link Manager Qualification	438
2.3.3	HCI Qualification.....	440

1 TEST METHODOLOGY

This section describes the test modes for hardware and low-level functionality tests of Bluetooth devices.

The BR/EDR test mode supports testing of the Bluetooth transmitter and receiver including transmitter tests (packets with constant bit patterns) and loopback tests. It is intended mainly for certification/compliance testing of the radio and baseband layer, and may also be used for regulatory approval or in-production and after-sales testing.

For AMP, this section describes the AMP test mode for hardware and low-level functionality tests of Bluetooth AMP devices. The AMP test mode provides the ability to control an AMP device using HCI Commands for both MAC conformance testing and PHY transmitter and receiver tests utilising the BR/EDR connection. The AMP test mode is intended mainly for certification/compliance testing of the MAC and PHY and may also be used for regulatory approval or in production and after sales testing.

1.1 BR/EDR TEST SCENARIOS

A device in BR/EDR test mode shall not support normal operation. For security reasons the BR/EDR test mode is designed such that it offers no benefit to the user. Therefore, no data output or acceptance on a HW or SW interface shall be allowed.

1.1.1 Test Setup

The setup consists of a device under test (DUT) and a tester. Optionally, additional measurement equipment may be used.

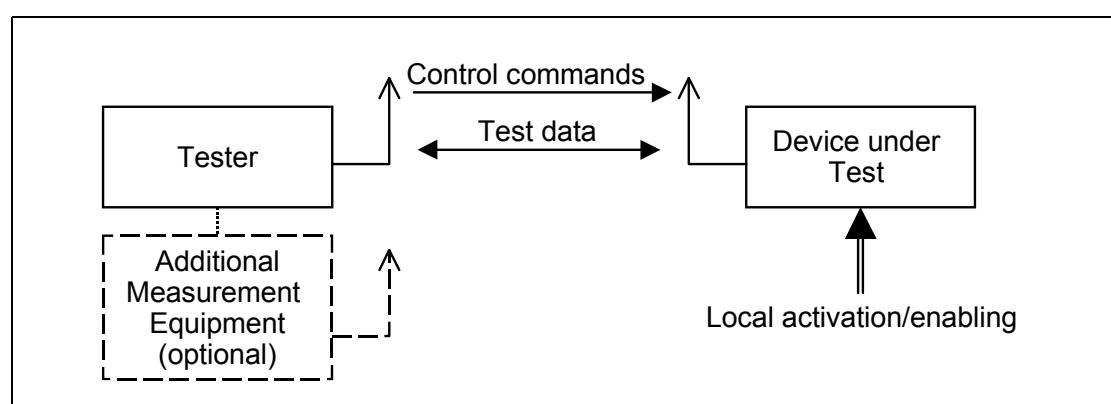


Figure 1.1: Setup for Test Mode

Tester and DUT form a piconet where the tester acts as master and has full control over the test procedure. The DUT acts as slave.

The control is done via the air interface using LMP commands (see [\[Vol. 2, Part C\] Section 4.7.3 on page 344](#)). Hardware interfaces to the DUT may exist, but are not subject to standardization.

The test mode is a special state of the Bluetooth model. For security and type approval reasons, a Bluetooth device in test mode shall not support normal operation. When the DUT leaves the test mode it enters the standby state. After power-off the Bluetooth device shall return to the standby state.

1.1.2 Transmitter Test

The Bluetooth device transmits a constant bit pattern. This pattern is transmitted periodically with packets aligned to the slave TX timing of the piconet formed by tester and DUT. The same test packet is repeated for each transmission.

The transmitter test is started when the master sends the first POLL packet. In non-hopping mode the agreed frequency is used for this POLL packet.

The tester (master) transmits control or POLL packets in the master-to-slave transmission slots. The DUT (slave) shall transmit packets in the following slave-to-master transmission slot. The tester's polling interval is fixed and defined by the LMP_test_control PDU. The device under test may transmit its burst according to the normal timing even if no packet from the tester was received. In this case, the ARQN bit is shall be set to NAK.

The burst length may exceed the length of a one slot packet. In this case the tester may take the next free master TX slot for polling. The timing is illustrated in [Figure 1.2](#).

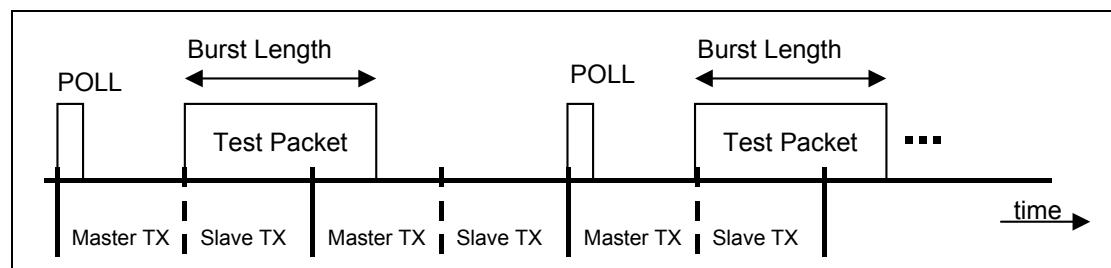


Figure 1.2: Timing for Transmitter Test

1.1.2.1 Packet Format

The test packet is a normal Bluetooth packet, see [Figure 1.3](#). For the payload itself see below.

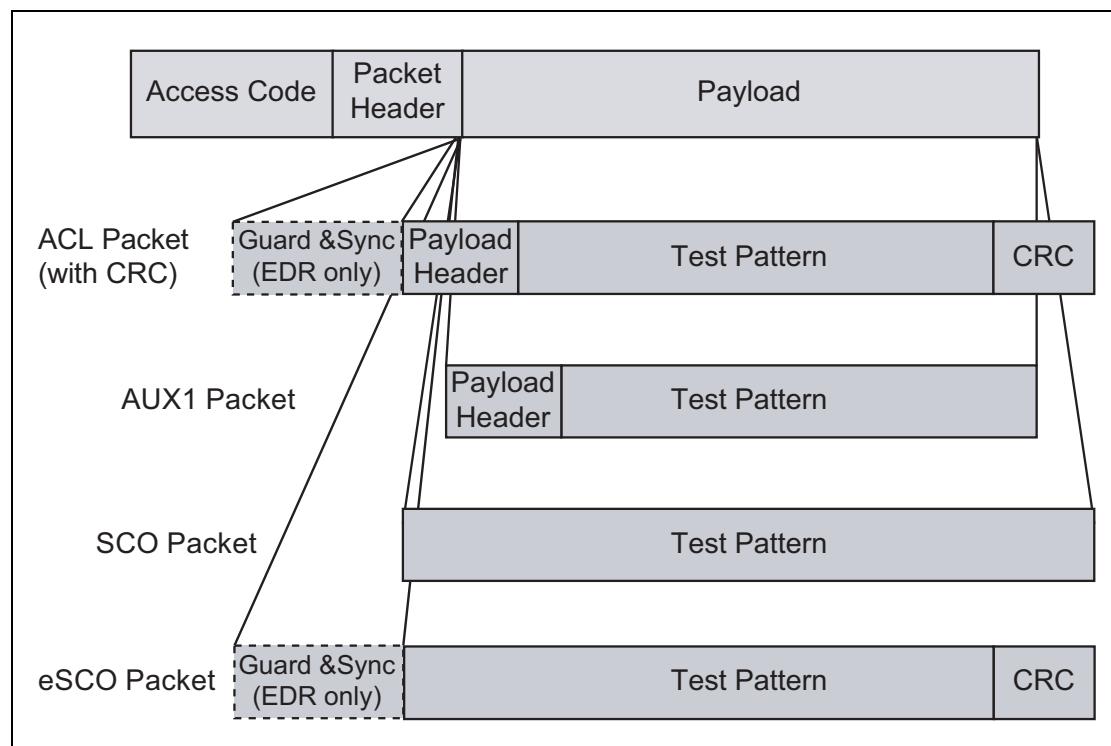


Figure 1.3: General Format of TX Packet

During configuration the tester defines:

- the packet type to be used
- payload length

For the payload length, the restrictions from the baseband specification shall apply (see [“Baseband Specification” on page 58\[vol. 2\]](#)). In case of ACL, SCO and eSCO packets the payload structure defined in the baseband specification is preserved as well, see [Figure 1.3 on page 416](#).

For the transmitter test mode, only packets without FEC should be used; i.e. HV3, EV3, EV5, DH1, DH3, DH5, 2-EV3, 2-EV5, 3-EV3, 3-EV5, 2-DH1, 2-DH3, 2-DH5, 3-DH1, 3-DH3, 3-DH5 and AUX1 packets.

In transmitter test mode, the packets exchanged between the tester and the DUT shall not be scrambled with the whitening sequence. Whitening shall be turned off when the DUT has accepted to enter the transmitter test mode, and shall be turned on when the DUT has accepted to exit the transmitter test mode, see [Figure 1.4 on page 417](#). Implementations shall ensure that retransmissions of the LMP_accepted messages use the same whitening status as used in the original LMP_accepted.

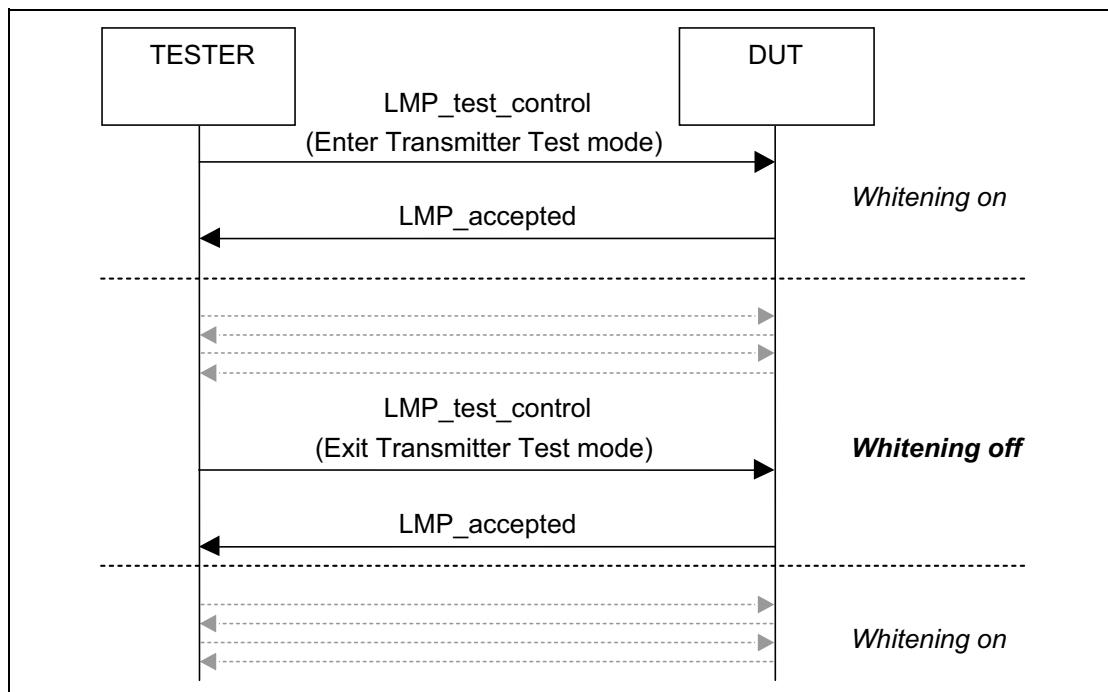


Figure 1.4: Use of whitening in Transmitter mode

1.1.2.2 Pseudorandom Sequence

The same pseudorandom sequence of bits shall be used for each transmission (i.e. the packet is repeated). A PRBS-9 Sequence is used, see [2] and [3].

The properties of this sequence are as follows (see [3]). The sequence may be generated in a nine-stage shift register whose 5th and 9th stage outputs are added in a modulo-two addition stage (see Figure 1.5), and the result is fed back to the input of the first stage. The sequence begins with the first ONE of 9 consecutive ONEs; i.e. the shift register is initialized with nine ones.

- Number of shift register stages: 9
- Length of pseudo-random sequence: $2^9 - 1 = 511$ bits
- Longest sequence of zeros: 8 (non-inverted signal)

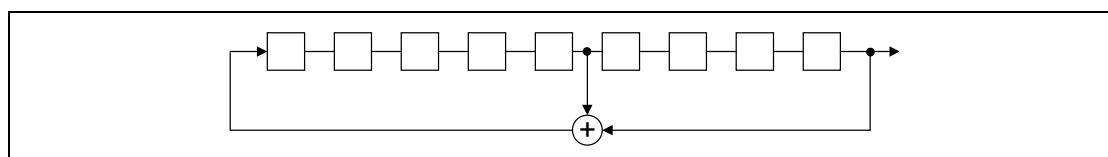


Figure 1.5: Linear Feedback Shift Register for Generation of the PRBS sequence

1.1.2.3 Control of Transmit Parameters

The following parameters can be set to configure the transmitter test:

1. Bit pattern:
 - Constant zero
 - Constant one
 - Alternating 1010...¹
 - Alternating 1111 0000 1111 0000...¹
 - Pseudorandom bit pattern
 - Transmission off
2. Frequency selection:
 - Single frequency
 - Normal hopping
3. TX frequency
 - $k \rightarrow f := (2402 + k)$ MHz
4. Default poll period in TDD frames ($n * 1.25$ ms)
5. Packet Type
6. Length of Test Sequence (user data of packet definition in “[Baseband Specification](#)” on page 58[vol. 2])

1.1.2.4 Power Control

When the legacy power control mechanism is tested the DUT shall start transmitting at the maximum power and shall reduce/increase its power by one step on every LMP_incr_power_req or LMP_decr_power_req PDU received. When the enhanced power control mechanism is tested a DUT shall start transmitting at the maximum power and shall reduce/increase its power by one step or go to the maximum power level when a LMP_power_control_req PDU is received.

1.1.2.5 Switch Between Different Frequency Settings

A change in the frequency selection becomes effective when the LMP procedure is completed:

When the tester receives the LMP_accepted it shall then transmit POLL packets containing the ACK for at least 8 slots (4 transmissions). When these transmissions have been completed the tester shall change to the new frequency hop and whitening settings.

1. It is recommended that the sequence starts with a one; but, as this is irrelevant for measurements, it is also allowed to start with a zero.

After sending LMP_accepted the DUT shall wait for the LC level ACK for the LMP_accepted. When this is received it shall change to the new frequency hop and whitening settings.

There will be an implementation defined delay after sending the LMP_accepted before the TX or loopback test starts. Testers shall be able to cope with this.

Note: Loss of the LMP_accepted PDU will eventually lead to a loss of frequency synchronization that cannot be recovered. Similar problems occur in normal operation, when the hopping pattern changes.

1.1.2.6 Adaptive Frequency Hopping

Adaptive Frequency Hopping (AFH) shall only be used when the Hopping Mode is set to 79 channels (e.g. Hopping Mode = 1) in the LMP_test_control PDU. If AFH is used, the normal LMP commands and procedures shall be used. When AFH is enabled prior to entering test mode it shall continue to be used with the same parameters if Hopping Mode = 1 until the AFH parameters are changed by the LMP_set_AFH PDU.

The channel classification reporting state shall be retained upon entering or exiting Test Mode. The DUT shall change the channel classification reporting state in Test Mode based on control messages from the tester (LMP_channel_classification_req) and from the Host (HCI Write_AFH_Channel_Classification_Mode).

1.1.3 LoopBack Test

In loopback, the device under test receives normal baseband packets containing payload Accepted from the tester. The received packets shall be decoded in the DUT, and the payload shall be sent back using the same packet type. The return packet shall be sent back in either the slave-to-master transmission slot directly following the transmission of the tester, or it is delayed and sent back in the slave-to-master transmission slot after the next transmission of the tester (see [Figure 1.7 to Figure 1.9 on page 422](#)).

There is no signaling to determine or control the mode. The device behavior shall be fixed or adjusted by other means, and shall not change randomly.

The tester can select, whether whitening is on or off. This setting holds for both uplink and downlink. For switching the whitening status, the same rules as in [Section 1.1.2 on page 415 \(Figure 1.4\)](#) shall apply.

The following rules apply (for illustration see [Figure 1.6 on page 421](#)):

- If the synch word was not detected, the DUT shall not reply.
- If the header error check (HEC) fails, the DUT shall either reply with a NULL packet with the ARQN bit set to NAK or send nothing.

- If the packet contains an LMP message relating to the control of the test mode this command shall be executed and the packet shall not be returned, though ACK or NAK shall be returned as per the usual procedure. Other LMP commands are ignored and no packet is returned.
- The payload FEC is decoded and the payload shall be encoded again for transmission. This allows testing of the FEC handling. If the pure bit error rate shall be determined the tester chooses a packet type without FEC.
- The CRC shall be evaluated. In the case of a failure, ARQN=NAK shall be returned. The payload shall be returned as received.
A new CRC for the return packet shall be calculated for the returned payload regardless of whether the CRC was valid or not.
- If the CRC fails for a packet with a CRC and a payload header, the number of bytes as indicated in the (possibly erroneous) payload header shall be looped back.

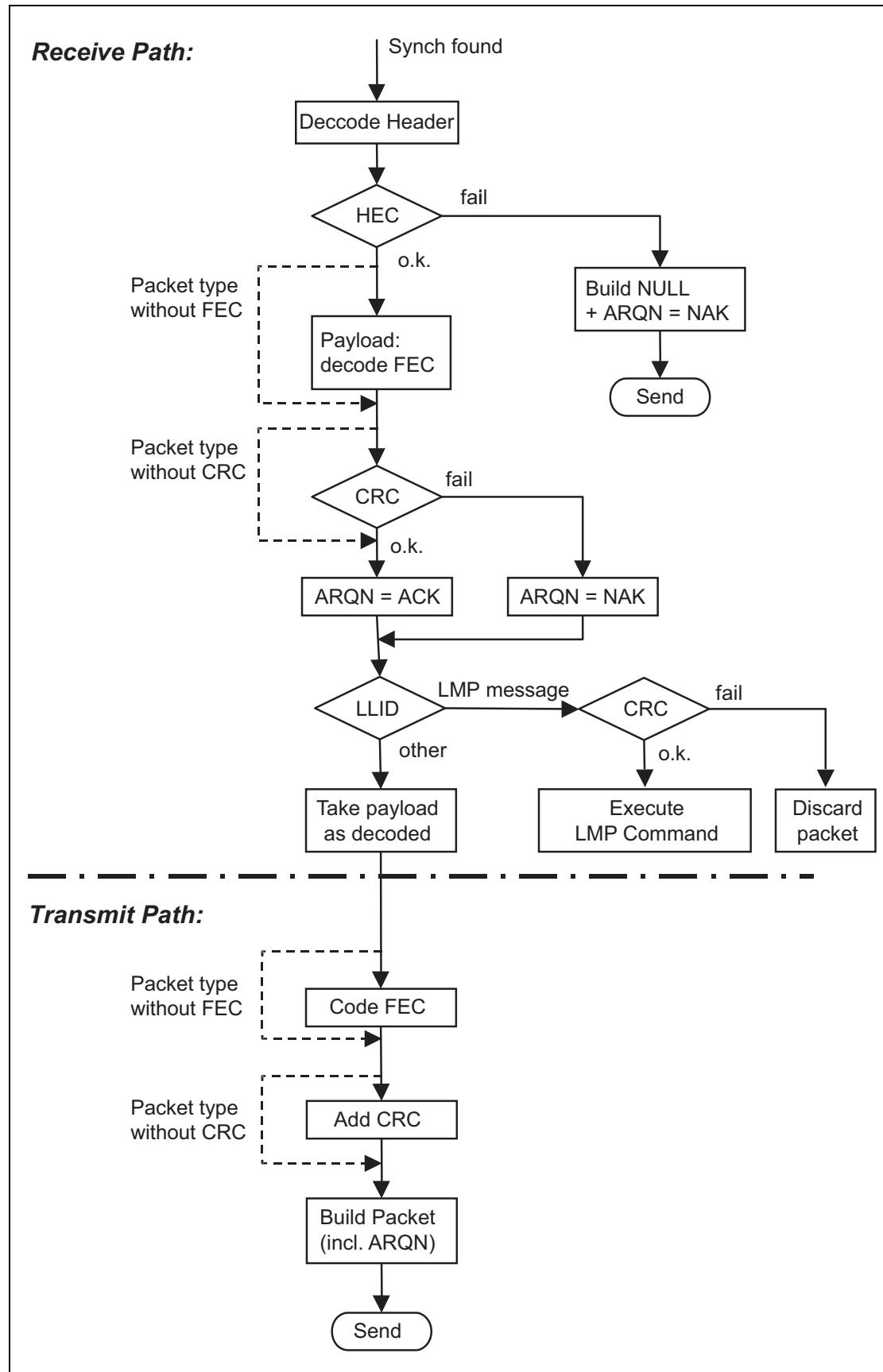


Figure 1.6: DUT Packet Handling in Loop Back Test

The timing for normal and delayed loopback is illustrated in [Figure 1.7](#) to [Figure 1.9](#):

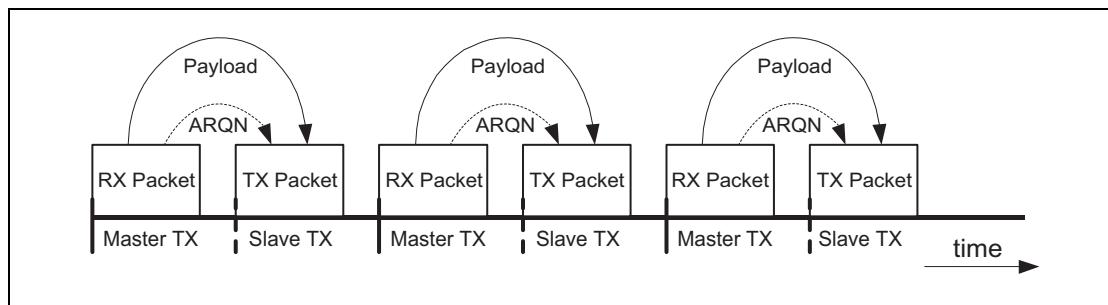


Figure 1.7: Payload & ARQN handling in normal loopback.

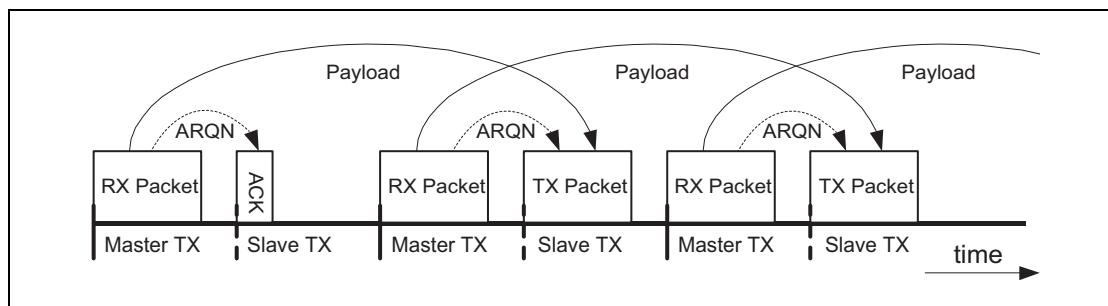


Figure 1.8: Payload & ARQN handling in delayed loopback - start.

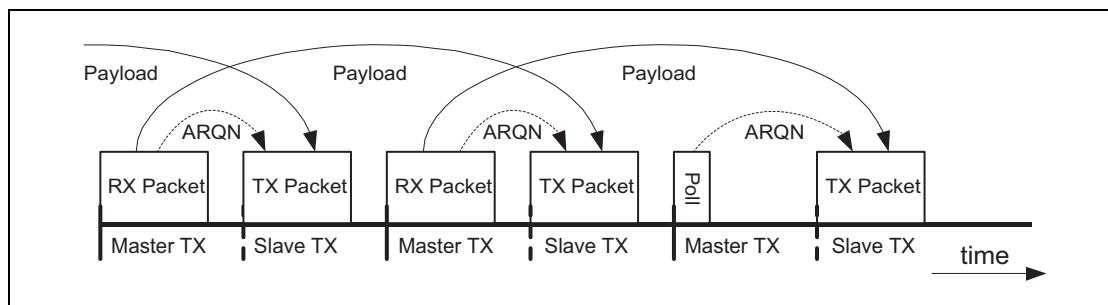


Figure 1.9: Payload & ARQN handling in delayed loopback - end.

The whitening is performed in the same way as it is used in normal active mode.

The following parameters can be set to configure the loop back test:

1. Packet Class¹
 - ACL Packets
 - SCO Packets
 - eSCO Packets

1. This is included because, in the future, the packet type numbering may not remain unambiguous.

Test Support

ACL Packets without whitening
SCO Packets without whitening
eSCO Packets without whitening

2. Frequency Selection
 - Single frequency (independent for RX and TX)
 - Normal hopping
3. Power level: (To be used according radio specification requirements)
power control or fixed TX power

The switch of the frequency setting is done exactly as for the transmitter test (see [Section 1.1.2.5 on page 418](#)).

1.1.4 Pause Test

Pause test is used by testers to put the device under test into Pause Test mode from either the loopback or transmitter test modes.

When an LMP_test_control PDU that specifies Pause Test is received the DUT shall stop the current test and enter Pause Test mode. In the case of a transmitter test this means that no more packets shall be transmitted. While in Pause Test mode the DUT shall respond normally to POLL packets (i.e. responds with a NULL packet). The DUT shall also respond normally to all the LMP packets that are allowed in test mode.

When the test scenario is set to Pause Test all the other fields in the LMP_test_control PDU shall be ignored. There shall be no change in hopping scheme or whitening as a result of a request to pause test.

1.2 AMP TEST SCENARIOS

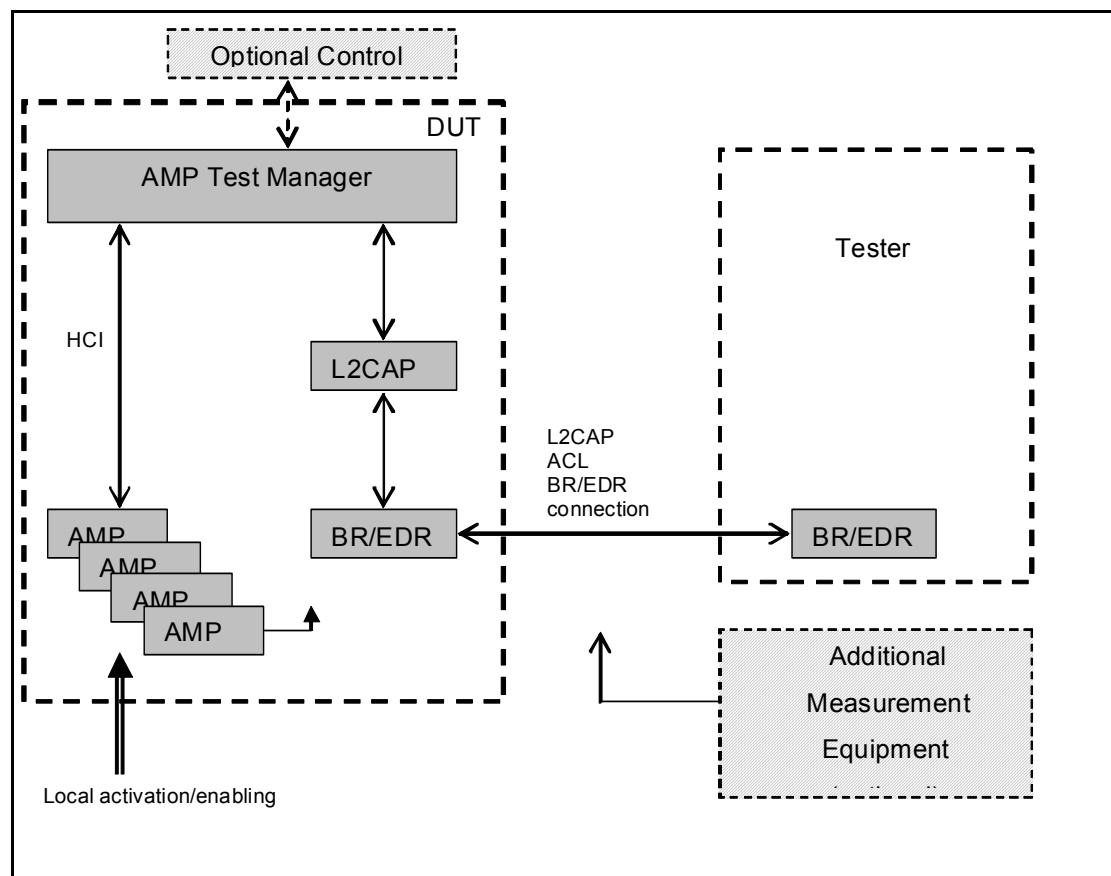


Figure 1.10: Test Architecture

1.2.1 Methodology Overview

To perform PHY tests on an AMP, the DUT shall first have AMP Test Mode enabled locally as required for a BR/EDR Controller using the HCI Enable_Device_Under_Test_Mode command. When this command is received by the AMP controller it shall enable the AMP to accept HCI Test Commands and when received by the BR/EDR controller it shall enable the AMP Test Manager to pass HCI commands and events between the tester and the AMP device utilizing the BR/EDR connection. A DUT AMP shall reject all AMP related HCI Test Commands, except HCI_Enable_Device_Under_Test_Mode, until the HCI_Enable_Device_Under_Test_Mode command has been issued by the local Host. Although AMP Test Mode is enabled the AMP shall not enter AMP Test Mode until the HCI_AMP_Test_Command is received.

Note: Enabling test mode with HCI_Enable_Device_Under_Test_Mode command allows the AMP to enter test mode when an HCI_AMP_Test_Command is received.

The local control of an AMP in AMP Test Mode shall be by the AMP Test Manager as shown in [Figure 1.10](#). The AMP Test Manager will receive AMP Test Manager commands and HCI test commands via ACL data packets as

shown in [Figure 1.10](#), using an L2CAP connectionless fixed test channel (channel 0x003F) from a Bluetooth test device. This does not exclude driving the AMP Test Manager directly from an implementation vendor specific interface as shown from the "Optional Control" in [Figure 1.10](#).

The AMP Test Manager commands allow the tester to discover the number and type of AMPs supported and to read their PHY capability support via the AMP Test Manager.

The tester can use HCI commands and HCI Test Commands to control all of the AMPs to perform qualification, IOP testing as well as entering AMP Test Mode to perform AMP PHY tests. The HCI events and responses shall be routed back to the tester via the fixed test channel (0x003F).

If an HCI command is sent to an invalid Controller ID (a controller not reported in the Discovery Response) the AMP Command Rejected Event shall be returned with an Invalid Controller Id as the reason.

An AMP shall be taken out of AMP Test Mode by the HCI_Reset command. This will reset only the AMP receiving the HCI_Reset.

The AMP Test Manager shall not be enabled to support HCI commands over the fixed test channel (0x003F) unless AMP Test Mode has been enabled locally with the HCI_Enable_Device_Under_Test_Mode command.

The "Optional Control" interface shown in [Figure 1.10](#) is a vendor specific interface that allows local control of the AMP Test Manager when a BR/EDR link may not be available.

1.2.1.1 Initiation Example Description

The AMP Test Manager is enabled to receive AMP test commands and HCI commands over the fixed test channel once it has been locally enabled. Using the fixed test channel, the tester shall use the AMP test commands and events to identify the number and type of AMPs available.

When the AMP Test Manager has been enabled and the AMPs identified the Tester shall use the AMP Test Manager to

- Transfer HCI commands and Events to and from the AMPs for qualification and IOP testing.
- Put the AMPs into test mode using HCI AMP test commands and events for AMP PHY testing.

1.2.2 Control and Configuration

Control and configuration of AMP tests shall be performed using two groups of commands/events over the ACL BR/EDR connection on the fixed test channel. The two groups are the AMP Test Manager commands/events which are to the AMP Test Manager and the HCI AMP test commands/events which are routed via the AMP Test Manager between the AMP indicated as part of the message structure and the tester.

AMP Test Manager commands and events	AMP Command Rejected Event
	AMP Discover Request Command
	AMP Discover Response Event
	AMP Read PHY Capability Bit Map Request
	AMP Read PHY Capability Bit Map Response Event
HCI AMP test commands and events	Read Loopback Mode Command
	Write Loopback Mode Command
	Enable AMP Receiver Reports Command
	AMP Test End Command
	AMP Test Command
	AMP Start Test Event
	AMP Test End Event
	AMP Receiver Report Event

1.2.3 AMP Test Manager

The AMP Test Manager provides the interface to the AMPs available so that qualification, IOP and PHY testing can be performed using the BR/EDR connection. The AMP Test Manager provides the following services:

1. Identification of the number and type of AMPs available
2. Enumeration of the available AMPs so that they can be communicated to individually
3. Routing of the HCI commands and events between the AMPs and the tester over the ACL BR/EDR connection on the fixed test channel, or the vendor specific interface (that is, a local interface to the AMP Test Manager other than the BR/EDR radio).

1.2.4 Test Commands/Events Format

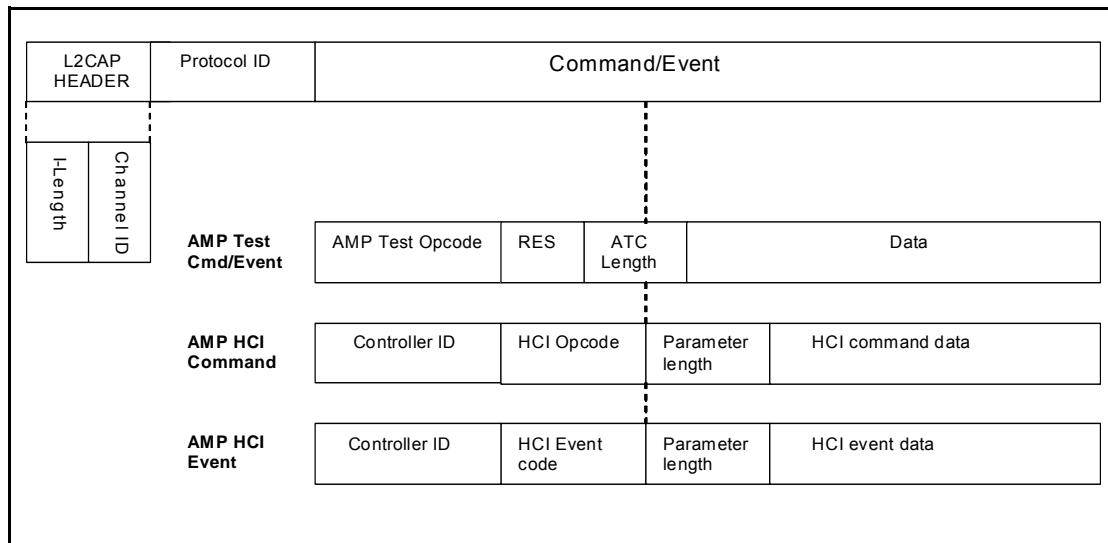


Figure 1.11: L2CAP test command/event formats

The fields shown are:

L2CAP header (4 octets)

This is the Basic L2CAP header with the Channel ID always set to test channel (0x003F). (See section [\[Vol. 3\], Part A Section 3.1](#))

- *Length (2 octets)*
Length of the Information payload which includes the Protocol ID and the Command/Event
- *Channel ID (2 octets)*
Fixed value of (0x003F)

Protocol ID (1 octet)

This identifies the protocol in the command/event data. The IDs used are an extension of the UART transport packet types.

Protocol ID	Command/event type
0x01	HCI Command
0x02	Reserved
0x03	Reserved
0x04	HCI Event
0x05	AMP Test Command
0x06	AMP Test Event

Table 1.1: L2CAP valid test protocol IDs

Test Support

Controller ID (1 octet)

Each Controller in a device is assigned a one-octet unique identifier by the local AMP Test Manager. This field determines the destination of the HCI Command or the source of the HCI Event. The Controller ID values shall be in the range of 1 to 255. Controller ID 0 has been reserved for the primary BR/EDR controller.

Note: "Controller" is a generic term which covers both the primary BR/EDR controller and any AMPs

ATO (AMP Test Opcode) (1 octet)

This is the AMP Test Manager commands or event code as described in [Section 1.2.5](#).

RES (1 octet)

Reserved = 0

AMP Test Command (ATC) Length (2 octets)

This is the number of octets in the data portion of this message

AMP HCI Commands and Events

The format of these packets follow the standard HCI format described in [\[Vol. 2\], Part E](#), after the Controller ID.

Examples

	Protocol ID	Command/Event			
AMP Test command	0x05	AMP Test Opcode	0x00	Length	AMP Test Command
AMP Test Event	0x06	AMP Test Opcode	0x00	Length	AMP Test Event
AMP HCI Command	0x01	Controller ID	HCI Command		
AMP HCI Event	0x04	Controller ID	HCI Event		

1.2.5 AMP Test Manager Commands/Events

The AMP Test Manager commands are sent to the AMP Test Manager from the tester and the events are from the AMP Test Manager to the tester.

The data fields are:

- **AMP Test Opcode (1 octet)**
Identifies the type of AMP Test Manager command/event.
- **RES (1 octet)**
Reserved and shall be set to 0.
- **Amp Test Command (ATC) length (2 octets)**
Length of the data field of the command/event. This length does not include the AMP Test Opcode, reserved octet or the ATC length.

AMP Test Opcode	Description
0x00	Reserved
0x01	AMP Command Rejected Event
0x02	AMP Discover Request
0x03	AMP Discover Response Event
0x04	AMP Read PHY Capability Bit Map Command
0x05	AMP Read PHY Capability Bit Map Response Event
0x06 - 0xFF	Reserved

Table 1.2: AMP Test Opcodes

1.2.5.1 AMP Command Rejected Event

This event shall be sent by the DUT to the tester when a command cannot be accepted. The parameter returned indicates the reason.

Opcode = 0x01	Reserved = 0	ATC length = 2	Reason
---------------	--------------	----------------	--------

The data fields are:

- **AMP Test Opcode (1 octet)**
0x01
- **RES (1 octet)**
Reserved and shall be set to 0
- **ATC length (2 octets)**
Always set to 0x0002

- *Reason (2 octets)*

See [Table 1.3](#).

Reason Value	Description
0x0000	Command not recognized
0x0001	Invalid Controller ID
0x0002	Invalid Protocol ID
0x0003 to 0xFFFF	Reserved

Table 1.3: AMP Command reasons for rejection

1.2.5.2 AMP Discover Request

The tester shall send this command to the AMP Test Manager to obtain the available Controllers and their types.

Opcode = 0x02	Reserved = 0	ATC length = 0
---------------	--------------	----------------

The data fields are:

- *AMP Test Opcode (1 octet)*
Shall be set to 0x02
- *RES (1 octet)*
Reserved and shall be set to 0x00
- *ATC length (2 octet)*
Shall be set to 0x0000

Events generated

On receipt of the AMP_Discover_Request command the AMP Test Manager shall return the AMP_Discover_Response event.

1.2.5.3 AMP Discover Response Event

When the AMP Test Manager receives an AMP_Discover_Request command it shall reply to the tester with an AMP_Discover_Response event indicating for each controller the Controller ID and Controller Type. The first Controller ID and Type pair in the response shall be 0 for the BR/EDR primary controller followed by the available additional Controllers ID and Type pairs.

Test Support



Opcode = 0x03	Reserved = 0	ATC length	Controller list
---------------	--------------	------------	-----------------

Controller list

Controller ID=0	Controller Type=0	Additional Controller ID and Type pairs
-----------------	-------------------	---

- **AMP Test Opcode (1 octet)**
Shall be set to 0x03
- **RES (1 octet)**
Reserved and shall be set to 0x00.
- **ATC length (2 octets)**
ATClength = (Controller ID length + Controller Type length) * number of Controllers
- **Controller Types (1 octet)**
Each type of Controller is assigned a unique one-octet value. These values are defined in [Assigned Numbers](#). A device may have multiple AMPs of the same type.

Note: The AMP Discovery Response shall include the primary BR/EDR Controller as the first Controller ID and Controller Type pair.

1.2.5.4 AMP Read PHY Capability Bit Map Command

To perform AMP PHY tests the tester needs to know the supported PHY capabilities of the DUT. This information shall be provided in the PHY Capability Bit Map. This command shall be sent from the tester to the DUT to request the PHY Capability Bit Map from an AMP with the passed Controller ID.

Opcode = 0x04	Reserved = 0	ATC length = 1	Controller ID
---------------	--------------	----------------	---------------

The data fields are:

- **AMP Test Opcode (1 octet)**
Always set to 0x04
- **RES (1 octet)**
Reserved and shall be set to 0
- **ATC length (2 octets)**
Always set to 0x0001
- **Controller ID (1 octet)**
The Controller from which the PHY Capability Bit Map is being requested. This is an invalid request from a BR/EDR Controller and will be rejected with the reason of Invalid Controller ID.

1.2.5.5 AMP Read PHY Capability Bit Map Response Event

When the AMP_Read_PHY_Capability_Bit_Map Command is received by an AMP Test Manager it shall reply with the capability bit map for the AMP indicated if the Controller ID is valid.

- *AMP Test Opcode (1 octet)*
Always set to 0x05
- *RES (1 octet)*
Reserved and shall be set to 0
- *ATC length (2 octets)*
The length of the PHY Capabilities Bit Map returned depending on the AMP type.
- *Status (1 octet)*
See the following table.

Reason Value	Description
0x00	Success
0x01	Invalid Controller ID
0x02 to 0xFF	Reserved

Note: If the status is not Success the remaining parameters shall be zero.

- *Controller ID (1 octet)*
This is the Controller ID from which the PHY Capability Bit Map has been sent.
- *PHY Capabilities Bit Map*
See [\[Volume 5\]](#) for the PHY Capabilities Bit Map.

Opcode = 0x05	Reserved = 0	ATC length	Status	Controller ID	PHY Capabilities Bit Map
---------------	--------------	------------	--------	---------------	--------------------------

1.3 REFERENCES

- [1] Bluetooth Link Manager Protocol.
- [2] CCITT Recommendation O.153 (1992), Basic parameters for the measurement of error performance at bit rates below the primary rate.
- [3] ITU-T Recommendation O.150 (1996), General requirements for instrumentation for performance measurements on digital transmission equipment.
- [4] Bluetooth Baseband Specification.

2 TEST CONTROL INTERFACE (TCI)

This section describes the Bluetooth Test Control Interface (TCI). The TCI provides a uniform method of accessing the upper interface of the implementation being tested. This facilitates the use of a standardized interface on test equipment used for formal Qualification of implementations.

2.1 INTRODUCTION

2.1.1 Terms Used

Conformance testing	Testing according to the applicable procedures given in the Bluetooth Protocol Test Specifications and the Bluetooth Profile Conformance Test Specification when tested against a test system.
HCI	Host Controller Interface
IUT	Implementation Under Test: An implementation of one or more Bluetooth protocols and profiles which is to be studied by testing. This term is used when describing the test concept for products and components equipped with Bluetooth wireless technology as defined in the PRD.
PRD	Bluetooth Qualification Program Reference Document: This document is maintained by the Bluetooth Qualification Review Board and is the reference to specify the functions, organization and processes inside the Bluetooth Qualification program.
TCI	Test Control Interface: The interface and protocol used by the test equipment to send and receive messages to and from the upper interface of the IUT.

2.1.2 Usage of the Interface

For all products and components equipped with Bluetooth wireless technology, conformance testing is used to verify the implemented functionality in the lower layers. Conformance testing of the lowest layers requires an upper tester to test the implementation sufficiently well.

In order to avoid that the tester will have to adapt to each and every product and component equipped with Bluetooth wireless technology, the use of the standardized TCI is mandated. This concept puts some burden upon the manufacturer of the IUT in terms of supplying an adapter providing the necessary conversion from/to the IUT's specific interface to the TCI. The adapter can consist of hardware, firmware and software. After qualification testing has been performed the TCI may be removed from the product or component equipped with Bluetooth wireless technology. It is the

manufacturer's option to remove it from the qualified product or component equipped with Bluetooth wireless technology.

The TCI is used when qualifying the implemented functionality of the:

- Baseband layer, BB
- Link Manager layer, LM

If support of the Host Controller Interface is claimed by the manufacturer the TCI is used to qualify it.

2.2 TCI CONFIGURATIONS

This section describes the test configurations used when verifying the different Bluetooth requirements. Each layer in the Bluetooth stack is qualified using the procedures described in the layer specific test specification.

2.2.1 Bluetooth RF Requirements

For qualification of the Bluetooth Radio Frequency requirements the defined Test Mode is used, see [Section 1 on page 414](#).

Similar to TCI, the specific test mode functionality may be removed from the product or component after qualification, at the discretion of the manufacturer.

2.2.1.1 *Required interfaces*

For RF qualification only the air interface is required, see [Figure 2.1](#). Depending on the physical design of the IUT it might be necessary to temporarily attach an RF connector for executing the RF tests. As stated in [Section 1 on page 414](#), the Test Mode shall be locally enabled on the IUT for security reasons. The implementation of this local enabling is not subject to standardization.

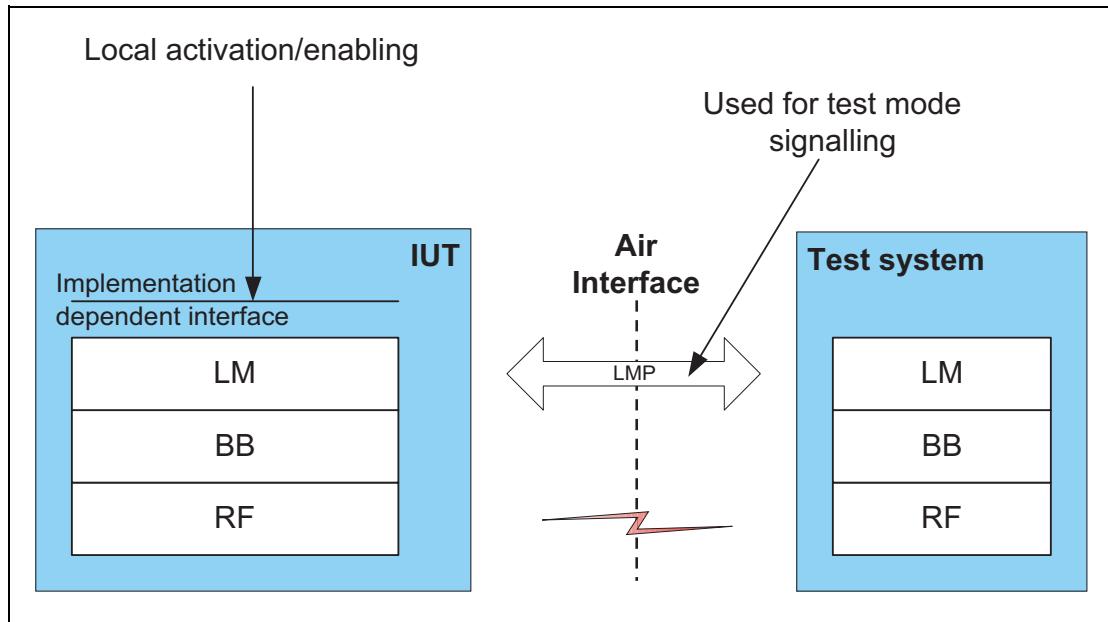


Figure 2.1: General test set-up for RF qualification

2.2.2 Bluetooth Protocol Requirements

Depending on which of the Bluetooth layers BB, LM or HCI are implemented in the product subject to qualification, the amount of testing needed to verify the Bluetooth protocol requirements differs. Also, the TCI used during the qualification is slightly different. The commands and events necessary for qualification are detailed in the test specifications and only those commands indicated in the test cases to be executed need be implemented.

For other protocols in the Bluetooth stack the TCI is not used. An implementation specific user interface is used to interact with the IUT's upper interface.

2.2.2.1 Required interfaces

For BB, LM and HCI qualification both the air interface of the IUT and the TCI are required. For other protocols both the air interface and the user interface are used as described in the test specification.

2.2.3 Bluetooth Profile Requirements

For each Bluetooth profile for which conformance is claimed, profile qualification testing is performed to verify the Bluetooth profile requirements. With higher layer protocols the TCI is not used during testing. A user interface specific to the implementation is used to interact with the IUT's upper interface.

2.2.3.1 Required interfaces

For this type of qualification both the air interface and the user interface of the IUT are used as described in the test specification.

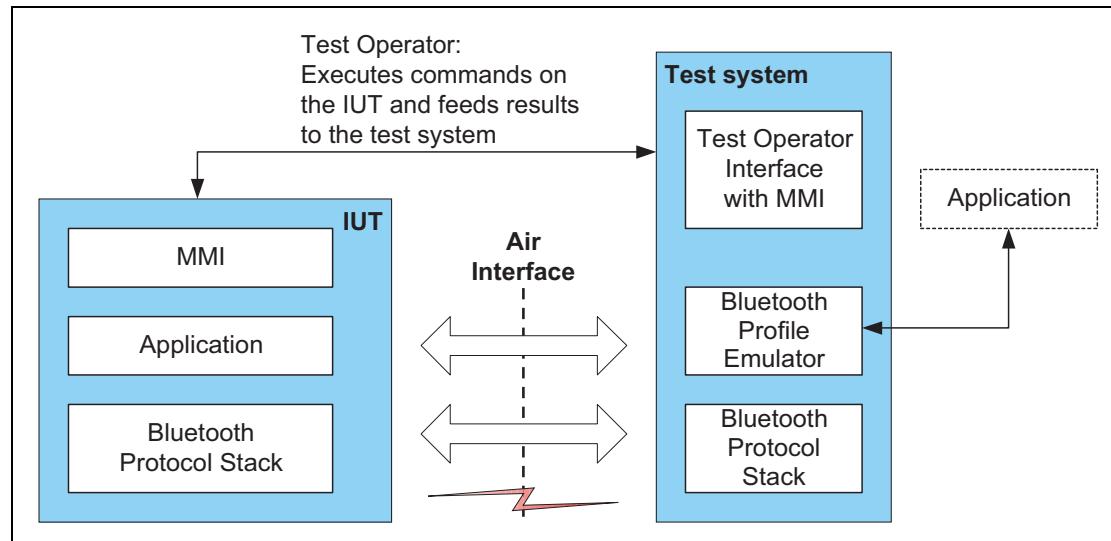


Figure 2.2: General test set-up for profile qualification

2.3 TCI CONFIGURATION AND USAGE

This interface is semantically and syntactically identical to the HCI interface described in “[Host Controller Interface Functional Specification](#)” on [page 386\[vol. 2\]](#). The complete HCI interface is not required in the TCI, but the subset of HCI commands and events necessary for verifying the functionality of the IUT. The exact set of commands and events is specified in the test specifications for the layers subject to testing.

It is worth emphasizing again the TCI is an adapter, logically attached to the upper interface of the IUT. As such the TCI adapts the standardized signaling described here to the implementation specific interface of the IUT.

2.3.1 Transport Layers

The method used to convey commands and events between the tester and the IUT’s upper interface can be either physical bearer or “software bearer”.

2.3.1.1 *Physical bearer*

It is recommended to use one of the transport layers specified for the HCI in [\[Volume 4\] Host Controller Interface \[Transport Layer\]](#). However, other physical bearers are not excluded and may be provided on test equipment. The use of a physical bearer is required for test cases active in category A. Please see the PRD for the definitions of test case categories. Please see the current active Test Case Reference List for the categorization of specific test cases.

2.3.1.2 *Software bearer*

There is no physical connection between the tester and the IUT’s upper interface. In this case, the manufacturer of the IUT shall supply test software and hardware that can be operated by a test operator. The operator will receive instructions from the tester and will execute them on the IUT. The “software bearer” shall support the same functionality as if using the TCI with a physical bearer. Use of the “software bearer” shall be agreed upon between the manufacturer of the IUT and the test facility that performs the qualification tests. The test facilities can themselves specify requirements placed on such a “software bearer”. Furthermore, the use of a “software bearer” is restricted to test cases active in one of the three lower categories B, C and D.

2.3.2 Baseband and Link Manager Qualification

For the qualification of the link control part of the Baseband layer and for the Link Manager layer, the TCI is used as the interface between the test system and the upper interface of the IUT. The test system accesses the upper interface of the IUT by sending HCI commands and receiving HCI events from the IUT as described in the HCI specification. The required functionality on the TCI depends on the IUT's implemented functionality of the BB and LM layers, and therefore which test cases are executed.

A schematic example in [Figure 2.3](#) shows the test configuration for BB and LM qualification of Bluetooth products which do not support HCI, and use a physical bearer for the TCI. In this example the Test Control (TC) Software represents what the manufacturer has to supply with the IUT when using an external test facility for qualification. The function of the TC Software is to adapt the implementation dependent interface to the TCI.

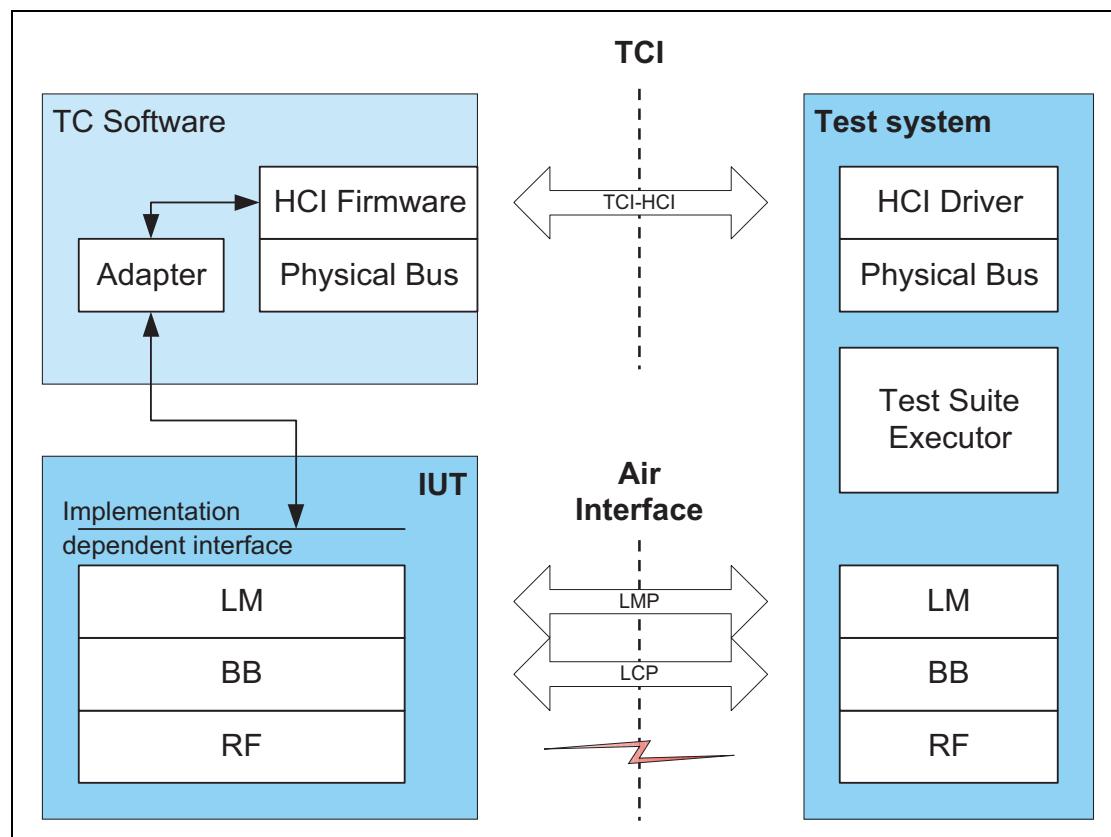


Figure 2.3: BB and LM qualification with TCI physical bearer

[Figure 2.4](#) shows a schematic example of the test configuration for the same Bluetooth product using a “software bearer” for the TCI. Here the function of the Test Control Software is to represent the application that can be controlled by the test operator.

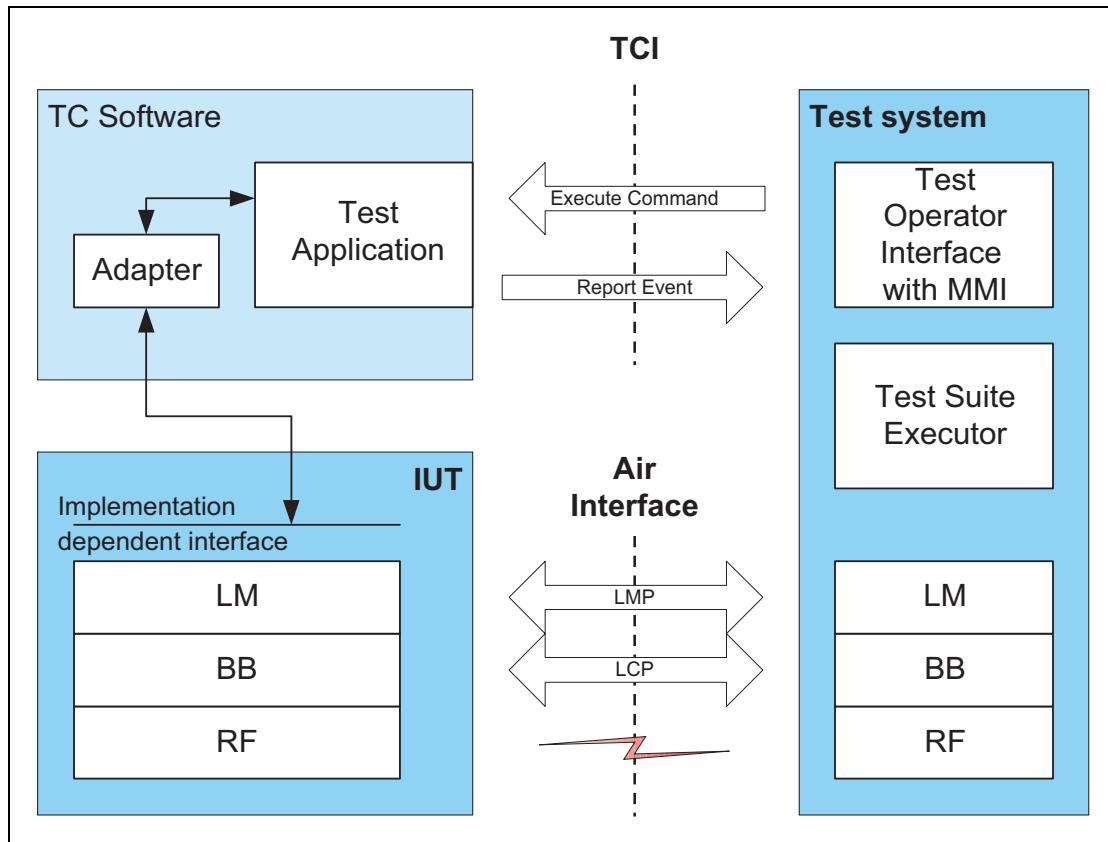


Figure 2.4: BB and LM qualification with “software bearer”

2.3.3 HCI Qualification

The TCI may also be used for HCI signaling verification and qualification. The HCI signaling is only verified if support of the HCI functionality is claimed by the manufacturer.

A schematic example in [Figure 2.5](#) shows the test configuration for HCI qualification of Bluetooth products. As can be seen in the figure the implemented HCI is used as the interface to the tester.

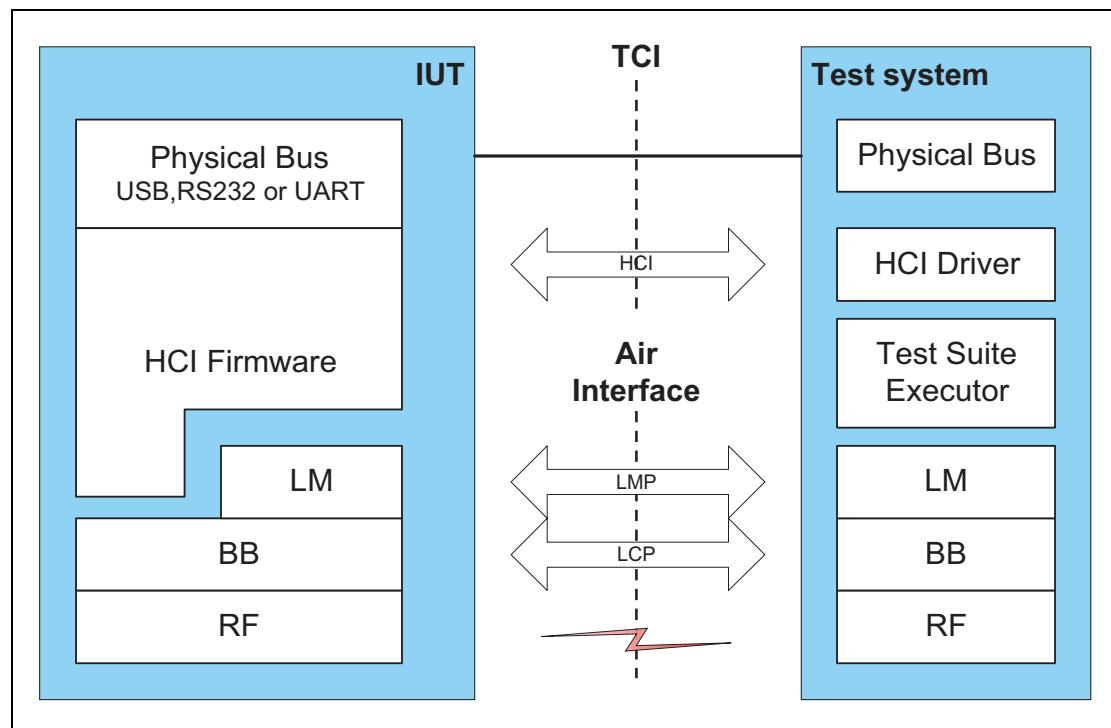


Figure 2.5: General test set-up for HCI qualification

Core System Package [Host volume]
Part E

AMP MANAGER PROTOCOL SPECIFICATION

This document specifies the changes to the Core specification required to incorporate the AMP Manager Protocol (A2MP) for the Alternate MAC/PHY feature.

CONTENTS

1	Introduction	443
1.1	General Description	443
2	General Operation	444
2.1	Basic Capabilities	444
2.2	AMP Manager Channel Over L2CAP	445
2.3	Using the AMP Manager Protocol	446
2.3.1	Discovering a Remote AMP Manager.....	446
2.3.2	Discovering Available Controllers on a Remote Device	446
2.3.3	Creation of AMP Physical Links.....	447
2.4	Controller IDs.....	448
2.5	Controller Types	448
3	Protocol Description	449
3.1	Packet Formats.....	449
3.2	AMP Command Reject (Code 0x01)	451
3.3	AMP Discover Request (Code 0x02).....	452
3.4	AMP Discover Response (Code 0x03).....	453
3.5	AMP Change Notify (Code 0x04)	456
3.6	AMP Change Response (Code 0x05)	457
3.7	AMP Get Info Request (Code 0x06)	457
3.8	AMP Get Info Response (Code 0x07)	458
3.9	AMP Get AMP Assoc Request (Code 0x08)	460
3.10	AMP Get AMP Assoc Response (Code 0x09).....	460
3.11	AMP Create Physical Link Request (Code 0x0A).....	461
3.12	AMP Create Physical Link Response (Code 0x0B).....	462
3.13	AMP Disconnect Physical Link Request (Code 0x0C)	465
3.14	AMP Disconnect Physical Link Response (Code 0x0D).....	465
3.15	Create Physical Link Collision Resolution	467
3.16	Response Timeout.....	467
3.17	Unexpected BR/EDR Physical Link Disconnect	467

1 INTRODUCTION

1.1 GENERAL DESCRIPTION

The AMP Manager Protocol (A2MP) provides a means for one device to solicit information regarding AMP capabilities from another device. Each device contains an abstract entity called an AMP Manager which uses the AMP Manager Protocol to communicate with a peer AMP Manager on another device.

2 GENERAL OPERATION

2.1 BASIC CAPABILITIES

The AMP Manager entity has the following responsibilities and capabilities:

1. Ability to discover remote AMP Managers
2. Ability to discover available Controllers
3. Ability to query remote AMP Controller information
4. Ability to manage AMP physical links
5. Responsible for the creation of dedicated AMP keys

The AMP Manager communicates with the local AMP PAL and communicates with remote AMP Managers using the AMP Manager Protocol over a fixed L2CAP channel. The AMP Manager entity exists within the Bluetooth stack as depicted in [Figure 2.1](#).

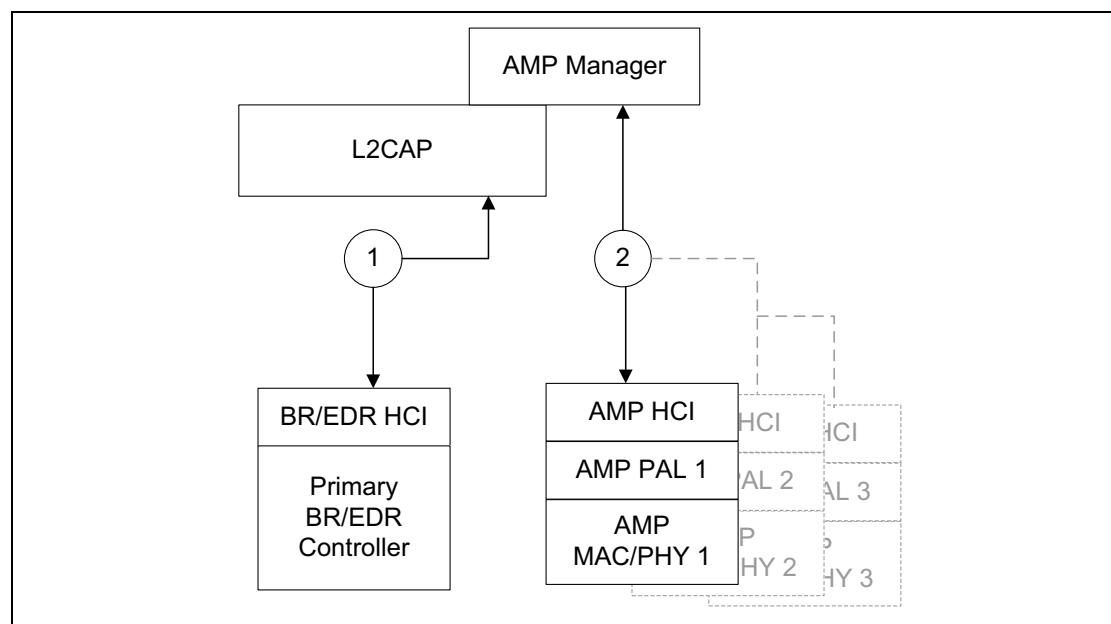


Figure 2.1: Overview of the Lower Software Layers

[Figure 2.1](#) shows the two basic communication paths for the AMP Manager. The AMP Manager uses path 1 via L2CAP and the Primary BR/EDR Controller to first discover remote AMP Managers. The AMP Manager can query for the possibility of available remote AMPs and their capabilities as well as other information. The AMP Managers communicate over the AMP Manager Protocol Channel (A2MP Channel). See [Section 2.2](#) for more information on the A2MP Channel.

After discovering the AMPs supported by the remote device, the AMP Manager uses communication path 2 to manage the physical links between its local AMP(s) and AMPs on remote devices. Note that there can be more than one local AMP and there also may be more than one AMP located on the remote device.

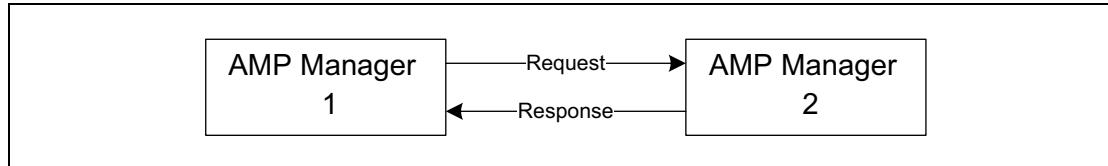


Figure 2.2: AMP Manager Peers

The AMP Manager Protocol is a request / response protocol between two AMP Managers. Requests and responses go in both directions. Either peer can request the same category of information from its corresponding peer.

2.2 AMP MANAGER CHANNEL OVER L2CAP

Peer AMP Managers communicate over the A2MP Channel which is an L2CAP fixed channel. The configuration parameters for the AMP Manager channel shall be as shown below. The MTU value shall be any valid value greater than or equal to 670 bytes.

Parameter	Value
MTU	≥ 670
Flush Timeout	0xFFFF (Infinite)
QoS	Best Effort
Mode	Enhanced Retransmission Mode
FCS Option	16-bit FCS

Table 2.1: AMP Manager Channel configuration parameters

The parameters for the Enhanced Retransmission Mode used on the fixed channel shall be as shown below. The Retransmission Time-out, Monitor Time-out and MPS shall be any valid value that is greater than or equal to the values stated in the following tables.

Parameter	Value
TxWindow size	1
Max Transmit	0xff
Retransmission time-out	≥ 2 seconds
Monitor time-out	≥ 12 seconds

Table 2.2: AMP Manager Channel Enhanced Retransmission Mode parameters

Parameter	Value
Maximum PDU size (MPS)	≥ 670
16-bit FCS	Enabled

Table 2.2: AMP Manager Channel Enhanced Retransmission Mode parameters

The minimum value for both MTU and MPS is 670 bytes. A larger MTU and MPS size can be communicated using the AMP Discover request (see section 3.3) and AMP Discover response (see [Section 3.4](#)) packets. The MTU parameter and the Enhanced Retransmission mode MPS parameter are numerically equal to allow an AMP manager to send any A2MP packet using a single L2CAP PDU. A device may segment any A2MP packet into multiple L2CAP PDUs.

2.3 USING THE AMP MANAGER PROTOCOL

The following text describes how the AMP Manager Protocol is used to establish an AMP physical link between two devices.

2.3.1 Discovering a Remote AMP Manager

Once an ACL connection is established over the Primary BR/EDR Controller, the local AMP Manager shall examine the L2CAP extended features bits of the remote device to determine if L2CAP fixed channels are supported (see [\[Part A\], Logical Link Control and Adaptation Protocol Specification, Section 4.12 on page 74](#)). If so, a query for the existence of its peer (AMP Manager) may be made by issuing an Information Request for all available fixed channels.

2.3.2 Discovering Available Controllers on a Remote Device

The next step is to discover the existence and capabilities of the AMP Controllers of the peer. An AMP Discover Request packet can be issued over the AMP Manager Protocol channel for this purpose.

In an AMP Discover Response packet, the peer AMP Manager will return information about its local AMP capabilities in the form of a Controller List. A Controller List is a sequence of Controller ID/Controller Type/Controller Status triples identifying the Controllers that are available for communication.

The Controller List contains a basic top level description of the available peer Controllers and their status. The AMP Manager parses this list for physical radio compatibility. If more information is required about a remote AMP Controller, the local AMP Manager can subsequently issue an AMP Get Info Request packet using the Controller ID retrieved via the AMP Discover exchange. An AMP Get Info Response packet is returned from the peer with AMP Controller Info for the specified AMP Controller. The AMP Controller Info contains information about the AMP Controller that can be used by the host to determine whether or not to create a physical link to the AMP Controller.

2.3.3 Creation of AMP Physical Links

Once it has been determined that a Bluetooth connection over an AMP is desired, the AMP Manager retrieves the remote AMP Controller's AMP_Assoc structure by issuing an AMP Get AMP Assoc Request packet. It then passes the AMP_Assoc structure received from the remote device to its local AMP PAL and signals it to begin the physical discovery and connection process. In devices that support HCI this is done via the HCI_Create_Physical_Link command followed by the HCI_Write_Remote_AMP_ASSOC command.

Note that the AMP Manager only holds the contents of the AMP_Assoc structure for use in AMP Manager operations. The AMP Manager does not directly attempt to interpret the contents of the AMP_Assoc structure.

The AMP manager is responsible for generating the Physical Link Handle used in the HCI_Create_Physical_Link and HCI_Accept_Physical_Link commands. See section [\[Vol 2 Part E\] Section 5.3.2 on page 469](#) for details. When the AMP PAL is ready it will generate an HCI_Channel_Selected event which is a signal to read the local AMP_Assoc structure using the HCI_Read_Local_AMP_ASSOC command.

The AMP Manager shall not create or accept a physical link over an AMP if mutual authentication was not performed or encryption was not enabled over BR/EDR.

When the AMP Manager does not have a dedicated AMP key for the selected Controller type between the two devices and the BR/EDR link key type is not "Debug", the AMP Manager shall call the Secure Simple Pairing h2 function (see [\[Vol 2 Part H\] Section 7.7.5.2 on page 1355](#)) to generate a dedicated AMP key. This dedicated AMP key shall be used during subsequent connections.

If the Bluetooth device supports more than one AMP type, the AMP Manager shall call the h2 function a second time to regenerate Key_GAMP as defined in [\[Vol 2 Part H\] Section 7.7.5.3 on page 1356](#) before generating a Dedicated AMP link key for a different AMP type.

When the BR/EDR link key type is set to "Debug", the AMP Manager shall use Key_GAMP directly as the key for the AMP regardless of the AMP Type. The key length, key type and the link key itself are passed to the AMP Controller in the HCI_Create_Physical_Link command.

After receiving the HCI_Channel_Selected event and reading the local AMP_Assoc structure the AMP Manager then issues an AMP Create Physical Link Request packet to send the local AMP_Assoc structure to the peer device in order to engage the peer device in the AMP physical link creation.

If the peer accepts the request it issues an HCI_Accept_Physical_Link command to its local AMP and passes the AMP_Assoc structure received in the AMP Create Physical Link Request packet to its local AMP Controller. The

peer responds with an AMP Create Physical Link Response packet to either accept or reject the request to create a physical link.

Both hosts will receive an HCI_Physical_Link_Complete event indicating the completion of the physical link creation.

On success, the procedure for creation of a logical connection can then commence.

2.4 CONTROLLER IDS

Each Controller in a device is assigned a one-octet unique identifier by the local AMP Manager. From the perspective of a remote device, Controller IDs are only valid during the life time of the AMP Manager Protocol channel. The Controller IDs on a remote device become invalid when the A2MP Channel is disconnected. If a local Controller becomes unavailable during the life time of the A2MP channel, the Controller ID may be reclaimed after receiving the AMP Change Response to the AMP Change Notify packets sent to all affected remote AMP Managers. It can then be assigned to any new AMP that is added. The value 0 is reserved for the Primary BR/EDR Controller. Other Controllers in the device shall be assigned values from 1 to 255.

If an AMP Controller becomes unavailable and later becomes available again during the life time of the A2MP Channel, remote AMP Managers shall not assume that the AMP Controller will be assigned the Controller ID that it was previously assigned.

2.5 CONTROLLER TYPES

Each type of Controller is assigned a unique one-octet value. These values are defined in [Assigned Numbers](#). A device may have multiple AMPs of the same type.

3 PROTOCOL DESCRIPTION

The AMP Manager Protocol defines the procedures and format of the packets used to exchange information between two peer AMP Managers.

3.1 PACKET FORMATS

This section describes the packets used in the AMP Manager Protocol. Unless otherwise stated an implementation shall include all specified fields in a packet.

[Figure 3.1](#) displays the general format of all AMP Manager Protocol packets.

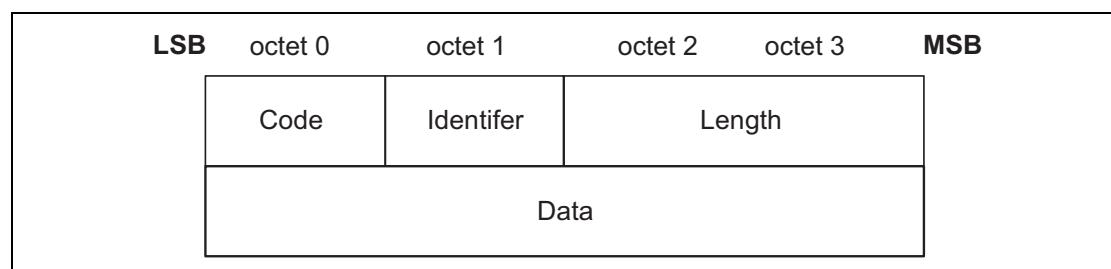


Figure 3.1: AMP Manager Protocol Packet Format

The fields shown are:

- **Code (1 octet)**
The code identifies the type of packet. When an AMP manager receives a packet with an unknown Code field it shall send a Command Reject packet in response. [Table 3.1](#) lists the codes defined by this document. All codes are specified with the most significant bit in the left most position.

Code	Description
0x00	Reserved
0x01	AMP Command Reject
0x02	AMP Discover request
0x03	AMP Discover response
0x04	AMP Change notify
0x05	AMP Change response
0x06	AMP Get Info request
0x07	AMP Get Info response
0x08	AMP Get AMP Assoc request
0x09	AMP Get AMP Assoc response

Table 3.1: Codes

Code	Description
0x0A	AMP Create Physical Link request
0x0B	AMP Create Physical Link response
0x0C	AMP Disconnect Physical Link request
0x0D	AMP Disconnect Physical Link response
0x0E - 0xFF	Reserved

Table 3.1: Codes

- *Identifier (1 octet)*

The Identifier field matches responses with requests. The requesting device sets this field and the responding devices uses the same value in the response. For each A2MP channel an AMP manger shall use different identifiers for each request sent (this includes the AMP Change Notify command). Following the original transmission of an identifier in a request, the identifier may be recycled if all other identifiers have subsequently been used.

A response packet with an invalid identifier is discarded. Identifier 0x00 is an illegal identifier and shall never be used in any packet. [Table 3.2](#) shows the valid values for the Identifier field,

Value	Description
0x00	Invalid
0x01 - 0xFF	Valid

Table 3.2: Identifier

- *Length (2 octets)*

The Length field indicates the size in octets of the data field of the packet only, i.e. it does not cover the Code, Identifier and Length fields.

- *Data (0 or more octets)*

The Data field is variable in length. The Code field determines the format of the Data field. The Length field determines the length of the Data field.

3.2 AMP COMMAND REJECT (CODE 0X01)

An AMP Manager shall send an AMP Command Reject packet in response to a request packet with an unknown code or when sending the corresponding response is inappropriate. The identifier shall match the identifier of the request packet being rejected. AMP Command Reject packets shall not be sent in response to an unknown response packet.

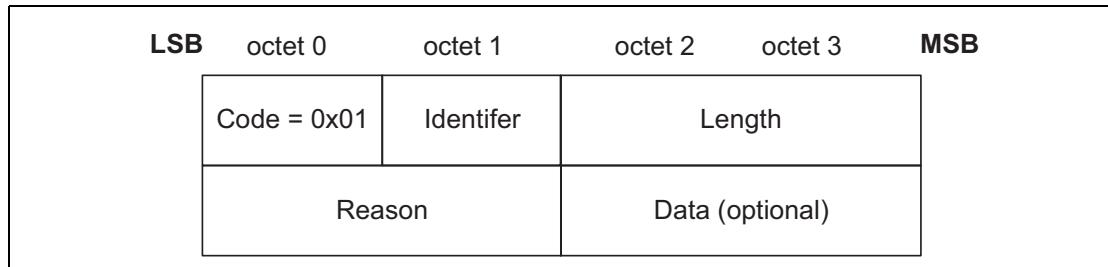


Figure 3.2: AMP Command Reject Packet

The data fields are

- *Reason (2 octets)*

The Reason field describes why the request packet was rejected, and is set to one of the following Reason codes.

Reason value	Description
0x0000	Command not recognized
Other	Reserved

Table 3.3: AMP Command Reject reasons

- *Data (0 or more octets)*

The length and content of the Data field depends on the Reason code. If the reason code is 0x0000 "Command not recognized" no Data field is used.



3.3 AMP DISCOVER REQUEST (CODE 0X02)

An AMP Manager sends an AMP Discover Request packet to a peer AMP Manager to obtain the list of the available Controllers supported by the peer device.

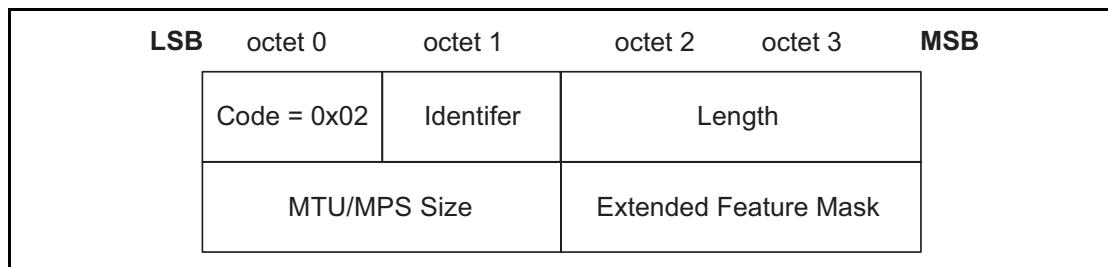


Figure 3.3: AMP Discover Request Packet

The data fields are

- *MTU/MPS size (2 octets)*

The MTU/MPS Size field in the AMP Discover Request packet indicates the MTU/MPS that the sender of the request can receive on the AMP Manager Protocol Channel. The minimum value is 670 (see [Section 2.2](#)). An AMP manager shall send the same value for the AMP Manager Protocol Channel MTU/MPS in both AMP Discover Request and AMP Discover Response packets.

- *Extended Feature Mask* (2 octets)

The extended features supported by the AMP Manager are represented by a bit mask. Each feature is represented by a single bit which shall be set to 1 if the feature is supported and set to 0 otherwise. All reserved feature bits shall be set to 0 by the sender.

The feature mask is defined in Table 3.4.

Supported Feature	Octet	Bit
Reserved	0	0-7
Reserved	1	0-6
Extension Bit	1	7

Table 3.4: A2MP Extended Feature Mask

Bit 7 of Octet 1 of this field is used to extend the A2MP Extended Feature Mask. If the Extension Bit is set then the Extended Feature Mask field is extended by two octets. Each two octet extension includes an Extension Bit that allows a further two octets to be added to the field. [Figure 3.4](#) shows how the Extension Bit (E) can be used to extend the A2MP Extended Feature Mask.

LSB	octet 0	octet 1	MSB
	Extended Feature Mask (Octet 0)	Extended Feature Mask (Octet 1)	E=1
	Extended Feature Mask (Octet 2)	Extended Feature Mask (Octet 3)	E=1
	Extended Feature Mask (Octet 4)	Extended Feature Mask (Octet 5)	E=0

Figure 3.4: Example A2MP Extended Feature Mask

3.4 AMP DISCOVER RESPONSE (CODE 0X03)

When an AMP Manager receives an AMP Discover Request packet it shall reply with an AMP Discover Response packet indicating the IDs, types and status of the controllers that are currently available on the local device. If there is a change in the number or status of supported controllers after sending an AMP Discover Response packet and before the A2MP Channel is disconnected, the AMP Manager shall send an AMP Change Notify packet to the peer AMP Manager.

LSB	octet 0	octet 1	octet 2	octet 3	MSB
	Code = 0x03	Identifier	Length		
	MTU/MPS Size		Extended Feature Mask		
	Controller List				

Figure 3.5: AMP Discover Response Packet

The data fields are

- *MTU/MPS size (2 octets)*

The MTU/MPS Size field in the AMP Discover Response packet indicates the MTU/MPS that the sender of the response can receive on the AMP Manager Protocol Channel. The minimum value is 670 (see [Section 2.2](#)). An AMP Manager shall send the same value for the AMP Manager Protocol Channel MTU/MPS supported in both AMP Discover Request and AMP Discover Response packets.

- *Extended Feature Mask (2 octets)*

The extended features supported by the AMP Manager are represented by a bit mask. Each feature is represented by a single bit which shall be set to 1

if the feature is supported and set to 0 otherwise. All unknown, reserved, or unassigned feature bits shall be set to 0.

The feature mask is defined in [Table 3.4](#) and [Figure 3.4](#).

- *Controller List (3 or more octets)*

Controller List is variable in length. It consists of 1 or more entries. Each entry is comprised of a Controller ID, a Controller Type and a Controller status.

An entry for Controller ID 0x00 (Primary BR/EDR controller) shall always be sent and shall be the first entry in the Controller List. The Controller Type for this entry shall be set to 0x00 (Primary BR/EDR controller). The Controller status for this entry shall be set to 0x01 (The AMP Controller can only be used by Bluetooth technology). If the Primary BR/EDR Controller's Link Manager does not support Secure Simple Pairing, it shall be the only Controller in the Controller List.

[Figure 3.6](#) displays the format of the Controller list.

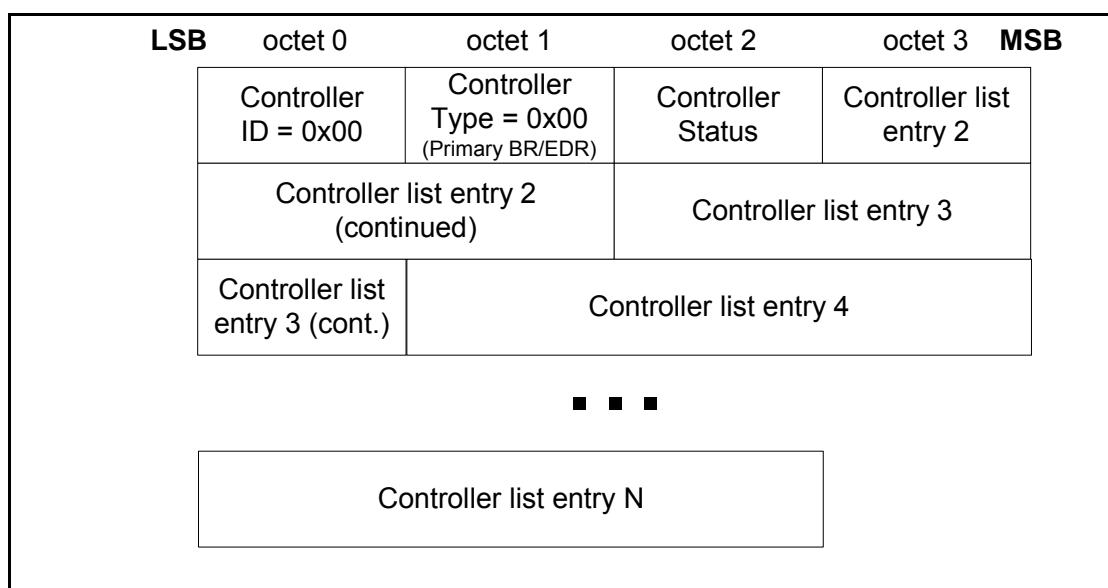


Figure 3.6: Controller list format

- *Controller ID (1 octet)*

The Controller ID uniquely identifies a Controller (see [Section 2.4](#) for more information about Controller IDs).

- *Controller Type (1 octet)*

The Controller Type specifies the type of the Controller. The Controller type values are defined in [Assigned Numbers](#).

- *Controller Status (1 octet)*

The Controller Status field indicates the current status of the AMP. The status information can be used by the receiving device to help decide if it should use the AMP. If multiple AMPs are available the status can also be used by an implementation to determine which would be the optimal AMP to

use. None of the AMP status values indicate that a Physical Link shall not be created to the AMP. If an implementation wishes to ensure it does not receive Create Physical Link Requests for a given AMP then it should remove the associated entry from the Controller list it sends.

Table 3.5 contains the different values of the Controller Status field.

Value	Parameter Description
0x00	<p>The Controller radio is available but is currently physically powered down. This value should be used if the AMP Controller is present and can be powered up by the AMP Manager. A Controller List entry shall not exist for an AMP that is not present or cannot be powered up by the AMP manager.</p> <p>This value indicates that there may be a cost of time and power to use this AMP Controller (i.e., the time taken and power required to power up the AMP Controller). These costs are AMP type and AMP implementation dependent.</p>
0x01	<p>This value indicates that the AMP Controller is only used by the Bluetooth technology and will not be shared with other non-Bluetooth technologies.</p> <p>This value shall only be used if the AMP Controller is powered up.</p> <p>This value does not indicate how much bandwidth is currently free on the AMP Controller.</p>
0x02	<p>The AMP Controller has no capacity available for Bluetooth operation.</p> <p>This value indicates that all of the AMP Controller's bandwidth is currently allocated to servicing a non Bluetooth technology.</p> <p>A device may create a Physical Link to an AMP Controller that has this status.</p> <p>This value shall only be used if the AMP Controller is powered up.</p>
0x03	<p>The AMP Controller has low capacity available for Bluetooth operation.</p> <p>This value indicates that the majority of the AMP Controller's bandwidth is currently allocated to servicing a non Bluetooth technology.</p> <p>An AMP Controller with capacity in the approximate range of $0 < \text{capacity} < 30\%$ should indicate this value.</p> <p>This value does not indicate how much of the capacity available for Bluetooth operation is currently being used.</p> <p>This value shall only be used if the AMP Controller is powered up.</p>
0x04	<p>The AMP Controller has medium capacity available for Bluetooth operation.</p> <p>An AMP Controller with capacity in the approximate range of $30\% < \text{capacity} < 70\%$ should indicate this value.</p> <p>This value does not indicate how much of the capacity available for Bluetooth operation is currently being used.</p> <p>This value shall only be used if the AMP Controller is powered up.</p>

Table 3.5: Controller Status

Value	Parameter Description
0x05	<p>The AMP Controller has high capacity available for Bluetooth operation.</p> <p>This value indicates that the majority of the AMP Controller's bandwidth is currently allocated to servicing the Bluetooth technology.</p> <p>An AMP Controller with capacity in the approximate range of $70\% < \text{capacity} < 100\%$ should indicate this value.</p> <p>This value does not indicate how much of the capacity available for Bluetooth operation is currently being used.</p> <p>This value shall only be used if the AMP Controller is powered up.</p>
0x06	<p>The AMP Controller has full capacity available for Bluetooth operation.</p> <p>This value indicates that while currently the AMP is only being used by Bluetooth the device allows a different technology to share the radio.</p> <p>This value shall be used as the default Controller Status value for devices that are not capable of determining the available capacity for Bluetooth operation for an AMP Controller.</p> <p>This value does not indicate how much of the capacity available for Bluetooth operation is currently being used.</p> <p>This value shall only be used if the AMP Controller is powered up.</p>
Other	Reserved

Table 3.5: Controller Status

The available capacity ranges given for values 0x03, 0x04, and 0x05 are only approximate and permit an implementation to not send an AMP Change Notify packet if the current availability is moving frequently from one range to another (e.g., if the current availability is moving between 68% (corresponding to Controller status 0x04) and 72% (corresponding to Controller status 0x05) frequently an implementation is not mandated to send an AMP Change Notify packet every time the value crosses the approximate 70% threshold between the two Controller Status values).

3.5 AMP CHANGE NOTIFY (CODE 0X04)

The AMP Change Notify packet indicates to the peer device that something has changed in the list of supported Controllers. An AMP Change Notify packet shall only be sent to a remote AMP Manager if it has previously requested a Controller List via an AMP Discover Request. Examples of when an AMP Change Notify packet is sent is when a new Controller becomes available or one or several Controllers advertised in the previous AMP Discover Response or AMP Change Notify packet become unavailable or if the status of a previously advertised Controller changes. The AMP Change Notify carries the new list of the available Controllers. Based on the changes from the previous Change Notify or Discovery Response, the AMP Manager can identify added or deleted controllers or controllers that have changed status.

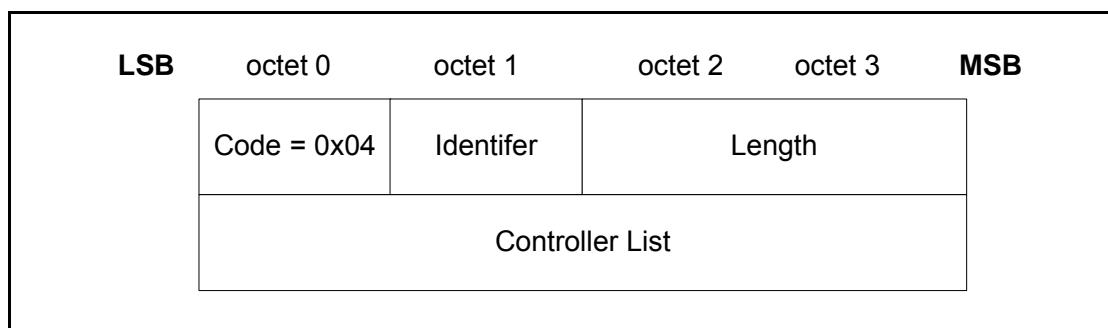


Figure 3.7: AMP Change Notify Packet

- *Controller List (3 or more octets)*

Controller List is variable in length. It consists of 1 or more entries. Each entry is comprised of a Controller ID, a Controller Type and a Controller status (see [Section 3.4](#) for a description of Controller List).

3.6 AMP CHANGE RESPONSE (CODE 0X05)

The AMP Change Response packet shall be sent to acknowledge receipt of the AMP Change Notify packet.

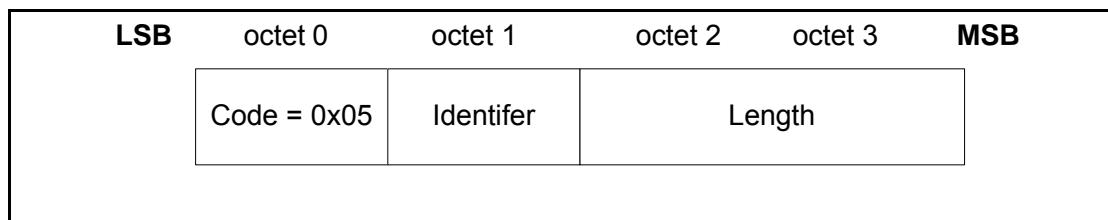


Figure 3.8: AMP Change Response Packet

3.7 AMP GET INFO REQUEST (CODE 0X06)

To determine if a physical link should be established to a particular AMP Controller, an AMP Manager may need to obtain further information for a remote AMP Controller. For each Controller ID reported in the AMP Discover Response or AMP Change Notify packet the AMP Manager can query this information by issuing an AMP Get Info Request packet. The exception is Controller ID 0. Information for the Primary BR/EDR radio shall not be obtained with an AMP Get Info Request packet.

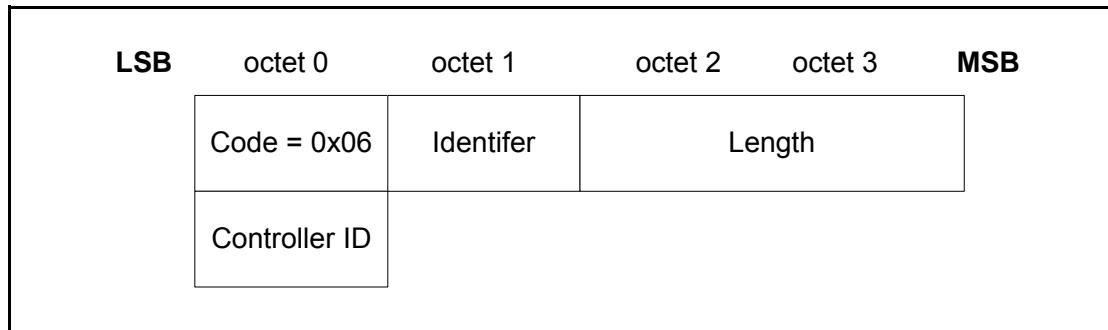


Figure 3.9: AMP Get Info Request Packet

The data fields are

- *Controller ID (1 octet)*

Controller ID is one of the IDs reported in the latest AMP Discover Response, or AMP Change Notify packet received from the peer AMP Manager.

3.8 AMP GET INFO RESPONSE (CODE 0X07)

When an AMP Manager receives an AMP Get Info Request packet it shall reply with the AMP Get Info Response packet. This packet carries Controller Information including bandwidth and latency capabilities. The receiver of the AMP Get Info Response packet can assume that the value of all fields shall remain constant while the remote AMP remains active.

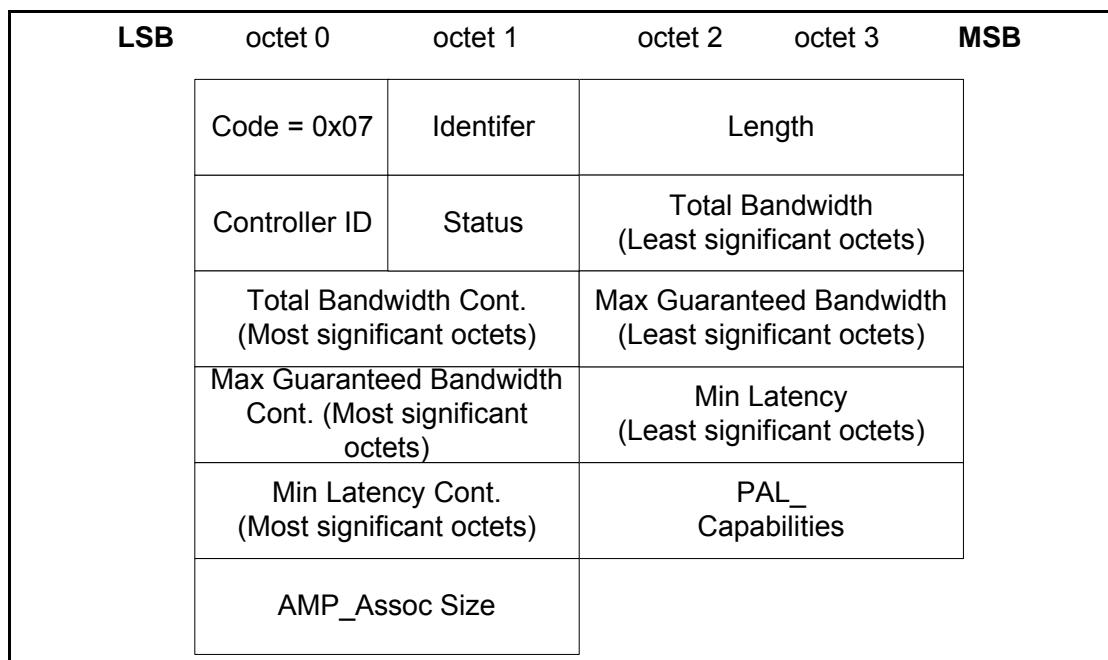


Figure 3.10: AMP Get Info Response Packet

- *Controller ID (1 octet)*

Controller ID is the identifier requested in the AMP Get Info Request.

- *Status (1 octet)*

The Status field indicates the status of the request

Status Code	Description
0x00	Success
0x01	Invalid Controller ID
Other	Reserved

Table 3.6: AMP Get Info Response Status values

An AMP Get Info Response with status of Invalid Controller ID shall be sent by an AMP manager that has received an AMP Get Info Request containing either:

- a) A Controller ID that has not been allocated by the AMP manager
- b) A Controller ID of 0

If the Status field is set to Invalid Controller ID all subsequent fields in the AMP Get Info Response shall be ignored by the receiver.

- *Total Bandwidth (4 octets)*

The Total Bandwidth field indicates the total data rate in kilobits per second (1000 b/s) that can be achieved by the Controller for applications. This value shall account for any bandwidth limitations of the Host and the HCI transport if present. The Host is responsible for determining these bandwidth limitations.

- *Max Guaranteed Bandwidth (4 octets)*

The Maximum Guaranteed Bandwidth field indicates the maximum data rate in kilobits per second that the AMP can guarantee for a logical channel. Any request made by an application above this level will be rejected. This value shall account for any bandwidth limitations of the Host and the HCI transport if present. The Host is responsible for determining these bandwidth limitations.

- *Min Latency (4 octets)*

The Min Latency field indicates the minimum latency in microseconds that the AMP can guarantee for a logical channel. This value shall account for any latency limitations of the Host and the HCI transport if present. The Host is responsible for determining these latency limitations.

- *PAL_Capabilities (2 octets)*

The PAL_Capabilities field contains the PAL capabilities returned via HCI Read Local AMP Info Command (see [\[Vol 2 Part E\] Section 7.5.8 on page 814](#) for more information).

- *AMP_Assoc Structure Size (2 octets)*

The maximum size in octets of the requested AMP's AMP_Assoc structure. The value of the AMP_Assoc Structure Size for a given AMP is determined by the controller. If an HCI is being used this value can be retrieved by the Host sending the HCI_Read_Local_AMP_Info command to the Controller.

3.9 AMP GET AMP ASSOC REQUEST (CODE 0X08)

To establish a physical link to a particular Controller, an AMP Manager needs to obtain the AMP_Assoc structure for the remote Controller. The remote AMP is identified by its Controller ID as reported in the AMP Discover Response or AMP Change Notify packet. AMP Get AMP Assoc Request packet shall not be used to obtain an AMP_Assoc structure for the Primary BR/EDR Controller.

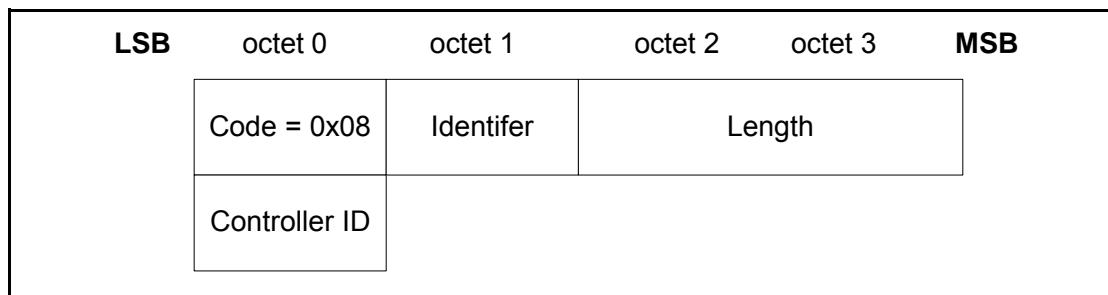


Figure 3.11: AMP Get AMP Assoc Request Packet

The data fields are:

- *Controller ID (1 octet)*

Controller ID is one of the IDs reported in the latest AMP Discover Response or AMP Change Notify packet received from the peer AMP Manager.

3.10 AMP GET AMP ASSOC RESPONSE (CODE 0X09)

When an AMP Manager receives an AMP Get AMP Assoc Request packet it shall reply with the AMP Get AMP Assoc Response packet. This packet carries the AMP_Assoc structure. The AMP_Assoc structure is used in creating the physical AMP link. The exact length and format of the AMP_Assoc structure shall be defined in each AMP PAL specification.

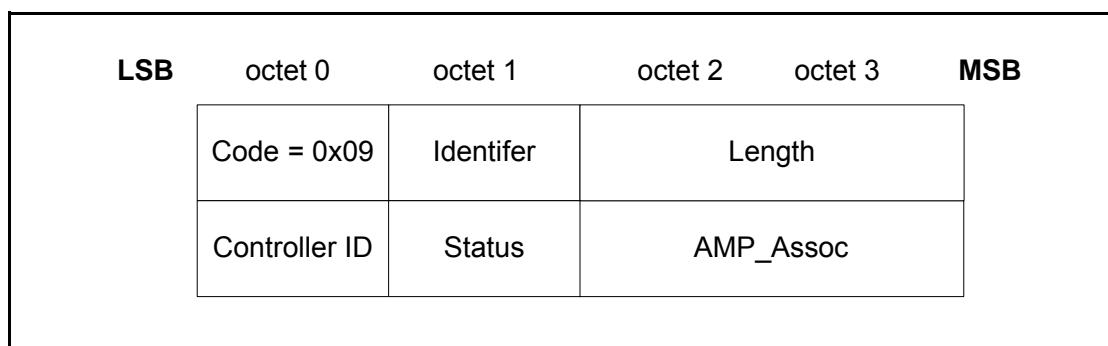


Figure 3.12: AMP Get AMP Assoc Response Packet

The data fields are

- *Controller ID (1 octet)*

Controller ID is the identifier requested in the AMP Get AMP Assoc Request packet.

- *Status (1 octet)*

The Status field indicates the status of the request.

Status Code	Description
0x00	Success
0x01	Invalid Controller ID
Other	Reserved

Table 3.7: AMP Get AMP Assoc Response Status values

An AMP Get AMP Assoc Response packet with status of Invalid Controller ID shall be sent by an AMP manager that has received an AMP Get AMP Assoc Request packet containing either

- a) A Controller ID that has not been allocated by the AMP manager
- b) A Controller ID of 0

If the Status field is set to Invalid Controller ID (0x01) then the AMP_Assoc structure shall not be included in the Get AMPAssoc Response packet.

- *AMP_Assoc structure (from 0 to (A2MP MTU - 6) octets)*

AMP_Assoc structure is variable in length. The Controller Type of the Controller ID referenced in the AMP Get AMP Assoc Request packet determines the format and the length of the AMP_Assoc structure. The length of the AMP_Assoc structure can be determined by subtracting two from the Length field.

The length of the AMP_Assoc structure shall not exceed any previously communicated AMP_Assoc length (the AMP_Assoc length is included in the AMP Get Info Response packet. See [Section 3.8](#)).

3.11 AMP CREATE PHYSICAL LINK REQUEST (CODE 0X0A)

Establishing an AMP physical link requires signaling between the AMP Managers in order to turn the AMP Controller on, if it is not already turned on, and to provide information to set the required state of the AMP Controller, setup addressing, security, etc. When an AMP Manager receives an AMP Create Physical Link request packet it shall initiate its part of the AMP physical link creation and send the AMP Create Physical Link Response packet. AMP Create Physical Link request shall not be used to establish a physical link to the Primary BR/EDR Controller.

For some AMP Controllers it may be required to send additional information to the peer AMP Manager. This information is packaged in a variable size

AMP_Assoc structure. The exact length and format of the AMP_Assoc structure shall be defined in each AMP PAL specification.

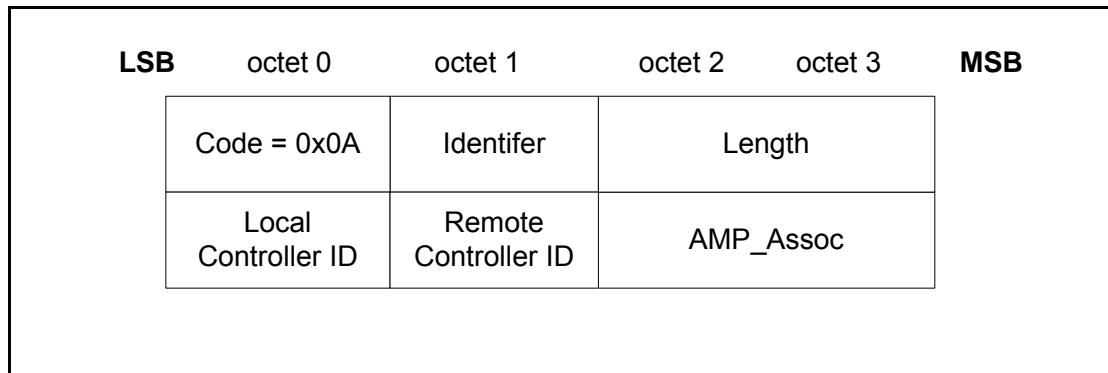


Figure 3.13: AMP Create Physical Link Request Packet

- *Local Controller ID (1 octet)*
Local Controller ID is the ID of the Controller to be used on the device sending the AMP Create Physical Request packet.
- *Remote Controller ID (1 octet)*
The ID of the AMP Controller to use on the device that receives the AMP Create Physical Link Request packet. The sender of the AMP Create Physical Link Request packet obtains the Remote Controller ID from the latest Discover Response, or the Change Notify packet received from the peer AMP Manager.
- *AMP_Assoc structure (from 0 to (A2MP MTU - 6) octets)*
AMP_Assoc structure is variable in length. See section 3.10 for a description of AMP_Assoc structure.

3.12 AMP CREATE PHYSICAL LINK RESPONSE (CODE 0X0B)

For each AMP Create Physical Link Request the AMP Manager shall reply with an AMP Create Physical Link Response packet. If the device has sent its own AMP Create Physical Link Request to create a physical link with the same AMP Controller a collision has occurred and one of the requests shall be rejected while the other request shall proceed. The algorithm described in [Section 3.15](#) shall be used to determine which request to reject.

When the AMP Manager receives an AMP Create Physical Link Request packet it shall try to perform the requested actions and send a response identifying the status of the operation.

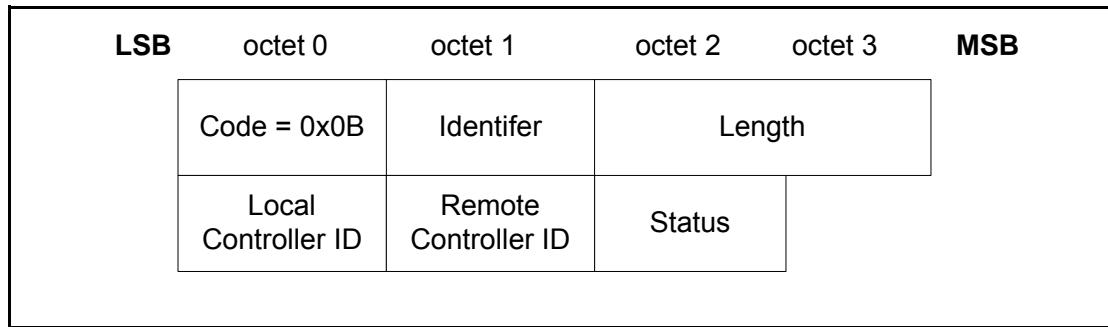


Figure 3.14: AMP Create Physical Link Response Packet

The data fields are

- *Local Controller ID (1 octet)*

This field contains the Controller ID local to the sender of the AMP Create Physical Link Response packet. The ID value is the same as the Remote Controller ID that was in the associated AMP Create Physical Link Request packet.

- *Responder Controller ID (1 octet)*

This field contains the Controller ID used by the receiver of the AMP Create Physical Link Response packet. The ID value is the same as the Local Controller ID that was in the associated AMP Create Physical Link Request packet.

- *Status (1 octet)*

The Status field indicates the result of the request.

Status Code	Description
0x00	Success - Link creation is started
0x01	Invalid Controller ID
0x02	Failed - Unable to start link creation
0x03	Failed - Collision Occurred
0x04	Failed - AMP Disconnected Physical Link Request packet received
0x05	Failed - Physical Link Already Exists
0x06	Failed - Security violation
Other	Reserved

Table 3.8: AMP Create Physical Link Response Status values

An AMP Create Physical Link Response with status of Invalid Controller ID shall be sent by an AMP manager that has received an AMP Create Physical Link Request containing either

- a) A Controller ID that has not been allocated by the AMP manager
- b) A Controller ID of 0

An AMP Create Physical Link Response with status of Failed - Security Violation shall be sent by an AMP manager that has received an AMP Create Physical Link Request without mutual authentication being performed over the BR/EDR Controller and without encryption being enabled on the BR/EDR Controller.

3.13 AMP DISCONNECT PHYSICAL LINK REQUEST (CODE 0X0C)

AMP Disconnect Physical Link request is an optional request that can be used by either initiator or responder to either abort the creation of an AMP physical link or disconnect an existing Physical Link.

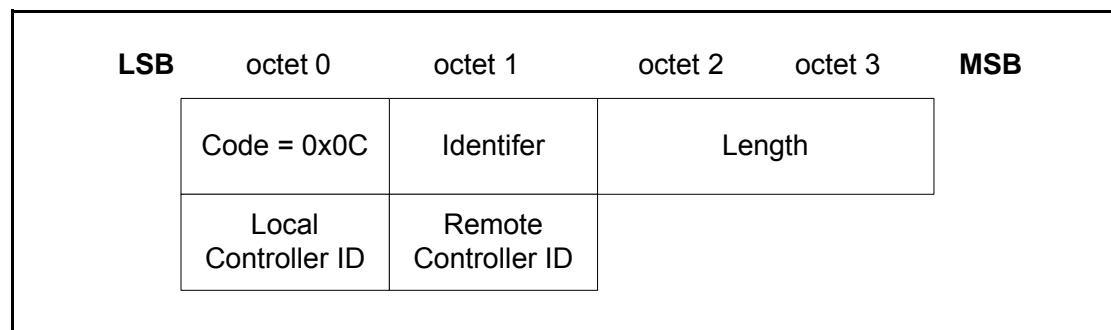


Figure 3.15: AMP Disconnect Physical Link Request Packet

The data fields are:

- *Local Controller ID (1 octet)*

Local Controller ID is the ID of the Controller on the device sending the AMP Disconnect Physical Link Request packet.

- *Remote Controller ID (1 octet)*

The ID of the AMP Controller on the device that receives the AMP Disconnect Physical Link Request packet.

3.14 AMP DISCONNECT PHYSICAL LINK RESPONSE (CODE 0X0D)

If an AMP Manager receives an AMP Disconnect Physical Link request packet while creation of a physical link is outstanding, it shall abort the operation. In devices that support HCI the Host may send an `HCI_Disconnect_Physical_Link` command to cause the controller to abort the creation of a Physical Link.

If an AMP Disconnect Physical Link request packet is received from a device that has an AMP Create Physical Link request packet outstanding, the receiver shall send both an AMP Create Physical Link Response packet with a status of 'Failed - AMP Disconnected Physical Link Request packet received' and an AMP Disconnect Physical Link response packet with a status of 'Success'.

If an AMP manager receives an AMP Disconnect Physical Link request packet when creation of a physical link is not outstanding and a Physical Link does not exist for the given local and remote AMP IDs it shall send an AMP Disconnect Physical Link Response with a status of 'Failed - No Physical Link exists and no Physical Link creation is in progress'.

The status of 'Invalid Controller ID' shall only be used if the receiver of the AMP Disconnect Request packet does not have an AMP with an ID that matches the Remote Controller ID in the received AMP Disconnect Request packet.

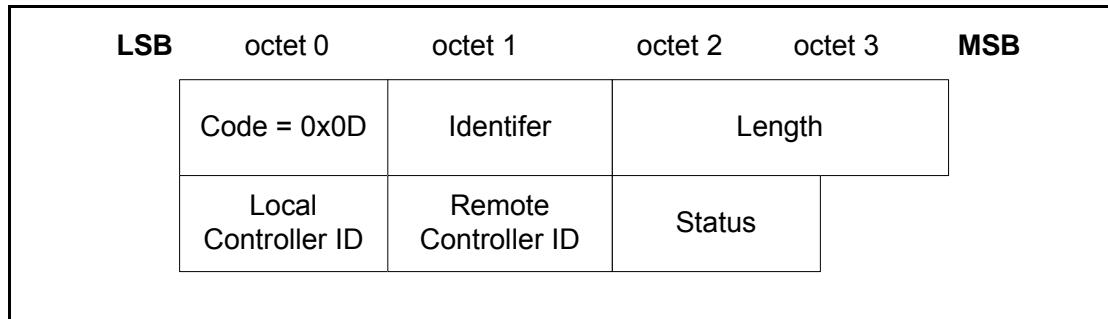


Figure 3.16: AMP Disconnect Physical Link Response Packet

The data fields are:

- *Local Controller ID (1 octet)*

This field contains the Controller ID local to the sender of the AMP Disconnect Physical Link Response packet. The ID value is the same as the Remote Controller ID that was in the associated AMP Disconnect Physical Link Request packet.

- *Remote Controller ID (1 octet)*

This field contains the Controller ID used by the receiver of the AMP Disconnect Physical Link Response packet. The ID value is the same as the Local Controller ID that was in the associated AMP Disconnect Physical Link Request packet.

- *Status (1 octet)*

The Status field indicates the result of the request.

Status Code	Description
0x00	Success
0x01	Invalid Controller ID
0x02	Failed - No Physical Link exists and no Physical Link creation is in progress
Other	Reserved

Table 3.9: AMP Disconnect Physical Link Response Status values

3.15 CREATE PHYSICAL LINK COLLISION RESOLUTION

If two devices simultaneously attempt to create a physical link a collision will occur. In this situation one of the requests shall be canceled while the other request shall proceed. Both devices must know which request will be rejected. The following algorithm shall be used by both devices to determine which request to reject.

1. Set i=0 (representing the least significant octet of the BD_ADDR).
2. Compare the least significant octet, octet[i] of the BD_ADDR, of both devices. If the octets are not equal go to step 4.
3. Increment i by 1. Go to step 2.
4. The device with the larger BD_ADDR octet shall send an AMP Create Physical Link Response with the Status field set to "Failed - Collision Occurred."

3.16 RESPONSE TIMEOUT

The Response Timeout is used to monitor the response to request packets. The exact timeout value is implementation dependent and may be chosen within the range of 15 - 60 seconds. The recommended value is 20 seconds. If a response to a request packet is not received within the Response Timeout then the ACL link shall be disconnected.

3.17 UNEXPECTED BR/EDR PHYSICAL LINK DISCONNECT

It is possible that the BR/EDR link between two devices might unexpectedly disconnect (Link Supervision Timeout) while AMP links between the same two devices remain active. This situation is problematic since the BR/EDR link is used for L2CAP and AMP Manager signaling between the two devices.

When the BR/EDR link with a remote device unexpectedly disconnects due to Link Supervision Timeout the AMP manager shall disconnect all AMP physical links to the same remote device.

Core System Package [Host volume]

Part F

ATTRIBUTE PROTOCOL (ATT)

This specification defines the Attribute Protocol; a protocol for discovering, reading, and writing attributes on a peer device

CONTENTS

1	Introduction	471
1.1	Scope	471
1.2	Conformance	471
2	Protocol Overview	472
3	Protocol Requirements	473
3.1	Introduction	473
3.2	Basic Concepts.....	473
3.2.1	Attribute Type.....	473
3.2.2	Attribute Handle	473
3.2.3	Attribute Handle Grouping	474
3.2.4	Attribute Value.....	474
3.2.5	Attribute Permissions.....	474
3.2.6	Control-Point Attributes.....	476
3.2.7	Protocol Methods	476
3.2.8	Exchanging MTU Size	476
3.2.9	Long Attribute Values.....	476
3.2.10	Atomic Operations	477
3.3	Attribute PDU	477
3.3.1	Attribute PDU Format.....	478
3.3.2	Sequential Protocol.....	479
3.3.3	Transaction	480
3.4	Attribute Protocol PDUs.....	481
3.4.1	Error Handling.....	481
3.4.1.1	Error Response	481
3.4.2	MTU Exchange	483
3.4.2.1	Exchange MTU Request	483
3.4.2.2	Exchange MTU Response	483
3.4.3	Find Information	485
3.4.3.1	Find Information Request	485
3.4.3.2	Find Information Response	485
3.4.3.3	Find By Type Value Request	487
3.4.3.4	Find By Type Value Response	488
3.4.4	Reading Attributes	489
3.4.4.1	Read By Type Request	489
3.4.4.2	Read By Type Response.....	491
3.4.4.3	Read Request	492
3.4.4.4	Read Response.....	492
3.4.4.5	Read Blob Request	493

Attribute Protocol (ATT)

3.4.4.6	Read Blob Response	494
3.4.4.7	Read Multiple Request	495
3.4.4.8	Read Multiple Response	496
3.4.4.9	Read by Group Type Request.....	496
3.4.4.10	Read by Group Type Response	498
3.4.5	Writing Attributes.....	499
3.4.5.1	Write Request.....	499
3.4.5.2	Write Response.....	500
3.4.5.3	Write Command.....	501
3.4.5.4	Signed Write Command	502
3.4.6	Queued Writes	503
3.4.6.1	Prepare Write Request.....	503
3.4.6.2	Prepare Write Response	505
3.4.6.3	Execute Write Request.....	505
3.4.6.4	Execute Write Response	506
3.4.7	Server Initiated.....	507
3.4.7.1	Handle Value Notification	507
3.4.7.2	Handle Value Indication.....	507
3.4.7.3	Handle Value Confirmation.....	508
3.4.8	Attribute Opcode Summary.....	509
3.4.9	Attribute PDU Response Summary	511
4	Security Considerations	514

1 INTRODUCTION

1.1 SCOPE

The attribute protocol allows a device referred to as the server to expose a set of attributes and their associated values to a peer device referred to as the client. These attributes exposed by the server can be discovered, read, and written by a client, and can be indicated and notified by the server.

1.2 CONFORMANCE

If conformance to this protocol is claimed, all capabilities indicated as mandatory for this protocol shall be supported in the specified manner (process-mandatory). This also applies to all optional and conditional capabilities for which support is indicated. All mandatory capabilities, and optional and conditional capabilities for which support is indicated, are subject to verification as part of the Bluetooth qualification program.

2 PROTOCOL OVERVIEW

The attribute protocol defines two roles; a server role and a client role. It allows a server to expose a set of attributes to a client that are accessible using the attribute protocol.

An attribute is a discrete value that has the following three properties associated with it:

- a) attribute type, defined by a UUID
- b) attribute handle
- c) a set of permissions that are defined by each higher layer specification that utilizes the attribute; these permissions cannot be accessed using the Attribute Protocol.

The attribute type specifies what the attribute represents. Bluetooth SIG defined attribute types are defined in the Bluetooth SIG assigned numbers page, and used by an associated higher layer specification. Non-Bluetooth SIG attribute types may also be defined.

The attribute handle uniquely identifies an attribute on a server, allowing a client to reference the attribute in read or write requests; see [Section 3.4.4](#), [Section 3.4.5](#), and [Section 3.4.6](#). It allows a client to uniquely identify the attribute being notified or indicated, see [Section 3.4.7](#). Clients are able to discover the handles of the server's attributes; see [Section 3.4.3](#). Permissions may be applied to an attribute to prevent applications from obtaining or altering an attribute's value. An attribute may be defined by a higher layer specification to be readable or writable or both, and may have additional security requirements. For more information, see [Section 3.2.5](#).

A client may send attribute protocol requests to a server, and the server shall respond to all requests that it receives. A device can implement both client and server roles, and both roles can function concurrently in the same device and between the same devices. There shall be only one instance of a server on each Bluetooth device; this implies that the attribute handles shall be identical for all supported bearers. For a given client, the server shall have one set of attributes. The server can support multiple clients. Note: multiple services may be exposed on a single server by allocating separate ranges of handles for each service. The discovery of these handle ranges is defined by a higher layer specification.

The attribute protocol has notification and indication capabilities that provide an efficient way of sending attribute values to a client without the need for them to be read; see [Section 3.3](#).

3 PROTOCOL REQUIREMENTS

3.1 INTRODUCTION

Each attribute has an attribute type that identifies, by means of a UUID (Universally Unique Identifier), what the attribute represents so that a client can understand the attributes exposed by a server. Each attribute has an attribute handle that is used for accessing the attribute on a server, as well as an attribute value.

An attribute value is accessed using its attribute handle. The attribute handles are discovered by a client using attribute protocol PDUs (Protocol Data Unit). Attributes that have the same attribute type may exist more than once in a server. Attributes also have a set of permissions that controls whether they can be read or written, or whether the attribute value shall be sent over an encrypted link. Security aspects of the attribute protocol are defined in [Section 4](#).

3.2 BASIC CONCEPTS

3.2.1 Attribute Type

A universally unique identifier (UUID) is used to identify every attribute type. A UUID is considered unique over all space and time. A UUID can be independently created by anybody and distributed or published as required. There is no central registry for UUIDs, as they are based off a unique identifier that is not duplicated. The attribute protocol allows devices to identify attribute types using UUIDs regardless of the local handle used to identify them in a read or write request.

Universal unique identifiers are defined in SDP [\[Vol 3\] Part B, Section 2.5.1](#).

All 32-bit Attribute UUIDs shall be converted to 128-bit UUIDs when the Attribute UUID is contained in an ATT PDU.

3.2.2 Attribute Handle

An attribute handle is a 16-bit value that is assigned by each server to its own attributes to allow a client to reference those attributes. An attribute handle shall not be reused while an ATT Bearer exists between a client and its server.

Attribute handles on any given server shall have unique, non-zero values. Attributes are ordered by attribute handle.

An attribute handle of value 0x0000 is reserved, and shall not be used. An attribute handle of value 0xFFFF is known as the maximum attribute handle.

Note: Attributes can be added or removed while an ATT Bearer is active, however, an attribute that has been removed cannot be replaced by another attribute with the same handle while an ATT Bearer is active.

3.2.3 Attribute Handle Grouping

Grouping is defined by a specific attribute placed at the beginning of a range of other attributes that are grouped with that attribute, as defined by a higher layer specification. Clients can request the first and last handles associated with a group of attributes.

3.2.4 Attribute Value

An attribute value is an octet array that may be either fixed or variable length. For example, it can be a one octet value, or a four octet integer, or a variable length string. An attribute may contain a value that is too large to transmit in a single PDU and can be sent using multiple PDUs. The values that are transmitted are opaque to the attribute protocol. The encoding of these octet arrays is defined by the attribute type.

When transmitting attribute values in a request, a response, a notification or an indication, the attribute value length is not sent in any field of the PDU. The length of a variable length field in the PDU is implicitly given by the length of the packet that carries this PDU. This implies that:

- a) only one attribute value can be placed in a single request, response, notification or indication unless the attribute values have lengths known by both the server and client, as defined by the attribute type.
- b) This attribute value will always be the only variable length field of a request, response, notification or indication.
- c) The bearer protocol (e.g. L2CAP) preserves datagram boundaries.

Note: Some responses include multiple attribute values, for example when client requests multiple attribute reads. For the client to determine the attribute value boundaries, the attribute values must have a fixed size defined by the attribute type.

3.2.5 Attribute Permissions

An attribute has a set of permission values associated with it. The permissions associated with an attribute specifies that it may be read and/or written. The permissions associated with the attribute specifies the security level required for read and/or write access, as well as notification and/or indication. The permissions of a given attribute are defined by a higher layer specification, and are not discoverable using the attribute protocol.

If access to a secure attribute requires an authenticated link, and the client is not already authenticated with the server with sufficient security, then an error response shall be sent with the error code «Insufficient Authentication». When a client receives this error code it may try to authenticate the link, and if the authentication is successful, it can then access the secure attribute.

If access to a secure attribute requires an encrypted link, and the link is not encrypted, then an error response shall be sent with the error code «Insufficient Encryption». When a client receives this error code it may try to encrypt the link and if the encryption is successful, it can then access the secure attribute.

If access to a secure attribute requires an encrypted link, and the link is encrypted but with an encryption key size that is too short for the level of security required, then an error response shall be sent with the error code «Insufficient Encryption Key Size». When a client receives this error code it may try to encrypt the link with a larger key size, and if the encryption is successful, it can then access the secure attribute.

Attribute permissions are a combination of access permissions, encryption permissions, authentication permissions and authorization permissions.

The following access permissions are possible:

- Readable
- Writeable
- Readable and writable

The following encryption permissions are possible:

- Encryption required
- No encryption required

The following authentication permissions are possible:

- Authentication Required
- No Authentication Required

The following authorization permissions are possible:

- Authorization Required
- No Authorization Required

Access permissions are used by a server to determine if a client can read and/or write an attribute value.

Authentication permissions are used by a server to determine if an authenticated physical link is required when a client attempts to access an attribute. Authentication permissions are also used by a server to determine if

an authenticated physical link is required before sending a notification or indication to a client.

Authorization permissions determine if a client needs to be authorized before accessing an attribute value.

3.2.6 Control-Point Attributes

Attributes that cannot be read, but can only be written, notified or indicated are called control-point attributes. These control-point attributes can be used by higher layers to enable device specific procedures, for example the writing of a command or the indication when a given procedure on a device has completed.

3.2.7 Protocol Methods

The attribute protocol uses methods defined in [Section 3.4](#) to find, read, write, notify, and indicate attributes. A method is categorized as either a request, a response, a command, a notification, an indication or a confirmation method; see [Section 3.3](#). Some attribute protocol PDUs can also include an Authentication Signature, to allow authentication of the originator of this PDU without requiring encryption. The method and signed bit are known as the opcode.

3.2.8 Exchanging MTU Size

ATT_MTU is defined as the maximum size of any packet sent between a client and a server. A higher layer specification defines the default ATT_MTU value.

The client and server may optionally exchange the maximum size of a packet that can be received using the Exchange MTU Request and Response PDUs. Both devices then use the minimum of these exchanged values for all further communication (see [Section 3.4.2](#)).

A device that is acting as a server and client at the same time shall use the same value for Client Rx MTU and Server Rx MTU.

The ATT_MTU value is a per ATT Bearer value. Note: A device with multiple ATT Bearers may have a different ATT_MTU value for each ATT Bearer.

3.2.9 Long Attribute Values

The longest attribute that can be sent in a single packet is (ATT_MTU - 1) octets in size. At a minimum, the Attribute Opcode is included in an Attribute PDU.

An attribute value may be defined to be larger than (ATT_MTU - 1) octets in size. These attributes are called long attributes.

To read the entire value of an attributes larger than (ATT_MTU-1) octets, the read blob request is used. It is possible to read the first (ATT_MTU-1) octets of a long attribute value using the read request.

To write the entire value of an attribute larger than (ATT_MTU-3) octets, the prepare write request and execute write request is used. It is possible to write the first (ATT_MTU-3) octets of a long attribute value using the write request.

It is not possible to determine if an attribute value is longer than (ATT_MTU-3) octets using this protocol. A higher layer specification will state that a given attribute can have a maximum length larger than (ATT_MTU-3) octets.

The maximum length of an attribute value shall be 512 octets.

Note: The protection of an attribute value changing when reading the value using multiple attribute protocol PDUs is the responsibility of the higher layer.

3.2.10 Atomic Operations

The server shall treat each request or command as an atomic operation that cannot be affected by another client sending a request or command at the same time. If a physical link is disconnected for any reason (user action or loss of the radio link), the value of any modified attribute is the responsibility of the higher layer specification.

Long attributes cannot be read or written in a single atomic operation.

3.3 ATTRIBUTE PDU

Attribute PDUs are one of six method types:

- Requests—PDUs sent to a server by a client, and invoke responses.
- Responses—PDUs sent to a client in response to a request to a server.
- Commands—PDUs sent to a server by a client.
- Notifications—Unsolicited PDUs sent to a client by a server.
- Indications—Unsolicited PDUs sent to a client by a server, and invoke confirmations.
- Confirmations—PDUs sent to an attribute server to confirm receipt of an indication by a client.

A server shall be able to receive and properly respond to the following requests:

- *Find Information Request*
- *Read Request*

Support for all other PDU types in a server can be specified in a higher layer specification, see [Section 3.4.8](#).

If a client sends a request, then the client shall support all possible responses PDUs for that request.

If a server receives a request that it does not support, then the server shall respond with the *Error Response* with the Error Code «Request Not Supported», with the Attribute Handle In Error set to 0x0000.

If a server receives a command that it does not support, indicated by the Command Flag of the PDU set to one, then the server shall ignore the Command.

If the server receives an invalid request – for example, the PDU is the wrong length – then the server shall respond with the *Error Response* with the Error Code «Invalid PDU», with the Attribute Handle In Error set to 0x0000.

If a server does not have sufficient resources to process a request, then the server shall respond with the *Error Response* with the Error Code «Insufficient Resources», with the Attribute Handle In Error set to 0x0000.

If a server cannot process a request because an error was encountered during the processing of this request, then the server shall respond with the *Error Response* with the Error Code «Unlikely Error», with the Attribute Handle In Error set to 0x0000.

3.3.1 Attribute PDU Format

Attribute PDUs has the following format:

Name	Size (octets)	Description
Attribute Opcode	1	The attribute PDU operation code bit7: Authentication Signature Flag bit6: Command Flag bit5-0: Method
Attribute Parameters	0 to (ATT_MTU - X)	The attribute PDU parameters X = 1 if Authentication Signature Flag of the Attribute Opcode is 0 X = 13 if Authentication Signature Flag of the Attribute Opcode is 1
Authentication Signature	0 or 12	Optional authentication signature for the Attribute Opcode and Attribute Parameters

Table 3.1: Format of Attribute PDU

Multi-octet fields within the attribute protocol shall be sent least significant octet first (little endian) with the exception of the Attribute Value field. The endian-ness of the Attribute Value field is defined by a higher layer specification.

The Attribute Opcode is composed of three fields, the Authentication Signature Flag, the Command Flag, and the Method. The Method is a 6-bit value that determines the format and meaning of the Attribute Parameters.

If the Authentication Signature Flag of the Attribute Opcode is set to one, the Authentication Signature value shall be appended to the end of the attribute PDU, and X is 13. If the Authentication Signature Flag of the Attribute Opcode is set to zero, the Authentication Signature value shall not be appended, and X is 1.

The Authentication Signature field is calculated as defined in Security Manager (see [\[Vol. 3\] Part H, Section 2.4.5](#)). This value provides an Authentication Signature for the variable length message (m) consisting of the following values in this order: Attribute Opcode, Attribute Parameters.

An Attribute PDU that includes an Authentication Signature should not be sent on an encrypted link. Note: an encrypted link already includes authentication data on every packet and therefore adding more authentication data is not required.

If the Command Flag of the Attribute Opcode is set to one, the PDU shall be considered to be a Command.

Only the Write Command may include an Authentication Signature.

3.3.2 Sequential Protocol

Many attribute protocol PDUs use a sequential request-response protocol.

Once a client sends a request to a server, that client shall send no other request to the same server until a response PDU has been received.

Indications sent from a server also use a sequential indication-confirmation protocol. No other indications shall be sent to the same client from this server until a confirmation PDU has been received. The client, however, is free to send commands and requests prior to sending a confirmation.

For notifications, which do not have a response PDU, there is no flow control and a notification can be sent at any time.

Commands that do not require a response do not have any flow control. Note: a server can be flooded with commands, and a higher layer specification can define how to prevent this from occurring.

Commands and notifications that are received but cannot be processed, due to buffer overflows or other reasons, shall be discarded. Therefore, those PDUs must be considered to be unreliable.

Note: Flow control for each client and a server is independent.

Note: It is possible for a server to receive a request, send one or more notifications, and then the response to the original request. The flow control of requests is not affected by the transmission of the notifications.

Note: It is possible for a server to receive a request and then a command before responding to the original request. The flow control of requests is not affected by the transmission of commands.

Note: It is possible for a notification from a server to be sent after an indication has been sent but the confirmation has not been received. The flow control of indications is not affected by the transmission of notifications.

Note: It is possible for a client to receive an indication from a server and then send a request or command to that server before sending the confirmation of the original indication.

3.3.3 Transaction

An attribute protocol request and response or indication-confirmation pair is considered a single transaction. A transaction shall always be performed on one ATT Bearer, and shall not be split over multiple ATT Bearers.

On the client, a transaction shall start when the request is sent by the client. A transaction shall complete when the response is received by the client.

On a server, a transaction shall start when a request is received by the server. A transaction shall complete when the response is sent by the server.

On a server, a transaction shall start when an indication is sent by the server. A transaction shall complete when the confirmation is received by the server.

On a client, a transaction shall start when an indication is received by the client. A transaction shall complete when the confirmation is sent by the client.

A transaction not completed within 30 seconds shall time out. Such a transaction shall be considered to have failed and the local higher layers shall be informed of this failure. No more attribute protocol requests, commands, indications or notifications shall be sent to the target device on this ATT Bearer.

Note: To send another attribute protocol PDU, a new ATT Bearer must be established between these devices. The existing ATT Bearer may need to be disconnected or the bearer terminated before the new ATT Bearer is established.

If the ATT Bearer is disconnected during a transaction, then the transaction shall be considered to be closed, and any values that were being modified on the server will be in an undetermined state, and any queue that was prepared by the client using this ATT Bearer shall be cleared.

Note: Each Prepare Write Request is a separate request and is therefore a separate transaction.

Note: Each Read Blob Request is a separate request and is therefore a separate transaction.

3.4 ATTRIBUTE PROTOCOL PDUS

3.4.1 Error Handling

3.4.1.1 Error Response

The *Error Response* is used to state that a given request cannot be performed, and to provide the reason.

Note: The Write Command does not generate an Error Response.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x01 = Error Response
Request Opcode In Error	1	The request that generated this error response
Attribute Handle In Error	2	The attribute handle that generated this error response
Error Code	1	The reason why the request has generated an error response

Table 3.2: Format of Error Response

The Request Opcode In Error parameter shall be set to the Attribute Opcode of the request that generated this error.

The Attribute Handle In Error parameter shall be set to the attribute handle in the original request that generated this error. If there was no attribute handle in the original request or if the request is not supported, then the value 0x0000 shall be used for this field.

The Error Code parameter shall be set to one of the following values:

Name	Error Code	Description
Invalid Handle	0x01	The attribute handle given was not valid on this server.
Read Not Permitted	0x02	The attribute cannot be read.
Write Not Permitted	0x03	The attribute cannot be written.
Invalid PDU	0x04	The attribute PDU was invalid.
Insufficient Authentication	0x05	The attribute requires authentication before it can be read or written.
Request Not Supported	0x06	Attribute server does not support the request received from the client.

Table 3.3: Error Codes

Attribute Protocol (ATT)

Name	Error Code	Description
Invalid Offset	0x07	Offset specified was past the end of the attribute.
Insufficient Authorization	0x08	The attribute requires authorization before it can be read or written.
Prepare Queue Full	0x09	Too many prepare writes have been queued.
Attribute Not Found	0x0A	No attribute found within the given attribute handle range.
Attribute Not Long	0x0B	The attribute cannot be read or written using the Read Blob Request.
Insufficient Encryption Key Size	0x0C	The Encryption Key Size used for encrypting this link is insufficient.
Invalid Attribute Value Length	0x0D	The attribute value length is invalid for the operation.
Unlikely Error	0x0E	The attribute request that was requested has encountered an error that was unlikely, and therefore could not be completed as requested.
Insufficient Encryption	0x0F	The attribute requires encryption before it can be read or written.
Unsupported Group Type	0x10	The attribute type is not a supported grouping attribute as defined by a higher layer specification.
Insufficient Resources	0x11	Insufficient Resources to complete the request.
Reserved	0x012 – 0x7F	Reserved for future use.
Application Error	0x80-0x9F	Application error code defined by a higher layer specification.
Reserved	0xA0 – 0xDF	Reserved for future use
Common Profile and Service Error Codes	0xE0 – 0xFF	Common profile and service error codes defined in [Core Specification Supplement] , Part B.

Table 3.3: Error Codes

If an error code is received in the *Error Response* that is not understood by the client, for example an error code that was reserved for future use that is now being used in a future version of this specification, then the *Error Response* shall still be considered to state that the given request cannot be performed for an unknown reason.

3.4.2 MTU Exchange

3.4.2.1 Exchange MTU Request

The *Exchange MTU Request* is used by the client to inform the server of the client's maximum receive MTU size and request the server to respond with its maximum receive MTU size.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x02 = Exchange MTU Request
Client Rx MTU	2	Client receive MTU size

Table 3.4: Format of Exchange MTU Request

The Client Rx MTU shall be greater than or equal to the default ATT_MTU.

This request shall only be sent once during a connection by the client. The Client Rx MTU parameter shall be set to the maximum size of the attribute protocol PDU that the client can receive.

3.4.2.2 Exchange MTU Response

The *Exchange MTU Response* is sent in reply to a received *Exchange MTU Request*.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x03 = Exchange MTU Response
Server Rx MTU	2	Attribute server receive MTU size

Table 3.5: Format of Exchange MTU Response

The Server Rx MTU shall be greater than or equal to the default ATT_MTU.

The Server Rx MTU parameter shall be set to the maximum size of the attribute protocol PDU that the server can receive.

The server and client shall set ATT_MTU to the minimum of the Client Rx MTU and the Server Rx MTU. The size is the same to ensure that a client can correctly detect the final packet of a long attribute read.

This ATT_MTU value shall be applied in the server after this response has been sent and before any other attribute protocol PDU is sent.

This ATT_MTU value shall be applied in the client after this response has been received and before any other attribute protocol PDU is sent.

Attribute Protocol (ATT)

If either Client Rx MTU or Service Rx MTU are incorrectly less than the default ATT_MTU, then the ATT_MTU shall not be changed and the ATT_MTU shall be the default ATT_MTU.

If a device is both a client and a server, the following rules shall apply:

1. A device's *Exchange MTU Request* shall contain the same MTU as the device's *Exchange MTU Response* (i.e. the MTU shall be symmetric).
2. If MTU is exchanged in one direction, that is sufficient for both directions.
3. It is permitted, (but not necessary - see 2.) to exchange MTU in both directions, but the MTUs shall be the same in each direction (see 1.)
4. If an Attribute Protocol Request is received after the MTU Exchange Request is sent and before the MTU Exchange Response is received, the associated Attribute Protocol Response shall use the default MTU. [Figure 3.1](#) shows an example that is covered by this rule. In this case device A and device B both use the default MTU for the Attribute Protocol Response.
5. Once the MTU Exchange Request has been sent, the initiating device shall not send an Attribute Protocol Indication or Notification until after the MTU Exchange Response has been received. Note: This stops the risk of a cross-over condition where the MTU size is unknown for the Indication or Notification.

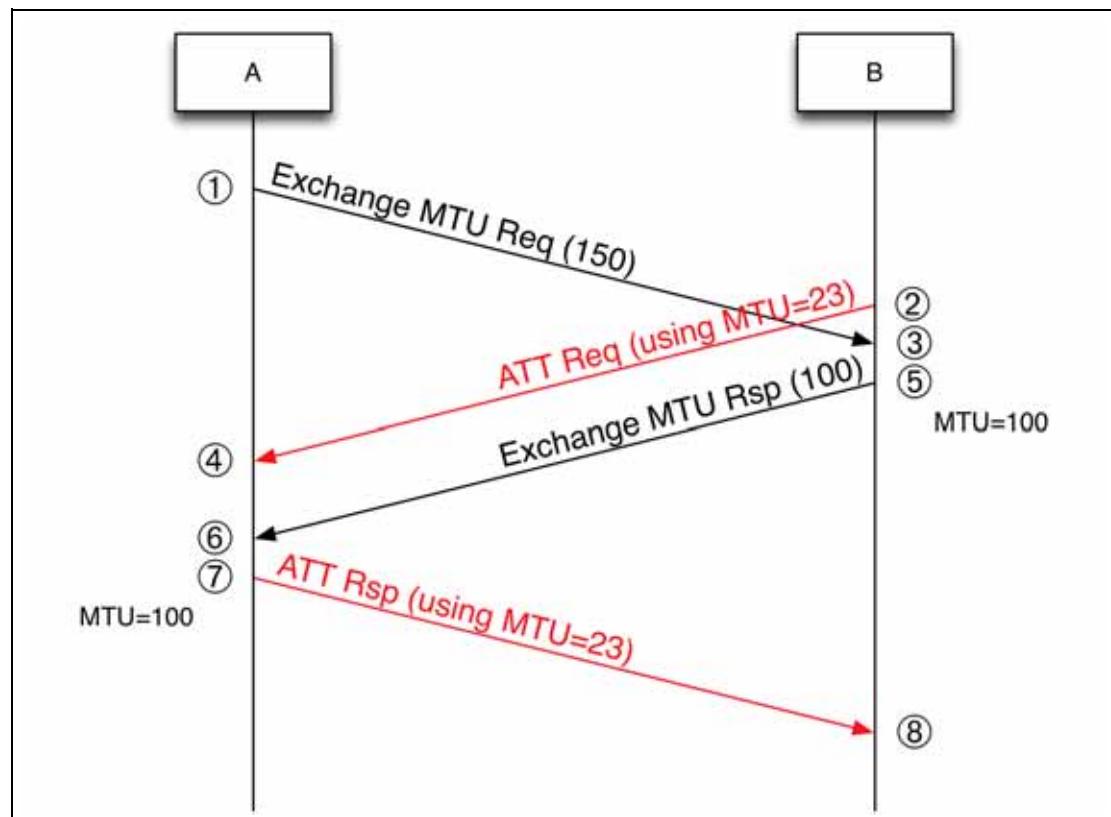


Figure 3.1: MTU request and response exchange

3.4.3 Find Information

3.4.3.1 Find Information Request

The *Find Information Request* is used to obtain the mapping of attribute handles with their associated types. This allows a client to discover the list of attributes and their types on a server.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x04 = Find Information Request
Starting Handle	2	First requested handle number
Ending Handle	2	Last requested handle number

Table 3.6: Format of Find Information Request

Only attributes with attribute handles between and including the Starting Handle parameter and the Ending Handle parameter will be returned. To read all attributes, the Starting Handle parameter shall be set to 0x0001, and the Ending Handle parameter shall be set to 0xFFFF. The Starting Handle parameter shall be less than or equal to the Ending Handle parameter.

If one or more attributes will be returned, a *Find Information Response* PDU shall be sent.

If a server receives a *Find Information Request* with the Starting Handle parameter greater than the Ending Handle parameter or the Starting Handle parameter is 0x0000, an *Error Response* shall be sent with the «Invalid Handle» error code; the Attribute Handle In Error parameter shall be set to the Starting Handle parameter.

If no attributes will be returned, an *Error Response* shall be sent with the «Attribute Not Found» error code; the Attribute Handle In Error parameter shall be set to the Starting Handle parameter.

The server shall not respond to the *Find Information Request* with an *Error Response* with the «Insufficient Authentication», «Insufficient Authorization», «Insufficient Encryption Key Size» or «Application Error» error code.

3.4.3.2 Find Information Response

The *Find Information Response* is sent in reply to a received *Find Information Request* and contains information about this server.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x05 = Find Information Response

Table 3.7: Format of Find Information Response

Attribute Protocol (ATT)

Parameter	Size (octets)	Description
Format	1	The format of the information data.
Information Data	4 to (ATT_MTU-2)	The information data whose format is determined by the Format field

Table 3.7: Format of Find Information Response

The *Find Information Response* shall have complete handle-UUID pairs. Such pairs shall not be split across response packets; this also implies that a handle-UUID pair shall fit into a single response packet. The handle-UUID pairs shall be returned in ascending order of attribute handles.

The Format parameter can contain one of two possible values.

Name	Format	Description
Handle(s) and 16-bit Bluetooth UUID(s)	0x01	A list of 1 or more handles with their 16-bit Bluetooth UUIDs
Handle(s) and 128-bit UUID(s)	0x02	A list of 1 or more handles with their 128-bit UUIDs

Table 3.8: Format field values

The information data field is comprised of a list of data defined in the tables below depending on the value chosen for the format.

Handle	16-bit Bluetooth UUID
2 octets	2 octets

Table 3.9: Format 0x01 - handle and 16-bit Bluetooth UUIDs

Handle	128-bit UUID
2 octets	16 octets

Table 3.10: Format 0x02 - handle and 128-bit UUIDs

Note: If sequential attributes have differing UUID sizes, it may happen that a *Find Information Response* is not filled with the maximum possible amount of (handle, UUID) pairs. This is because it is not possible to include attributes with differing UUID sizes into a single response packet. In that case, the following attribute would have to be read using another *Find Information Request* with its starting handle updated.

3.4.3.3 Find By Type Value Request

The *Find By Type Value Request* is used to obtain the handles of attributes that have a 16-bit UUID attribute type and attribute value. This allows the range of handles associated with a given attribute to be discovered when the attribute type determines the grouping of a set of attributes. Note: Generic Attribute Profile defines grouping of attributes by attribute type.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x06 = Find By Type Value Request
Starting Handle	2	First requested handle number
Ending Handle	2	Last requested handle number
Attribute Type	2	2 octet UUID to find
Attribute Value	0 to (ATT_MTU-7)	Attribute value to find

Table 3.11: Format of *Find By Type Value Request*

Only attributes with attribute handles between and including the Starting Handle parameter and the Ending Handle parameter that match the requested attribute type and the attribute value that have sufficient permissions to allow reading will be returned. To read all attributes, the Starting Handle parameter shall be set to 0x0001, and the Ending Handle parameter shall be set to 0xFFFF.

If one or more handles will be returned, a *Find By Type Value Response* PDU shall be sent.

Note: Attribute values will be compared in terms of length and binary representation.

Note: It is not possible to use this request on an attribute that has a value longer than (ATT_MTU-7).

If a server receives a *Find By Type Value Request* with the Starting Handle parameter greater than the Ending Handle parameter or the Starting Handle parameter is 0x0000, an *Error Response* shall be sent with the «Invalid Handle» error code. The Attribute Handle In Error parameter shall be set to the Starting Handle parameter.

If no attributes will be returned, an *Error Response* shall be sent by the server with the error code «Attribute Not Found». The Attribute Handle In Error parameter shall be set to the starting handle.

The server shall not respond to the *Find By Type Value Request* with an *Error Response* with the «Insufficient Authentication», «Insufficient Authorization», «Insufficient Encryption Key Size», «Insufficient Encryption» or «Application Error» error code.

3.4.3.4 Find By Type Value Response

The *Find By Type Value Response* is sent in reply to a received *Find By Type Value Request* and contains information about this server.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x07 = Find By Type Value Response
Handles Information List	4 to (ATT_MTU-1)	A list of 1 or more Handle Informations

Table 3.12: Format of *Find By Type Value Response*

The Handles Information List field is a list of one or more Handle Informations. The Handles Information field is an attribute handle range as defined in [Table 3.13](#).

Found Attribute Handle	Group End Handle
2 octets	2 octets

Table 3.13: Format of the Handles Information

The *Find By Type Value Response* shall contain one or more complete Handles Information. Such Handles Information shall not be split across response packets. The Handles Information List is ordered sequentially based on the found attribute handles.

For each handle that matches the attribute type and attribute value in the *Find By Type Value Request* a Handles Information shall be returned. The Found Attribute Handle shall be set to the handle of the attribute that has the exact attribute type and attribute value from the *Find By Type Value Request*. If the attribute type in the *Find By Type Value Request* is a grouping attribute as defined by a higher layer specification, the Group End Handle shall be defined by that higher layer specification. If the attribute type in the *Find By Type Value Request* is not a grouping attribute as defined by a higher layer specification, the Group End Handle shall be equal to the Found Attribute Handle.

Note: The Group End Handle may be greater than the Ending Handle in the *Find By Type Value Request*.

If a server receives a *Find By Type Value Request*, the server shall respond with the *Find By Type Value Response* containing as many handles for attributes that match the requested attribute type and attribute value that exist in the server that will fit into the maximum PDU size of (ATT_MTU-1).

3.4.4 Reading Attributes

3.4.4.1 Read By Type Request

The *Read By Type Request* is used to obtain the values of attributes where the attribute type is known but the handle is not known.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x08 = Read By Type Request
Starting Handle	2	First requested handle number
Ending Handle	2	Last requested handle number
Attribute Type	2 or 16	2 or 16 octet UUID

Table 3.14: Format of Read By Type Request

Only the attributes with attribute handles between and including the Starting Handle and the Ending Handle with the attribute type that is the same as the Attribute Type given will be returned. To search through all attributes, the starting handle shall be set to 0x0001 and the ending handle shall be set to 0xFFFF.

Note: All attribute types are effectively compared as 128-bit UUIDs, even if a 16-bit UUID is provided in this request or defined for an attribute. See [Section 3.2.1](#).

The starting handle shall be less than or equal to the ending handle. If a server receives a *Read By Type Request* with the Starting Handle parameter greater than the Ending Handle parameter or the Starting Handle parameter is 0x0000, an *Error Response* shall be sent with the «Invalid Handle» error code; The Attribute Handle In Error parameter shall be set to the Starting Handle parameter.

If no attribute with the given type exists within the handle range, then no attribute handle and value will be returned, and an *Error Response* shall be sent with the error code «Attribute Not Found». The Attribute Handle In Error parameter shall be set to the starting handle.

The attributes returned shall be the attributes with the lowest handles within the handle range. These are known as the requested attributes.

If the attributes with the requested type within the handle range have attribute values that have the same length, then these attributes can all be read in a single request.

The attribute server shall include as many attributes as possible in the response in order to minimize the number of PDUs required to read attributes of the same type.

Note: If the attributes with the requested type within the handle range have attribute values with different lengths, then multiple *Read By Type Requests* must be made.

When multiple attributes match, then the rules below shall be applied to each in turn.

- Only attributes that can be read shall be returned in a *Read By Type Response*.
- If an attribute in the set of requested attributes would cause an *Error Response* then this attribute cannot be included in a *Read By Type Response* and the attributes before this attribute shall be returned.
- If the first attribute in the set of requested attributes would cause an *Error Response* then no other attributes in the requested attributes can be considered.

The server shall respond with a *Read By Type Response* if the requested attributes have sufficient permissions to allow reading.

If the client has insufficient authorization to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authorization». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

If the client has insufficient security to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authentication». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

If the client has an insufficient encryption key size to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Encryption Key Size». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

If the client has not enabled encryption, and encryption is required to read the requested attribute, then an *Error Response* shall be sent with the error code «Insufficient Encryption». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

If the requested attribute's value cannot be read due to permissions then an *Error Response* shall be sent with the error code «Read Not Permitted». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

Note: if there are multiple attributes with the requested type within the handle range, and the client would like to get the next attribute with the requested type, it would have to issue another *Read By Type Request* with its starting handle updated. The client can be sure there are no more such attributes remaining once it gets an *Error Response* with the error code «Attribute Not Found».

3.4.4.2 Read By Type Response

The *Read By Type Response* is sent in reply to a received *Read By Type Request* and contains the handles and values of the attributes that have been read.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x09 = Read By Type Response
Length	1	The size of each attribute handle-value pair
Attribute Data List	2 to (ATT_MTU - 2)	A list of Attribute Data

Table 3.15: Format of Read By Type Response

The *Read By Type Response* shall contain complete handle-value pairs. Such pairs shall not be split across response packets. The handle-value pairs shall be returned sequentially based on the attribute handle.

The Length parameter shall be set to the size of one attribute handle-value pair.

The maximum length of an attribute handle-value pair is 255 octets, bounded by the Length parameter that is one octet. Therefore, the maximum length of an attribute value returned in this response is $(\text{Length} - 2) = 253$ octets.

The attribute handle-value pairs shall be set to the value of the attributes identified by the attribute type within the handle range within the request. If the attribute value is longer than $(\text{ATT_MTU} - 4)$ or 253 octets, whichever is smaller, then the first $(\text{ATT_MTU} - 4)$ or 253 octets shall be included in this response.

Note: the *Read Blob Request* would be used to read the remaining octets of a long attribute value.

The Attribute Data field is comprised of a list of attribute handle and value pairs as defined in [Table 3.16](#).

Attribute Handle	Attribute Value
2 octets	$(\text{Length} - 2)$ octets

Table 3.16: Format of the Attribute Data

3.4.4.3 Read Request

The *Read Request* is used to request the server to read the value of an attribute and return its value in a *Read Response*.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x0A = Read Request
Attribute Handle	2	The handle of the attribute to be read

Table 3.17: Format of Read Request

The attribute handle parameter shall be set to a valid handle.

The server shall respond with a *Read Response* if the handle is valid and the attribute has sufficient permissions to allow reading.

If the client has insufficient authorization to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authorization».

If the client has insufficient security to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authentication».

If the client has an insufficient encryption key size to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Encryption Key Size».

If the client has not enabled encryption, and encryption is required to read the requested attribute, then an *Error Response* shall be sent with the error code «Insufficient Encryption».

If the handle is invalid, then an *Error Response* shall be sent with the error code «Invalid Handle».

If the attribute value cannot be read due to permissions then an *Error Response* shall be sent with the error code «Read Not Permitted».

3.4.4.4 Read Response

The read response is sent in reply to a received *Read Request* and contains the value of the attribute that has been read.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x0B = Read Response
Attribute Value	0 to (ATT_MTU-1)	The value of the attribute with the handle given

Table 3.18: Format of Read Response

The attribute value shall be set to the value of the attribute identified by the attribute handle in the request. If the attribute value is longer than (ATT_MTU-1) then the first (ATT_MTU-1) octets shall be included in this response.

Note: the *Read Blob Request* would be used to read the remaining octets of a long attribute value.

3.4.4.5 Read Blob Request

The *Read Blob Request* is used to request the server to read part of the value of an attribute at a given offset and return a specific part of the value in a *Read Blob Response*.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x0C = Read Blob Request
Attribute Handle	2	The handle of the attribute to be read
Value Offset	2	The offset of the first octet to be read

Table 3.19: Format of Read Blob Request

The attribute handle parameter shall be set to a valid handle.

The value offset parameter is based from zero; the first value octet has an offset of zero, the second octet has a value offset of one, etc.

The server shall respond with a *Read Blob Response* if the handle is valid and the attribute and value offset is not greater than the length of the attribute value and has sufficient permissions to allow reading.

If the client has insufficient authorization to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authorization».

If the client has insufficient security to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authentication».

If the client has an insufficient encryption key size to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Encryption Key Size».

If the client has not enabled encryption, and encryption is required to read the requested attribute, then an *Error Response* shall be sent with the error code «Insufficient Encryption».

If the handle is invalid, then an *Error Response* shall be sent with the error code «Invalid Handle».

If the attribute value cannot be read due to permissions then an *Error Response* shall be sent with the error code «Read Not Permitted».

If the value offset of the *Read Blob Request* is greater than the length of the attribute value, an *Error Response* shall be sent with the error code «Invalid Offset».

If the attribute value has a fixed length that is less than or equal to (ATT_MTU - 3) octets in length, then an *Error Response* can be sent with the error code «Attribute Not Long».

If the value offset of the *Read Blob Request* is equal to the length of the attribute value, then the length of the part attribute value in the response shall be zero.

Note: if the attribute is longer than (ATT_MTU-1) octets, the *Read Blob Request* is the only way to read the additional octets of a long attribute. The first (ATT_MTU-1) octets may be read using a *Read Request*, an *Handle Value Notification* or an *Handle Value Indication*.

Note: Long attributes may or may not have their length specified by a higher layer specification. If the long attribute has a variable length, the only way to get to the end of it is to read it part by part until the value in the *Read Blob Response* has a length shorter than (ATT_MTU-1) or an *Error Response* with the error code «Invalid Offset».

Note: the value of a Long Attribute may change between one *Read Blob Request* and the next *Read Blob Request*. A higher layer specification should be aware of this and define appropriate behavior.

3.4.4.6 Read Blob Response

The *Read Blob Response* is sent in reply to a received *Read Blob Request* and contains part of the value of the attribute that has been read.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x0D = Read Blob Response
Part Attribute Value	0 to (ATT_MTU-1)	Part of the value of the attribute with the handle given

Table 3.20: Format of Read Blob Response

The part attribute value shall be set to part of the value of the attribute identified by the attribute handle and the value offset in the request. If the value offset is equal to the length of the attribute value, then the length of the part attribute value shall be zero. If the attribute value is longer than (Value Offset + ATT_MTU-1) then (ATT_MTU-1) octets from Value Offset shall be included in this response.

3.4.4.7 Read Multiple Request

The *Read Multiple Request* is used to request the server to read two or more values of a set of attributes and return their values in a *Read Multiple Response*. Only values that have a known fixed size can be read, with the exception of the last value that can have a variable length. The knowledge of whether attributes have a known fixed size is defined in a higher layer specification.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x0E = Read Multiple Request
Set Of Handles	4 to (ATT_MTU-1)	A set of two or more attribute handles.

Table 3.21: Format of Read Multiple Request

The attribute handles in the Set Of Handles parameter shall be valid handles.

The server shall respond with a *Read Multiple Response* if all the handles are valid and all attributes have sufficient permissions to allow reading.

Note: The attribute values for the attributes in the Set Of Handles parameters do not have to all be the same size.

Note: The attribute handles in the Set Of Handles parameter do not have to be in attribute handle order; they are in the order that the values are required in the response.

If the client has insufficient authorization to read any of the attributes then an *Error Response* shall be sent with the error code «Insufficient Authorization».

If the client has insufficient security to read any of the attributes then an *Error Response* shall be sent with the error code «Insufficient Authentication».

If the client has an insufficient encryption key size to read any of the attributes then an *Error Response* shall be sent with the error code «Insufficient Encryption Key Size».

If the client has not enabled encryption, and encryption is required to read the requested attribute, then an *Error Response* shall be sent with the error code «Insufficient Encryption».

If any of the handles are invalid, then an *Error Response* shall be sent with the error code «Invalid Handle».

If any of the attribute values cannot be read due to permissions then an *Error Response* shall be sent with the error code «Read Not Permitted».

If an *Error Response* is sent, the Attribute Handle In Error parameter shall be set to the handle of the first attribute causing the error.

3.4.4.8 Read Multiple Response

The read response is sent in reply to a received *Read Multiple Request* and contains the values of the attributes that have been read.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x0F = Read Multiple Response
Set Of Values	0 to (ATT_MTU-1)	A set of two or more values

Table 3.22: Format of Read Multiple Response

The Set Of Values parameter shall be a concatenation of attribute values for each of the attribute handles in the request in the order that they were requested. If the Set Of Values parameter is longer than (ATT_MTU-1) then only the first (ATT_MTU-1) octets shall be included in this response.

Note: a client should not use this request for attributes when the Set Of Values parameter could be (ATT_MTU-1) as it will not be possible to determine if the last attribute value is complete, or if it overflowed.

3.4.4.9 Read by Group Type Request

The *Read By Group Type Request* is used to obtain the values of attributes where the attribute type is known, the type of a grouping attribute as defined by a higher layer specification, but the handle is not known.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x10 = Read By Group Type Request
Starting Handle	2	First requested handle number
Ending Handle	2	Last requested handle number
Attribute Group Type	2 or 16	2 or 16 octet UUID

Table 3.23: Format of Read By Group Type Request

Only the attributes with attribute handles between and including the Starting Handle and the Ending Handle with the attribute type that is the same as the Attribute Group Type given will be returned. To search through all attributes, the starting handle shall be set to 0x0001 and the ending handle shall be set to 0xFFFF.

Note: All attribute types are effectively compared as 128-bit UUIDs, even if a 16-bit UUID is provided in this request or defined for an attribute. See [Section 3.2.1](#).

The starting handle shall be less than or equal to the ending handle. If a server receives a *Read By Group Type Request* with the Starting Handle parameter greater than the Ending Handle parameter or the Starting Handle parameter is

0x0000, an *Error Response* shall be sent with the «Invalid Handle» error code; The Attribute Handle In Error parameter shall be set to the Starting Handle parameter.

If the Attribute Group Type is not a supported grouping attribute as defined by a higher layer specification then an *Error Response* shall be sent with the error code «Unsupported Group Type». The Attribute Handle In Error parameter shall be set to the Starting Handle.

If no attribute with the given type exists within the handle range, then no attribute handle and value will be returned, and an *Error Response* shall be sent with the error code «Attribute Not Found». The Attribute Handle In Error parameter shall be set to the starting handle.

The attributes returned shall be the attributes with the lowest handles within the handle range. These are known as the requested attributes.

If the attributes with the requested type within the handle range have attribute values that have the same length, then these attributes can all be read in a single request.

The attribute server shall include as many attributes as possible in the response in order to minimize the number of PDUs required to read attributes of the same type.

Note: If the attributes with the requested type within the handle range have attribute values with different lengths, then multiple *Read By Group Type Requests* must be made.

When multiple attributes match, then the rules below shall be applied to each in turn.

- Only attributes that can be read shall be returned in a *Read By Group Type Response*.
- If an attribute in the set of requested attributes would cause an *Error Response* then this attribute cannot be included in a *Read By Group Type Response* and the attributes before this attribute shall be returned.
- If the first attribute in the set of requested attributes would cause an *Error Response* then no other attributes in the requested attributes can be considered.

The server shall respond with a *Read By Group Type Response* if the requested attributes have sufficient permissions to allow reading.

If the client has insufficient authorization to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authorization». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

If the client has insufficient security to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authentication». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

If the client has an insufficient encryption key size to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Encryption Key Size». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

If the client has not enabled encryption, and encryption is required to read the requested attribute, then an *Error Response* shall be sent with the error code «Insufficient Encryption». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

If the requested attribute's value cannot be read due to permissions then an *Error Response* shall be sent with the error code «Read Not Permitted». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

Note: if there are multiple attributes with the requested type within the handle range, and the client would like to get the next attribute with the requested type, it would have to issue another *Read By Group Type Request* with its starting handle updated. The client can be sure there are no more such attributes remaining once it gets an *Error Response* with the error code «Attribute Not Found».

3.4.4.10 Read by Group Type Response

The *Read By Group Type Response* is sent in reply to a received *Read By Group Type Request* and contains the handles and values of the attributes that have been read.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x11 = Read By Group Type Response
Length	1	The size of each Attribute Data
Attribute Data List	2 to (ATT_MTU - 2)	A list of Attribute Data

Table 3.24: Format of Read By Group Type Response

The *Read By Group Type Response* shall contain complete Attribute Data. An Attribute Data shall not be split across response packets. The Attribute Data List is ordered sequentially based on the attribute handles

The Length parameter shall be set to the size of the one Attribute Data.

The maximum length of an Attribute Data is 255 octets, bounded by the Length parameter that is one octet. Therefore, the maximum length of an attribute value returned in this response is (Length – 4) = 251 octets.

The Attribute Data List shall be set to the value of the attributes identified by the attribute type within the handle range within the request. If the attribute value is longer than (ATT_MTU - 6) or 251 octets, whichever is smaller, then the first (ATT_MTU - 6) or 251 octets shall be included in this response.

Note: the *Read Blob Request* would be used to read the remaining octets of a long attribute value.

The Attribute Data List is comprised of a list of Attribute Data as defined in [Table 3.25](#).

Attribute Handle	End Group Handle	Attribute Value
2 octets	2 octets	(Length - 4) octets

Table 3.25: Format of the Attribute Data

3.4.5 Writing Attributes

3.4.5.1 Write Request

The *Write Request* is used to request the server to write the value of an attribute and acknowledge that this has been achieved in a *Write Response*.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x12 = Write Request
Attribute Handle	2	The handle of the attribute to be written
Attribute Value	0 to (ATT_MTU-3)	The value to be written to the attribute

Table 3.26: Format of Write Request

The Attribute Handle shall be set to a valid handle.

The Attribute Value shall be set to the new value of the attribute.

If the attribute value has a variable length, then the attribute value shall be truncated or lengthened to match the length of the Attribute Value parameter.

Note: If an attribute value has a variable length and if the Attribute Value parameter is of zero length, the attribute value will be fully truncated.

If the attribute value has a fixed length and the Attribute Value parameter length is less than or equal to the length of the attribute value, the octets of the attribute value parameter length shall be written; all other octets in this attribute value shall be unchanged.

The server shall respond with a *Write Response* if the handle is valid, the attribute has sufficient permissions to allow writing, and the attribute value has a valid size and format, and it is successful in writing the attribute.

If the attribute value has a variable length and the Attribute Value parameter length exceeds the maximum valid length of the attribute value then the server shall respond with an *Error Response* with the error code «Invalid Attribute Value Length».

If the attribute value has a fixed length and the requested attribute value parameter length is greater than the length of the attribute value then the server shall respond with an *Error Response* with the error code «Invalid Attribute Value Length».

If the client has insufficient authorization to write the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authorization».

If the client has insufficient security to write the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authentication».

If the client has an insufficient encryption key size to write the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Encryption Key Size».

If the client has not enabled encryption, and encryption is required to write the requested attribute, then an *Error Response* shall be sent with the error code «Insufficient Encryption».

If the handle is invalid, then an *Error Response* shall be sent with the error code «Invalid Handle».

If the attribute value cannot be written due to permissions then an *Error Response* shall be sent with the error code «Write Not Permitted».

If the attribute value cannot be written due to an application error then an *Error Response* shall be sent with an error code defined by a higher layer specification.

3.4.5.2 Write Response

The *Write Response* is sent in reply to a valid *Write Request* and acknowledges that the attribute has been successfully written.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x13 = Write Response

Table 3.27: Format of Write Response

The *Write Response* shall be sent after the attribute value is written.

3.4.5.3 Write Command

The *Write Command* is used to request the server to write the value of an attribute, typically into a control-point attribute.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x52 = Write Command
Attribute Handle	2	The handle of the attribute to be set
Attribute Value	0 to (ATT_MTU-3)	The value of be written to the attribute

Table 3.28: Format of Write Command

The attribute handle parameter shall be set to a valid handle.

The attribute value parameter shall be set to the new value of the attribute.

If the attribute value has a variable length, then the attribute value shall be truncated or lengthened to match the length of the attribute value parameter.

Note: If an attribute value has a variable length and if the attribute value parameter is of zero length, the attribute value will be fully truncated.

If the attribute value has a fixed length and the attribute value parameter length is less than or equal to the length of the attribute value, the octets up to the attribute value parameter length shall be written; all other octets in this attribute value shall be unchanged.

If the attribute value has a variable length and the attribute value parameter length exceeds the maximum valid length of the attribute value then the server shall ignore the command.

If the attribute value has a fixed length and the requested attribute value parameter length is greater than the length of the attribute value then the server shall ignore the command.

No *Error Response* or *Write Response* shall be sent in response to this command. If the server cannot write this attribute for any reason the command shall be ignored.

3.4.5.4 Signed Write Command

The *Signed Write Command* is used to request the server to write the value of an attribute with an authentication signature, typically into a control-point attribute.

Parameter	Size (Octets)	Description
Attribute Opcode	1	0xD2 = Signed Write Command
Attribute Handle	2	The handle of the attribute to be set
Attribute Value	0 to (ATT_MTU - 15)	The value to be written to the attribute
Authentication Signature	12	Authentication signature for the Attribute Opcode, Attribute Handle and Attribute Value Parameters

Table 3.29: Format of Signed Write Command

The attribute handle parameter shall be set to a valid handle.

The attribute value parameter shall be set to the new value of the attribute.

The attribute signature shall be calculated as defined in [Section 3.3.1](#).

If the attribute value has a variable length, then the attribute value shall be truncated or lengthened to match the length of the attribute value parameter.

Note: If an attribute value has a variable length and if the attribute value parameter is of zero length, the attribute value will be fully truncated.

If the attribute value has a fixed length and the attribute value parameter length is less than or equal to the length of the attribute value, the octets up to the attribute value parameter length shall be written; all other octets in this attribute value shall be unchanged.

If the attribute value has a variable length and the attribute value parameter length exceeds the maximum valid length of the attribute value then the server shall ignore the command.

If the attribute value has a fixed length and the requested attribute value parameter length is greater than the length of the attribute value then the server shall ignore the command.

If the authentication signature verification fails, then the server shall ignore the command.

No *Error Response* or *Write Response* shall be sent in response to this command. If the server cannot write this attribute for any reason the command shall be ignored.

3.4.6 Queued Writes

The purpose of queued writes is to queue up writes of values of multiple attributes in a first-in first-out queue and then execute the write on all of them in a single atomic operation.

3.4.6.1 Prepare Write Request

The *Prepare Write Request* is used to request the server to prepare to write the value of an attribute. The server will respond to this request with a *Prepare Write Response*, so that the client can verify that the value was received correctly.

A client may send more than one *Prepare Write Request* to a server, which will queue and send a response for each handle value pair.

A server may limit the number of prepare write requests that it can accept. A higher layer specification should define this limit.

After a *Prepare Write Request* has been issued, and the response received, any other attribute command or request can be issued from the same client to the same server.

Each client's queued values are separate; the execution of one queue shall not affect the preparation or execution of any other client's queued values.

Any actions on attributes that exist in the prepare queue shall proceed as if the prepare queue did not exist, and the prepare queue shall be unaffected by these actions. A subsequent execute write will write the values in the prepare queue even if the value of the attribute has changed since the prepare writes were started.

The attribute protocol makes no determination on the validity of the Part Attribute Value or the Value Offset. A higher layer specification determines the meaning of the data.

Each *Prepare Write Request* will be queued even if the attribute handle is the same as a previous *Prepare Write Request*. These will then be executed in the order received, causing multiple writes for this attribute to occur.

If the link is lost while a number of prepared write requests have been queued, the queue will be cleared and no writes will be executed.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x16 = Prepare Write Request
Attribute Handle	2	The handle of the attribute to be written

Table 3.30: Format of Prepare Write Request

Parameter	Size (octets)	Description
Value Offset	2	The offset of the first octet to be written
Part Attribute Value	0 to (ATT_MTU-5)	The value of the attribute to be written

Table 3.30: Format of Prepare Write Request

The Attribute Handle parameter shall be set to a valid handle.

The Value Offset parameter shall be set to the offset of the first octet where the Part Attribute Value parameter is to be written within the attribute value. The Value Offset parameter is based from zero; the first octet has an offset of zero, the second octet has an offset of one, etc.

The server shall respond with a *Prepare Write Response* if the handle is valid, the attribute has sufficient permissions to allow writing at this time, and the prepare queue has sufficient space.

Note: The Attribute Value validation is done when an Execute Write Request is received. Hence, any Invalid Offset or Invalid Attribute Value Length errors are generated when an Execute Write Request is received.

If the client has insufficient authorization to write the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authorization».

If the client has insufficient security to write the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authentication».

If the client has an insufficient encryption key size to write the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Encryption Key Size».

If the client has not enabled encryption, and encryption is required to write the requested attribute, then an *Error Response* shall be sent with the error code «Insufficient Encryption».

If the server does not have sufficient space to queue this request then an *Error Response* shall be sent with the error code «Prepare Queue Full».

If the handle is invalid, then an *Error Response* shall be sent with the error code «Invalid Handle».

If the attribute value cannot be written then an *Error Response* shall be sent with the error code «Write Not Permitted».

The server shall not change the value of the attribute until an *Execute Write Request* is received.

If a *Prepare Write Request* was invalid, and therefore an *Error Response* has been issued, then this prepared write will be considered to have not been received. All existing prepared writes in the prepare queue shall not be affected by this invalid request.

3.4.6.2 Prepare Write Response

The *Prepare Write Response* is sent in response to a received *Prepare Write Request* and acknowledges that the value has been successfully received and placed in the prepare write queue.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x17 = Prepare Write Response
Attribute Handle	2	The handle of the attribute to be written
Value Offset	2	The offset of the first octet to be written
Part Attribute Value	0 to (ATT_MTU-5)	The value of the attribute to be written

Table 3.31: Format of Prepare Write Response

The attribute handle shall be set to the same value as in the corresponding *Prepare Write Request*.

The value offset and part attribute value shall be set to the same values as in the corresponding *Prepare Write Request*.

3.4.6.3 Execute Write Request

The *Execute Write Request* is used to request the server to write or cancel the write of all the prepared values currently held in the prepare queue from this client. This request shall be handled by the server as an atomic operation.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x18 = Execute Write Request
Flags	1	0x00 – Cancel all prepared writes 0x01 – Immediately write all pending prepared values

Table 3.32: Format of Execute Write Request

When the flags parameter is set to 0x01, values that were queued by the previous prepare write requests shall be written in the order they were received in the corresponding *Prepare Write Request*. The queue shall then be cleared, and an *Execute Write Response* shall be sent.

When the flags parameter is set to 0x00 all pending prepare write values shall be discarded for this client. The queue shall then be cleared, and an *Execute Write Response* shall be sent.

If the prepared Attribute Value exceeds the maximum valid length of the attribute value then all pending prepare write values shall be discarded for this client, the queue shall then be cleared, and an *Error Response* shall be sent with the error code «Invalid Attribute Value Length».

If the prepare Value Offset is greater than the current length of the attribute value then all pending prepare write values shall be discarded for this client, the queue shall be cleared and then an *Error Response* shall be sent with the «Invalid Offset».

If the prepare write requests cannot be written, due to an application error, the queue shall be cleared and then an *Error Response* shall be sent with a higher layer specification defined error code. The Attribute Handle In Error parameter shall be set to the attribute handle of the attribute from the prepare queue that caused this application error. The state of the attributes that were to be written from the prepare queue is not defined in this case.

3.4.6.4 Execute Write Response

The *Execute Write Response* is sent in response to a received *Execute Write Request*.

Parameter	Size	Description
Attribute Opcode	1	0x19 - Execute Write Response

Table 3.33: Format of Execute Write Response

The *Execute Write Response* shall be sent after the attributes are written. In case an action is taken in response to the write, an indication may be used once the action is complete.

3.4.7 Server Initiated

3.4.7.1 Handle Value Notification

A server can send a notification of an attribute's value at any time.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x1B = Handle Value Notification
Attribute Handle	2	The handle of the attribute
Attribute Value	0 to (ATT_MTU-3)	The current value of the attribute

Table 3.34: Format of Handle Value Notification

The attribute handle shall be set to a valid handle.

The attribute value shall be set to the current value of attribute identified by the attribute handle.

If the attribute value is longer than (ATT_MTU-3) octets, then only the first (ATT_MTU-3) octets of this attribute's value can be sent in a notification.

Note: for a client to get a long attribute, it would have to use the *Read Blob Request* following the receipt of this notification.

If the attribute handle or the attribute value is invalid, then this notification shall be ignored upon reception.

3.4.7.2 Handle Value Indication

A server can send an indication of an attribute's value.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x1D = Handle Value Indication
Attribute Handle	2	The handle of the attribute
Attribute Value	0 to (ATT_MTU-3)	The current value of the attribute

Table 3.35: Format of Handle Value Indication

The attribute handle shall be set to a valid handle.

The attribute value shall be set to the current value of attribute identified by the attribute handle.

If the attribute value is longer than (ATT_MTU-3) octets, then only the first (ATT_MTU - 3) octets of this attribute's value can be sent in an indication.

Note: For a client to get a long attribute, it would have to use the *Read Blob Request* following the receipt of this indication.

The client shall send a *Handle Value Confirmation* in response to a *Handle Value Indication*. No further indications to this client shall occur until the confirmation has been received by the server.

If the attribute handle or the attribute value is invalid, the client shall send a handle value confirmation in response and shall discard the handle and value from the received indication.

3.4.7.3 Handle Value Confirmation

The *Handle Value Confirmation* is sent in response to a received *Handle Value Indication* and confirms that the client has received an indication of the given attribute.

Parameter	Size (octets)	Description
Attribute Opcode	1	0x1E = Handle Value Confirmation

Table 3.36: Format of Handle Value Confirmation

3.4.8 Attribute Opcode Summary

Table 3.37 gives a summary of the attribute protocol PDUs.

Attribute PDU Name	Attribute Opcode	Parameters
Error Response	0x01	Request Opcode in Error, Attribute Handle In Error, Error Code
Exchange MTU Request	0x02	Client Rx MTU
Exchange MTU Response	0x03	Server Rx MTU
Find Information Request	0x04	Starting Handle, Ending Handle, UUID
Find Information Response	0x05	Format, Information Data
Find By Type Value Request	0x06	Starting Handle, Ending Handle, Attribute Type, Attribute Value
Find By Type Value Response	0x07	Handles Information List
Read By Type Request	0x08	Starting Handle, Ending Handle, UUID
Read By Type Response	0x09	Length, Attribute Data List
Read Request	0x0A	Attribute Handle
Read Response	0x0B	Attribute Value
Read Blob Request	0x0C	Attribute Handle, Value Offset
Read Blob Response	0x0D	Part Attribute Value
Read Multiple Request	0x0E	Handle Set
Read Multiple Response	0x0F	Value Set
Read by Group Type Request	0x10	Start Handle, Ending Handle, UUID
Read by Group Type Response	0x11	Length, Attribute Data List
Write Request	0x12	Attribute Handle, Attribute Value
Write Response	0x13	-

Table 3.37: Attribute Protocol Summary

Attribute PDU Name	Attribute Opcode	Parameters
Write Command	0x52	Attribute Handle, Attribute Value
Prepare Write Request	0x16	Attribute Handle, Value Offset, Part Attribute Value
Prepare Write Response	0x17	Attribute Handle, Value Offset, Part Attribute Value
Execute Write Request	0x18	Flags
Execute Write Response	0x19	-
Handle Value Notification	0x1B	Attribute Handle, Attribute Value
Handle Value Indication	0x1D	Attribute Handle, Attribute Value
Handle Value Confirmation	0x1E	
Signed Write Command	0xD2	Attribute Handle, Attribute Value, Authentication Signature

Table 3.37: *Attribute Protocol Summary*

3.4.9 Attribute PDU Response Summary

Table 3.38 gives a summary of the Attribute PDU Method responses that are allowed. Each method indicates the method that should be sent as a successful response, and whether an Error Response can be sent in response instead. If an Error Response can be sent, then the table also indicates the Error Codes that are valid within this Error Response for the given method.

Attribute PDU Method	Successful Response	Error Response Allowed	Error Response Error Codes
Exchange MTU Request	Exchange MTU Response	Yes	Request Not Supported
Find Information Request	Find Information Response	Yes	Invalid Handle, Attribute Not Found
Find By Type Value Request	Find By Type Value Response	Yes	Invalid Handle, Request Not Supported, Attribute Not Found
Read By Type Request	Read By Type Response	Yes	Invalid Handle, Request Not Supported, Attribute Not Found, Insufficient Authorization, Insufficient Authentication, Insufficient Encryption, Insufficient Encryption Key Size, Read Not Permitted, Application Error
Read Request	Read Response	Yes	Invalid Handle, Insufficient Authorization, Insufficient Authentication, Insufficient Encryption, Insufficient Encryption Key Size, Read Not Permitted, Application Error

Table 3.38: Attribute Request and Response Summary

Attribute Protocol (ATT)

Attribute PDU Method	Successful Response	Error Response Allowed	Error Response Error Codes
Read Blob Request	Read Blob Response	Yes	Invalid Handle, Request Not Supported, Insufficient Authorization, Insufficient Authentication, Insufficient Encryption, Insufficient Encryption Key Size, Read Not Permitted, Invalid Offset, Attribute Not Long, Application Error
Read Multiple Request	Read Multiple Response	Yes	Invalid Handle, Request Not Supported, Insufficient Authorization, Insufficient Authentication, Insufficient Encryption, Insufficient Encryption Key Size, Read Not Permitted, Application Error
Read by Group Type Request	Read by Group Type Response	Yes	Invalid Handle, Request Not Supported, Attribute Not Found, Insufficient Authorization, Insufficient Authentication, Insufficient Encryption, Insufficient Encryption Key Size Read Not Permitted, Unsupported Group Type, Application Error
Write Request	Write Response	Yes	Invalid Handle, Request Not Supported, Insufficient Authorization, Insufficient Authentication, Insufficient Encryption, Insufficient Encryption Key Size, Write Not Permitted, Invalid Attribute Value Length, Application Error

Table 3.38: Attribute Request and Response Summary

Attribute Protocol (ATT)

Attribute PDU Method	Successful Response	Error Response Allowed	Error Response Error Codes
Write Command	N/A	No	
Signed Write Command	N/A	No	
Prepare Write Request	Prepare Write Response	Yes	Invalid Handle, Request Not Supported, Insufficient Authorization, Insufficient Authentication, Write Not Permitted, Prepare Queue Full, Insufficient Encryption, Insufficient Encryption Key Size, Application Error
Execute Write Request	Execute Write Response	Yes	Application Error, Invalid Offset, Invalid Attribute Value Length
Handle Value Notification	N/A	No	
Handle Value Indication	Handle Value Confirmation	No	

Table 3.38: Attribute Request and Response Summary

4 SECURITY CONSIDERATIONS

The attribute protocol can be used to access information that may require both authorization and an authenticated and encrypted physical link before an attribute can be read or written.

If such a request is issued when the client has not been authorized to access this information, the server shall send an *Error Response* with the error code set to «Insufficient Authorization». The authorization requirements for access to a given attribute are not defined in this specification. Each device implementation will determine how authorization occurs. Authorization procedures are defined in GAP, and may be further refined in a higher layer specification.

If such a request is issued when the physical link is unauthenticated, the server shall send an *Error Response* with the error code set to «Insufficient Authentication». A client wanting to read or write this attribute can then request that the physical link be authenticated, and once this has been completed, send the request again.

The attribute protocol can be used to notify or indicate the value of an attribute that may require an authenticated and encrypted physical link before an attribute notification or indication is performed. A server wanting to notify or indicate this attribute can then request that the physical link be authenticated, and once this has been completed, send the notification or indication.

The list of attributes that a device supports is not considered private or confidential information, and therefore the *Find Information Request* shall always be permitted. This implies that an «Insufficient Authorization» or «Insufficient Authentication» error code shall not be used in an *Error Response* for a *Find Information Request*.

For example, an attribute value may be allowed to be read by any device, but only written by an authenticated device. An implementation should take this into account, and not assume that just because it can read an attribute's value, it will also be able to write the value. Similarly, just because an attribute value can be written, does not mean that an attribute value can also be read. Each individual attribute could have different security requirements.

When a client accesses an attribute, the order of checks that are performed on the server will have security implications. A server shall check authentication and authorization requirements before any other check is performed.

Note: For example, if the authentication and authorization requirement checks are not performed first then the size of an attribute could be determined by performing repeated read blob requests on an attribute that a client does not have access to, because either an «Invalid Offset» error code or «Insufficient Authentication» error codes would be returned.

Core System Package [Host volume]
Part G

GENERIC ATTRIBUTE PROFILE (GATT)

This specification defines the Generic Attribute Profile that describes a service framework using the Attribute Protocol for discovering services, and for reading and writing characteristic values on a peer device.

CONTENTS

1	Introduction	519
1.1	Scope	519
1.2	Profile Dependency	519
1.3	Conformance	519
1.4	Bluetooth Specification Release Compatibility.....	520
1.5	Conventions.....	520
2	Profile Overview.....	521
2.1	Protocol Stack.....	521
2.2	Configurations and Roles	521
2.3	User Requirements and Scenarios.....	522
2.4	Profile Fundamentals.....	522
2.5	Attribute Protocol	523
2.5.1	Overview	523
2.5.2	Attribute Caching	524
2.5.3	Attribute Grouping.....	525
2.5.4	UUIDs	526
2.6	GATT Profile Hierarchy.....	527
2.6.1	Overview	527
2.6.2	Service	528
2.6.3	Included Services.....	528
2.6.4	Characteristic.....	529
2.7	Configured Broadcast.....	529
3	Service Interoperability Requirements	530
3.1	Service Definition.....	530
3.2	Include Definition	531
3.3	Characteristic Definition.....	531
3.3.1	Characteristic Declaration.....	532
3.3.1.1	Characteristic Properties	533
3.3.1.2	Characteristic Value Attribute Handle.....	533
3.3.1.3	Characteristic UUID.....	533
3.3.2	Characteristic Value Declaration	534
3.3.3	Characteristic Descriptor Declarations.....	534
3.3.3.1	Characteristic Extended Properties	535
3.3.3.2	Characteristic User Description	536
3.3.3.3	Client Characteristic Configuration	536
3.3.3.4	Server Characteristic Configuration	538
3.3.3.5	Characteristic Presentation Format	539
3.3.3.6	Characteristic Aggregate Format	542
3.4	Summary of GATT Profile Attribute Types.....	543

4	GATT Feature Requirements	544
4.1	Overview.....	544
4.2	Feature Support and Procedure Mapping	544
4.3	Server Configuration.....	546
4.3.1	Exchange MTU	546
4.4	Primary Service Discovery.....	547
4.4.1	Discover All Primary Services.....	547
4.4.2	Discover Primary Service by Service UUID	548
4.5	Relationship Discovery	550
4.5.1	Find Included Services.....	550
4.6	Characteristic Discovery	551
4.6.1	Discover All Characteristics of a Service	551
4.6.2	Discover Characteristics by UUID.....	553
4.7	Characteristic Descriptor Discovery.....	554
4.7.1	Discover All Characteristic Descriptors.....	554
4.8	Characteristic Value Read	556
4.8.1	Read Characteristic Value	556
4.8.2	Read Using Characteristic UUID	556
4.8.3	Read Long Characteristic Values.....	557
4.8.4	Read Multiple Characteristic Values	558
4.9	Characteristic Value Write	559
4.9.1	Write Without Response	559
4.9.2	Signed Write Without Response	560
4.9.3	Write Characteristic Value.....	560
4.9.4	Write Long Characteristic Values	561
4.9.5	Reliable Writes.....	562
4.10	Characteristic Value Notification	564
4.10.1	Notifications	564
4.11	Characteristic Value Indications.....	565
4.11.1	Indications.....	565
4.12	Characteristic Descriptors.....	566
4.12.1	Read Characteristic Descriptors	566
4.12.2	Read Long Characteristic Descriptors	566
4.12.3	Write Characteristic Descriptors	567
4.12.4	Write Long Characteristic Descriptors	568
4.13	GATT Procedure Mapping to ATT Protocol Opcodes	569
4.14	Procedure Timeouts	571
5	L2CAP Interoperability Requirements	572
5.1	BR/EDR L2CAP Interoperability Requirements	572

Generic Attribute Profile (GATT)

5.1.1	ATT_MTU.....	572
5.1.2	BR/EDR Channel Requirements.....	572
5.1.3	BR/EDR Channel Establishment Collisions	573
5.2	LE L2CAP Interoperability Requirements	573
5.2.1	ATT_MTU.....	573
5.2.2	LE Channel Requirements.....	574
6	GAP Interoperability Requirements	575
6.1	BR/EDR GAP Interoperability Requirements.....	575
6.1.1	Connection Establishment	575
6.2	LE GAP Interoperability Requirements.....	575
6.2.1	Connection Establishment	575
6.2.2	Profile Roles.....	575
6.3	Disconnected Events	576
6.3.1	Notifications and Indications While Disconnected	576
7	Defined Generic Attribute Profile Service	577
7.1	Service Changed	577
8	Security Considerations	579
8.1	Authentication Requirements.....	579
8.2	Authorization Requirements	580
9	SDP Interoperability Requirements.....	581
10	References	582
Appendix A	Example Attribute Server Attributes	583

1 INTRODUCTION

1.1 SCOPE

The Generic Attribute Profile (GATT) defines a service framework using the Attribute Protocol. This framework defines procedures and formats of services and their characteristics. The procedures defined include discovering, reading, writing, notifying and indicating characteristics, as well as configuring the broadcast of characteristics.

1.2 PROFILE DEPENDENCY

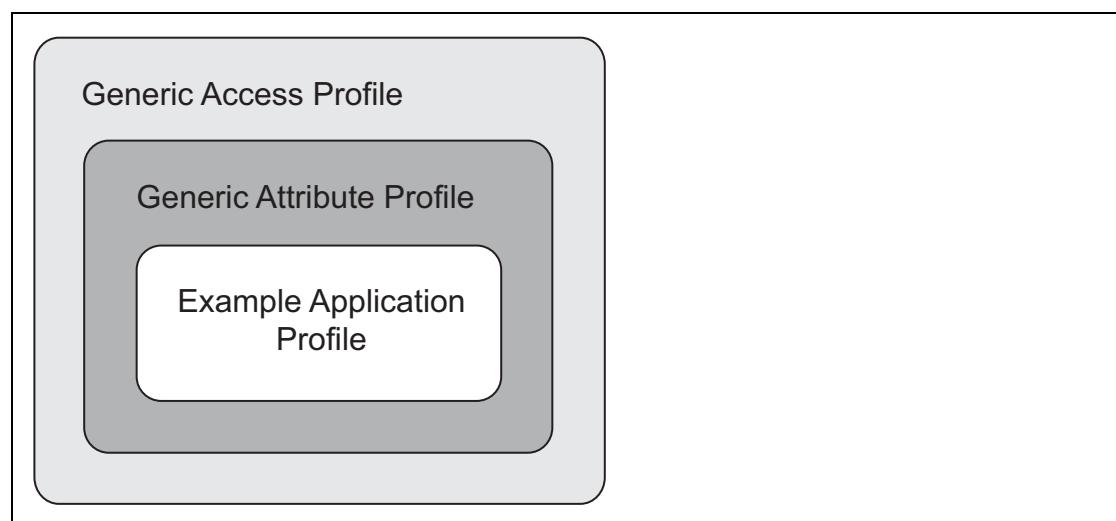


Figure 1.1: *Profile dependencies*

Figure 1.1 depicts the structure and the dependencies of the profiles. A profile is dependent upon another profile if it re-uses parts of that profile by implicitly or explicitly referencing it.

1.3 CONFORMANCE

If conformance to this profile is claimed, all capabilities indicated as mandatory for this profile shall be supported in the specified manner (process-mandatory). This also applies for all optional and conditional capabilities for which support is indicated. All mandatory capabilities, and optional and conditional capabilities for which support is indicated, are subject to verification as part of the Bluetooth qualification program.

1.4 BLUETOOTH SPECIFICATION RELEASE COMPATIBILITY

This specification can be used with Bluetooth Core Specification Version 1.2 or later when using the profile on the BR/EDR physical link and Bluetooth Core Specification Version 4.0 or later when using the profile on the LE physical link.

1.5 CONVENTIONS

In this specification the use of literal terms such as procedure, PDUs, opcodes or function names appear in italics. Specific names of fields in structures, packets, etc. also appear in italics. The use of « » (e.g. «Primary Service») indicate a Bluetooth SIG-defined UUID.

2 PROFILE OVERVIEW

The GATT profile is designed to be used by an application or another profile, so that a client can communicate with a server. The server contains a number of attributes, and the GATT Profile defines how to use the Attribute Protocol to discover, read, write and obtain indications of these attributes, as well as configuring broadcast of attributes.

2.1 PROTOCOL STACK

[Figure 2.1](#) shows the peer protocols used by this profile.

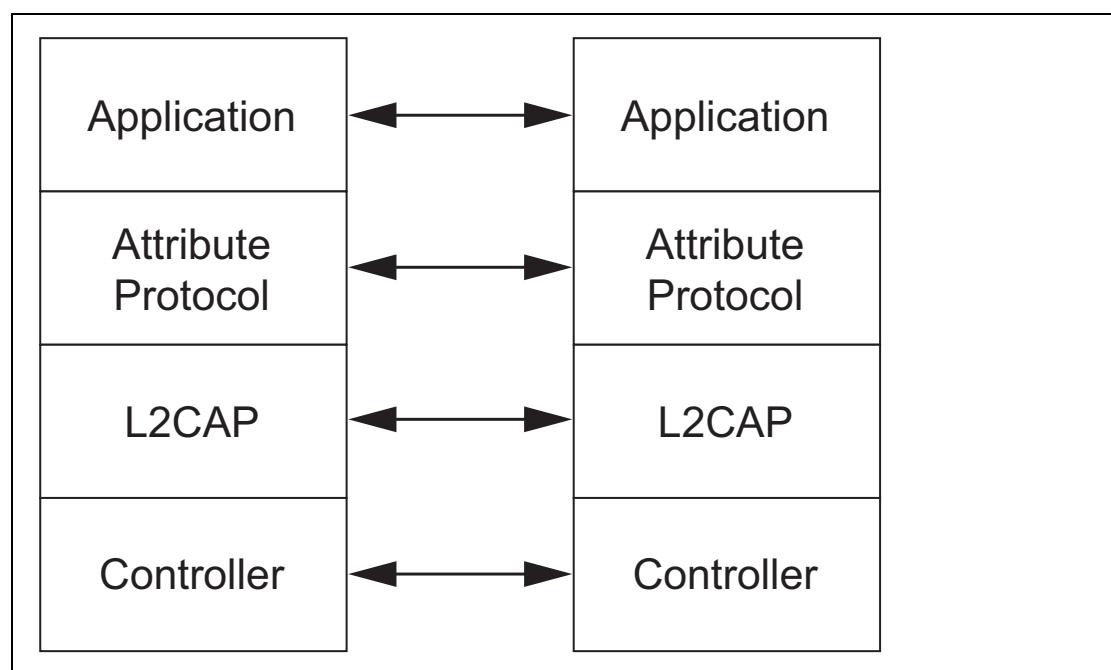


Figure 2.1: Protocol model

2.2 CONFIGURATIONS AND ROLES

The following roles are defined for devices that implement this profile:

Client—This is the device that initiates commands and requests towards the server and can receive responses, indications and notifications sent by the server.

Server—This is the device that accepts incoming commands and requests from the client and sends responses, indications and notifications to a client.

Note: The roles are not fixed to the device. The roles are determined when a device initiates a defined procedure, and they are released when the procedure ends.

A device can act in both roles at the same time.

An example of configurations illustrating the roles for this profile is depicted in [Figure 2.2](#).

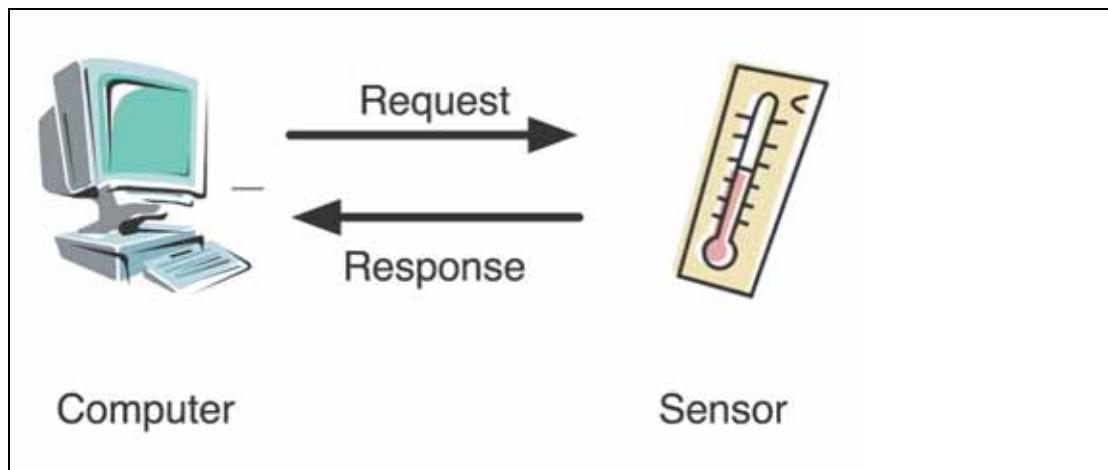


Figure 2.2: Examples of configuration

In [Figure 2.2](#), the computer is the temperature service client and the sensor is the temperature service server. The computer initiates procedures to configure the sensor or to read the sensor values. In this example the sensor provides information about the characteristics the sensor device exposes as part of the temperature service and may permit some characteristics to be written. Also, the sensor responds to read requests with the appropriate values.

2.3 USER REQUIREMENTS AND SCENARIOS

The following scenarios are covered by this profile:

- Exchanging configuration
- Discovery of services and characteristics on a device
- Reading a characteristic value
- Writing a characteristic value
- Notification of a characteristic value
- Indication of a characteristic value

2.4 PROFILE FUNDAMENTALS

This profile can be used over any physical link, using the Attribute Protocol L2CAP channel, known as the ATT Bearer. Here is a brief summary of lower layer requirements communication between the client and the server.

- An ATT Bearer is established using “Channel Establishment” as defined in [Section 6](#).

- The profile roles are not tied to the controller master/slave roles.
- On an LE Physical link, use of security features such as authorization, authentication and encryption are optional. On a BR/EDR physical link encryption is mandatory.
- Multi-octet fields within the GATT Profile shall be sent least significant octet first (little endian).

2.5 ATTRIBUTE PROTOCOL

The GATT Profile requires the implementation of the [Attribute Protocol \(ATT\)](#) and the required Attribute opcodes indicated in [Section 4.2](#) and [Section 4.13](#).

2.5.1 Overview

The GATT Profile uses the Attribute Protocol to transport data in the form of commands, requests, responses, indications, notifications and confirmations between devices. This data is contained in Attribute Protocol PDUs.

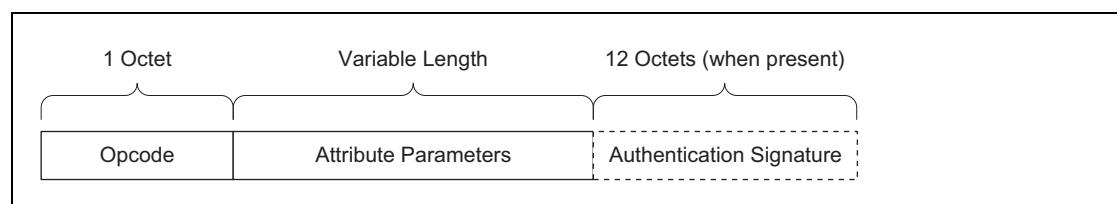


Figure 2.3: Attribute Protocol PDU

The *Opcode* contains the specific command, request, response, indication, notification or confirmation opcode and a flag for authentication. The *Attribute Parameters* contain data for the specific command or request or the data returned in a response, indication or notification. The *Authentication Signature* is optional and is described in [\[Vol. 3\] Part H, Section 2.4.5](#).

Attribute Protocol commands and requests act on values stored in Attributes on the server device. An Attribute is composed of four parts: *Attribute Handle*, *Attribute Type*, *Attribute Value*, and *Attribute Permissions*. [Figure 2.4](#) shows a logical representation of an Attribute. The actual representation for a given implementation is specific to that implementation.

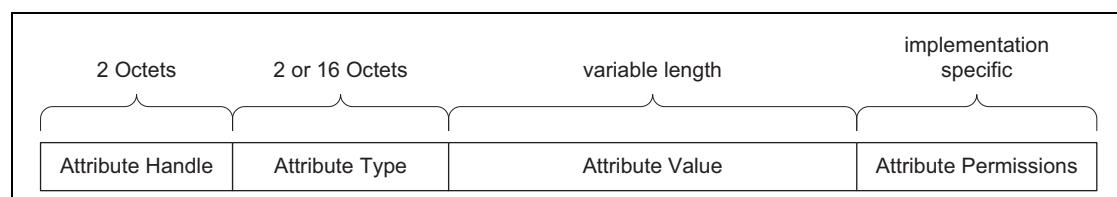


Figure 2.4: Logical Attribute Representation

The *Attribute Handle* is an index corresponding to a specific Attribute. The *Attribute Type* is a UUID that describes the *Attribute Value*. The *Attribute Value* is the data described by the *Attribute Type* and indexed by the *Attribute Handle*. The Attributes are ordered by increasing *Attribute Handle* values. *Attribute Handle* values may begin at any value between 0x0001 and 0xFFFF. Although the *Attribute Handle* values are in increasing order, following *Attribute Handle* values may differ by more than one. That is to say there may be gaps between successive *Attribute Handles*.

Attribute Permissions is part of the Attribute that cannot be read from or written to using the Attribute Protocol. It is used by the server to determine whether read or write access is permitted for a given attribute. *Attribute Permissions* are established by the GATT profile, a higher layer profile or are implementation specific if not specified.

2.5.2 Attribute Caching

Attribute caching is an optimization that allows the client to discover the Attribute information such as *Attribute Handles* used by the server once and use the same Attribute information across reconnections without rediscovery. Without caching the Attribute information, the client shall rediscover the Attribute information at each reconnection. With caching, time is saved and a significant amount of packets exchanged between the client and server is not required. The Attribute information that shall be cached by a client is the *Attribute Handles* of all server attributes and the GATT service characteristics values.

Attribute Handles used by the server should not change over time. This means that once an *Attribute Handle* is discovered by a client the *Attribute Handle* for that Attribute should not be changed.

Some circumstances may cause servers to change the *Attribute Handles* used for services, perhaps due to a factory reset or a firmware upgrade procedure being performed. The following is only required on the server if the services on the server can be added, modified or removed. If GATT based services on the server cannot be changed during the usable lifetime of the device, the *Service Changed* characteristic shall not exist on the server and the client does not need to ever perform service discovery after the initial service discovery for that server.

To support caching when a server supports changes in GATT based services, an indication is sent by the server to clients when a service is added, removed, or modified on the server. A GATT based service is considered modified if the binding of the *Attribute Handles* to the associated Attributes grouped within a service definition are changed. Any change to the GATT service definition characteristic values other than the *Service Changed* characteristic value itself shall also be considered a modification.

For clients that have a trusted relationship (i.e. bond) with the server, the attribute cache is valid across connections. For clients with a trusted relationship and not in a connection when a service change occurs, the server shall send an indication when the client reconnects to the server. For clients that do not have a trusted relationship with the server, the attribute cache is valid only during the connection. Clients without a trusted relationship shall receive an indication when the service change occurs only during the current connection.

Note: Clients without a trusted relationship must perform service discovery on each connection if the server supports the *Service Changed* characteristic.

The server shall send a *Handle Value Indication* containing the range of affected *Attribute Handles* that shall be considered invalid in the client's attribute cache. The start *Attribute Handle* shall be the start *Attribute Handle* of the service definition containing the change and the end *Attribute Handle* shall be the last *Attribute Handle* of the service definition containing the change. The value in the indication is composed of two 16-bit *Attribute Handles* concatenated to indicate the affected *Attribute Handle* range.

Note: A server may set the affected *Attribute Handle* range to 0x0001 to 0xFFFF to indicate to the client to rediscover the entire set of *Attribute Handles* on the server.

The client, upon receiving a Handle Value Indication containing the range of affected Attribute Handles, shall consider the attribute cache invalid over the affected Attribute Handle range. Any outstanding request transaction shall be considered invalid if the Attribute Handle is contained within the affected Attribute Handle range. The client must perform service discovery before the client uses any service that has an attribute within the affected Attribute Handle range.

Once the server has received the Handle Value Confirmation, the server can consider the client to be aware of the updated Attribute Handles.

The client shall consider the affected *Attribute Handle* range to be invalid in its attribute cache and perform the discovery procedures to restore the attribute cache. The server shall store service changed information for all bonded devices.

2.5.3 Attribute Grouping

Generic attribute profile defines grouping of attributes for three attribute types: «Primary Service», «Secondary Service» and «Characteristic». A group begins with a declaration, and ends as defined in [Section 3.1](#) for services and [Section 3.3](#) for characteristics. Not all of the grouping attributes can be used in the ATT Read By Group Type Request. The «Primary Service» and «Secondary Service» grouping types may be used in the Read By Group Type Request.

The «Characteristic» grouping type shall not be used in the ATT Read By Group Type Request.

2.5.4 UUIDs

All 16-bit UUIDs shall be contained in exactly 2 octets. All 128-bit UUIDs shall be contained in exactly 16 octets.

All 32-bit UUIDs shall be converted to 128-bit UUIDs when the UUID is contained in an ATT PDU. See [\[Vol. 3\] Part F, Section 3.2.1](#) for the method of conversion.

2.6 GATT PROFILE HIERARCHY

2.6.1 Overview

The GATT Profile specifies the structure in which profile data is exchanged. This structure defines basic elements such as services and characteristics, used in a profile. All of the elements are contained by Attributes. Attributes used in the Attribute Protocol are containers that carry this profile data.

The top level of the hierarchy is a profile. A profile is composed of one or more services necessary to fulfill a use case. A service is composed of characteristics or references to other services. Each characteristic contains a value and may contain optional information about the value. The service and characteristic and the components of the characteristic (i.e. value and descriptors) contain the profile data and are all stored in Attributes on the server.

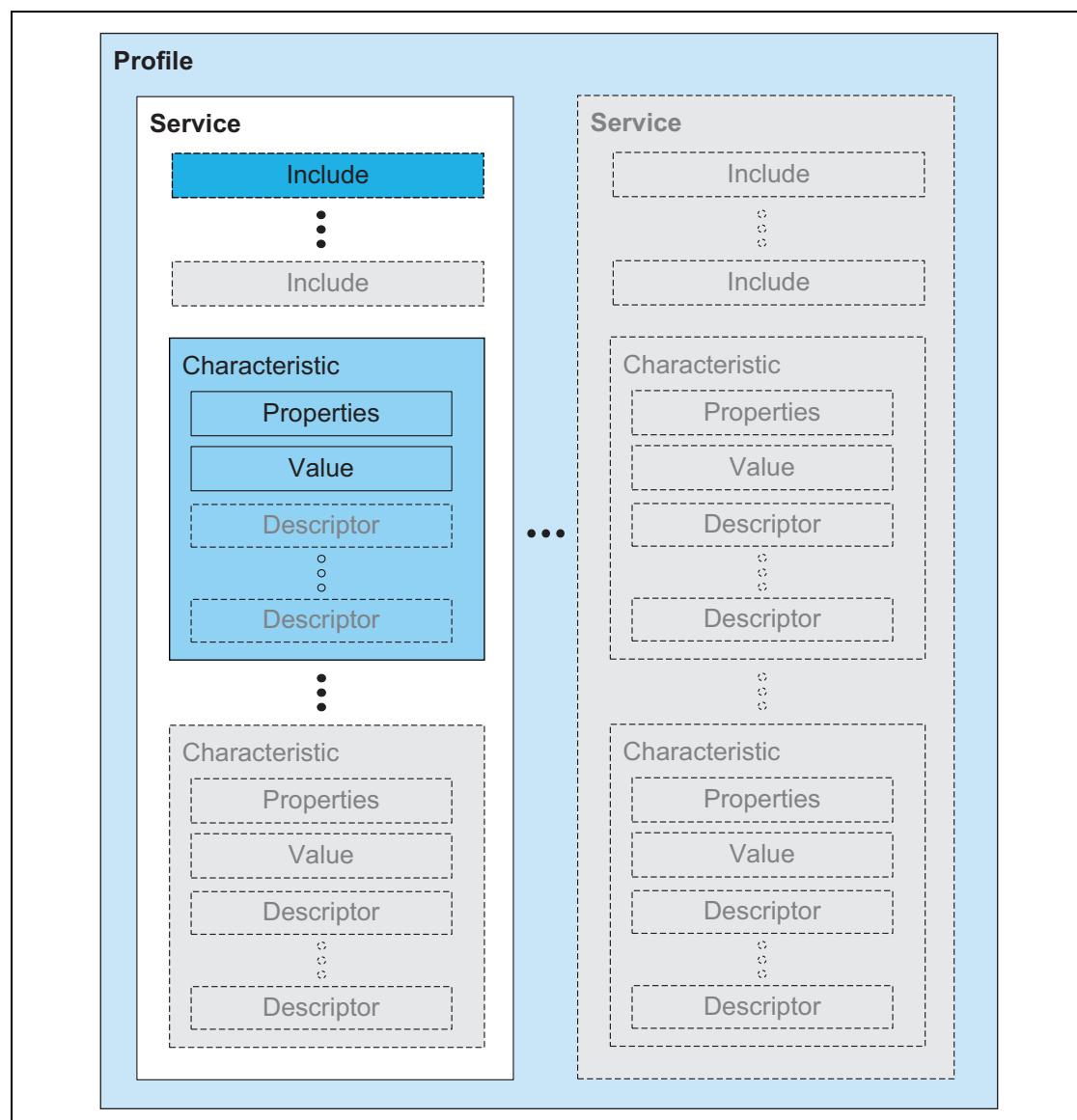


Figure 2.5: GATT Profile hierarchy

2.6.2 Service

A service is a collection of data and associated behaviors to accomplish a particular function or feature. In GATT, a service is defined by its service definition. A service definition may contain referenced services, mandatory characteristics and optional characteristics.

To maintain backward compatibility with earlier clients, later versions of a service definition can only add new referenced services or optional characteristics. Later versions of a service definition are forbidden from changing behaviors from previous versions of the service definition.

There are two types of services: primary service and secondary service. A primary service is a service that exposes the primary usable functionality of this device. A primary service can be included by another service. Primary services can be discovered using Primary Service Discovery procedures. A secondary service is a service that is only intended to be referenced from a primary service or another secondary service or other higher layer specification. A secondary service is only relevant in the context of the entity that references it.

The determination of whether a service is either a primary or secondary service can be mandated by a higher layer specification.

Services may be used in one or more higher layer specifications to fulfill a particular use case.

The service definition is described in [Section 3.1](#).

2.6.3 Included Services

An included service is a method to reference another service definition existing on the server into the service being defined. To include another service, an include definition is used at the beginning of the service definition. When a service definition uses an include definition to reference the included service, the entire included service definition becomes part of the new service definition. This includes all the included services and characteristics of the included service. The included service still exists as an independent service. A service that is included by another service shall not be changed by the act of inclusion or by the including service. There are no limits to the number of include definitions or the depth of nested includes in a service definition.

The include definition is described in [Section 3.2](#).

2.6.4 Characteristic

A characteristic is a value used in a service along with properties and configuration information about how the value is accessed and information about how the value is displayed or represented. In GATT, a characteristic is defined by its characteristic definition. A characteristic definition contains a characteristic declaration, characteristic properties, and a value and may contain descriptors that describe the value or permit configuration of the server with respect to the characteristic.

The characteristic definition is described in [Section 3.3](#).

2.7 CONFIGURED BROADCAST

For LE physical links, Configured Broadcast is a method for a client to indicate to a server which *Characteristic Value* shall be broadcast in the advertising data when the server is executing the broadcast mode procedure. For BR/EDR physical links, Configured Broadcast is not supported.

To configure a *Characteristic Value* to be broadcast by the server when in broadcast mode, the client sets the broadcast configuration bit described in [Section 3.3.3.3](#). The frequency of the broadcast is part of the service or characteristic behavior definition.

3 SERVICE INTEROPERABILITY REQUIREMENTS

3.1 SERVICE DEFINITION

A service definition shall contain a service declaration and may contain include definitions and characteristic definitions. The service definition ends before the next service declaration or after the maximum *Attribute Handle* is reached. Service definitions appear on the server in an order based on *Attribute Handle*.

All include definitions and characteristic definitions contained within the service definition are considered to be part of the service. All include definitions shall immediately follow the service declaration and precede any characteristic definitions. A service definition may have zero or more include definitions. All characteristic definitions shall be immediately following the last include definition or in the event of no include definitions, immediately following the service declaration. A service definition may have zero or more characteristic definitions. There is no upper limit for include or characteristic definitions.

A service declaration is an Attribute with the *Attribute Type* set to the UUID for «Primary Service» or «Secondary Service». The *Attribute Value* shall be the 16-bit Bluetooth UUID or 128-bit UUID for the service, known as the service UUID. A client shall support the use of both 16-bit and 128-bit UUIDs. A client may ignore any service definition with an unknown service UUID. An unknown service UUID is a UUID for an unsupported service. The *Attribute Permissions* shall be read-only and shall not require authentication or authorization.

When multiple services exist, services definitions with service declarations using 16-bit Bluetooth UUID should be grouped together (i.e. listed sequentially) and services definitions with service declarations using 128-bit UUID should be grouped together.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	0x2800 – UUID for «Primary Service» OR 0x2801 for «Secondary Service»	16-bit Bluetooth UUID or 128-bit UUID for Service	Read Only, No Authentication, No Authorization

Table 3.1: Service Declaration

A device or higher level specification may have multiple service definitions and may have multiple service definitions with the same service UUID.

All Attributes on a Server shall either contain a service declaration or exist within a service definition.

Service definitions contained in a server may appear in any order; a client shall not assume the order of service definitions on a server.

3.2 INCLUDE DEFINITION

An include definition shall contain only one include declaration.

The include declaration is an Attribute with the *Attribute Type* set to the UUID for «Include». The *Attribute Value* shall be set to the included service *Attribute Handle*, the End Group Handle, and the *service UUID*. The Service UUID shall only be present when the UUID is a 16-bit Bluetooth UUID. The *Attribute Permissions* shall be read only and not require authentication or authorization.

Attribute Handle	Attribute Type	Attribute Value			Attribute Permission
0xNNNN	0x2802 – UUID for «Include»	Included Service Attribute Handle	End Group Handle	Service UUID	Read Only, No Authentication, No Authorization

Table 3.2: *Include Declaration*

A server shall not contain a service definition with an include definition to another service that references the original service. This applies to each of the services the included definition references. This is referred to as a circular reference.

If the client detects a circular reference or detects nested include declarations to a greater level than it expects, it should terminate the ATT Bearer.

3.3 CHARACTERISTIC DEFINITION

A characteristic definition shall contain a characteristic declaration, a Characteristic Value declaration and may contain characteristic descriptor declarations. A characteristic definition ends at the start or the next characteristic declaration or service declaration or after the maximum *Attribute Handle*. Characteristic definitions appear on the server within a service definition in an order based on *Attribute Handle*.

Each declaration above is contained in a separate Attribute. The two required declarations are the characteristic declaration and the Characteristic Value declaration. The Characteristic Value declaration shall exist immediately following the characteristic declaration. Any optional characteristic descriptor declarations are placed after the Characteristic Value declaration. The order of the optional characteristic descriptor declarations is not significant.

A characteristic definition may be defined to concatenate several Characteristic Values into a single aggregated Characteristic Value. This may be used to optimize read and writes of multiple Characteristic Values through the reading and writing of a single aggregated Characteristic Value. This type of characteristic definition is the same as a normal characteristic definition. The characteristic declaration shall use a characteristic UUID that is unique to the

aggregated characteristic definition. The aggregated characteristic definition may also contain a characteristic aggregate format descriptor that describes the display format of the aggregated Characteristic Value.

3.3.1 Characteristic Declaration

A characteristic declaration is an Attribute with the Attribute Type set to the UUID for «Characteristic» and *Attribute Value* set to the Characteristic Properties, Characteristic Value *Attribute Handle* and Characteristic UUID. The Attribute Permissions shall be readable and not require authentication or authorization.

The characteristic declaration *Attribute Value* shall not change while the server has a trusted relationship with any client.

Attribute Handle	Attribute Types	Attribute Value			Attribute Permissions
0xNNNN	0x2803–UUID for «Characteristic»	Characteristic Properties	Characteristic Value Attribute Handle	Characteristic UUID	Read Only, No Authentication, No Authorization

Table 3.3: Characteristic declaration

The *Attribute Value* of a characteristic declaration is read only.

Attribute Value	Size	Description
Characteristic Properties	1 octets	Bit field of characteristic properties
Characteristic Value Handle	2 octets	Handle of the Attribute containing the value of this characteristic
Characteristic UUID	2 or 16 octets	16-bit Bluetooth UUID or 128-bit UUID for Characteristic Value

Table 3.4: Attribute Value Field in characteristic declaration

A service may have multiple characteristic definitions with the same Characteristic UUID.

Within a service definition, some characteristics may be mandatory and those characteristics shall be located after the include declarations and before any optional characteristics within the service definition. A client shall not assume any order of those characteristics that are mandatory or any order of those characteristics that are optional within a service definition. Whenever possible and within the requirements stated earlier, characteristics definitions with characteristic declarations using 16-bit Bluetooth UUIDs should be group together (i.e. listed sequentially) and characteristics definitions with characteristic declarations using 128-bit UUIDs should be grouped together.

3.3.1.1 Characteristic Properties

The Characteristic Properties bit field determines how the Characteristic Value can be used, or how the characteristic descriptors (see Section 3.3.3) can be accessed. If the bits defined in Table 3.5 are set, the action described is permitted. Multiple Characteristic Properties can be set.

These bits shall be set according to the procedures this characteristic supports, without regard to security requirements.

Properties	Value	Description
Broadcast	0x01	If set, permits broadcasts of the Characteristic Value using Server Characteristic Configuration Descriptor. If set, the Server Characteristic Configuration Descriptor shall exist.
Read	0x02	If set, permits reads of the Characteristic Value using procedures defined in Section 4.8
Write Without Response	0x04	If set, permit writes of the Characteristic Value without response using procedures defined in Section 4.9.1 .
Write	0x08	If set, permits writes of the Characteristic Value with response using procedures defined in Section 4.9.3 or Section 4.9.4 .
Notify	0x10	If set, permits notifications of a Characteristic Value without acknowledgement using the procedure defined in Section 4.10 . If set, the Client Characteristic Configuration Descriptor shall exist.
Indicate	0x20	If set, permits indications of a Characteristic Value with acknowledgement using the procedure defined in Section 4.11 . If set, the Client Characteristic Configuration Descriptor shall exist.
Authenticated Signed Writes	0x40	If set, permits signed writes to the Characteristic Value using the procedure defined in Section 4.9.2 .
Extended Properties	0x80	If set, additional characteristic properties are defined in the Characteristic Extended Properties Descriptor defined in Section 3.3.3.1 . If set, the Characteristic Extended Properties Descriptor shall exist.

Table 3.5: Characteristic Properties bit field

3.3.1.2 Characteristic Value Attribute Handle

The Characteristic Value *Attribute Handle* field is the *Attribute Handle* of the Attribute that contains the *Characteristic Value*.

3.3.1.3 Characteristic UUID

The *Characteristic UUID* field is a 16-bit Bluetooth UUID or 128-bit UUID that describes the type of *Characteristic Value*. A client shall support the use of both 16-bit and 128-bit *Characteristic UUIDs*. A client may ignore any characteristic definition with an unknown *Characteristic UUID*. An unknown characteristic UUID is a UUID for an unsupported characteristic.

3.3.2 Characteristic Value Declaration

The *Characteristic Value* declaration contains the value of the characteristic. It is the first Attribute after the characteristic declaration. All characteristic definitions shall have a *Characteristic Value* declaration.

A Characteristic Value declaration is an Attribute with the Attribute Type set to the 16-bit Bluetooth or 128-bit UUID for the Characteristic Value used in the characteristic declaration. The *Attribute Value* is set to the *Characteristic Value*. The *Attribute Permissions* are specified by the service or may be implementation specific if not specified otherwise.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0xuuuu – 16-bit Bluetooth UUID or 128-bit UUID for Characteristic UUID	Characteristic Value	Higher layer profile or implementation specific

Table 3.6: *Characteristic Value declaration*

3.3.3 Characteristic Descriptor Declarations

Characteristic descriptors are used to contain related information about the *Characteristic Value*. The GATT profile defines a standard set of characteristic descriptors that can be used by higher layer profiles. Higher layer profiles may define additional characteristic descriptors that are profile specific. Each characteristic descriptor is identified by the characteristic descriptor UUID. A client shall support the use of both 16-bit and 128-bit characteristic descriptor UUIDs. A client may ignore any characteristic descriptor declaration with an unknown characteristic descriptor UUID. An unknown characteristic descriptor UUID is a UUID for an unsupported characteristic descriptor.

Characteristic descriptors if present within a characteristic definition shall follow the *Characteristic Value* declaration. The characteristic descriptor declaration may appear in any order within the characteristic definition. The client shall not assume the order in which a characteristic descriptor declaration appears in a characteristic definition following the *Characteristic Value* declaration.

Characteristic descriptor declaration permissions are defined by a higher layer profile or are implementation specific. A client shall not assume all characteristic descriptor declarations are readable.

3.3.3.1 Characteristic Extended Properties

The *Characteristic Extended Properties* declaration is a descriptor that defines additional *Characteristic Properties*. If the *Extended Properties* bit of the *Characteristic Properties* is set then this characteristic descriptor shall exist. The characteristic descriptor may occur in any position within the characteristic definition after the Characteristic Value. Only one *Characteristic Extended Properties* declaration shall exist in a characteristic definition.

The characteristic descriptor is contained in an Attribute and the *Attribute Type* shall be set to the UUID for «Characteristic Extended Properties» and the *Attribute Value* shall be the *Characteristic Extended Properties Bit Field*. The *Attribute Permissions* shall be readable without authentication and authorization being required.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2900 – UUID for «Characteristic Extended Properties»	Characteristic Extended Properties Bit Field	Read Only, No Authentication, No Authorization

Table 3.7: *Characteristic Extended Properties declaration*

The *Characteristic Extended Properties* bit field describes additional properties on how the Characteristic Value can be used, or how the characteristic descriptors (see [Section 3.3.3.3](#)) can be accessed. If the bits defined in Table 3 8 are set, the action described is permitted. *Multiple Characteristic Properties* can be set.

Properties	Value	Description
Reliable Write	0x0001	If set, permits reliable writes of the Characteristic Value using the procedure defined in Section 4.9.5
Writable Auxiliaries	0x0002	If set, permits writes to the characteristic descriptor defined in Section 3.3.3.2
Reserved for Future Use	0xFFFF	Reserved for Future Use

Table 3.8: *Characteristic Extended Properties bit field*

3.3.3.2 Characteristic User Description

The *Characteristic User Description* declaration is an optional characteristic descriptor that defines a UTF-8 string of variable size that is a user textual description of the *Characteristic Value*. If the *Writable Auxiliary* bit of the *Characteristic Properties* is set then this characteristic descriptor can be written. The characteristic descriptor may occur in any position within the characteristic definition after the *Characteristic Value*. Only one *Characteristic User Description* declaration shall exist in a characteristic definition.

The characteristic descriptor is contained in an Attribute and the *Attribute Type* shall be set to the UUID for «Characteristic User Description» and the *Attribute Value* shall be set to the characteristic user description UTF-8 string. The *Attribute Permissions* are specified by the profile or may be implementation specific if not specified otherwise.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2901 – UUID for «Characteristic User Description»	Characteristic User Description UTF-8 String	Higher layer profile or implementation specific

Table 3.9: *Characteristic User Description* declaration

3.3.3.3 Client Characteristic Configuration

The *Client Characteristic Configuration* declaration is an optional characteristic descriptor that defines how the characteristic may be configured by a specific client. The Client Characteristic Configuration descriptor value shall be persistent across connections for bonded devices. The Client Characteristic Configuration descriptor value shall be set to the default value at each connection with non-bonded devices. The characteristic descriptor value is a bit field. When a bit is set, that action shall be enabled, otherwise it will not be used. The *Client Characteristic Configuration* descriptor may occur in any position within the characteristic definition after the *Characteristic Value*. Only one *Client Characteristic Configuration* declaration shall exist in a characteristic definition.

A client may write this configuration descriptor to control the configuration of this characteristic on the server for the client. Each client has its own instantiation of the *Client Characteristic Configuration*. Reads of the *Client Characteristic Configuration* only shows the configuration for that client and writes only affect the configuration of that client. Authentication and authorization may be required by the server to write the configuration descriptor. The *Client Characteristic Configuration* declaration shall be readable and writable.

Generic Attribute Profile (GATT)

The characteristic descriptor is contained in an Attribute. The *Attribute Type* shall be set to the UUID for «Client Characteristic Configuration». The *Attribute Value* shall be set to the characteristic descriptor value. The *Attribute Permissions* are specified by the profile or may be implementation specific if not specified otherwise.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	02902 – UUID for «Client Characteristic Configuration»	Characteristic Configuration Bits	Readable with no authentication or authorization. Writable with authentication and authorization defined by a higher layer specification or is implementation specific.

Table 3.10: Client Characteristic Configuration declaration

The following Client Characteristic Configuration bits are defined:

Configuration	Value	Description
Notification	0x0001	The Characteristic Value shall be notified. This value can only be set if the characteristic's property has the notify bit set.
Indication	0x0002	The Characteristic Value shall be indicated. This value can only be set if the characteristic's property has the indicate bit set.
Reserved for Future Use	0xFFFF4	Reserved for future use.

Table 3.11: Client Characteristic Configuration bit field definition

The default value for the *Client Characteristic Configuration* descriptor value shall be 0x0000.

In the case where multiple ATT bearers from the same device are supported by the GATT server, each ATT bearer shall be considered to have a separate GATT client instance. Therefore each GATT client shall have a separate Client Characteristic Configuration.

3.3.3.4 Server Characteristic Configuration

The *Server Characteristic Configuration* declaration is an optional characteristic descriptor that defines how the characteristic may be configured for the server. The characteristic descriptor value is a bit field. When a bit is set, that action shall be enabled, otherwise it will not be used. The *Server Characteristic Configuration* descriptor may occur in any position within the characteristic definition after the *Characteristic Value*. Only one *Server Characteristic Configuration* declaration shall exist in a characteristic definition. The *Server Characteristic Configuration* declaration shall be readable and writable.

A client may write this configuration descriptor to control the configuration of this characteristic on the server for all clients. There is a single instantiation of the *Server Characteristic Configuration* for all clients. Reads of the *Server Characteristic Configuration* shows the configuration all clients and writes affect the configuration for all clients. Authentication and authorization may be required by the server to write the configuration descriptor.

The characteristic descriptor is contained in an Attribute. The *Attribute Type* shall be set to the UUID for «Server Characteristic Configuration». The *Attribute Value* shall be set to the characteristic descriptor value. The *Attribute Permissions* are specified by the profile or may be implementation specific if not specified otherwise.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2903 – UUID for «Server Characteristic Configuration»	Characteristic Configuration Bits	Readable with no authentication or authorization. Writable with authentication and authorization defined by a higher layer specification or is implementation specific.

Table 3.12: Server Characteristic Configuration declaration

The following *Server Characteristic Configuration* bits are defined:

Configuration	Value	Description
Broadcast	0x0001	The Characteristic Value shall be broadcast when the server is in the broadcast procedure if advertising data resources are available. This value can only be set if the characteristic's property has the broadcast bit set.
Reserved for Future Use	0xFFFF2	Reserved for future use.

Table 3.13: Server Characteristic Configuration bit field definition

3.3.3.5 Characteristic Presentation Format

The *Characteristic Presentation Format* declaration is an optional characteristic descriptor that defines the format of the *Characteristic Value*. The characteristic descriptor may occur in any position within the characteristic definition after the *Characteristic Value*. If more than one *Characteristic Presentation Format* declarations exist, in a characteristic definition, then a *Characteristic Aggregate Format* declaration shall exist as part of the characteristic definition.

The characteristic format value is composed of five parts: format, exponent, unit, name space, and description.

The characteristic descriptor is contained in an Attribute. The *Attribute Type* shall be set to the UUID for «Characteristic Format». The *Attribute Value* shall be set to the characteristic descriptor value. The *Attribute Permissions* shall be read only and not require authentication or authorization.

Attribute Handle	Attribute Type	Attribute Value					Attribute Permissions
0xNNNN	0x2904 – UUID for «Characteristic Format»	Format	Exponent	Unit	Name Space	Description	Read only No Authentication, NO authorization

Table 3.14: Characteristic Format declaration

The definition of the Characteristic Presentation Format descriptor Attribute Value field is the following.

Field Name	Value Size	Description
Format	1 octet	Format of the value of this characteristic.
Exponent	1 octet	Exponent field to determine how the value of this characteristic is further formatted.
Unit	2 octets	The unit of this characteristic as defined in [1]
Name Space	1 octet	The name space of the description as defined in [1]
Description	2 octets	The description of this characteristic as defined in a higher layer profile.

Table 3.15: Characteristic Format Value definition

3.3.3.5.1 Bit Ordering

The bit ordering used for the Characteristic Format descriptor shall be little-endian.

3.3.3.5.2 Format

The format field determines how a single value contained in the *Characteristic Value* is formatted. If a format is not a whole number of octets, then the data shall be contained within the least significant bits of the value, and all other bits shall be set to zero on transmission and ignored upon receipt. If the *Characteristic Value* is less than an octet, it occupies an entire octet.

The following format values are defined:

Format	Short Name	Description	Exponent Value
0x00	rfu	Reserved for Future Use	No
0x01	boolean	unsigned 1-bit; 0 = false, 1 = true	No
0x02	2bit	unsigned 2-bit integer	No
0x03	nibble	unsigned 4-bit integer	No
0x04	uint8	unsigned 8-bit integer	Yes
0x05	uint12	unsigned 12-bit integer	Yes
0x06	uint16	unsigned 16-bit integer	Yes
0x07	uint24	unsigned 24-bit integer	Yes
0x08	uint32	unsigned 32-bit integer	Yes
0x09	uint48	unsigned 48-bit integer	Yes
0x0A	uint64	unsigned 64-bit integer	Yes
0x0B	uint128	unsigned 128-bit integer	Yes
0x0C	sint8	signed 8-bit integer	Yes
0x0D	sint12	signed 12-bit integer	Yes
0x0E	sint16	signed 16-bit integer	Yes
0x0F	sint24	signed 24-bit integer	Yes
0x10	sint32	signed 32-bit integer	Yes
0x11	sint48	signed 48-bit integer	Yes
0x12	sint64	signed 64-bit integer	Yes
0x13	sint128	signed 128-bit integer	Yes
0x14	float32	IEEE-754 32-bit floating point	No
0x15	float64	IEEE-754 64-bit floating point	No
0x16	SFLOAT	IEEE-11073 16-bit SFLOAT	No
0x17	FLOAT	IEEE-11073 32-bit FLOAT	No
0x18	duint16	IEEE-20601 format	No

Table 3.16: Characteristic Format types

Format	Short Name	Description	Exponent Value
0x19	utf8s	UTF-8 string	No
0x1A	utf16s	UTF-16 string	No
0x1B	struct	Opaque structure	No
0x1C – 0xFF	rfu	Reserved for Future Use	No

Table 3.16: Characteristic Format types

When encoding an IPv4 address, the uint32 Format type shall be used.

When encoding an IPv6 address, the uint128 Format type shall be used.

When encoding a Bluetooth BD_ADDR, the uint48 Format type shall be used.

A uint16 is two uint16 values concatenated together.

3.3.3.5.3 Exponent

The exponent field is used with integer data types to determine how the value is further formatted. The exponent field is only used on integer format types as indicated in the format field in Table 3.16. The exponent field is a signed integer.

$$\text{actual value} = \text{Characteristic Value} * 10^{\text{Exponent}}$$

As can be seen in the above equation, the actual value is a combination of the Characteristic Value and the value 10 to the power Exponent. This is sometimes known as a fixed point number.

For example, if the Exponent is 2 and the *Characteristic Value* is 23, the actual value would be 2300.

For example, if the Exponent is -3 and the *Characteristic Value* is 3892, the actual value would be 3.892.

3.3.3.5.4 Unit

The Unit is a UUID as defined in the Assigned Numbers document [1].

3.3.3.5.5 Name Space

The Name Space field is used to identify the organization as defined in the Assigned Numbers document [1], that is responsible for defining the enumerations for the description field.

3.3.3.5.6 Description

The *Description* is an enumerated value as defined in the Assigned Numbers document [1] from the organization identified by the Name Space field.

3.3.3.6 Characteristic Aggregate Format

The *Characteristic Aggregate Format* declaration is an optional characteristic descriptor that defines the format of an aggregated *Characteristic Value*.

The characteristic descriptor may occur in any position within the characteristic definition after the *Characteristic Value*. Only one *Characteristic Aggregate Format* declaration shall exist in a characteristic definition.

The *Characteristic Aggregate Format* value is composed of a list of Attribute Handles of *Characteristic Presentation Format* declarations, where each *Attribute Handle* points to a *Characteristic Presentation Format* declaration.

The *Attribute Permissions* shall be read only and not require authentication or authorization.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2905 – UUID for «Characteristic Aggregate Format»	List of <i>Attribute Handles</i> for the <i>Characteristic Presentation Format</i> Declarations	Read only No authentication No authorization

Table 3.17: *Characteristic Aggregate Format* declaration

The *List of Attribute Handles* is the concatenation of multiple 16-bit *Attribute Handle* values into a single *Attribute Value*. The list shall contain at least two *Attribute Handle for Characteristic Presentation Format declarations*. The *Characteristic Value* shall be decomposed by each of the *Characteristic Presentation Format* declarations pointed to by the *Attribute Handles*. The order of the *Attribute Handles* in the list is significant.

If more than one *Characteristic Presentation Format* declarations exist in a characteristic definition, there shall also be one *Characteristic Aggregate Format* declaration. The *Characteristic Aggregate Format* declaration shall include each *Characteristic Presentation Format* declaration in the characteristic definition in the list of *Attribute Handles*. *Characteristic Presentation Format* declarations from other characteristic definitions may also be used.

A *Characteristic Aggregate Format* declaration may exist without a *Characteristic Presentation Format* declaration existing in the characteristic definition. The *Characteristic Aggregate Format* declaration may use *Characteristic Presentation Format* declarations from other characteristic definitions.

3.4 SUMMARY OF GATT PROFILE ATTRIBUTE TYPES

The following table summarizes the Attribute Types defined by the GATT Profile.

Attribute Type	UUID	Description
«Primary Service»	0x2800	Primary Service Declaration
«Secondary Service»	0x2801	Secondary Service Declaration
«Include»	0x2802	Include Declaration
«Characteristic»	0x2803	Characteristic Declaration
«Characteristic Extended Properties»	0x2900	Characteristic Extended Properties
«Characteristic User Description»	0x2901	Characteristic User Description Descriptor
«Client Characteristic Configuration»	0x2902	Client Characteristic Configuration Descriptor
«Server Characteristic Configuration»	0x2903	Server Characteristic Configuration Descriptor
«Characteristic Format»	0x2904	Characteristic Format Descriptor
«Characteristic Aggregate Format»	0x2905	Characteristic Aggregate Format Descriptor

Table 3.18: Summary of GATT Profile Attribute types

4 GATT FEATURE REQUIREMENTS

4.1 OVERVIEW

There are 11 features defined in the GATT Profile:

1. Server Configuration
2. Primary Service Discovery
3. Relationship Discovery
4. Characteristic Discovery
5. Characteristic Descriptor Discovery
6. Reading a Characteristic Value
7. Writing a Characteristic Value
8. Notification of a Characteristic Value
9. Indication of a Characteristic Value
10. Reading a Characteristic Descriptor
11. Writing a Characteristic Descriptor

Each of the features is mapped to procedures and sub-procedures. These procedures and sub-procedures describe how the Attribute Protocol is used to accomplish the corresponding feature.

4.2 FEATURE SUPPORT AND PROCEDURE MAPPING

The table below maps each feature to the procedures used for that feature, and indicates whether the procedure is optional or mandatory for that feature. The procedures are described in the referenced section.

Item No.	Feature	Sub-Procedure	Ref.	Support in Client	Support in Server
1	Server Configuration	Exchange MTU	4.3.1	O	O
2	Primary Service Discovery	Discover All Primary Services	4.4.1	O	M
		Discover Primary Services By Service UUID	4.4.2	O	M
3	Relationship Discovery	Find Included Services	4.5.1	O	M
4	Characteristic Discovery	Discover All Characteristic of a Service	4.6.1	O	M
		Discover Characteristic by UUID	4.6.2	O	M

Table 4.1: GATT feature mapping to procedures

Generic Attribute Profile (GATT)



Item No.	Feature	Sub-Procedure	Ref.	Support in Client	Support in Server
5	Characteristic Descriptor Discovery	Discover All Characteristic Descriptors	4.7.1	O	M
6	Characteristic Value Read	Read Characteristic Value	4.8.1	O	M
		Read Using Characteristic UUID	4.8.1	O	M
		Read Long Characteristic Values	4.8.2	O	O
		Read Multiple Characteristic Values	4.8.3	O	O
7	Characteristic Value Write	Write Without Response	4.9.1	O	C.1
		Signed Write Without Response	4.9.2	O	O
		Write Characteristic Value	4.9.3	O	C.2
		Write Long Characteristic Values	4.9.4	O	O
		Characteristic Value Reliable Writes	4.9.5	O	O
8	Characteristic Value Notification	Notifications	4.10.1	O	O
9	Characteristic Value Indication	Indications	4.11.1	M	C3
10	Characteristic Descriptor Value Read	Read Characteristic Descriptors	4.12.1	O	O
		Read Long Characteristic Descriptors	4.12.2	O	O
11	Characteristic Descriptor Value Write	Write Characteristic Descriptors	4.12.3	O	O
		Write Long Characteristic Descriptors	4.12.4	O	O
<p>C1: Write Without Response is mandatory if Signed Write Without Response is supported otherwise optional</p> <p>C2: Write Characteristic Value is mandatory if Write Long Characteristic Values is supported otherwise optional</p> <p>C3: If <i>Service Changed Characteristic</i> is present, this feature is mandatory, otherwise optional.</p>					

Table 4.1: GATT feature mapping to procedures

4.3 SERVER CONFIGURATION

This procedure is used by the client to configure the Attribute Protocol. This procedure has only one sub-procedure used to set the MTU sizes.

4.3.1 Exchange MTU

This sub-procedure is used by the client to set the ATT_MTU to the maximum possible value that can be supported by both devices when the client supports a value greater than the default ATT_MTU for the Attribute Protocol. This sub-procedure shall only be initiated once during a connection.

This sub-procedure shall not be used on a BR/EDR physical link since the MTU size is negotiated using L2CAP channel configuration procedures.

The Attribute Protocol *Exchange MTU Request* is used by this sub-procedure. The Client Rx MTU parameter shall be set to the maximum MTU that this client can receive.

Two possible responses can be sent from the server for the *Exchange MTU Request*: *Exchange MTU Response* and *Error Response*.

Error Response is returned if an error occurred on the server.

The server shall respond to this message with an *Exchange MTU Response* with the Server Rx MTU parameter set to the maximum MTU that this server can receive.

If the *Error Response* is sent by the server with the *Error Code* set to *Request Not Supported*, the *Attribute Opcode* is not supported and the default MTU shall be used.

Once the messages have been exchanged, the ATT_MTU shall be set to the minimum of the Client Rx MTU and Server Rx MTU values.

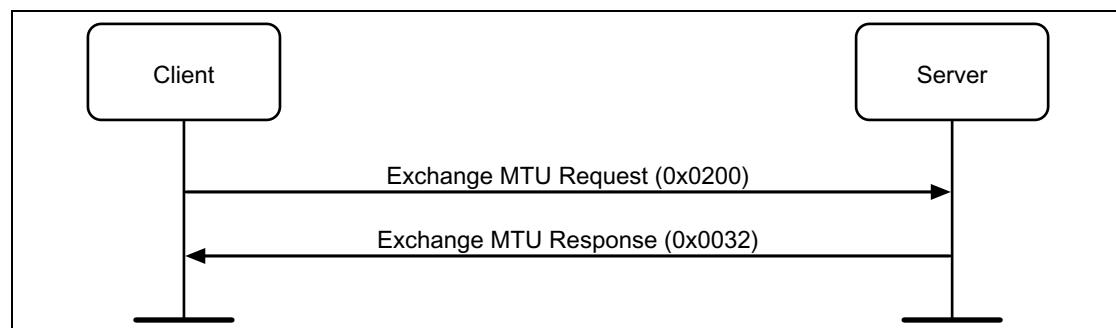


Figure 4.1: Exchange MTU

For example, in [Figure 4.1](#), based on the exchanged ATT_MTU value in the ATT_MTU would be 0x0032.

4.4 PRIMARY SERVICE DISCOVERY

This procedure is used by a client to discover primary services on a server. Once the primary services are discovered, additional information about the primary services can be accessed using other procedures, including characteristic discovery and relationship discovery to find other related primary and secondary services.

There are two sub-procedures that can be used for primary service discovery: Discover All Primary Services and Discover Primary Services by Service UUID.

4.4.1 Discover All Primary Services

This sub-procedure is used by a client to discover all the primary services on a server.

The Attribute Protocol *Read By Group Type Request* shall be used with the Attribute Type parameter set to the UUID for «Primary Service». The *Starting Handle* shall be set to 0x0001 and the *Ending Handle* shall be set to 0xFFFF.

Two possible responses can be sent from the server for the *Read By Group Type Request*: *Read By Group Type Response* and *Error Response*.

Error Response is returned if an error occurred on the server.

Read By Group Type Response returns a list of *Attribute Handle*, *End Group Handle*, and *Attribute Value* tuples corresponding to the services supported by the server. Each *Attribute Value* contained in the response is the Service UUID of a service supported by the server. The *Attribute Handle* is the handle for the service declaration. The *End Group Handle* is the handle of the last attribute within the service definition. The *End Group Handle* of the last service in a device can be 0xFFFF. The *Read By Group Type Request* shall be called again with the *Starting Handle* set to one greater than the last *End Group Handle* in the *Read By Group Type Response*.

This sub-procedure is complete when the *Error Response* is received and the *Error Code* is set to «Attribute Not Found» or when the *End Group Handle* in the *Read by Type Group Response* is 0xFFFF.

It is permitted to end the sub-procedure early if a desired primary service is found prior to discovering all the primary services on the server.

Note: The service declaration described in [Section 3.1](#) specifies that the service declaration is readable and requires no authentication or authorization, therefore insufficient authentication or read not permitted errors shall not occur.

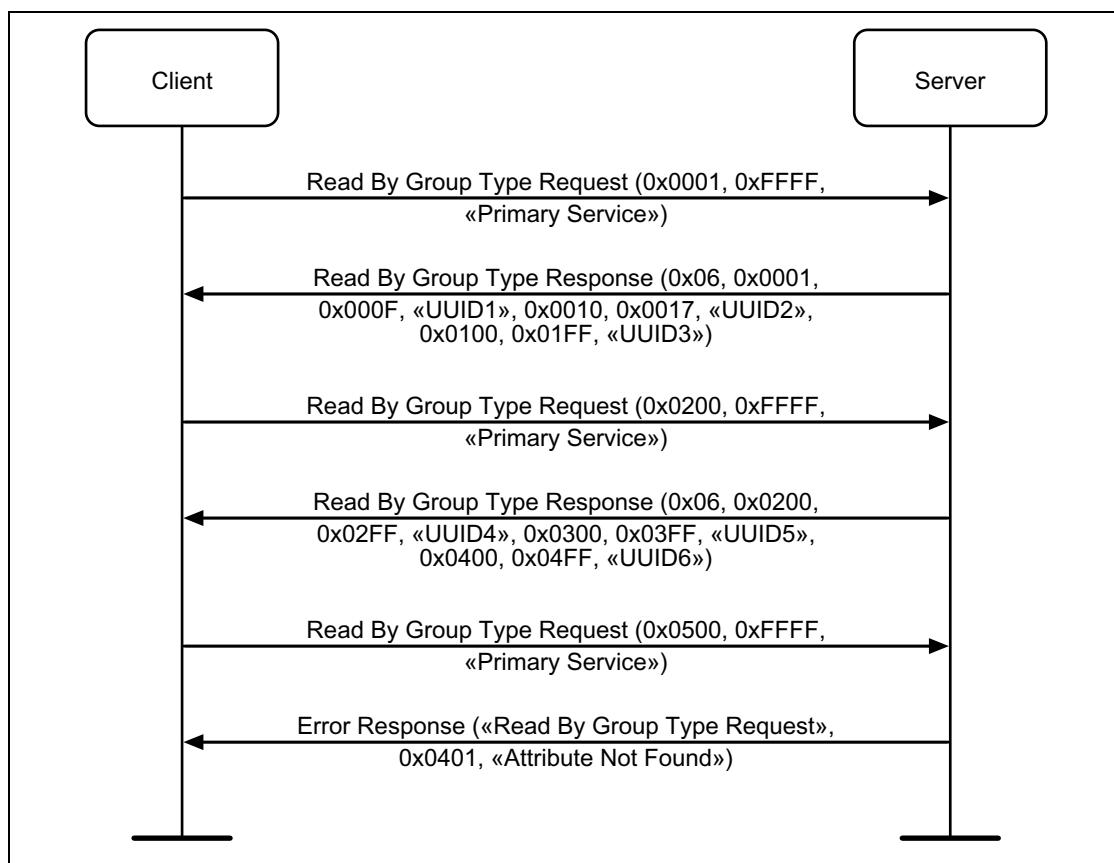


Figure 4.2: Discover All Primary Services example

4.4.2 Discover Primary Service by Service UUID

This sub-procedure is used by a client to discover a specific primary service on a server when only the Service UUID is known. The specific primary service may exist multiple times on a server. The primary service being discovered is identified by the service UUID.

The Attribute Protocol *Find By Type Value Request* shall be used with the *Attribute Type* parameter set to the UUID for «Primary Service» and the *Attribute Value* set to the 16-bit Bluetooth UUID or 128-bit UUID for the specific primary service. The *Starting Handle* shall be set to 0x0001 and the *Ending Handle* shall be set to 0xFFFF.

Two possible responses can be sent from the server for the *Find By Type Value Request*: *Find By Type Value Response* and *Error Response*.

Error Response is returned if an error occurred on the server.

Find By Type Value Response returns a list of *Attribute Handle* ranges. The *Attribute Handle* range is the starting handle and the ending handle of the service definition. The *End Group Handle* of the last service in a device can be 0xFFFF. If the *Attribute Handle* range for the Service UUID being searched is returned and the End Found Handle is not 0xFFFF, the *Find By Type Value*

Request may be called again with the *Starting Handle* set to one greater than the last *Attribute Handle* range in the Find By Type Value Response.

This sub-procedure is complete when the *Error Response* is received and the *Error Code* is set to «Attribute Not Found» or when the *End Group Handle* in the *Read by Type Group Response* is 0xFFFF.

It is permitted to end the sub-procedure early if a desired primary service is found prior to discovering all the primary services of the specified service UUID supported on the server.

Note: The service declaration described in [Section 3.1](#) specifies that the service declaration is readable and requires no authentication or authorization, therefore insufficient authentication or read not permitted errors shall not occur.

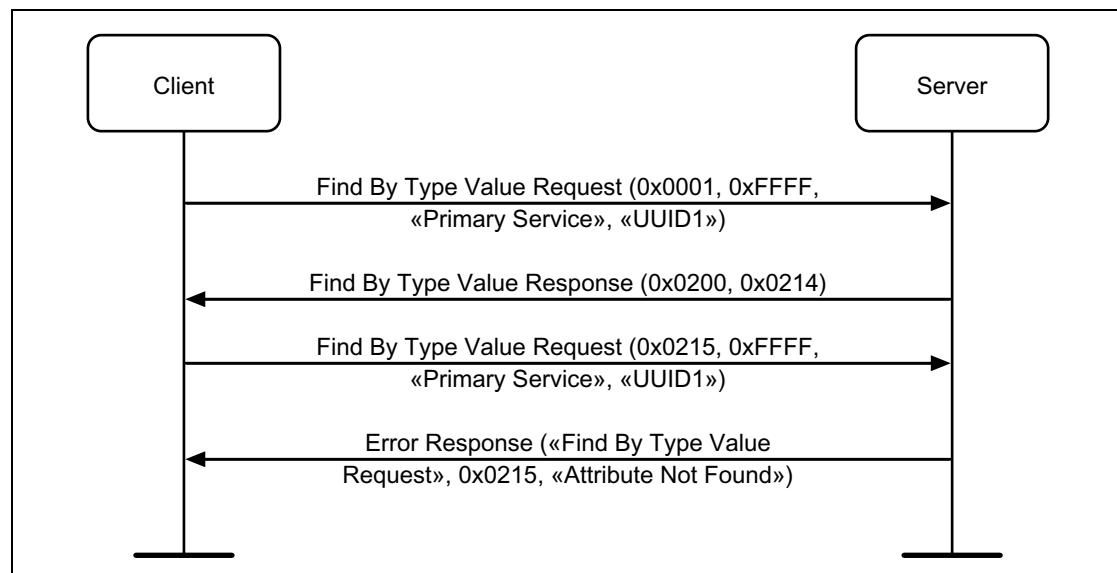


Figure 4.3: Discover Primary Service by Service UUID example

4.5 RELATIONSHIP DISCOVERY

This procedure is used by a client to discover service relationships to other services.

There is one sub-procedure that can be used for relationship discovery: Find Included Services.

4.5.1 Find Included Services

This sub-procedure is used by a client to find include service declarations within a service definition on a server. The service specified is identified by the service handle range.

The Attribute Protocol *Read By Type Request* shall be used with the *Attribute Type* parameter set to the UUID for «Include». The *Starting Handle* shall be set to starting handle of the specified service and the *Ending Handle* shall be set to the ending handle of the specified service. It is permitted to end the sub-procedure early if a desired included service is found prior to discovering all the included services of the specified service supported on the server.

Two possible responses can be sent from the server for the *Read By Type Request*: *Read By Type Response* and *Error Response*.

Error Response is returned if an error occurred on the server.

Read By Type Response returns a set of *Attribute Handle* and *Attribute Value* pairs corresponding to the included services in the service definition. Each *Attribute Value* contained in the response is composed of the *Attribute Handle* of the included service declaration and the *End Group Handle*. If the service UUID is a 16-bit Bluetooth UUID it is also returned in the response. The *Read By Type Request* shall be called again with the *Starting Handle* set to one greater than the last *Attribute Handle* in the *Read By Type Response*.

The sub-procedure is complete when either the *Error Response* is received with the *Error Code* set to *Attribute Not Found* or the *Read By Type Response* has an *Attribute Handle* of the included service declaration that is equal to the *Ending Handle* of the request.

To get the included service UUID when the included service uses a 128-bit UUID, the *Read Request* is used. The *Attribute Handle* for the *Read Request* is the *Attribute Handle* of the included service.

Note: The include declaration described in [Section 3.2](#) specifies that the include declaration is readable and requires no authentication or authorization, therefore insufficient authentication or read not permitted errors shall not occur.

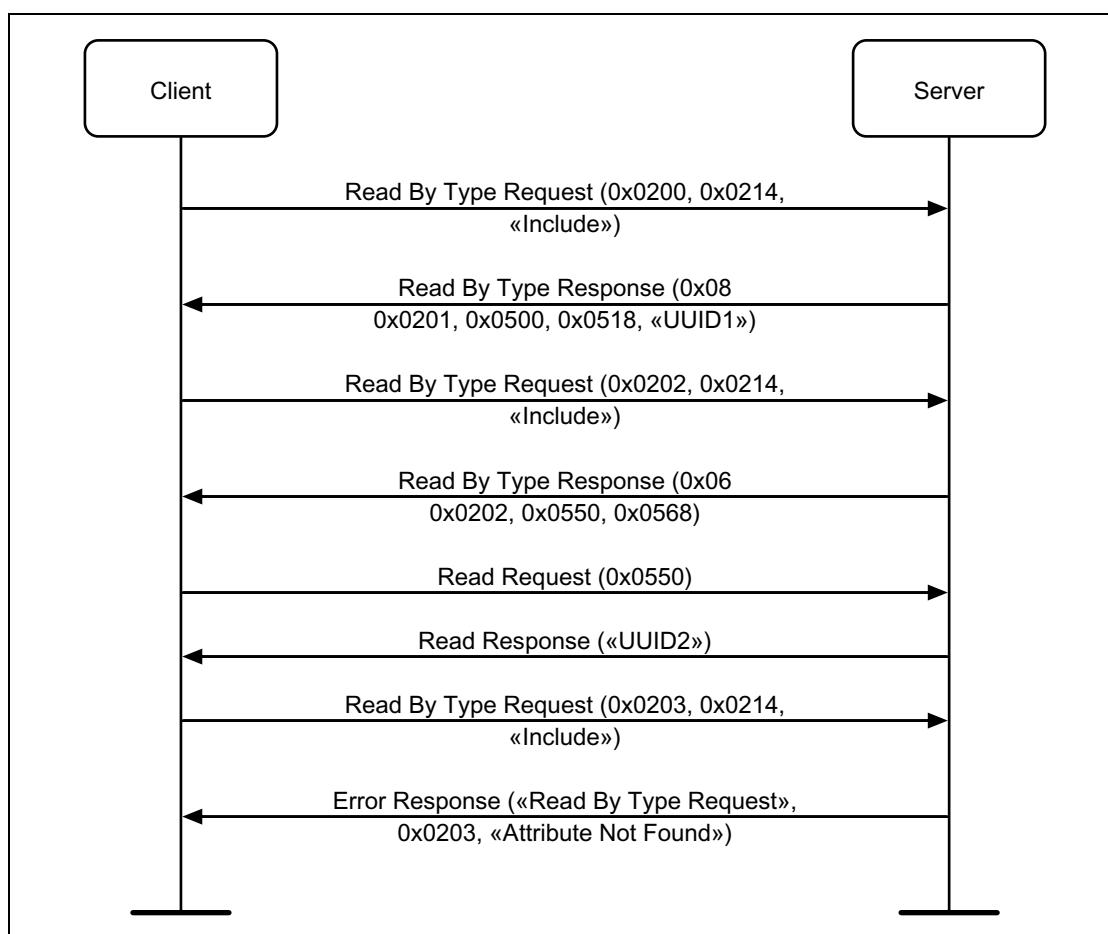


Figure 4.4: Find Included Services example

4.6 CHARACTERISTIC DISCOVERY

This procedure is used by a client to discover service characteristics on a server. Once the characteristics are discovered additional information about the characteristics can be discovered or accessed using other procedures.

There are two sub-procedures that can be used for characteristic discovery:
Discover All Characteristics of a Service and Discover Characteristics by
UUID.

4.6.1 Discover All Characteristics of a Service

This sub-procedure is used by a client to find all the characteristic declarations within a service definition on a server when only the service handle range is known. The service specified is identified by the service handle range.

The Attribute Protocol *Read By Type Request* shall be used with the *Attribute Type* parameter set to the UUID for «Characteristic». The *Starting Handle* shall be set to starting handle of the specified service and the *Ending Handle* shall be set to the ending handle of the specified service.

Two possible responses can be sent from the server for the *Read By Type Request*: *Read By Type Response* and *Error Response*.

Error Response is returned if an error occurred on the server.

Read By Type Response returns a list of *Attribute Handle* and *Attribute Value* pairs corresponding to the characteristics in the service definition. The *Attribute Handle* is the handle for the characteristic declaration. The *Attribute Value* is the Characteristic Properties, Characteristic Value Handle and Characteristic UUID. The Read By Type Request shall be called again with the *Starting Handle* set to one greater than the last *Attribute Handle* in the *Read By Type Response*.

The sub-procedure is complete when the *Error Response* is received and the *Error Code* is set to *Attribute Not Found* or the *Read By Type Response* has an *Attribute Handle* that is equal to the *Ending Handle* of the request.

It is permitted to end the sub-procedure early if a desired characteristic is found prior to discovering all the characteristics of the specified service supported on the server.

Note: The characteristic declaration described in [Section 3.3](#) specifies that the characteristic declaration is readable and requires no authentication or authorization, therefore insufficient authentication or read not permitted errors should not occur.

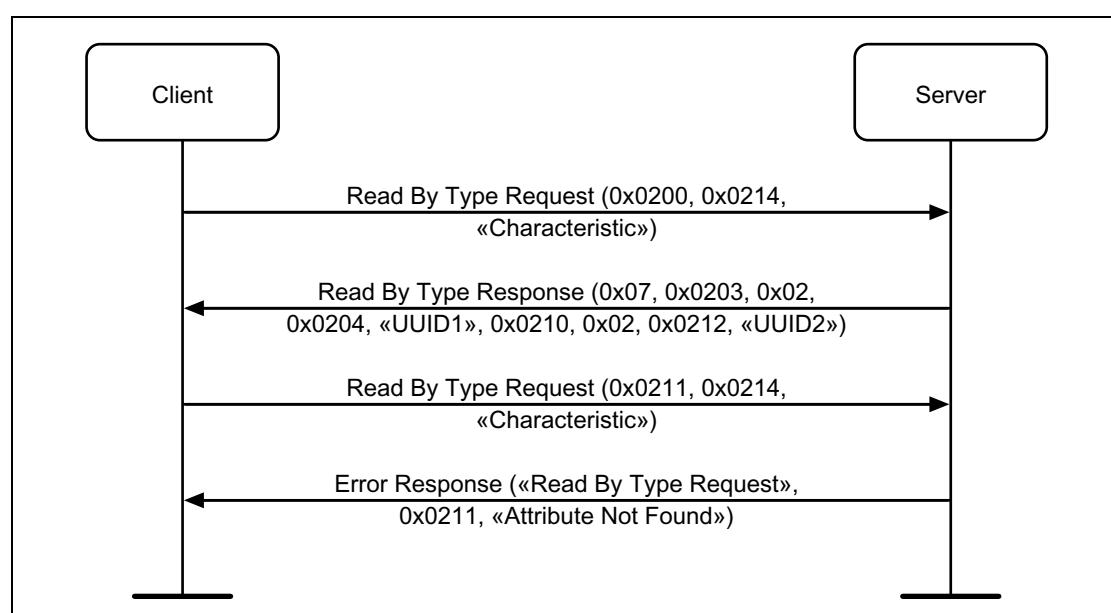


Figure 4.5: Discover All Characteristics of a Service example

4.6.2 Discover Characteristics by UUID

This sub-procedure is used by a client to discover service characteristics on a server when only the service handle ranges are known and the characteristic UUID is known. The specific service may exist multiple times on a server. The characteristic being discovered is identified by the characteristic UUID.

The Attribute Protocol *Read By Type Request* is used to perform the beginning of the sub-procedure. The *Attribute Type* is set to the UUID for «Characteristic» and the *Starting Handle* and *Ending Handle* parameters shall be set to the service handle range.

Two possible responses can be sent from the server for the *Read By Type Request*: *Read By Type Response* and *Error Response*.

Error Response is returned if an error occurred on the server.

Read By Type Response returns a list of *Attribute Handle* and *Attribute Value* pairs corresponding to the characteristics contained in the handle range provided. Each *Attribute Value* in the list is the *Attribute Value* for the characteristic declaration. The *Attribute Value* contains the characteristic properties, Characteristic Value Handle and characteristic UUID. The *Attribute Value* for each *Attribute Handle* and *Attribute Value* pairs are checked for a matching characteristic UUID. Once found, the sub-procedure continues until the end of the service handle range is exhausted. The *Read By Type Request* is called again with the *Starting Handle* set to one greater than the last *Attribute Handle* in the *Read By Type Response*.

If the *Error Response* is sent by the server with the *Error Code* set to *Attribute Not Found*, the characteristic does not exist on the server within the handle range provided.

It is permitted to end the sub-procedure early if a desired characteristic is found prior to discovering all the characteristics for the specified service supported on the server.

Note: The characteristic declaration described in [Section 3.3](#) specifies that the characteristic declaration is readable and requires no authentication or authorization, therefore insufficient authentication or read not permitted errors shall not occur.

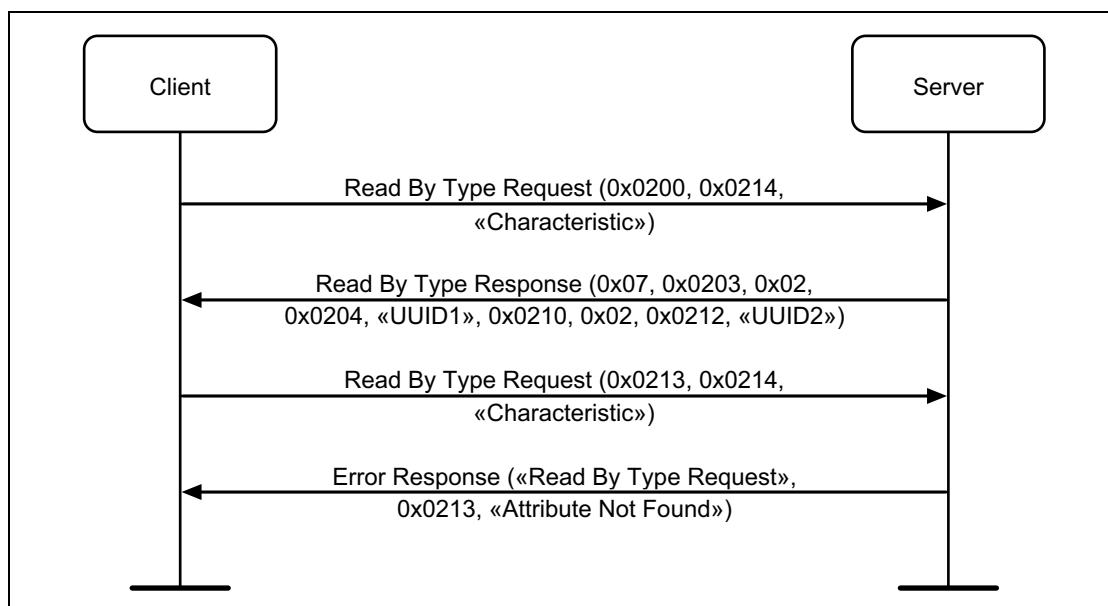


Figure 4.6: Discover Characteristics by UUID example

4.7 CHARACTERISTIC DESCRIPTOR DISCOVERY

This procedure is used by a client to discover characteristic descriptors of a characteristic. Once the characteristic descriptors are discovered additional information about the characteristic descriptors can be accessed using other procedures.

There is one sub-procedure that can be used for characteristic descriptor discovery: Discover All Characteristic Descriptors.

4.7.1 Discover All Characteristic Descriptors

This sub-procedure is used by a client to find all the characteristic descriptor's Attribute Handles and Attribute Types within a characteristic definition when only the characteristic handle range is known. The characteristic specified is identified by the characteristic handle range.

The Attribute Protocol *Find Information Request* shall be used with the *Starting Handle* set to the handle of the specified characteristic value + 1 and the *Ending Handle* set to the ending handle of the specified characteristic.

Two possible responses can be sent from the server for the *Find Information Request*: *Find Information Response* and *Error Response*.

Error Response is returned if an error occurred on the server.

Find Information Response returns a list of *Attribute Handle* and *Attribute Value* pairs corresponding to the characteristic descriptors in the characteristic definition. The *Attribute Handle* is the handle for the characteristic descriptor declaration. The *Attribute Value* is the Characteristic Descriptor UUID. The

Find Information Request shall be called again with the *Starting Handle* set to one greater than the last *Attribute Handle* in the *Find Information Response*.

The sub-procedure is complete when the *Error Response* is received and the *Error Code* is set to *Attribute Not Found* or the *Find Information Response* has an *Attribute Handle* that is equal to the *Ending Handle* of the request.

It is permitted to end the sub-procedure early if a desired Characteristic Descriptor is found prior to discovering all the characteristic descriptors of the specified characteristic.

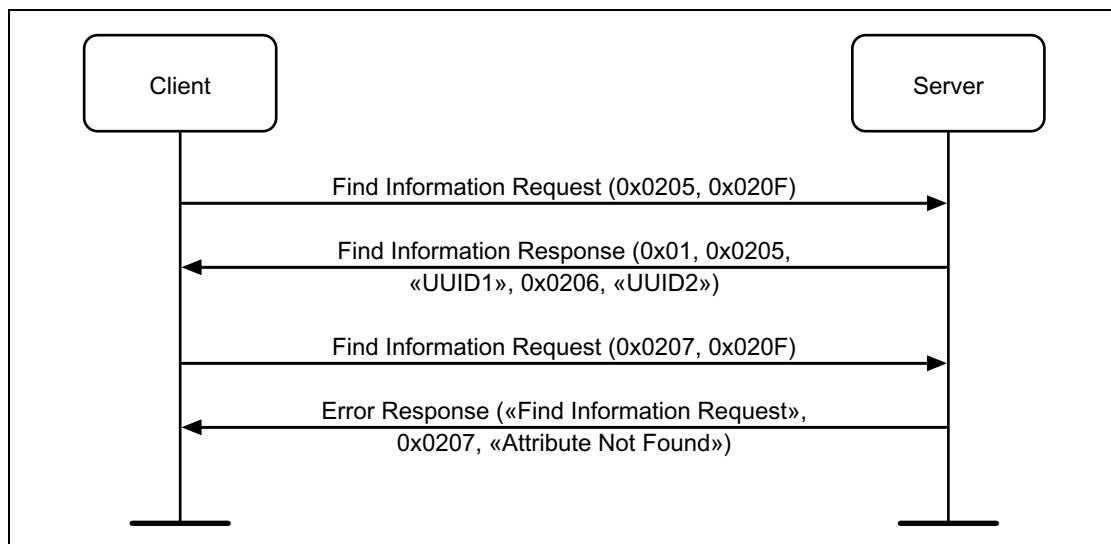


Figure 4.7: Discover All Characteristic Descriptors example

4.8 CHARACTERISTIC VALUE READ

This procedure is used to read a *Characteristic Value* from a server. There are four sub-procedures that can be used to read a *Characteristic Value*: Read Characteristic Value, Read Using Characteristic UUID, Read Long Characteristic Values, and Read Multiple Characteristic Values.

4.8.1 Read Characteristic Value

This sub-procedure is used to read a *Characteristic Value* from a server when the client knows the *Characteristic Value Handle*. The Attribute Protocol *Read Request* is used with the *Attribute Handle* parameter set to the *Characteristic Value Handle*. The Read Response returns the *Characteristic Value* in the *Attribute Value* parameter.

The *Read Response* only contains a *Characteristic Value* that is less than or equal to ($\text{ATT_MTU} - 1$) octets in length. If the *Characteristic Value* is greater than ($\text{ATT_MTU} - 1$) octets in length, the Read Long Characteristic Value procedure may be used if the rest of the *Characteristic Value* is required.

An *Error Response* shall be sent by the server in response to the *Read Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a read operation is not permitted on the *Characteristic Value*. The *Error Code* parameter is set as specified in the Attribute Protocol.

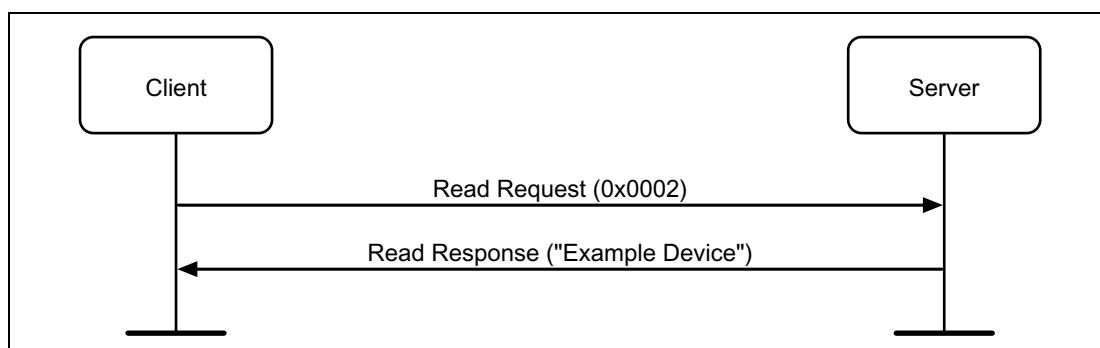


Figure 4.8: Read Characteristic Value example

4.8.2 Read Using Characteristic UUID

This sub-procedure is used to read a *Characteristic Value* from a server when the client only knows the characteristic UUID and does not know the handle of the characteristic.

The Attribute Protocol *Read By Type Request* is used to perform the sub-procedure. The Attribute Type is set to the known characteristic UUID and the Starting Handle and Ending Handle parameters shall be set to the range over which this read is to be performed. This is typically the handle range for the service in which the characteristic belongs.

Two possible responses can be sent from the server for the *Read By Type Request*: *Read By Type Response* and *Error Response*.

Error Response is returned if an error occurred on the server.

Read By Type Response returns a list of *Attribute Handle* and *Attribute Value* pairs corresponding to the characteristics contained in the handle range provided.

If the *Error Response* is sent by the server with the *Error Code* set to *Attribute Not Found*, the characteristic does not exist on the server within the handle range provided.

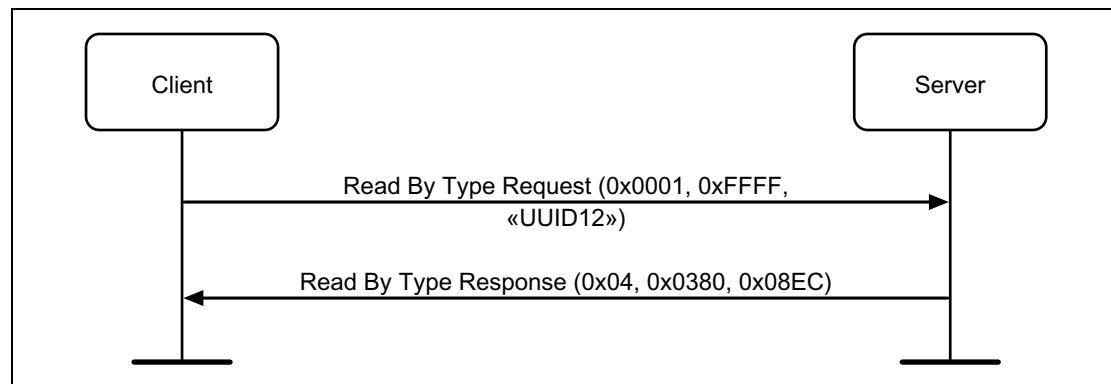


Figure 4.9: Read Using Characteristic UUID example

4.8.3 Read Long Characteristic Values

This sub-procedure is used to read a *Characteristic Value* from a server when the client knows the *Characteristic Value Handle* and the length of the *Characteristic Value* is longer than can be sent in a single *Read Response* Attribute Protocol message.

The Attribute Protocol *Read Blob Request* is used to perform this sub-procedure. The *Attribute Handle* shall be set to the *Characteristic Value Handle* of the *Characteristic Value* to be read. The *Value Offset* parameter shall be the offset within the *Characteristic Value* to be read. To read the complete *Characteristic Value* the offset should be set to 0x00 for the first *Read Blob Request*. The offset for subsequent *Read Blob Requests* is the next octet that has yet to be read. The *Read Blob Request* is repeated until the *Read Blob Response*'s *Part Attribute Value* parameter is shorter than $(\text{ATT_MTU} - 1)$.

For each *Read Blob Request* a *Read Blob Response* is received with a portion of the *Characteristic Value* contained in the *Part Attribute Value* parameter.

An *Error Response* shall be sent by the server in response to the *Read Blob Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a read operation is not permitted on the *Characteristic Value*. The *Error Code* parameter is set as specified in the

Attribute Protocol. If the *Characteristic Value* is not longer than (ATT_MTU – 1) an *Error Response* with the *Error Code* set to Attribute Not Long shall be received on the first *Read Blob Request*.

Note: The *Read Blob Request* may be used to read the remainder of an Attribute where the first part was read using a simple *Read Request*.

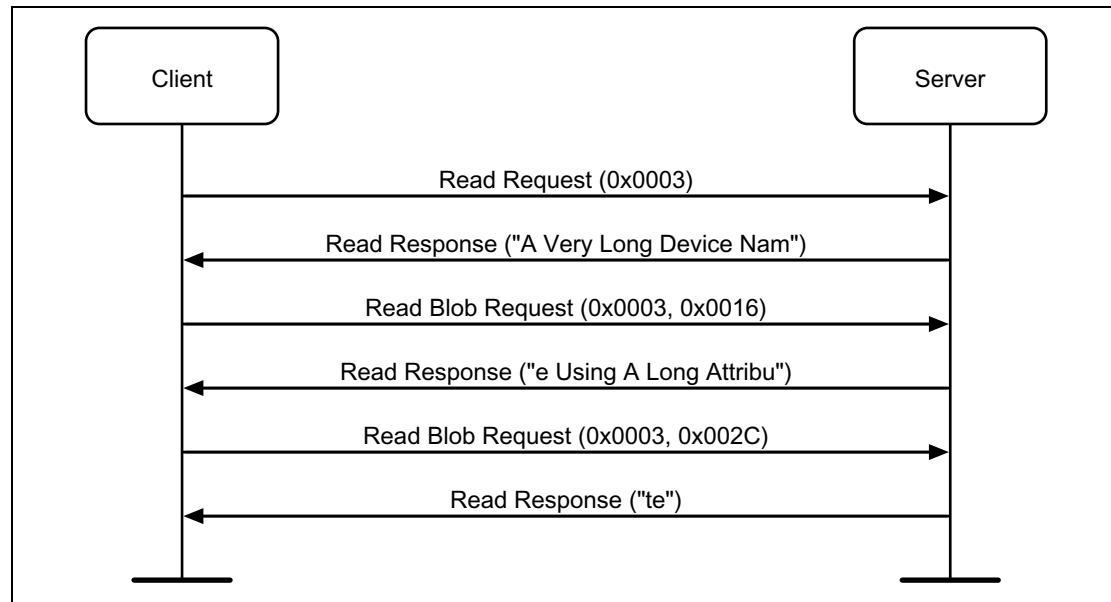


Figure 4.10: *Read Long Characteristic Values* example

4.8.4 Read Multiple Characteristic Values

This sub-procedure is used to read multiple *Characteristic Values* from a server when the client knows the *Characteristic Value Handles*. The Attribute Protocol *Read Multiple Requests* is used with the *Set Of Handles* parameter set to the *Characteristic Value Handles*. The *Read Multiple Response* returns the *Characteristic Values* in the *Set Of Values* parameter.

The *Read Multiple Response* only contains a set of *Characteristic Values* that is less than or equal to (ATT_MTU – 1) octets in length. If the *Set Of Values* is greater than (ATT_MTU – 1) octets in length, only the first (ATT_MTU – 1) octets are included in the response.

Note: A client should not request multiple *Characteristic Values* when the response's *Set Of Values* parameter is equal to (ATT_MTU – 1) octets in length since it is not possible to determine if the last *Characteristic Value* was read or additional *Characteristic Values* exist but were truncated.

An *Error Response* shall be sent by the server in response to the *Read Multiple Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a read operation is not permitted on any of the *Characteristic Values*. The *Error Code* parameter is set as specified in the Attribute Protocol.

Refer to the Attribute Protocol specification for the format of the *Set Of Handles* and *Set Of Values* parameter.

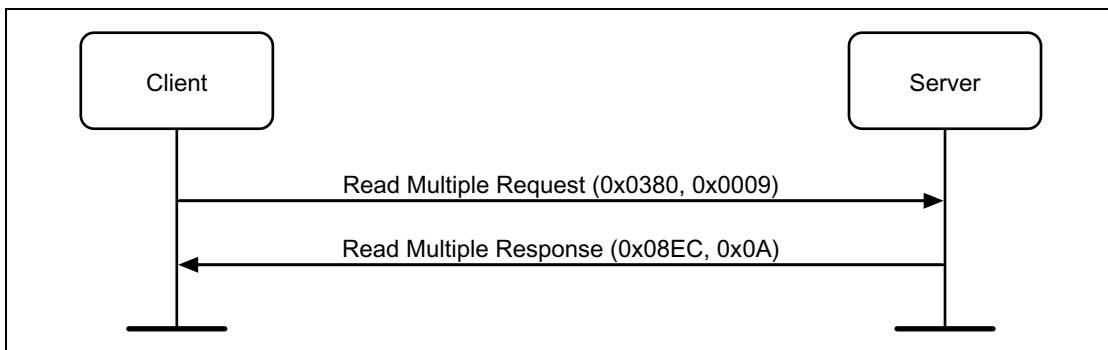


Figure 4.11: Read Multiple Characteristic Values example

4.9 CHARACTERISTIC VALUE WRITE

This procedure is used to write a *Characteristic Value* to a server.

There are five sub-procedures that can be used to write a *Characteristic Value*: Write Without Response, Signed Write Without Response, Write Characteristic Value, Write Long Characteristic Values and Reliable Writes.

4.9.1 Write Without Response

This sub-procedure is used to write a *Characteristic Value* to a server when the client knows the *Characteristic Value Handle* and the client does not need an acknowledgement that the write was successfully performed. This sub-procedure only writes the first (ATT_MTU – 3) octets of a *Characteristic Value*. This sub-procedure cannot be used to write a long characteristic; instead the *Write Long Characteristic Values* sub-procedure should be used.

The Attribute Protocol *Write Command* is used for this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Value Handle*. The *Attribute Value* parameter shall be set to the new *Characteristic Value*.

If the *Characteristic Value* write request is the wrong size, or has an invalid value as defined by the profile, then the write shall not succeed and no error shall be generated by the server.

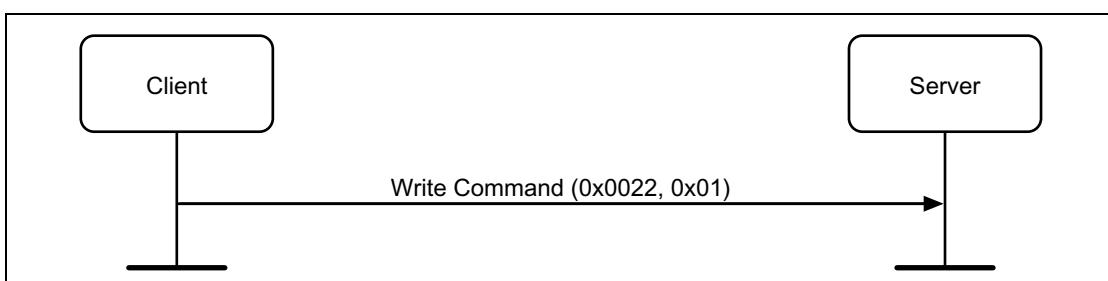


Figure 4.12: Write Without Response example

4.9.2 Signed Write Without Response

This sub-procedure is used to write a *Characteristic Value* to a server when the client knows the *Characteristic Value Handle* and the ATT Bearer is not encrypted. This sub-procedure shall only be used if the *Characteristic Properties* authenticated bit is enabled and the client and server device share a bond as defined in [\[Vol. 3\] Part C, Generic Access Profile](#).

This sub-procedure only writes the first (ATT_MTU – 15) octets of an *Attribute Value*. This sub-procedure cannot be used to write a long Attribute.

The Attribute Protocol *Signed Write Command* is used for this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Value Handle*. The *Attribute Value* parameter shall be set to the new *Characteristic Value* authenticated by signing the value, as defined in the Security Manager [\[Vol. 3\] Part H, Section 2.4.5](#).

If the authenticated *Characteristic Value* that is written is the wrong size, has an invalid value as defined by the profile, or the signed value does not authenticate the client, then the write shall not succeed and no error shall be generated by the server.

If a connection is already encrypted with LE security mode 1, level 2 or level 3 as defined in [\[Vol 3\] Part C, Section 10.2](#) then, a Write Without Response as defined in [Section 4.9.1](#) shall be used instead of a Signed Write Without Response.

Note: On BR/EDR, the ATT Bearer is always encrypted, due to the use of Security Mode 4, therefore this sub-procedure shall not be used.

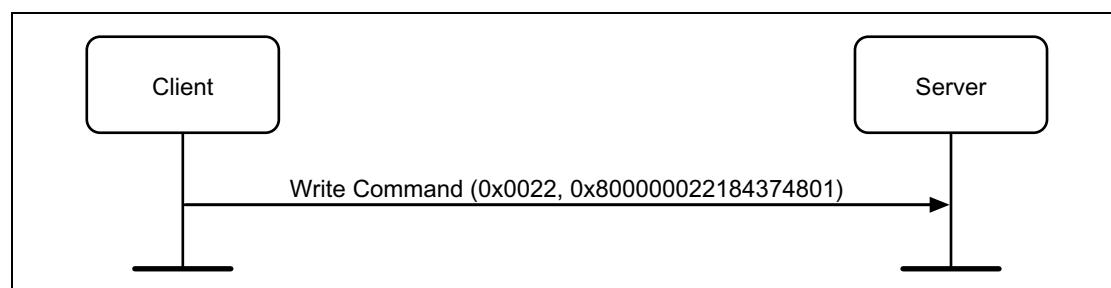


Figure 4.13: Signed Write Without Response example

4.9.3 Write Characteristic Value

This sub-procedure is used to write a *Characteristic Value* to a server when the client knows the *Characteristic Value Handle*. This sub-procedure only writes the first (ATT_MTU – 3) octets of a *Characteristic Value*. This sub-procedure cannot be used to write a long Attribute; instead the *Write Long Characteristic Values* sub-procedure should be used.

The Attribute Protocol *Write Request* is used to for this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Value Handle*. The *Attribute Value* parameter shall be set to the new characteristic.

A *Write Response* shall be sent by the server if the write of the *Characteristic Value* succeeded.

An *Error Response* shall be sent by the server in response to the *Write Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a write operation is not permitted on the *Characteristic Value*. The *Error Code* parameter is set as specified in the Attribute Protocol. If the *Characteristic Value* that is written is the wrong size, or has an invalid value as defined by the profile, then the value shall not be written and an *Error Response* shall be sent with the *Error Code* set to *Application Error* by the server.

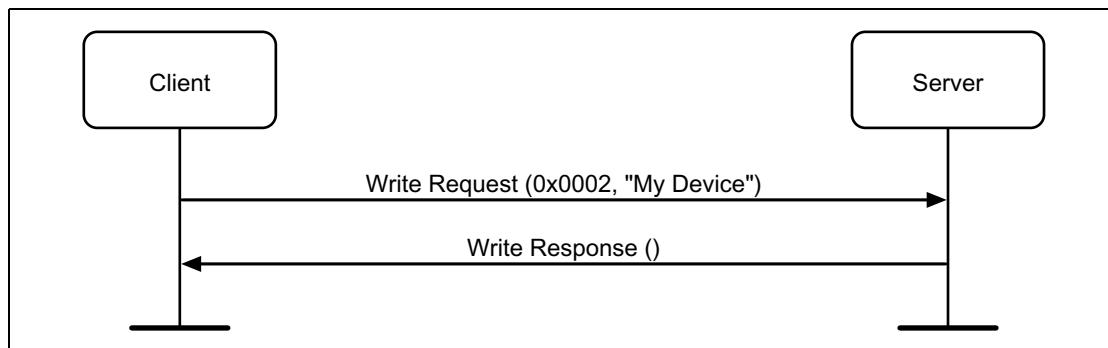


Figure 4.14: Write Characteristic Value example

4.9.4 Write Long Characteristic Values

This sub-procedure is used to write a *Characteristic Value* to a server when the client knows the *Characteristic Value Handle* but the length of the *Characteristic Value* is longer than can be sent in a single *Write Request* Attribute Protocol message.

The Attribute Protocol *Prepare Write Request* and *Execute Write Request* are used to perform this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Value Handle* of the *Characteristic Value* to be written. The *Part Attribute Value* parameter shall be set to the part of the *Attribute Value* that is being written. The *Value Offset* parameter shall be the offset within the *Characteristic Value* to be written. To write the complete *Characteristic Value* the offset should be set to 0x0000 for the first *Prepare Write Request*. The offset for subsequent *Prepare Write Requests* is the next octet that has yet to be written. The *Prepare Write Request* is repeated until the complete *Characteristic Value* has been transferred, after which an *Executive Write Request* is used to write the complete value.

Note: The values in the *Prepare Write Response* do not need to be verified in this sub-procedure.

An *Error Response* shall be sent by the server in response to the *Prepare Write Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a write operation is not permitted on the *Characteristic Value*. The *Error Code* parameter is set as specified in the Attribute Protocol. If the *Attribute Value* that is written is the wrong size, or has an invalid value as defined by the profile, then the write shall not succeed and an *Error Response* shall be sent with the *Error Code* set to *Application Error* by the server.

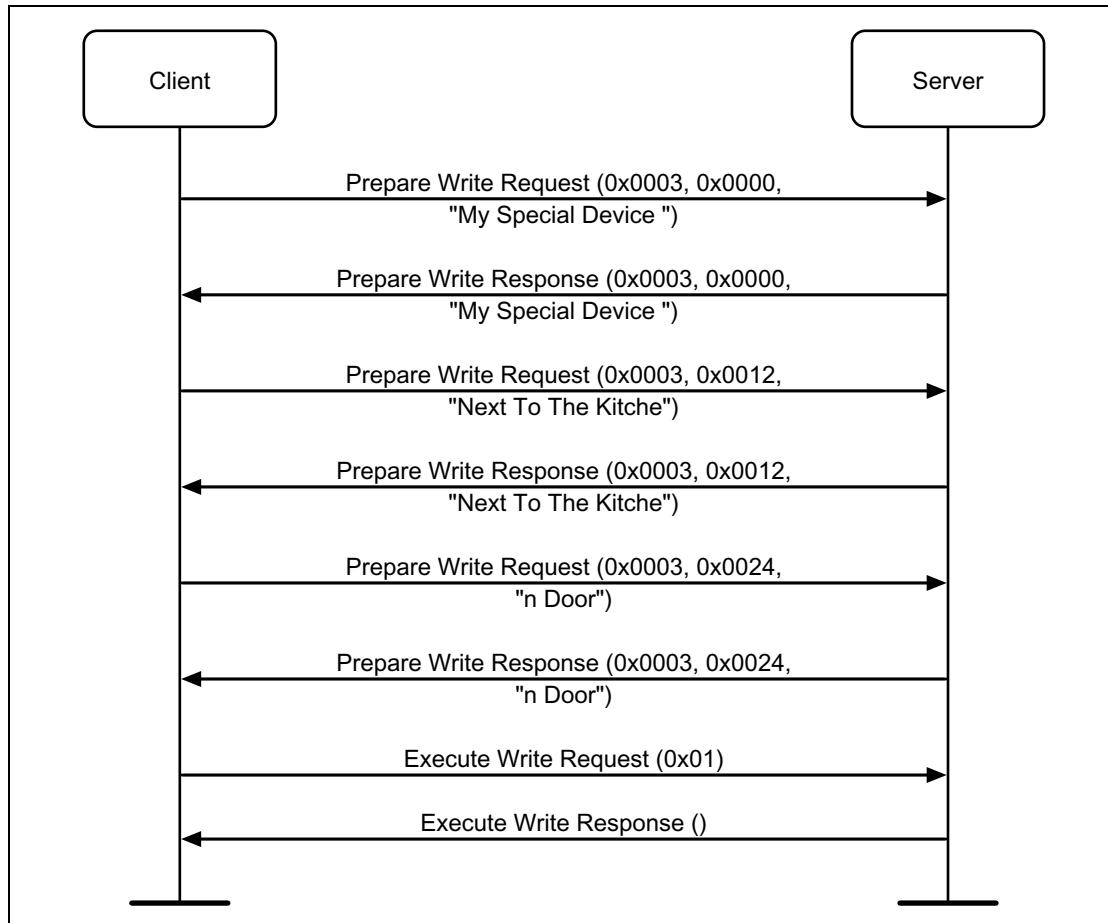


Figure 4.15: Write Long Characteristic Values example

4.9.5 Reliable Writes

This sub-procedure is used to write a *Characteristic Value* to a server when the client knows the *Characteristic Value Handle*, and assurance is required that the correct *Characteristic Value* is going to be written by transferring the *Characteristic Value* to be written in both directions before the write is performed. This sub-procedure can also be used when multiple values must be written, in order, in a single operation.

The sub-procedure has two phases; the first phase prepares the *Characteristic Values* to be written. To do this, the client transfers the *Characteristic Values* to the server. The server checks the validity of the *Characteristic Values*. The

client also checks each *Characteristic Value* to verify it was correctly received by the server using the server responses. Once this is complete, the second phase performs the execution of all of the prepared Characteristic Value writes on the server from this client.

In the first phase, the Attribute Protocol *Prepare Write Request* is used. The *Attribute Handle* shall be set to the *Characteristic Value Handle* that is to be prepared to write. The *Value Offset* and *Part Attribute Value* parameter shall be set to the new *Characteristic Value*.

There are two possible responses; *Prepare Write Response* or *Error Response*.

If the number of prepared write requests exceeds the number of prepared writes supported, then an *Error Response* with the *Error Code* set to *Prepare Queue Full* shall be sent by the server.

An *Error Response* shall be sent by the server in response to the *Prepare Write Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a write operation is not permitted on the *Characteristic Value*. The *Error Code* parameter is set as specified in the Attribute Protocol.

If a *Characteristic Value* is prepared two or more times during this sub-procedure, then all prepared values are written to the same *Characteristic Value* in the order that they were prepared.

If a *Prepare Write Response* is returned, then the *Value Offset* and *Part Attribute Value* parameter in the response shall be checked with the *Value Offset* and *Part Attribute Value* parameter that was sent in the *Prepare Write Request*; if they are different, then the value has been corrupted during transmission, and the sub-procedure shall be aborted by sending an *Execute Write Request* with the *Flags* parameter set to 0x00 to cancel all prepared writes. The complete sub-procedure may be restarted.

Multiple Prepare Write Requests can be sent by a client, each of which will be queued by the server.

In the second phase, the Attribute Protocol *Execute Write Request* is used. The Attribute *Flags* parameter shall be set to 0x01 to immediately write all pending prepared values in the order that they were prepared. The server shall write the prepared writes once it receives this request and shall only send the *Execute Write Response* once all the prepared values have been successfully written. If the *Characteristic Value* that is written is the wrong size, or has an invalid value as defined by the profile, then the write shall not succeed, and an *Error Response* with the *Error Code* set to *Application Error* shall be sent by the server. The state of the Characteristic Values that were prepared is undefined.

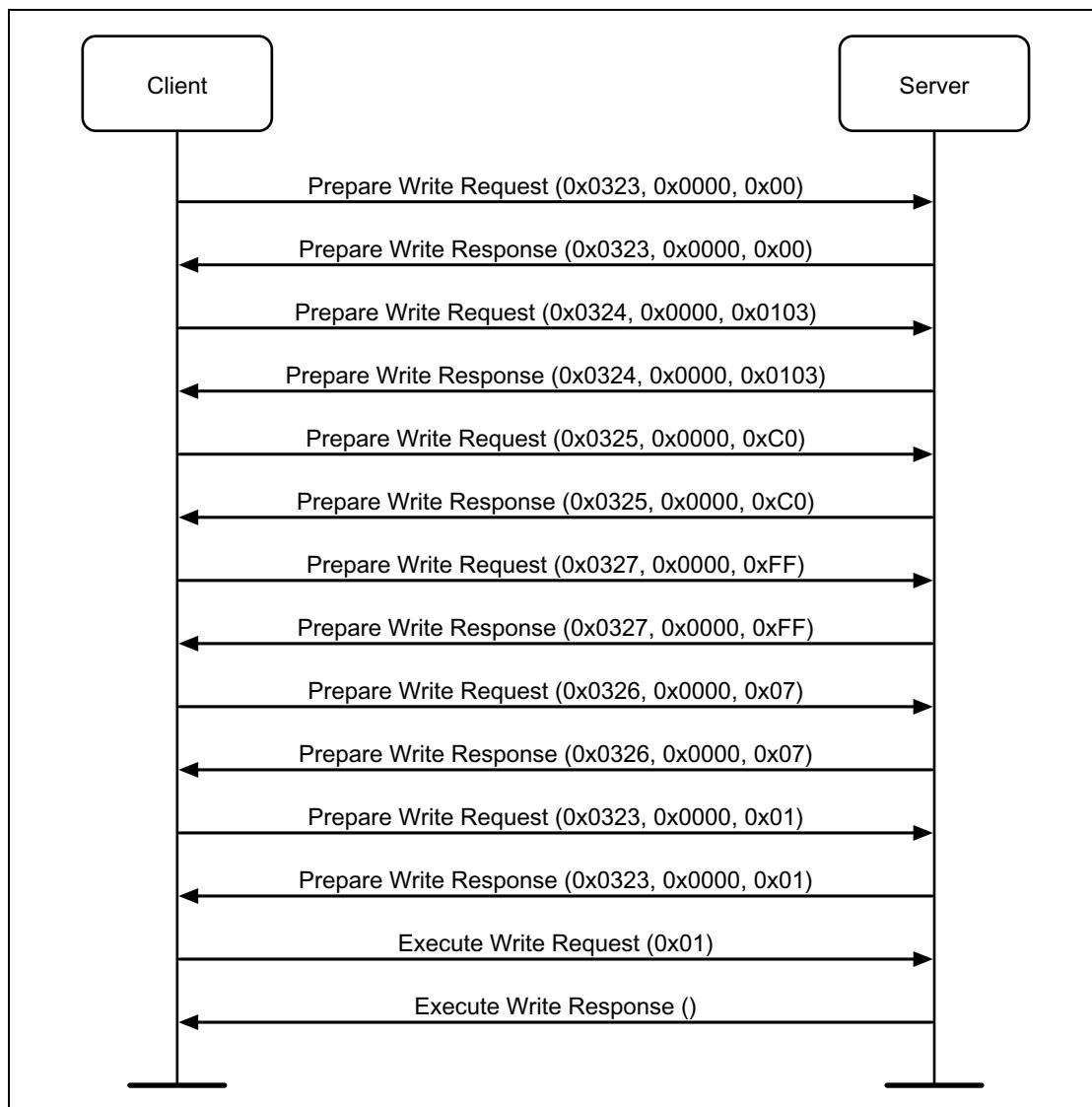


Figure 4.16: Reliable Writes example

4.10 CHARACTERISTIC VALUE NOTIFICATION

This procedure is used to notify a client of the value of a *Characteristic Value* from a server. There is one sub-procedure that can be used to notify a value: Notifications. Notifications can be configured using the Client Characteristic Configuration descriptor (See [Section 3.3.3.3](#)).

A profile defines when to use Notifications.

4.10.1 Notifications

This sub-procedure is used when a server is configured to notify a *Characteristic Value* to a client without expecting any Attribute Protocol layer acknowledgment that the notification was successfully received.

The Attribute Protocol *Handle Value Notification* is used to perform this sub-procedure. The Attribute Handle parameter shall be set to the *Characteristic Value Handle* being notified, and the Attribute Value parameter shall be set to the *Characteristic Value*.

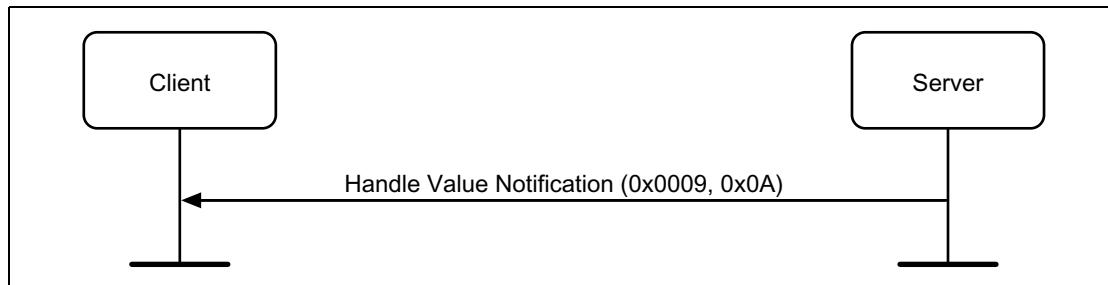


Figure 4.17: Notifications example

4.11 CHARACTERISTIC VALUE INDICATIONS

This procedure is used to indicate the *Characteristic Value* from a server to a client. There is one sub-procedure that can be used to indicate a value: Indications. Indications can be configured using the Client Characteristic Configuration descriptor (See [Section 3.3.3.3](#)).

A profile defines when to use Indications.

4.11.1 Indications

This sub-procedure is used when a server is configured to indicate a *Characteristic Value* to a client and expects an Attribute Protocol layer acknowledgement that the indication was successfully received.

The Attribute Protocol *Handle Value Indication* is used to perform this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Value Handle* being indicated, and the *Attribute Value* parameter shall be set to the characteristic. Once the *Handle Value Indication* is received by the client, the client shall respond with a *Handle Value Confirmation*.

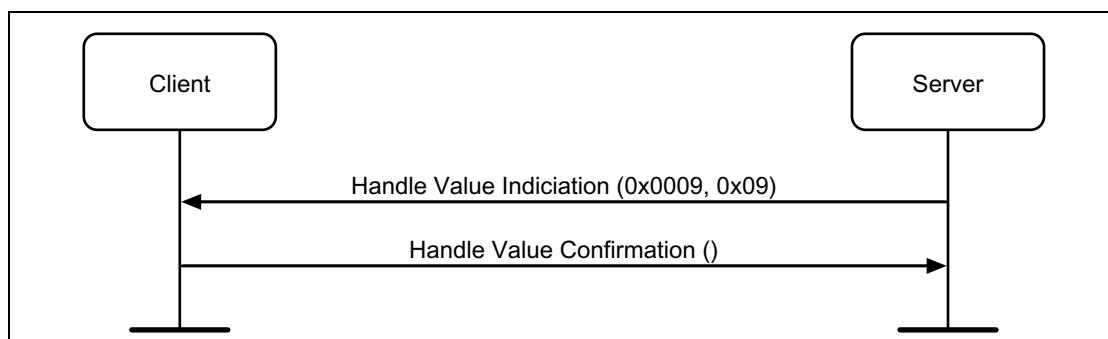


Figure 4.18: Indications example

4.12 CHARACTERISTIC DESCRIPTORS

This procedure is used to read and write characteristic descriptors on a server. There are two sub-procedures that can be used to read and write characteristic descriptors: Read Characteristic Descriptors and Write Characteristic Descriptors.

4.12.1 Read Characteristic Descriptors

This sub-procedure is used to read a characteristic descriptor from a server when the client knows the characteristic descriptor declaration's Attribute handle.

The Attribute Protocol *Read Request* is used for this sub-procedure. The *Read Request* is used with the *Attribute Handle* parameter set to the characteristic descriptor handle. The *Read Response* returns the characteristic descriptor value in the *Attribute Value* parameter.

An *Error Response* shall be sent by the server in response to the *Read Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a read operation is not permitted on the *Characteristic Value*. The *Error Code* parameter is set accordingly.

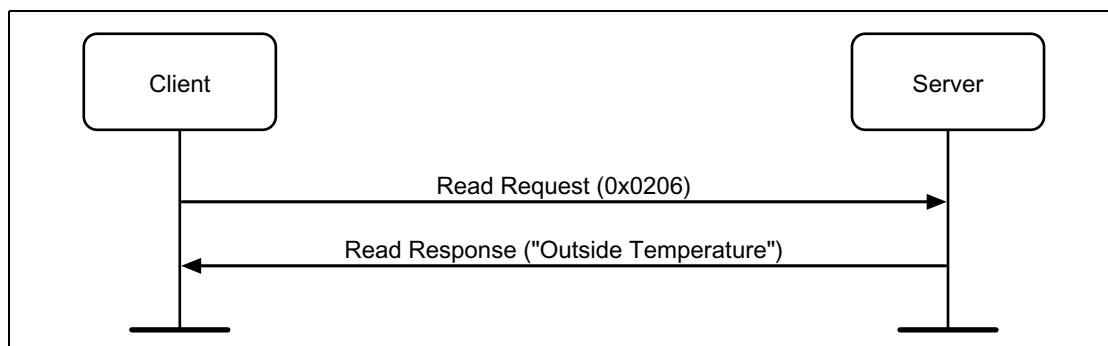


Figure 4.19: Read Characteristic Descriptors example

4.12.2 Read Long Characteristic Descriptors

This sub-procedure is used to read a characteristic descriptor from a server when the client knows the characteristic descriptor declaration's Attribute handle and the length of the characteristic descriptor declaration is longer than can be sent in a single Read Response Attribute Protocol message.

The Attribute Protocol Read Blob Request is used to perform this sub-procedure. The Attribute Handle parameter shall be set to the characteristic descriptor handle. The Value Offset parameter shall be the offset within the characteristic descriptor to be read. To read the complete characteristic descriptor the offset should be set to 0x00 for the first Read Blob Request. The offset for subsequent Read Blob Requests is the next octet that has yet to be read. The Read Blob Request is repeated until the Read Blob Response's Part

Attribute Value parameter is zero or an Error Response is sent by the server with the Error Code set to Invalid Offset.

For each Read Blob Request a Read Blob Response is received with a portion of the characteristic descriptor value contained in the Part Attribute Value parameter.

An Error Response shall be sent by the server in response to the Read Blob Request if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a read operation is not permitted on the characteristic descriptor. The Error Code parameter is set accordingly.

Note: The Read Blob Request may be used to read the remainder of an characteristic descriptor value where the first part was read using a simple Read Request.

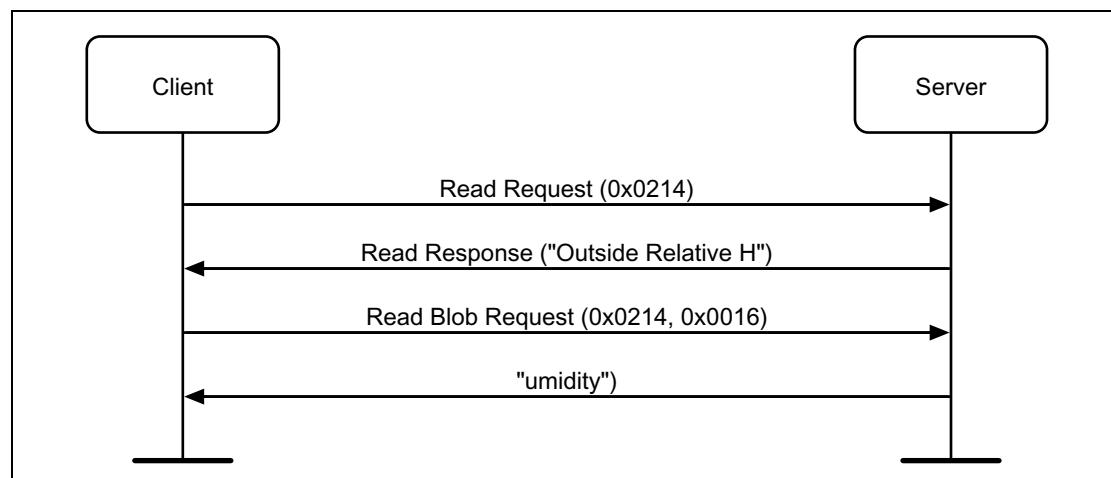


Figure 4.20: Read Long Characteristic Descriptors example

4.12.3 Write Characteristic Descriptors

This sub-procedure is used to write a characteristic descriptor value to a server when the client knows the characteristic descriptor handle.

The Attribute Protocol *Write Request* is used for this sub-procedure. The *Attribute Handle* parameter shall be set to the characteristic descriptor handle. The *Attribute Value* parameter shall be set to the new characteristic descriptor value.

A *Write Response* shall be sent by the server if the write of the characteristic descriptor value succeeded.

An *Error Response* shall be sent by the server in response to the *Write Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a write operation is not permitted on the *Characteristic Value*. The *Error Code* parameter shall be set as specified in the Attribute Protocol. If the characteristic descriptor value that is

written is the wrong size, or has an invalid value as defined by the profile, or the operation is not permitted at this time then the value shall not be written and an *Error Response* shall be sent with the *Error Code* set to *Application Error* by the server.

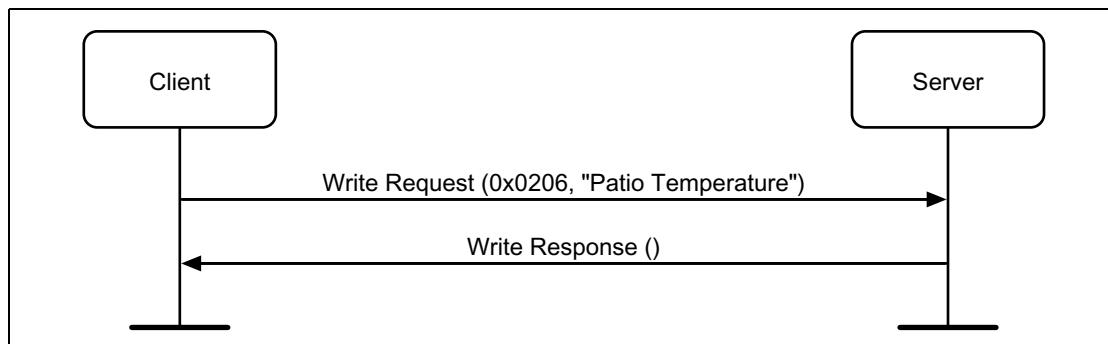


Figure 4.21: Write Characteristic Descriptors example

4.12.4 Write Long Characteristic Descriptors

This sub-procedure is used to write a characteristic descriptor value to a server when the client knows the characteristic descriptor handle but the length of the characteristic descriptor value is longer than can be sent in a single Write Request Attribute Protocol message.

The Attribute Protocol *Prepare Write Request* and *Execute Write Request* are used to perform this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Descriptor Handle* of the *Characteristic Value* to be written. The *Part Attribute Value* parameter shall be set to the part of the *Attribute Value* that is being written. The *Value Offset* parameter shall be the offset within the *Characteristic Value* to be written. To write the complete *Characteristic Value* the offset should be set to 0x0000 for the first *Prepare Write Request*. The offset for subsequent *Prepare Write Requests* is the next octet that has yet to be written. The *Prepare Write Request* is repeated until the complete *Characteristic Value* has been transferred, after which an *Executive Write Request* is used to write the complete value.

Note: The values in the *Prepare Write Response* do not need to be verified in this sub-procedure.

An *Error Response* shall be sent by the server in response to the *Prepare Write Request* or *Executive Write Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a write operation is not permitted on the *Characteristic Value*. The *Error Code* parameter is set as specified in the Attribute Protocol. If the *Attribute Value* that is written is the wrong size, or has an invalid value as defined by the profile, then the write shall not succeed and an *Error Response* shall be sent with the *Error Code* set to *Application Error* by the server.

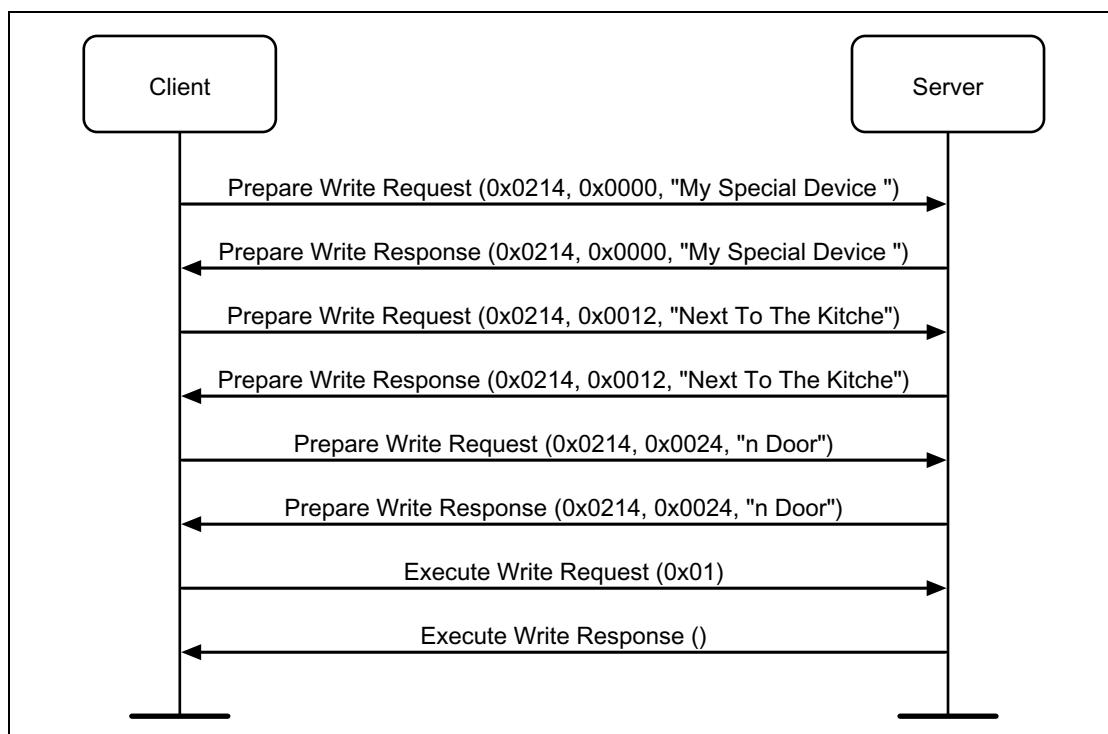


Figure 4.22: Write Long Characteristic Descriptors example

4.13 GATT PROCEDURE MAPPING TO ATT PROTOCOL OPCODES

The following table describes the mapping of the ATT protocol opcodes to the GATT procedures and sub-procedures. Only those portions of the ATT protocol requests, responses, notifications or indications necessary to implement the mandatory or supported optional sub-procedures is required.

Feature	Sub-Procedure	ATT Protocol Opcodes
Server Configuration	Exchange MTU	Exchange MTU Request Exchange MTU Response Error Response
Primary Service Discovery	Discover All Primary Services	Read By Group Type Request Read By Group Type Response Error Response
	Discover Primary Services By Service UUID	Find By Type Value Request Find By Type Value Response Error Response
Relationship Discovery	Find Included Services	Read By Type Request Read By Type Response Error Response

Table 4.2: GATT Procedure mapping to ATT protocol opcodes

Generic Attribute Profile (GATT)



Feature	Sub-Procedure	ATT Protocol Opcodes
Characteristic Discovery	Discover All Characteristic of a Service	Read By Type Request Read By Type Response Error Response
	Discover Characteristic by UUID	Read By Type Request Read By Type Response Error Response
Characteristic Descriptor Discovery	Discover All Characteristic Descriptors	Find Information Request Find Information Response Error Response
Characteristic Value Read	Read Characteristic Value	Read Request Read Response Error Response
	Read Using Characteristic UUID	Read By Type Request Read By Type Response Error Response
	Read Long Characteristic Values	Read Blob Request Read Blob Response Error Response
	Read Multiple Characteristic Values	Read Multiple Request Read Multiple Response Error Response
Characteristic Value Write	Write Without Response	Write Command
	Signed Write Without Response	Write Command
	Write Characteristic Value	Write Request Write Response Error Response
	Write Long Characteristic Values	Prepare Write Request Prepare Write Response Execute Write Request Execute Write Response Error Response
	Characteristic Value Reliable Writes	Prepare Write Request Prepare Write Response Execute Write Request Execute Write Response Error Response

Table 4.2: GATT Procedure mapping to ATT protocol opcodes

Feature	Sub-Procedure	ATT Protocol Opcodes
Characteristic Value Notification	Notifications	Handle Value Notification
Characteristic Value Indication	Indications	Handle Value Indication Handle Value Confirmation
Characteristic Descriptor Value Read	Read Characteristic Descriptors	Read Request Read Response Error Response
	Read Long Characteristic Descriptors	Read Blob Request Read Blob Response Error Response
Characteristic Descriptor Value Write	Write Characteristic Descriptors	Write Request Write Response Error Response
	Write Long Characteristic Descriptors	Prepare Write Request Prepare Write Response Prepare Write Request Prepare Write Response Error Response

Table 4.2: GATT Procedure mapping to ATT protocol opcodes

4.14 PROCEDURE TIMEOUTS

GATT procedures are protected from failure with an Attribute Protocol transaction timeout.

If the Attribute Protocol transaction times out, the procedure shall be considered to have failed, and the local higher layer shall be notified. No further GATT procedures shall be performed. A new GATT procedure shall only be performed when a new ATT Bearer has been established.

5 L2CAP INTEROPERABILITY REQUIREMENTS

The following default values shall be used by an implementation of this profile. The default values used may be different depending on the physical channel that the Attribute Protocol is being sent over.

5.1 BR/EDR L2CAP INTEROPERABILITY REQUIREMENTS

Over BR/EDR, L2CAP connection oriented channels are used to transmit Attribute Protocol PDUs. These channels use the channel establishment procedure from L2CAP using the fixed PSM (see [1]) including the configuration procedure to determine the ATT_MTU. Therefore, the ATT Bearer (or the logical link as referred to in the Attribute Protocol) is, in this case, an established L2CAP connection-oriented channel.

5.1.1 ATT_MTU

At the end of the L2CAP configuration phase, upon transition to the OPEN state, the ATT_MTU for this ATT Bearer shall be set to the minimum of the negotiated Maximum Transmission Unit configuration options.

Note: the minimum ATT_MTU for BR/EDR is 48 octets, as defined by L2CAP in [Vol. 3] Part A, Section 5.1.

5.1.2 BR/EDR Channel Requirements

All Attribute Protocol messages sent by GATT over an L2CAP channel are sent using a dynamic channel ID derived by connecting using a fixed PSM. The use of a fixed PSM allows rapid reconnection of the L2CAP channel for Attribute Protocol as a preliminary SDP query is not required.

All packets sent on this L2CAP channel shall be Attribute PDUs.

PDUs shall be reliably sent.

The flow specification for the Attribute Protocol shall be best effort.

If operating in Basic L2CAP mode, the information payload of the L2CAP B-frame shall be a single Attribute PDU.

The channel shall be encrypted. The Key_Type shall be either an Unauthenticated Combination Key or an Authenticated Combination Key.

5.1.3 BR/EDR Channel Establishment Collisions

The L2CAP connection can be initiated by the client or by the server.

Only one BR/EDR L2CAP connection shall be established between a client and a server. If the L2CAP connection already exists, the client or server shall not initiate the connection request.

If both devices open an L2CAP connection simultaneously both channels shall be closed and each device shall wait a random time (not more than 1 second and not less than 100 ms) and then try to open the L2CAP connection again. If it is known which device is the master, that device can re-try at once.

5.2 LE L2CAP INTEROPERABILITY REQUIREMENTS

Over LE, the ATT Bearer (or logical link as referred to in the Attribute Protocol) is the Attribute L2CAP fixed channel.

Note: To remove the ATT Bearer, the physical channel has to be disconnected.

5.2.1 ATT_MTU

Both a GATT client and server implementations shall support an ATT_MTU not less than the default value.

Default Value	Value for LE
ATT_MTU	23

Table 5.1: LE L2CAP ATT_MTU

5.2.2 LE Channel Requirements

All Attribute Protocol messages sent by GATT over an L2CAP logical link are sent using a fixed channel ID. This enables very fast transmission of the Attribute Protocol messages once a host connection setup is complete. The use of a fixed channel ID also enables the default channel properties to be defined.

L2CAP fixed CID 0x0004 shall be used for the Attribute Protocol. All packets sent on this fixed channel shall be Attribute Protocol PDUs.

The flow specification for the Attribute Protocol shall be best effort.

PDUs shall be reliably sent, and not flushed.

The retransmission and flow control mode for this channel shall be Basic L2CAP mode

The default parameters for the payload of the L2CAP B-frame shall be a single Attribute PDU.

Parameter	Value
MTU	23
Flush Timeout	0xFFFF (Infinite)
QoS	Best Effort
Mode	Basic Mode

Table 5.2: Attribute Protocol Fixed channel configuration parameters

6 GAP INTEROPERABILITY REQUIREMENTS

6.1 BR/EDR GAP INTEROPERABILITY REQUIREMENTS

6.1.1 Connection Establishment

To establish an ATT Bearer, the Channel Establishment procedure (as defined in [Generic Access Profile, Section 7.2](#)) shall be used.

Either device may terminate a link at any time.

No idle mode procedures or modes are defined by this profile.

6.2 LE GAP INTEROPERABILITY REQUIREMENTS

6.2.1 Connection Establishment

To establish a link, the Connection Establishment procedure (as defined in [Generic Access Profile, Section 9.3.5](#) through [Section 9.3.8](#)) shall be used.

Either device may terminate a link at any time.

No idle mode procedures or modes are defined by this profile.

6.2.2 Profile Roles

This profile can be used in the following profile roles (as defined in [Generic Access Profile](#)):

- Central
- Peripheral

6.3 DISCONNECTED EVENTS

6.3.1 Notifications and Indications While Disconnected

If a client has configured the server to send a notification or indication to the client, it shall be configured to allow re-establishment of the connection when it is disconnected.

If the client is disconnected, but intends to become a Central in the connection it shall perform a GAP connection establishment procedure. If the client is disconnected, but intends to become a Peripheral in the connection it shall go into a GAP connectable mode.

A server shall re-establish a connection with a client when an event or trigger operation causes a notification or indication to a client.

If the server is disconnected, but intends to become a Peripheral in the connection it shall go into a GAP connectable mode. If the server is disconnected, but intends to become a Central in the connection it shall perform a GAP connection establishment procedure.

If the server cannot re-establish a connection, then the notification or indication for this event shall be discarded and no further connection re-establishment shall occur, until another event occurs.

7 DEFINED GENERIC ATTRIBUTE PROFILE SERVICE

All characteristics defined within this section shall be contained in a primary service with the service UUID set to «GATT Service» as defined in [Section 3.1](#). Only one instance of the GATT service shall be exposed on a GATT server.

[Table 7.1](#) lists characteristics that may be present in the server and the characteristics that may be supported by the client.

Item No.	Characteristic	Ref.	Support in Client	Support in Server
1	Service Changed	7.1	M	C1

Table 7.1: GATT Profile service characteristic support

C1: Mandatory if service definitions on the server can be added, changed or removed; otherwise optional

The assigned UUIDs for these characteristics are defined in [\[1\]](#).

7.1 SERVICE CHANGED

The «Service Changed» characteristic is a control-point attribute (as defined in [\[Vol 3\] Part F, Section 3.2.6](#)) that shall be used to indicate to connected devices that services have changed (i.e., added, removed or modified). The characteristic shall be used to indicate to clients that have a trusted relationship (i.e. bond) with the server when GATT based services have changed when they re-connect to the server. See [Section 2.5.2](#).

This *Characteristic Value* shall be configured to be indicated, using the Client Characteristic Configuration descriptor by a client. Indications caused by changes to the Service Changed Characteristic Value shall be considered lost if the client has not enabled indications in the Client Configuration Characteristic Descriptor.

Attribute Handle	Attribute Type	Attribute Value			Attribute Permission
0xNNNN	0x2803 – UUID for «Characteristic»	Characteristic Properties = 0x20	0xMMMM = Handle of Characteristic Value	0x2A05 – UUID for «Service Changed»	No Authentication, No Authorization

Table 7.2: Service Changed Characteristic declaration

The *Service Changed Characteristic Value* is two 16-bit *Attribute Handles* concatenated together indicating the beginning and ending *Attribute Handles* affected by an addition, removal, or modification to a GATT-based service on the server. If a change has been made to any of the GATT service definition

Generic Attribute Profile (GATT)

characteristic values other than the *Service Changed* characteristic value, the range shall also include the beginning and ending *Attribute Handle* for the GATT service definition.

Attribute Handle	Attribute Type	Attribute Value		Attribute Permission
0xM-MMM	0x2A05 – UUID for «Service Changed»	0xuuuu – Start of Affected Attribute Handle Range	0xuuuu – End of Affected Attribute Handle Range	No Authentication, No Authorization, Not Readable, Not Writable

Table 7.3: *Service Changed Characteristic Value declaration*

There is only one instance of the *Service Changed* characteristic within the GATT service definition. A *Service Changed* characteristic value shall exist for each client with a trusted relationship.

If the list of GATT based services and the service definitions cannot change for the lifetime of the device then this characteristic shall not exist, otherwise this characteristic shall exist.

If the *Service Changed* characteristic exists on the server, the *Characteristic Value Indication* support on the server is mandatory.

The client shall support *Characteristic Value Indication* of the *Service Changed* characteristic.

The *Service Changed* characteristic *Attribute Handle* on the server shall not change if the server has a trusted relationship with any client.

8 SECURITY CONSIDERATIONS

8.1 AUTHENTICATION REQUIREMENTS

Authentication in the GATT Profile is applied to each characteristic independently. Authentication requirements are specified in this profile, related higher layer specifications or are implementation specific if not specified otherwise.

The GATT Profile procedures are used to access information that may require the client to be authenticated and have an encrypted connection before a characteristic can be read or written.

Over LE, if such a request is issued when the physical link is unauthenticated or unencrypted, the server shall send an *Error Response*. The client wanting to read or write this characteristic can then request that the physical link be authenticated using the GAP authentication procedure, and once this has been completed, send the request again.

Over BR/EDR, if such a request is issued when the physical link is unauthenticated or unencrypted, the server can either send an Error Response or perform the GAP authentication procedure and then upon completion of that procedure send the appropriate response. If the client receives this Error Response it can then request that the physical link be authenticated using the GAP authentication procedure, and once this has been completed send the request again.

The list of services and characteristics that a device supports is not considered private or confidential information, and therefore the Service and Characteristic Discovery procedures shall always be permitted. This implies that an *Insufficient Authentication* status code shall not be used in an *Error Response* for a *Find Information Request*.

Note: A characteristic may be allowed to be read by any device, but only written by an authenticated device. An implementation should take this into account, and not assume that if it can read a *Characteristic Value*, it will also be able to write the *Characteristic Value*. Similarly, if a characteristic can be written, it does not mean the characteristic can also be read. Each individual characteristic could have different security properties.

Once sufficient authentication of the client has been established to allow access to one characteristic within a service definition, a server may also allow access to other characteristics within the service definition depending on the higher level or implementation specific requirements.

A server may allow access to most characteristics within a service definition once sufficient authentication has been performed, but restrict access to other characteristics within the same service definition. This may result due to some

characteristics requiring stronger authentication requirements than currently enabled.

Once a server has authenticated a client for access to characteristics in one service definition, it may automatically allow access to characteristics in other service definitions.

8.2 AUTHORIZATION REQUIREMENTS

Authorization in the GATT Profile is applied to each characteristic independently. Authorization requirements may be specified in this profile, related higher layer specifications or are implementation specific if not specified otherwise.

The GATT Profile can be used to access information that may require authorization before a characteristic can be read or written.

If such a request is issued to a characteristic contained in a service definition that is not authorized, the responder shall send an *Error Response* with the status code set to *Insufficient Authorization*.

Once a server has authorized a client for access to characteristics in one group or service definition, it may automatically allow access to characteristics in other service definitions.

9 SDP INTEROPERABILITY REQUIREMENTS

A device that supports GATT over BR/EDR shall publish the following SDP record. The GATT Start Handle shall be set to the attribute handle of the «Generic Attribute Profile» service declaration. The GATT End Handle shall be set to the attribute handle of the last attribute within the «Generic Attribute Profile» service definition group.

Item	Definition	Type	Value	Status
Service Class ID List				M
Service Class #0		UUID	GATT Service	M
Protocol Descriptor List				M
Protocol #0		UUID	L2CAP	M
Parameter #0 for Protocol #0	PSM	Uint16	PSM = ATT	M
Protocol #1		UUID	ATT	M
Parameter #0 for Protocol #1	GATT Start Handle	Uint16		M
Parameter #1 for Protocol #1	GATT End Handle	Uint16		M
BrowseGroupList			PublicBrowseRoot	M

Table 9.1: SDP Record for the Generic Attribute Profile

10 REFERENCES

- [1] Assigned Numbers Specification:
<https://www.bluetooth.org/Technical/AssignedNumbers/home.htm>

APPENDIX A EXAMPLE ATTRIBUTE SERVER ATTRIBUTES

The following table shows an example Server and the Attributes contained on the server.

Handle	Attribute Type	Attribute Value
0x0001	«Primary Service»	«GAP Service»
0x0004	«Characteristic»	{0x02, 0x0006, «Device Name»}
0x0006	«Device Name»	“Example Device”
0x0010	«Primary Service»	«GATT Service»
0x0011	«Characteristic»	{0x26, 0x0012, «Service Changed»}
0x0012	«Service Changed»	0x0000, 0x0000
0x0100	«Primary Service»	«Battery State Service»
0x0106	«Characteristic»	{0x02, 0x0110, «Battery State»}
0x0110	«Battery State»	0x04
0x0200	«Primary Service»	«Thermometer Humidity Service»
0x0201	«Include»	{0x0500, 0x0504, «Manufacturer Service»}
0x0202	«Include»	{0x0550, 0x0568}
0x0203	«Characteristic»	{0x02, 0x0203, «Temperature»}
0x0204	«Temperature»	0x028A
0x0205	«Characteristic Format»	{0x0E, 0xFE, «degrees Celsius», 0x01, «Outside»}
0x0206	«Characteristic User Description»	“Outside Temperature”
0x0210	«Characteristic»	{0x02, 0x0212, «Relative Humidity»}
0x0212	«Relative Humidity»	0x27
0x0213	«Characteristic Format»	{0x04, 0x00, «Percent», «Bluetooth SIG», «Outside»}
0x0214	«Characteristic User Description»	“Outside Relative Humidity”
0x0280	«Primary Service»	«Weight Service»
0x0281	«Include»	{0x0505, 0x0509, «Manufacturer Service»}
0x0282	«Characteristic»	{0x02, 0x0283, «Weight kg»}
0x0283	«Weight kg»	0x00005582

Table A.1: Example Attribute Server Attributes

Generic Attribute Profile (GATT)



Handle	Attribute Type	Attribute Value
0x0284	«Characteristic Format»	{0x08, 0xFD, «kilogram», «Bluetooth SIG», «Hanging»}
0x0285	«Characteristic User Description»	“Rucksack Weight”
0x0300	«Primary Service»	«Position Service»
0x0301	«Characteristic»	{0x02, 0x0302, «Latitude Longitude»}
0x0302	«Latitude Longitude»	0x28BEAFA40B320FCE
0x0304	«Characteristic»	{0x02, 0x0305, «Latitude Longitude Elevation»}
0x0305	«Latitude Longitude Elevation»	0x28BEAFA40B320FCE0176
0x0400	«Primary Service»	«Alert Service»
0x0401	«Characteristic»	{0x0E, 0x0402, «Alert Enumeration»}
0x0402	«Alert Enumeration»	0x00
0x0500	«Secondary Service»	«Manufacturer Service»
0x0501	«Characteristic»	{0x02, 0x0502, «Manufacturer Name»}
0x0502	«Manufacturer Name»	“ACME Temperature Sensor”
0x0503	«Characteristic»	{0x02, 0x0504, «Serial Number»}
0x0504	«Serial Number»	“237495-3282-A”
0x0505	«Secondary Service»	«Manufacturer Service»
0x0506	«Characteristic»	{0x02, 0x0507, «Manufacturer Name»}
0x0550	«Secondary Service»	«Vendor Specific Service»
0x0560	«Characteristic»	{0x02, 0x0568, «Vendor Specific Type»}
0x0568	«Vendor Specific Type»	0x56656E646F72
0x0507	«Manufacturer Name»	“ACME Weighing Scales”
0x0508	«Characteristic»	{0x02, 0x0509, «Serial Number»}
0x0509	«Serial Number»	“11267-2327A00239”

Table A.1: Example Attribute Server Attributes

As can be seen, the attribute server indicates support for nine services: GAP Service, GATT Service, Battery State Service, Thermometer Humidity Service, Weight Service, Position Service, Alert Service, and two Manufacturer Services.

The server contains the following information about each of the services:

- The characteristic containing the name of the device is “Example Device”.
- The characteristic indicating the server supports all the attribute opcodes, and supports two prepared write values.

- The characteristic containing the battery state with a value of 0x04, meaning it is discharging.
- The characteristic containing the outside temperature with a value of 6.5 °C.
- The characteristic containing the outside relative humidity with a value of 39%.
- The characteristic containing the weight hanging off the device with a value of 21.89 kg.
- The characteristic containing the position of this device with the value of 68.3585444 degrees north, 18.7830222 degrees east, with an elevation of 374 meters.
- The characteristic containing the temperature sensor manufacturer with the value of ACME Temperature Sensor.
- The characteristic containing the serial number for the temperature sensor with a value of 237495-3282-A.
- The characteristic containing the weighing sensor manufacturer with a value of ACME Weight Scales.
- The characteristic containing the serial number for the weighing sensor with a value of 11267-2327A00239.

The device is therefore on the side of the Abisko Turiststation, Norrbottens Län, Sweden, with a battery in good state, measuring a relatively warm day, with low humidity, and a heavy rucksack.

Core System Package [Host volume]

Part H

SECURITY MANAGER SPECIFICATION

The Security Manager (SM) defines the protocol and behavior to manage pairing, authentication and encryption between LE-only or BR/EDR/LE devices.

CONTENTS

1	Introduction	591
1.1	Scope	591
1.2	Conventions.....	591
1.2.1	Bit and Byte Ordering Conventions.....	591
2	Security Manager.....	592
2.1	Introduction	592
2.2	Cryptographic Toolbox.....	594
2.2.1	Security function e	595
2.2.2	Random Address Hash function ah	595
2.2.3	Confirm value generation function c1 for LE Legacy Pairing.....	596
2.2.4	Key generation function s1 for LE Legacy Pairing	598
2.2.5	Function AES-CMAC	599
2.2.6	LE Secure Connections Confirm Value Generation Function f4	599
2.2.7	LE Secure Connections Key Generation Function f5	600
2.2.8	LE Secure Connections Check Value Generation Function f6	602
2.2.9	LE Secure Connections Numeric Comparison Value Generation Function g2	603
2.2.10	Link Key Conversion Function h6	604
2.3	Pairing Methods.....	605
2.3.1	Security Properties.....	606
2.3.2	IO Capabilities.....	607
2.3.3	OOB Authentication Data.....	608
2.3.4	Encryption Key Size.....	608
2.3.5	Pairing Algorithms.....	609
2.3.5.1	Selecting Key Generation Method.....	609
2.3.5.2	LE Legacy Pairing - Just Works	612
2.3.5.3	LE Legacy Pairing - Passkey Entry	612
2.3.5.4	Out of Band	613
2.3.5.5	LE Legacy Pairing Phase 2	613
2.3.5.6	LE Secure Connections Pairing Phase 2	615
2.3.5.7	Cross-transport Key Derivation	623
2.3.6	Repeated Attempts	623
2.4	Security in Bluetooth low energy	624
2.4.1	Definition of Keys and Values	624
2.4.2	Generation of Distributed Keys	624

2.4.2.1	Generation of IRK.....	624
2.4.2.2	Generation of CSRK.....	625
2.4.2.3	LE Legacy Pairing - Generation of LTK, EDIV and Rand	625
2.4.2.4	Derivation of BR/EDR Link Key from LE LTK626	
2.4.2.5	Derivation of LE LTK from BR/EDR Link Key626	
2.4.3	Distribution of Keys	627
2.4.3.1	LE Legacy Pairing Key Distribution	627
2.4.3.2	LE Secure Connections Key Distribution	628
2.4.4	Encrypted Session Setup.....	628
2.4.4.1	Encryption Setup using STK	628
2.4.4.2	Encryption Setup using LTK	629
2.4.5	Signing Algorithm.....	629
2.4.6	Slave Security Request.....	631
3	Security Manager Protocol	633
3.1	Introduction.....	633
3.2	Security Manager Channel over L2CAP.....	633
3.3	Command Format.....	634
3.4	SMP Timeout	635
3.5	Pairing Methods.....	635
3.5.1	Pairing Request	635
3.5.2	Pairing Response.....	638
3.5.3	Pairing Confirm	639
3.5.4	Pairing Random	640
3.5.5	Pairing Failed	642
3.5.6	Pairing Public Key.....	644
3.5.7	Pairing DHKey Check	644
3.5.8	Keypress Notification	645
3.6	Security in Bluetooth low energy	646
3.6.1	Key Distribution and Generation.....	646
3.6.2	Encryption Information	649
3.6.3	Master Identification.....	649
3.6.4	Identity Information	650
3.6.5	Identity Address Information	651
3.6.6	Signing Information	651
3.6.7	Security Request.....	652
Appendix A	EDIV and Rand Generation	654
A.1	EDIV Masking	654
A.1.1	DIV Mask generation function dm	654

A.1.2 EDIV Generation	655
A.1.3 DIV Recovery	655
Appendix B Key Management	656
B.1 Database Lookup	656
B.2 Key Hierarchy	656
B.2.1 Diversifying function d1	657
B.2.2 Generating Keys from ER	658
B.2.3 Generating Keys from IR	659
Appendix C Message Sequence Charts	660
C.1 Phase 1: Pairing Feature Exchange	660
C.1.1 Slave Security Request – Master Requests Pairing ..	661
C.2 Phase 2: Authenticating and Encrypting	662
C.2.1 LE Legacy Pairing	662
C.2.1.1 Legacy Phase 2: Short Term Key Generation – Just Works	662
C.2.1.2 Legacy Phase 2: Short Term Key Generation – Passkey Entry	663
C.2.1.3 Legacy Phase 2: Short Term Key Generation – Out of Band	664
C.2.2 LE Secure Connections	665
C.2.2.1 Public Key Exchange	665
C.2.2.2 Authentication Stage 1	665
C.2.2.3 Long Term Key Calculation	676
C.2.2.4 Authentication Stage 2 (DHKey checks)	677
C.3 Phase 3: Transport Specific Key Distribution	678
C.4 Security Re-established using Previously Distributed LTK ..	678
C.4.1 Master Initiated Security - Master Initiated Link Layer Encryption	678
C.4.2 Slave Security Request - Master Initiated Link Layer Encryption	679
C.5 Failure Conditions	679
C.5.1 Pairing Not Supported by Slave	679
C.5.2 Master Rejects Pairing Because of Key Size	680
C.5.3 Slave Rejects Pairing Because of Key Size	680
C.5.4 Passkey Entry Failure on Master	681
C.5.5 Passkey Entry Failure on Slave	682
C.5.6 Slave Rejects Master's Confirm Value	683
C.5.7 Master Rejects Slaves Confirm Value	684

Appendix D Sample Data	685
D.1 AES-CMAC RFC4493 Test Vectors	685
D.1.1 Example 1: Len = 0	685
D.1.2 Example 2: Len = 16	685
D.1.3 Example 3: Len = 40	685
D.1.4 Example 4: Len = 64	685
D.2 f4 LE SC Confirm Value Generation Function	686
D.3 f5 LE SC Key Generation Function	687
D.4 f6 LE SC Check Value Generation Function	687
D.5 g2 LE SC Numeric Comparison Generation Function	688
D.6 h6 LE SC Link Key Conversion Function	688
D.7 ah Random Address Hash Functions	688
D.8	688

1 INTRODUCTION

1.1 SCOPE

The Security Manager defines methods of pairing and key distribution, a protocol for those methods and a security toolbox to be used by those methods and other parts of an LE-only or BR/EDR/LE device.

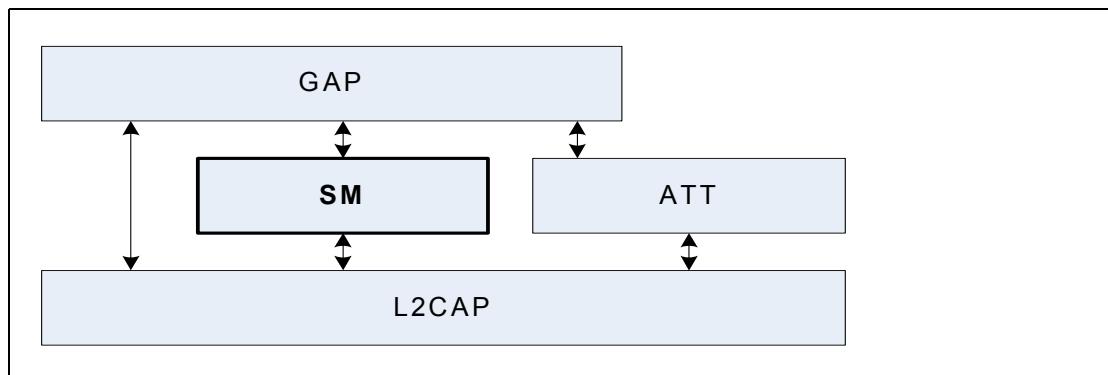


Figure 1.1: Relationship of the Security Manager to the rest of the LE Bluetooth architecture

The document describes master and slave roles in terms of protocol and requirements; these have the same meaning and are mapped to the LE device roles described in [\[Vol 1\] Part A, Section 1.2](#) or BR/EDR device roles (see [\[Vol 1\] Part A, Section 1.1](#)).

1.2 CONVENTIONS

1.2.1 Bit and Byte Ordering Conventions

When multiple bit fields are contained in a single octet and represented in a drawing in this specification, the least significant (low-order) bits are shown toward the left and most significant (high-order) bits toward the right.

Multiple-octet fields are drawn with the least significant octets toward the left and the most significant octets toward the right. Multiple-octet fields shall be transmitted with the least significant octet first.

Multiple-octet values written in hexadecimal notation have the most significant octet towards the left and the least significant octet towards the right, for example if '12' is the most significant octet and '34' is the least significant octet it would be written as 0x1234.

2 SECURITY MANAGER

2.1 INTRODUCTION

The Security Manager (SM) uses a key distribution approach to perform identity and encryption functionalities in radio communication. This means that each device generates and controls the keys it distributes and no other device affects the generation of these keys. The strength of a key is as strong as the algorithms implemented inside the distributing device.

The security architecture is designed such that memory and processing requirements for a responding device are lower than the memory and processing requirement for an initiating device.

Pairing is performed to establish keys which can then be used to encrypt a link. A transport specific key distribution is then performed to share the keys which can be used to encrypt a link in future reconnections, verify signed data and random address resolution.

Pairing is a three-phase process. The first two phases are always used and may be followed by an optional transport specific key distribution phase (see [Figure 2.1](#)):

- Phase 1: Pairing Feature Exchange
- Phase 2 (LE legacy pairing): Short Term Key (STK) Generation
- Phase 2 (LE Secure Connections): Long Term Key (LTK) Generation
- Phase 3: Transport Specific Key Distribution

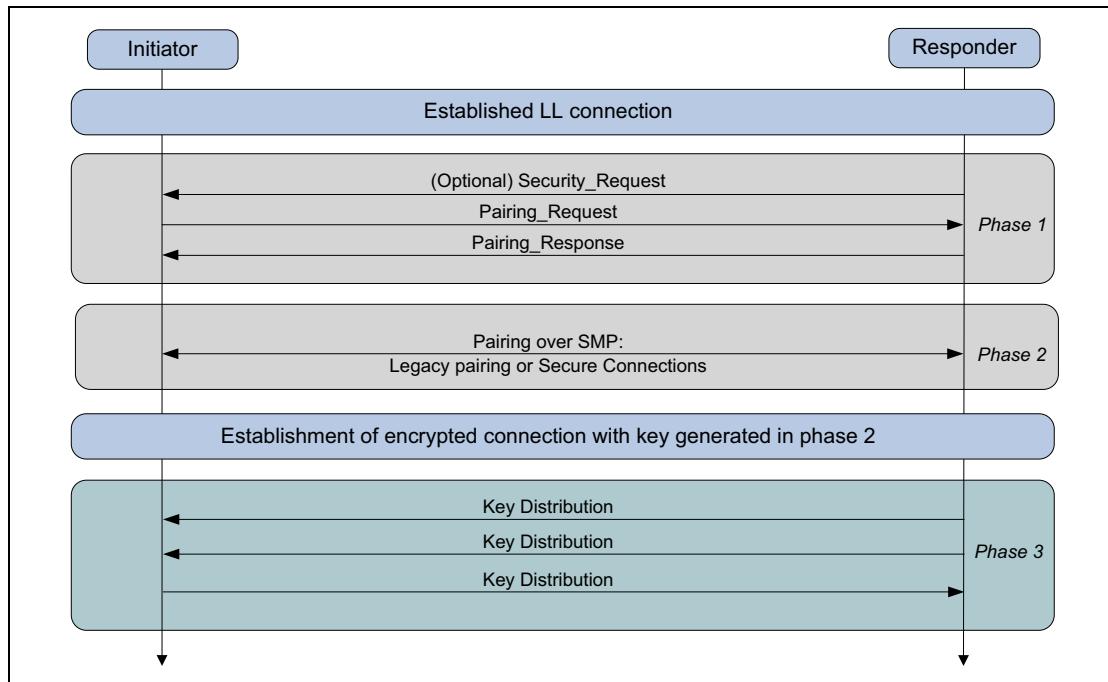


Figure 2.1: LE Pairing Phases

The devices shall first exchange authentication requirements and IO capabilities in the Pairing Feature Exchange to determine which of the following methods shall be used in Phase 2:

- Just Works
- Numeric Comparison (Only for LE Secure Connections)
- Passkey Entry
- Out Of Band (OOB)

Authentication requirements retrieved from the Pairing Feature Exchange also determine whether LE Secure Connections or LE legacy pairing is used.

Optionally, Phase 3 may then be performed to distribute transport specific keys, for example the Identity Resolving Key (IRK) value and Identity Address information. Phase 1 and Phase 3 are identical regardless of the method used in Phase 2.

Phase 3 shall only be performed on a link which is encrypted using:

- The STK generated in Phase 2 when using LE legacy pairing or
- The LTK generated in Phase 2 when using LE Secure Connections or
- The shared Link Key generated using BR/EDR pairing (see [Section 2.3.5.7](#)).

Phase 1 and Phase 2 may be performed on a link which is either encrypted or not encrypted.

2.2 CRYPTOGRAPHIC TOOLBOX

In order to support random addressing, pairing and other operations SM provides a toolbox of cryptographic functions. The following cryptographic functions are defined:

- ah is used to create a 24-bit hash used in random address creation and resolution.

The following cryptographic functions are defined to support the LE legacy pairing process:

- $c1$ is used to generate confirm values used during the pairing process.
- $s1$ is used to generate the STK during the pairing process.

The following cryptographic functions are defined to support the LE Secure Connections pairing process:

- $f4$ is used to generate confirm values during the pairing process.
- $f5$ is used to generate the LTK and the MacKey during the pairing process.
- $f6$ is used to generate the check values during authentication stage 2 in the pairing process.
- $g2$ is used to generate the 6-digit numeric comparison values during authentication stage 1 in the pairing process.
- $h6$ is used to generate the LE LTK from a BR/EDR link key derived from Secure Connections and is used to generate the BR/EDR link key from an LE LTK derived from Secure Connections.

The building block for the cryptographic functions ah , $c1$ and $s1$ is the security function e.

The building block for the cryptographic functions $f4$, $f5$, $f6$, $g2$ and $h6$ is the security function AES-CMAC.

Inside the $f4$, $f5$, $f6$, $g2$ and $h6$ functions when a multi-octet integer parameter is used as input to AES-CMAC the most significant octet of the integer shall be the first octet of the stream and the least significant octet of the integer shall be the last octet of the stream. The output of AES-CMAC inside these functions is a multi-octet integer where the first octet is MSB and the last octet is LSB of this integer.

2.2.1 Security function e

Security function e generates 128-bit $encryptedData$ from a 128-bit key and 128-bit $plaintextData$ using the AES-128-bit block cypher as defined in FIPS-197¹:

$$encryptedData = e(key, plaintextData)$$

The most significant octet of key corresponds to $key[0]$, the most significant octet of $plaintextData$ corresponds to $in[0]$ and the most significant octet of $encryptedData$ corresponds to $out[0]$ using the notation specified in FIPS-197¹.

Note: The security function e can be implemented in a Host or be implemented using the HCI_LE_Encrypt command (see [\[Vol 2\] Part E, Section 7.8.22](#)).

2.2.2 Random Address Hash function ah

The random address hash function ah is used to generate a hash value that is used in resolvable private addresses, see [Part C, Section 10.8.2](#).

The following are inputs to the random address hash function ah :

k is 128 bits

r is 24 bits

$padding$ is 104 bits

r is concatenated with $padding$ to generate r' which is used as the 128-bit input parameter $plaintextData$ to security function e :

$$r' = padding \parallel r$$

The least significant octet of r becomes the least significant octet of r' and the most significant octet of $padding$ becomes the most significant octet of r' .

For example, if the 24-bit value r is 0x423456 then r' is 0x000000000000000000000000423456.

The output of the random address function ah is:

$$ah(k, r) = e(k, r') \bmod 2^{24}$$

The output of the security function e is then truncated to 24 bits by taking the least significant 24 bits of the output of e as the result of ah .

1. NIST Publication FIPS-197 (<http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>)

2.2.3 Confirm value generation function *c1* for LE Legacy Pairing

During the LE legacy pairing process confirm values are exchanged. This confirm value generation function *c1* is used to generate the confirm values.

The following are inputs to the confirm value generation function *c1*:

- k* is 128 bits
- r* is 128 bits
- pres* is 56 bits
- preq* is 56 bits
- iat* is 1 bit
- ia* is 48 bits
- rat* is 1 bit
- ra* is 48 bits
- padding* is 32 bits or 0

iat is concatenated with 7-bits of 0 to create *iat'* which is 8 bits in length. *iat* is the least significant bit of *iat'*.

rat is concatenated 7-bits of 0 to create *rat'* which is 8 bits in length. *rat* is the least significant bit of *rat'*.

pres, *preq*, *rat'* and *iat'* are concatenated to generate *p1* which is XORed with *r* and used as 128-bit input parameter *plaintextData* to security function *e*:

$$p1 = pres \parallel preq \parallel rat' \parallel iat'$$

The octet of *iat'* becomes the least significant octet of *p1* and the most significant octet of *pres* becomes the most significant octet of *p1*.

For example, if the 8-bit *iat'* is 0x01, the 8-bit *rat'* is 0x00, the 56-bit *preq* is 0x07071000000101 and the 56 bit *pres* is 0x05000800000302 then *p1* is 0x0500080000030207071000001010001.

ra is concatenated with *ia* and *padding* to generate *p2* which is XORed with the result of the security function *e* using *p1* as the input parameter *plaintextData* and is then used as the 128-bit input parameter *plaintextData* to security function *e*:

$$p2 = padding \parallel ia \parallel ra$$

The least significant octet of *ra* becomes the least significant octet of *p2* and the most significant octet of *padding* becomes the most significant octet of *p2*.

For example, if 48-bit *ia* is 0xA1A2A3A4A5A6 and the 48-bit *ra* is 0xB1B2B3B4B5B6 then *p2* is 0x00000000A1A2A3A4A5A6B1B2B3B4B5B6.

The output of the confirm value generation function $c1$ is:

$$c1(k, r, \text{preq}, \text{pres}, \text{iat}, \text{rat}, \text{ia}, \text{ra}) = e(k, e(k, r \text{ XOR } p1) \text{ XOR } p2)$$

The 128-bit output of the security function e is used as the result of confirm value generation function $c1$.

For example, if the 128-bit k is 0x00000000000000000000000000000000, the 128-bit value r is 0x5783D52156AD6F0E6388274EC6702EE0, the 128-bit value $p1$ is 0x05000800000302070710000001010001 and the 128-bit value $p2$ is 0x00000000A1A2A3A4A5A6B1B2B3B4B5B6 then the 128-bit output from the $c1$ function is 0x1e1e3fef878988ead2a74dc5bef13b86.

2.2.4 Key generation function $s1$ for LE Legacy Pairing

The key generation function $s1$ is used to generate the STK during the LE legacy pairing process.

The following are inputs to the key generation function $s1$:

k is 128 bits

$r1$ is 128 bits

$r2$ is 128 bits

The most significant 64-bits of $r1$ are discarded to generate $r1'$ and the most significant 64-bits of $r2$ are discarded to generate $r2'$.

For example if the 128-bit value $r1$ is

0x000F0E0D0C0B0A091122334455667788 then $r1'$ is 0x1122334455667788.

If the 128-bit value $r2$ is 0x010203040506070899AABBCCDDEEFF00 then $r2'$ is 0x99AABBCCDDEEFF00.

$r1'$ is concatenated with $r2'$ to generate r' which is used as the 128-bit input parameter *plaintextData* to security function e :

$$r' = r1' \parallel r2'$$

The least significant octet of $r2'$ becomes the least significant octet of r' and the most significant octet of $r1'$ becomes the most significant octet of r' .

For example, if the 64-bit value $r1'$ is 0x1122334455667788 and $r2'$ is

0x99AABBCCDDEEFF00 then r' is

0x112233445566778899AABBCCDDEEFF00.

The output of the key generation function $s1$ is:

$$s1(k, r1, r2) = e(k, r')$$

The 128-bit output of the security function e is used as the result of key generation function $s1$.

For example if the 128-bit value k is

0x00

and the 128-bit value r' is

0x112233445566778899AABBCCDDEEFF00

then the output from the key generation function $s1$ is

0x9a1fe1f0e8b0f49b5b4216ae796da062.

2.2.5 Function AES-CMAC

RFC-4493¹ defines the Cipher-based Message Authentication Code (CMAC) that uses AES-128 as the block cipher function, also known as AES-CMAC.

The inputs to AES-CMAC are:

- m is the variable length data to be authenticated
- k is the 128-bit key

The 128-bit message authentication code (MAC) is generated as follows:²

$$\text{MAC} = \text{AES-CMAC}_k(m)$$

A device can implement AES functions in the Host or can use the HCI_LE_Encrypt command (see [\[Vol 2\] Part E, Section 7.8.22](#)) in order to use the AES function in the Controller.

2.2.6 LE Secure Connections Confirm Value Generation Function f4

During the LE Secure Connections pairing process, confirm values are exchanged. These confirm values are computed using the confirm value generation function f4.

This confirm value generation function makes use of the MAC function AES-CMAC_X, with a 128-bit key X.

The inputs to function f4 are:

- U is 256 bits
- V is 256 bits
- X is 128 bits
- Z is 8 bits

Z is zero (i.e. 8 bits of zeros) for Numeric Comparison and OOB protocol. In the Passkey Entry protocol, the most significant bit of Z is set equal to one and the least significant bit is made up from one bit of the passkey e.g. if the passkey bit is 1, then Z = 0x81 and if the passkey bit is 0, then Z = 0x80.

U, V and Z are concatenated and used as input m to the function AES-CMAC and X is used as the key k.

1. <http://www.ietf.org/rfc/rfc4493.txt>

2. RFC4493 uses the notation $\text{MAC} = \text{AES-CMAC}(k,m)$ where k is the key. This is functionally the same as the notation used in this specification $\text{MAC} = \text{AES-CMAC}_k(m)$

The inputs to $f4$ are different depending on different association models:

Numeric Comparison/ Just Works	Out-Of-Band	Passkey Entry
$Ca = f4(PKax, PKbx, Na, 0)$ $Cb = f4(PKbx, PKax, Nb, 0)$	$Ca = f4(PKax, PKax, ra, 0)$ $Cb = f4(PKbx, PKbx, rb, 0)$	$Cai = f4(PKax, PKbx, Nai, rai)$ $Cbi = f4(PKbx, PKax, Nbi, rbi)$

Table 2.1: Inputs to $f4$ for the different protocols

$PKax$ denotes the x-coordinate of the public key PKa of A.

Similarly, $PKbx$ denotes the x-coordinate of the public key PKb of B.

Nai is the nonce value of i^{th} round. For each round Nai value is a new 128-bit number. Similarly, rai is a one bit value of the passkey expanded to 8 bits (either 0x80 or 0x81).

Na and Nb are nonces from Devices A and B. ra and rb are random values generated by devices A and B.

The output of the confirm value generation function $f4$ is as follows:

$$f4(U, V, X, Z) = \text{AES-CMAC}_X(U \parallel V \parallel Z)$$

The least significant octet of Z becomes the least significant octet of the AES-CMAC input message m and the most significant octet of U becomes the most significant octet of the AES-CMAC input message m .

2.2.7 LE Secure Connections Key Generation Function $f5$

The LE Secure Connections key generation function $f5$ is used to generate derived keying material in order to create the LTK and keys for the commitment function $f6$ during the LE Secure Connections pairing process.

The definition of this key generation function makes use of the MAC function AES-CMAC_T with a 128-bit key T .

The inputs to function $f5$ are:

W is 256 bits

N_1 is 128 bits

N_2 is 128 bits

A_1 is 56 bits

A_2 is 56 bits

The key (T) is computed as follows:

$$T = \text{AES-CMAC}_{\text{SALT}}(W)$$

$SALT$ is the 128-bit value:

0x6C88 8391 AAF5 A538 6037 0BDB 5A60 83BE

Counter, keyID, N1, N2, A1, A2, and Length are concatenated and used as input m to the function AES-CMAC and T is used as the key k . Counter is one octet. Length is two octets.

The string “btle” is mapped into keyID using extended ASCII as follows:

```

keyID[0] = 0110 0101
keyID[1] = 0110 1100
keyID[2] = 0111 0100
keyID[3] = 0110 0010
keyID     = 0x62746c65

```

The output of the key generation function $f5$ is as follows:

$$f5(W, N1, N2, A1, A2) = \text{AES-CMAC}_T(\text{Counter} = 0 || \text{keyID} || N1 || N2 || A1 || A2 || \text{Length} = 256) || \text{AES-CMAC}_T(\text{Counter} = 1 || \text{keyID} || N1 || N2 || A1 || A2 || \text{Length} = 256)$$

The least significant octet of Length becomes the least significant octet of the AES-CMAC input message m and the most significant octet of Counter becomes the most significant octet of the AES-CMAC input message m .

The LTK and MacKey are calculated as:

$$\text{MacKey} || \text{LTK} = f5(\text{DHKey}, \text{N_master}, \text{N_slave}, \text{BD_ADDR_master}, \text{BD_ADDR_slave})$$

DHKey is the shared secret Diffie-Hellman key generated during LE Secure Connections pairing phase 2.

N_master is the random number sent by the master to the slave and N_slave is the random number sent by the slave to the master.

BD_ADDR_master is the device address of the master and BD_ADDR_slave is the device address of the slave. The device addresses are the values used during connection setup. The least significant bit in the most significant octet in both BD_ADDR_master and BD_ADDR_slave is set to 1 if the address is a random address and set to 0 if the address is a public address. The 7 most significant bits of the most significant octet in both BD_ADDR_master and BD_ADDR_slave are set to 0.

The LTK is the least significant 128 bits (Counter = 1) of $f5$. The MacKey (see [Section 2.2.8](#)) is the most significant 128 bits (Counter = 0) of $f5$.

A device can implement Diffie-Hellman key generation in the Host or can use the HCI_LE_Generate_DHKey command (see [\[Vol 2\] Part E, Section 7.8.37](#)) to generate the key in the Controller. Note: when using the HCI_LE_Generate_DHKey command, the device can only pair one remote device at a time.

2.2.8 LE Secure Connections Check Value Generation Function *f6*

The LE Secure Connections check value generation function *f6* is used to generate check values during authentication stage 2 of the LE Secure Connections pairing process.

The definition of the *f6* function makes use of the MAC function AES-CMAC_W with a 128-bit key W.

The inputs to function *f6* are:

- W is 128 bits
- N1 is 128 bits
- N2 is 128 bits
- R is 128 bits
- IOcap is 24 bits
- A1 is 56 bits
- A2 is 56 bits

N1, N2, R, IOcap, A1 and A2 are concatenated and used as input *m* to the function AES-CMAC and W is used as the key *k*.

The inputs to *f6* are different depending on different association models:

Numeric Comparison/ Just Works	Out-Of-Band	Passkey Entry
Ea = <i>f6</i> (MacKey, Na, Nb, 0, IOcapA, A, B) Eb = <i>f6</i> (MacKey, Nb, Na, 0, IOcapB, B, A)	Ea = <i>f6</i> (MacKey, Na, Nb, rb, IOcapA, A, B) Eb = <i>f6</i> (MacKey, Nb, Na, ra, IOcapB, B, A)	Ea = <i>f6</i> (MacKey, Na20, Nb20, rb, IOcapA, A, B) Eb = <i>f6</i> (MacKey, Nb20, Na20, ra, IOcapB, B, A)

Table 2.2: Inputs to *f6* for the different protocols

MacKey is the 128-bit MSBs of the output of *f5*.

Na is the random number sent by the master to the slave and Nb is the random number sent by the slave to the master.

IOcapA is the capabilities of the master and IOcapB is the capabilities of the slave. IOcapA and IOcapB are both three octets with the most significant octet as the AuthReq parameter, the middle octet as the OOB data flag and the least significant octet as the IO capability parameter. The AuthReq, OOB data flag and IO capability parameters are present in the Pairing Request and Pairing Response SMP packets.

In Passkey Entry, ra and rb are 6-digit passkey values expressed as a 128-bit integer. For instance, if the 6-digit value of ra is 131313 then

ra = 0x 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 f1

A is the device address of the master and B is the device address of the slave. The least significant bit in the most significant octet in both A and B is set to 1 if the address is a random address and set to 0 if the address is a public address. The 7 most significant bits of the most significant octet in both A and B are set to 0.

The output of the check value generation function $f6$ is as follows:

$$\begin{aligned} f6(W, N1, N2, R, IOcap, A1, A2) = \\ \text{AES-CMAC}_W(N1 \parallel N2 \parallel R \parallel IOcap \parallel A1 \parallel A2) \end{aligned}$$

The least significant octet of A2 becomes the least significant octet of the AES-CMAC input message m and the most significant octet of N1 becomes the most significant octet of the AES-CMAC input message m.

2.2.9 LE Secure Connections Numeric Comparison Value Generation Function $g2$

The LE Secure Connections numeric comparison value generation function $g2$ is used to generate the numeric comparison values during authentication stage 1 of the LE Secure Connections pairing process.

The definition of $g2$ makes use of the MAC function AES-CMAC_X with 128-bit key X.

The inputs to function $g2$ are:

U is 256 bits

V is 256 bits

X is 128 bits

Y is 128 bits

U, V, and Y are concatenated and used as input m to the function AES-CMAC and X is used as the key k .

The output of the numeric comparison value generation function $g2$ is as follows:

$$g2(U, V, X, Y) = \text{AES-CMAC}_X(U \parallel V \parallel Y) \bmod 2^{32}$$

The least significant octet of Y becomes the least significant octet of the AES-CMAC input message m and the most significant octet of U becomes the most significant octet of the AES-CMAC input message m.

The numeric verification value is taken as the six least significant digits of the 32-bit integer $g2(\text{PKax}, \text{PKbx}, \text{Na}, \text{Nb})$ where PKax denotes the x-coordinate of the public key PKa of A and PKbx denotes the x-coordinate of the public key PKb of B. Na and Nb are nonces from devices A and B. The value is then

converted to decimal numeric value. The checksum used for numeric comparison is the least significant six digits.

$$\text{Compare Value} = g2(U, V, X, Y) \bmod 10^6$$

For example, if output = 0x 01 2e b7 2a then decimal value = 19838762 and the checksum used for numeric comparison is 838762.

2.2.10 Link Key Conversion Function *h6*

The function *h6* is used to convert keys of a given size from one key type to another key type with equivalent strength.

The definition of the *h6* function makes use of the hashing function AES-CMAC_W with 128-bit key W.

The inputs to function *h6* are:

W is 128 bits

keyID is 32 bits

keyID is used as input m to the hashing function AES-CMAC and the most significant 128-bits of W are used as the key k (2.4.5)

The output of *h6* is as follows:

$$h6(W, \text{keyID}) = \text{AES-CMAC}_W(\text{keyID})$$

2.3 PAIRING METHODS

When pairing is started, the Pairing Feature Exchange shall be initiated by the initiating device. If the responding device does not support pairing or pairing cannot be performed then the responding device shall respond using the Pairing Failed message with the error code “Pairing Not Supported” otherwise it responds with a Pairing Response message.

The Pairing Feature Exchange is used to exchange IO capabilities, OOB authentication data availability, authentication requirements, key size requirements and which transport specific keys to distribute. The IO capabilities, OOB authentication data availability and authentication requirements are used to determine the key generation method used in Phase 2.

All of the LE legacy pairing methods use and generate 2 keys:

1. Temporary Key (TK): a 128-bit temporary key used in the pairing process which is used to generate STK (see [Section 2.3.5.5](#)).
2. Short Term Key (STK): a 128-bit temporary key used to encrypt a connection following pairing.

The LE Secure Connections pairing methods use and generate 1 key:

1. Long Term Key (LTK): a 128-bit key used to encrypt the connection following pairing and subsequent connections.

Authentication requirements are set by GAP, (see [\[Part C\], Generic Access Profile, Section 10.3](#)). The authentication requirements include the type of bonding and man-in-the-middle protection (MITM) requirements.

The initiating device indicates to the responding device which transport specific keys it would like to send to the responding device and which keys it would like the responding device to send to the initiator. The responding device replies with the keys that the initiating device shall send and the keys that the responding device shall send. The keys that can be distributed are defined in [Section 2.4.3](#). If the device receives a command with invalid parameters, it shall respond with Pairing Failed command with the error code “Invalid Parameters.”

2.3.1 Security Properties

Security properties provided by SM are classified into the following categories:

- LE Secure Connections pairing
- Authenticated MITM protection
- Unauthenticated no MITM protection
- No security requirements

LE Secure Connections pairing utilizes the P-256 elliptic curve.

In LE legacy pairing, Authenticated man-in-the-middle (MITM) protection is obtained by using the passkey entry pairing method or may be obtained using the out of band pairing method. In LE Secure Connections pairing, Authenticated man-in-the-middle (MITM) protection is obtained by using the passkey entry pairing method or the numeric comparison method or may be obtained using the out of band pairing method. To ensure that Authenticated MITM Protection is generated, the selected Authentication Requirements option must have MITM protection specified.

Unauthenticated no MITM Protection does not have protection against MITM attacks.

For LE Legacy Pairing, none of the pairing methods provide protection against a passive eavesdropper during the pairing process as predictable or easily established values for TK are used. If the pairing information is distributed without an eavesdropper being present then all the pairing methods provide confidentiality.

An initiating device shall maintain a record of the Security Properties for the distributed keys in a security database.

A responding device may maintain a record of the distributed key sizes and Security Properties for the distributed keys in a security database. Depending upon the key generation method and negotiated key size a responding device may have to reduce the key length (see [Section 2.3.4](#)) so that the initiator and responder are using identical keys.

Security properties of the key generated in phase 2 under which the keys are distributed shall be stored in the security database.

2.3.2 IO Capabilities

Input and output capabilities of a device are combined to generate its IO capabilities. The input capabilities are described in [Table 2.3](#). The output capabilities are described in [Table 2.4](#).

Capability	Description
No input	Device does not have the ability to indicate 'yes' or 'no'
Yes / No	Device has at least two buttons that can be easily mapped to 'yes' and 'no' or the device has a mechanism whereby the user can indicate either 'yes' or 'no' (see note below).
Keyboard	Device has a numeric keyboard that can input the numbers '0' through '9' and a confirmation. Device also has at least two buttons that can be easily mapped to 'yes' and 'no' or the device has a mechanism whereby the user can indicate either 'yes' or 'no' (see note below).

Table 2.3: User Input Capabilities

Note: 'yes' could be indicated by pressing a button within a certain time limit otherwise 'no' would be assumed.

Capability	Description
No output	Device does not have the ability to display or communicate a 6 digit decimal number
Numeric output	Device has the ability to display or communicate a 6 digit decimal number

Table 2.4: User Output Capabilities

The individual input and output capabilities are mapped to a single IO capability for that device which is used in the pairing feature exchange. The mapping is described in [Table 2.5](#).

		Local output capacity	
		No output	Numeric output
Local input capacity	No input	NoInputNoOutput	DisplayOnly
	Yes/No	NoInputNoOutput ¹	DisplayYesNo
	Keyboard	KeyboardOnly	KeyboardDisplay

Table 2.5: I/O Capabilities Mapping

1. None of the pairing algorithms can use Yes/No input and no output, therefore NoInputNoOutput is used as the resulting IO capability.

2.3.3 OOB Authentication Data

An out of band mechanism may be used to communicate information which is used during the pairing process. The information shall be a sequence of Advertising Data structures (see [\[Vol 3\] Part C, Section 11](#)).

The OOB data flag shall be set if a device has the peer device's out of band authentication data. A device uses the peer device's out of band authentication data to authenticate the peer device. In LE legacy pairing, the out of band method is used if both the devices have the other device's out of band authentication data available. In LE Secure Connections pairing, the out of band method is used if at least one device has the peer device's out of band authentication data available.

2.3.4 Encryption Key Size

Each device shall have maximum and minimum encryption key length parameters which defines the maximum and minimum size of the encryption key allowed in octets. The maximum and minimum encryption key length parameters shall be between 7 octets (56 bits) and 16 octets (128 bits), in 1 octet (8 bit) steps. This is defined by a profile or device application.

The smaller value of the initiating and responding devices maximum encryption key length parameters shall be used as the encryption key size.

Both the initiating and responding devices shall check that the resultant encryption key size is not smaller than the minimum key size parameter for that device and if it is, the device shall send the Pairing Failed command with error code "Encryption Key Size".

The encryption key size may be stored so it can be checked by any service that has minimum encryption key length requirements.

If a key has an encryption key size that is less than 16 octets (128 bits), it shall be created by masking the appropriate MSBs of the generated key to provide a resulting key that has the agreed encryption key size. The masking shall be done after generation and before being distributed, used or stored.

For example, if a 128-bit encryption key is

0x123456789ABCDEF0123456789ABCDEF0

and it is reduced to 7 octets (56 bits), then the resulting key is

0x00000000000000003456789ABCDEF0.

2.3.5 Pairing Algorithms

The information exchanged in Phase 1 is used to select which key generation method is used in Phase 2.

When LE legacy pairing is used, the pairing is performed by each device generating a Temporary Key (*TK*). The method to generate *TK* depends upon the pairing method chosen using the algorithm described in [Section 2.3.5.1](#). If Just Works is used then *TK* shall be generated as defined in [Section 2.3.5.2](#). If Passkey Entry is used then *TK* shall be generated as defined in [Section 2.3.5.3](#). If Out Of Band is used then *TK* shall be generated as defined in [Section 2.3.5.4](#). The *TK* value shall be used in the authentication mechanism defined in [Section 2.3.5.5](#) to generate the STK and encrypt the link.

2.3.5.1 Selecting Key Generation Method

If both devices have not set the MITM option in the Authentication Requirements Flags, then the IO capabilities shall be ignored and the Just Works association model shall be used.

In LE legacy pairing, if both devices have Out of Band authentication data, then the Authentication Requirements Flags shall be ignored when selecting the pairing method and the Out of Band pairing method shall be used. Otherwise, the IO capabilities of the device shall be used to determine the pairing method as defined in [Table 2.8](#).

In LE Secure Connections pairing, if one or both devices have out of band authentication data, then the Authentication Requirements Flags shall be ignored when selecting the pairing method and the Out of Band pairing method shall be used. Otherwise, the IO capabilities of the device shall be used to determine the pairing method as defined in [Table 2.8](#).

[Table 2.6](#) defines the STK generation method when at least one of the devices does not support LE Secure Connections.

		Initiator			
		OOB Set	OOB Not Set	MITM Set	MITM Not Set
Responder	OOB Set	Use OOB	Check MITM		
	OOB Not Set	Check MITM	Check MITM		
	MITM Set			Use IO Capabilities	Use IO Capabilities
	MITM Not Set			Use IO Capabilities	Use Just Works

Table 2.6: Rules for using Out-of-Band and MITM flags for LE legacy pairing

Table 2.7 defines the LTK generation method when both devices support LE Secure Connections.

		Initiator			
		OOB Set	OOB Not Set	MITM Set	MITM Not Set
Responder	OOB Set	Use OOB	Use OOB		
	OOB Not Set	Use OOB	Check MITM		
	MITM Set			Use IO Capabilities	Use IO Capabilities
	MITM Not Set			Use IO Capabilities	Use Just Works

Table 2.7: Rules for using Out-of-Band and MITM flags for LE Secure Connections Pairing

		Initiator			
Responder	DisplayOnly	Display YesNo	Keyboard Only	NoInput NoOutput	Keyboard Display
Display Only	Just Works Unauthenticated	Just Works Unauthenticated	Passkey Entry: responder displays, initiator inputs Authenticated	Just Works Unauthenticated	Passkey Entry: responder displays, initiator inputs Authenticated
	Just Works Unauthenticated	Just Works (For LE Legacy Pairing) Unauthenticated	Passkey Entry: responder displays, initiator inputs Authenticated	Just Works Unauthenticated	Passkey Entry (For LE Legacy Pairing): responder displays, initiator inputs Authenticated
Display YesNo	Just Works Unauthenticated	Numeric Comparison (For LE Secure Connections) Authenticated	Passkey Entry: responder displays, initiator inputs Authenticated	Just Works Unauthenticated	Numeric Comparison (For LE Secure Connections) Authenticated

Table 2.8: Mapping of IO Capabilities to Key Generation Method

		Initiator			
Responder	DisplayOnly	Display YesNo	Keyboard Only	NoInput NoOutput	Keyboard Display
Keyboard Only	Passkey Entry: initiator displays, responder inputs Authenticated	Passkey Entry: initiator displays, responder inputs Authenticated	Passkey Entry: initiator and responder inputs Authenticated	Just Works Unauthenticated	Passkey Entry: initiator displays, responder inputs Authenticated
	Just Works Unauthenticated	Just Works Unauthenticated	Just Works Unauthenticated	Just Works Unauthenticated	Just Works Unauthenticated
Keyboard Display	Passkey Entry: initiator displays, responder inputs Authenticated	Passkey Entry (For LE Legacy Pairing): initiator displays, responder inputs Authenticated	Passkey Entry: responder displays, initiator inputs Authenticated	Just Works Unauthenticated	Passkey Entry (For LE Legacy Pairing): initiator displays, responder inputs Authenticated
	Authentic- ated	Numeric Comparison (For LE Secure Connections) Authentic- ated	Authenti- cated		Numeric Comparison (For LE Secure Connections) Authentic- ated

Table 2.8: Mapping of IO Capabilities to Key Generation Method

The generated key will either be an Authenticated or Unauthenticated key. If the out of band authentication method is used and the Out of Band mechanism is known to be secure from eavesdropping the key is assumed to be Authenticated; however, the exact strength depends upon the method used to transfer the out of band information. If the Out of Band method is used and the Out of Band mechanism is not secure from eavesdropping or the level of eavesdropping protection is unknown, the key shall be Unauthenticated. The mapping of IO capabilities to an authenticated or unauthenticated key is described in [Table 2.8](#).

In LE legacy pairing, if the initiating device has Out of Band data and the responding device does not have Out of Band data then the responding device may send the Pairing Failed command with the error code “OOB Not Available” instead of the Pairing Response command.

If the key generation method does not result in a key that provides sufficient security properties then the device shall send the Pairing Failed command with the error code “Authentication Requirements”.

2.3.5.2 LE Legacy Pairing - Just Works

The Just Works STK generation method provides no protection against eavesdroppers or man in the middle attacks during the pairing process. If the attack is not present during the pairing process then confidentiality can be established by using encryption on a future connection.

Both devices set the TK value used in the authentication mechanism defined in [Section 2.3.5.5](#) to zero.

2.3.5.3 LE Legacy Pairing - Passkey Entry

The Passkey Entry STK generation method uses 6 numeric digits passed out of band by the user between the devices. A 6 digit numeric randomly generated passkey achieves approximately 20 bits of entropy.

If the IO capabilities of a device are DisplayOnly or if [Table 2.8](#) defines that the device displays the passkey, then that device shall display a randomly generated passkey value between 000,000 and 999,999. The display shall ensure that all 6 digits are displayed – including zeros. The other device shall allow the user to input a value between 000,000 and 999,999.

If entry of Passkey in UI fails to occur or is canceled then the device shall send Pairing Failed command with reason code “Passkey Entry Failed”.

For example, if the user entered passkey is ‘019655’ then TK shall be 0x0000000000000000000000000000004CC7.

The passkey Entry method provides protection against active “man-in-the-middle” (MITM) attacks as an active man-in-the-middle will succeed with a probability of 0.000001 on each invocation of the method.

The Passkey Entry STK generation method provides very limited protection against eavesdroppers during the pairing process because of the limited range of possible TK values which STK is dependent upon. If the attacker is not present during the pairing process then confidentiality and authentication can be established by using encryption on a future connection.

The TK value shall then be used in the authentication mechanism defined in [Section 2.3.5.5](#).

2.3.5.4 Out of Band

An out of band mechanism may be used to communicate information to help with device discovery, for example device address, and the 128-bit *TK* value used in the pairing process. The *TK* value shall be a 128-bit random number using the requirements for random generation defined in [\[Vol 2\] Part H, Section 2](#).

If the OOB communication is resistant to MITM attacks, then this association method is also resistant to MITM attacks. Also, in the Out of Band method, the size of authentication parameter (*TK*) need not be restricted by what the user can comfortably read or type. For that reason, the Out of Band method can be more secure than using the Passkey Entry or Just Works methods. However, both devices need to have matching OOB interfaces.

MITM protection is only provided if an active man-in-the-middle chance of a successful attack has a probability of 0.000001 or less in succeeding.

2.3.5.5 LE Legacy Pairing Phase 2

The initiating device generates a 128-bit random number (*Mrand*).

The initiating device calculates the 128-bit confirm value (*Mconfirm*) using the confirm value generation function *c1* (see [Section 2.2.3](#)) with the input parameter *k* set to *TK*, the input parameter *r* set to *Mrand*, the input parameter *preq* set to Pairing Request command, the input parameter *pres* set to the Pairing Response command, the input parameter *iat* set to the initiating device address type, *ia* set to the initiating device address, *rat* set to the responding device address type and *ra* set to the responding device address:

$$\begin{aligned} Mconfirm = c1(TK, Mrand, \\ \text{Pairing Request command, Pairing Response command,} \\ \text{initiating device address type, initiating device address,} \\ \text{responding device address type, responding device address}) \end{aligned}$$

Initiating and responding device addresses used for confirmation generation shall be device addresses used during connection setup, see [\[Part C\], Generic Access Profile, Section 9.3](#)

The responding device generates a 128-bit random number (*Srand*).

The responding device calculates the 128-bit confirm value (*Sconfirm*) using the confirm value generation function *c1* (see [Section 2.2.3](#)) with the input parameter *k* set to *TK*, the input parameter *r* set to *Srand*, the input parameter *preq* set to Pairing Request command, the input parameter *pres* set to the Pairing Response command, the input parameter *iat* set to the initiating device address type, *ia* set to the initiating device address, *rat* set to the responding device address type and *ra* set to the responding device address:

$$\begin{aligned} Sconfirm = c1(TK, Srand, \\ \text{Pairing Request command, Pairing Response command,} \end{aligned}$$

initiating device address type, initiating device address,
responding device address type, responding device address)

The initiating device transmits *Mconfirm* to the responding device. When the responding device receives *Mconfirm* it transmits *Sconfirm* to the initiating device. When the initiating device receives *Sconfirm* it transmits *Mrand* to the responding device.

The responding device verifies the *Mconfirm* value by repeating the calculation the initiating device performed, using the *Mrand* value received.

If the responding device's calculated *Mconfirm* value does not match the received *Mconfirm* value from the initiating device then the pairing process shall be aborted and the responding device shall send the Pairing Failed command with reason code "Confirm Value Failed".

If the responding device's calculated *Mconfirm* value matches the received *Mconfirm* value from the initiating device the responding device transmits *Srand* to the initiating device.

The initiating device verifies the received *Sconfirm* value by repeating the calculation the responding device performed, using the *Srand* value received.

If the initiating devices calculated *Sconfirm* value does not match the received *Sconfirm* value from the responding device then the pairing process shall be aborted and the initiating device shall send the Pairing Failed command with the reason code "Confirm Value Failed".

If the initiating device's calculated *Sconfirm* value matches the received *Sconfirm* value from the responding device the initiating device then calculates STK and tells the Controller to enable encryption.

STK is generated using the key generation function *s1* defined in [Section 2.2.4](#) with the input parameter *k* set to *TK*, the input parameter *r1* set to *Srand*, and the input parameter *r2* set to *Mrand*:

$$\text{STK} = s1(TK, Srand, Mrand)$$

If the encryption key size is less than 128 bits then the STK shall be masked to the correct key size as described in [Section 2.3.4](#).

The initiator shall use the generated STK to either enable encryption on the link or if encryption has already been enabled, perform the encryption pause procedure (see [Section 2.4.4.1](#)).

2.3.5.6 LE Secure Connections Pairing Phase 2

The Long Term Key is generated in LE Secure Connections pairing phase 2.

2.3.5.6.1 Public Key Exchange

Initially, each device generates its own Elliptic Curve Diffie-Hellman (ECDH) public-private key pair (phase 1). The public-private key pair contains a private (secret) key, and a public key. The private keys of devices A and B are denoted as SK_a and SK_b respectively. The public keys of devices A and B are denoted as PK_a and PK_b respectively. This key pair needs to be generated only once per device and may be computed in advance of pairing. A device may, at any time, choose to discard its public-private key pair and generate a new one, although there is not a requirement to do so.

Pairing is initiated by the initiating device sending its public key to the receiving device (phase 1a). The responding device replies with its own public key (phase 1b). These public keys are not regarded as secret although they may identify the devices. Note that phases 1a and 1b are the same in all three protocols.

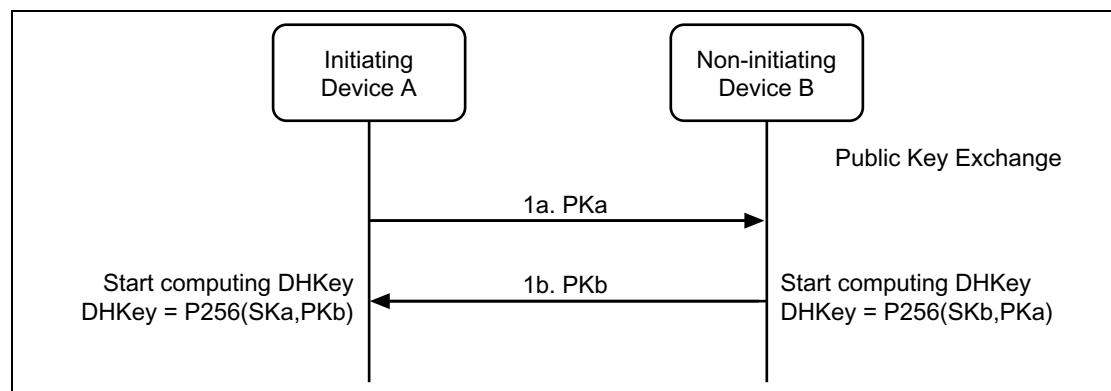


Figure 2.2: Public Key Exchange

After the public keys have been exchanged, the device can then start computing the Diffie-Hellman Key.

When the Security Manager is placed in a Debug mode it shall use the following Diffie-Hellman private / public key pair:

Private key: 3f49f6d4 a3c55f38 74c9b3e3 d2103f50 4aff607b eb40b799 5899b8a6 cd3c1abd

Public key (X): 20b003d2 f297be2c 5e2c83a7 e9f9a5b9 eff49111 acf4fdde cc030148 0e359de6

Public key (Y): dc809c49 652aeb6d 63329abf 5a52155c 766345c2 8fed3024 741c8ed0 1589d28b

Note: Only one side (initiator or responder) needs to set Secure Connections debug mode in order for debug equipment to be able to determine the LTK and, therefore, be able to monitor the encrypted connection.

2.3.5.6.2 Authentication Stage 1 – Just Works or Numeric Comparison

The Numeric Comparison association model will be used during pairing if the MITM bit is set to 1 in the Authentication Requirements in the Pairing Request PDU and/or Pairing Response PDU and both devices have IO capabilities set to either DisplayYesNo or KeyboardDisplay.

The sequence diagram of Authentication Stage 1 for the Just Works or Numeric Comparison protocol from the cryptographic point of view is shown in Figure 2.3.

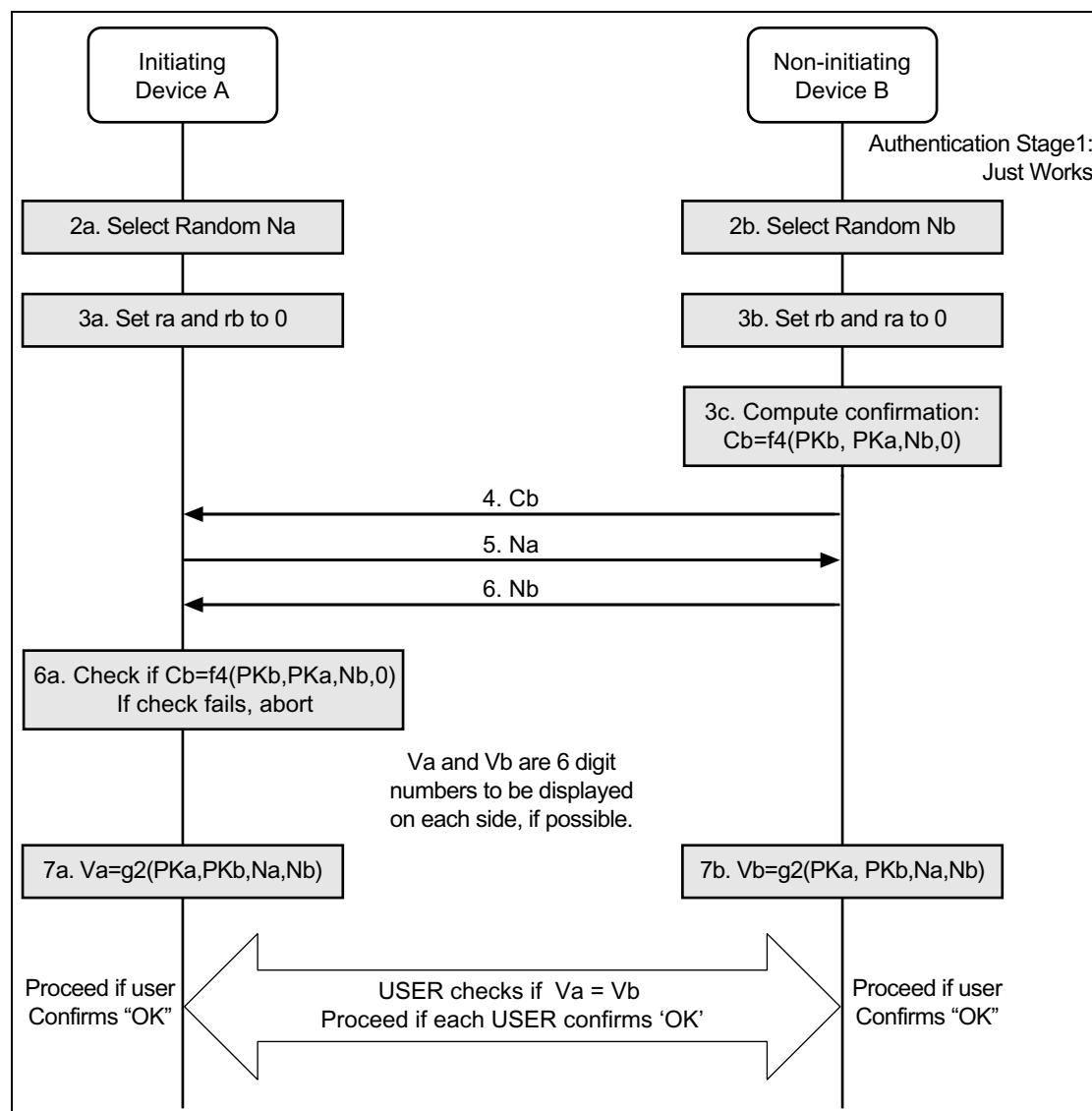


Figure 2.3: "Authentication Stage 1: Just Works or Numeric Comparison, LE Secure Connections

After the public keys have been exchanged, each device selects a pseudo-random 128-bit nonce (step 2). This value is used to prevent replay attacks and must be freshly generated with each instantiation of the pairing protocol. This value should be generated directly from a physical source of randomness or

with a good pseudo-random generator seeded with a random value from a physical source.

Following this the responding device then computes a commitment to the two public keys that have been exchanged and its own nonce value (step 3c). This commitment is computed as a one-way function of these values and is transmitted to the initiating device (step 4). The commitment prevents an attacker from changing these values at a later time.

The initiating and responding devices then exchange their respective nonce values (steps 5 and 6) and the initiating device confirms the commitment (step 6a). A failure at this point indicates the presence of an attacker or other transmission error and causes the protocol to abort. The protocol may be repeated with or without the generation of new public-private key pairs, but new nonces must be generated if the protocol is repeated.

When Just Works is used, the commitment checks (steps 7a and 7b) are not performed and the user is not shown the 6-digit values.

When Numeric Comparison is used, assuming that the commitment check succeeds, the two devices each compute 6-digit confirmation values that are displayed to the user on their respective devices (steps 7a, 7b, and 8). The user is expected to check that these 6-digit values match and to confirm if there is a match. If there is no match, the protocol aborts and, as before, new nonces must be generated if the protocol is to be repeated.

An active MITM must inject its own key material into this process to have any effect other than denial-of-service. A simple MITM attack will result in the two 6-digit display values being different with probability 0.999999. A more sophisticated attack may attempt to engineer the display values to match, but this is thwarted by the commitment sequence. If the attacker first exchanges nonces with the responding device, it must commit to the nonce that it will use with the initiating device before it sees the nonce from the initiating device. If the attacker first exchanges nonces with the initiating device, it must send a nonce to the responding device before seeing the nonce from the responding device. In each case, the attacker must commit to at least the second of its nonces before knowing the second nonce from the legitimate devices. It therefore cannot choose its own nonces in such a way as to cause the display values to match.

2.3.5.6.3 Authentication Stage 1 – Passkey Entry

The Passkey Entry protocol is used when SMP IO capability exchange sequence indicates that Passkey Entry shall be used.

The sequence diagram for Authentication Stage 1 for Passkey Entry from the cryptographic point of view is shown in [Figure 2.4](#).

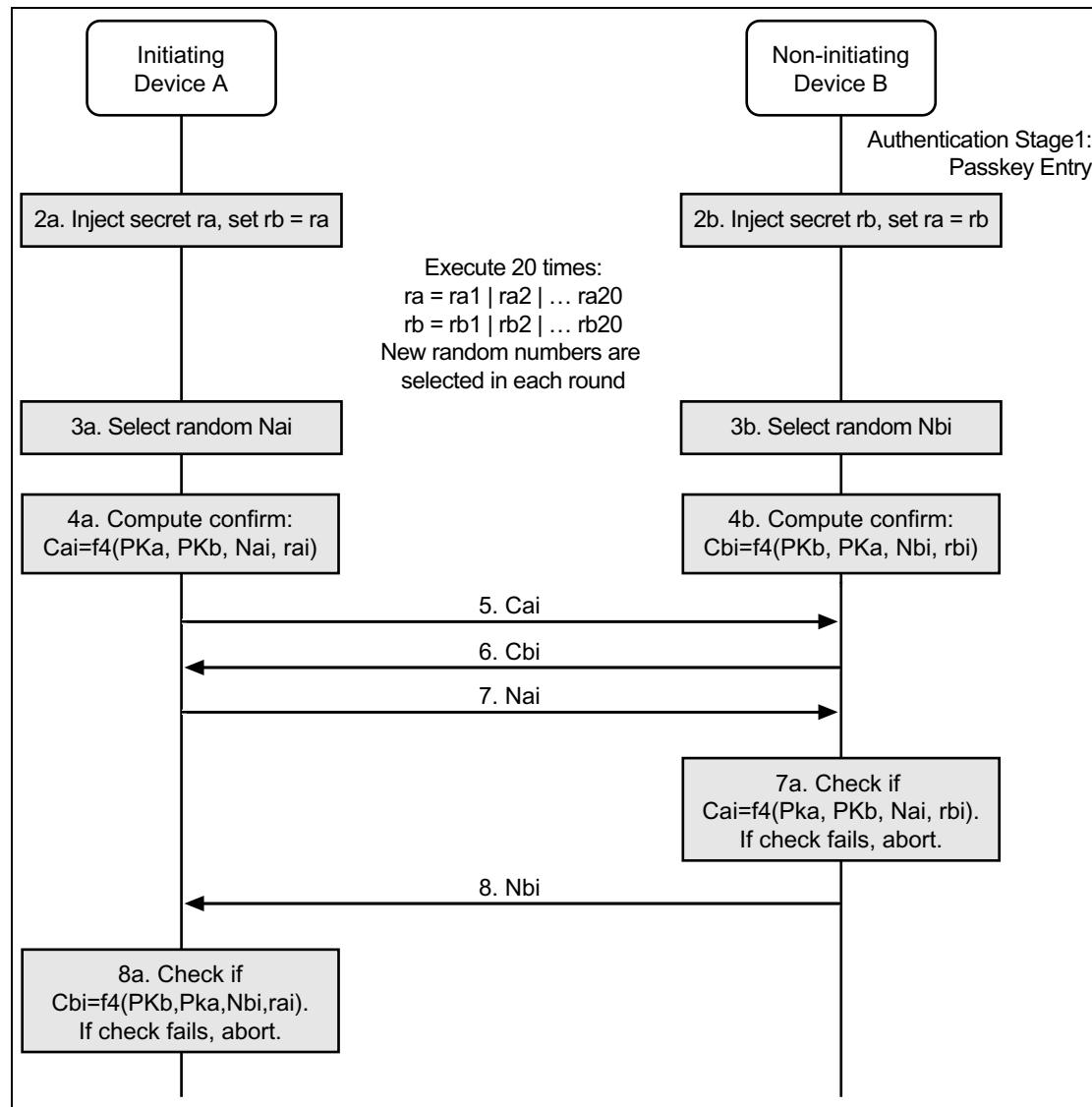


Figure 2.4: Authentication Stage 1: Passkey Entry, LE Secure Connections

The user inputs an identical Passkey into both devices. Alternately, the Passkey may be generated and displayed on one device, and the user then inputs it into the other (step 2). This short shared key will be the basis of the mutual authentication of the devices. Steps 3 through 8 are repeated k times for a k-bit Passkey --e.g., k=20 for a 6-digit Passkey (999999=0xF423F).

In Steps 3-8, each side commits to each bit of the Passkey, using a long nonce (128 bits), and sending the hash of the nonce, the bit of the Passkey, and both

public keys to the other party. The parties then take turns revealing their commitments until the entire Passkey has been mutually disclosed. The first party to reveal a commitment for a given bit of the Passkey effectively reveals that bit of the Passkey in the process, but the other party then has to reveal the corresponding commitment to show the same bit value for that bit of the Passkey, or else the first party will then abort the protocol, after which no more bits of the Passkey are revealed.

This "gradual disclosure" prevents leakage of more than 1 bit of un-guessed Passkey information in the case of a MITM attack. A MITM attacker with only partial knowledge of the Passkey will only receive one incorrectly-guessed bit of the Passkey before the protocol fails. Hence, a MITM attacker who engages first one side, then the other will only gain an advantage of at most two bits over a simple brute-force guesser which succeeds with probability 0.000001.

The long nonce is included in the commitment hash to make it difficult to brute force even after the protocol has failed. The public Diffie-Hellman values are included to tie the Passkey protocol to the original ECDH key exchange, to prevent a MITM from substituting the attacker's public key on both sides of the ECDH exchange in standard MITM fashion.

At the end of this stage, Na is set to Na20 and Nb is set to Nb20 for use in Authentication Stage 2.

2.3.5.6.4 Authentication Stage 1 – Out of Band

The Out-of-Band protocol is used when authentication information has been received by at least one of the devices and indicated in the OOB data flag parameter included in the SMP Pairing Request and SMP Pairing Response PDU. The mode in which the discovery of the peer device is first done in-band and then followed by the transmission of authentication parameters through OOB interface is not supported. The sequence diagram for Authentication Stage 1 for Out of Band from the cryptographic point of view is shown in [Figure 2.5](#).

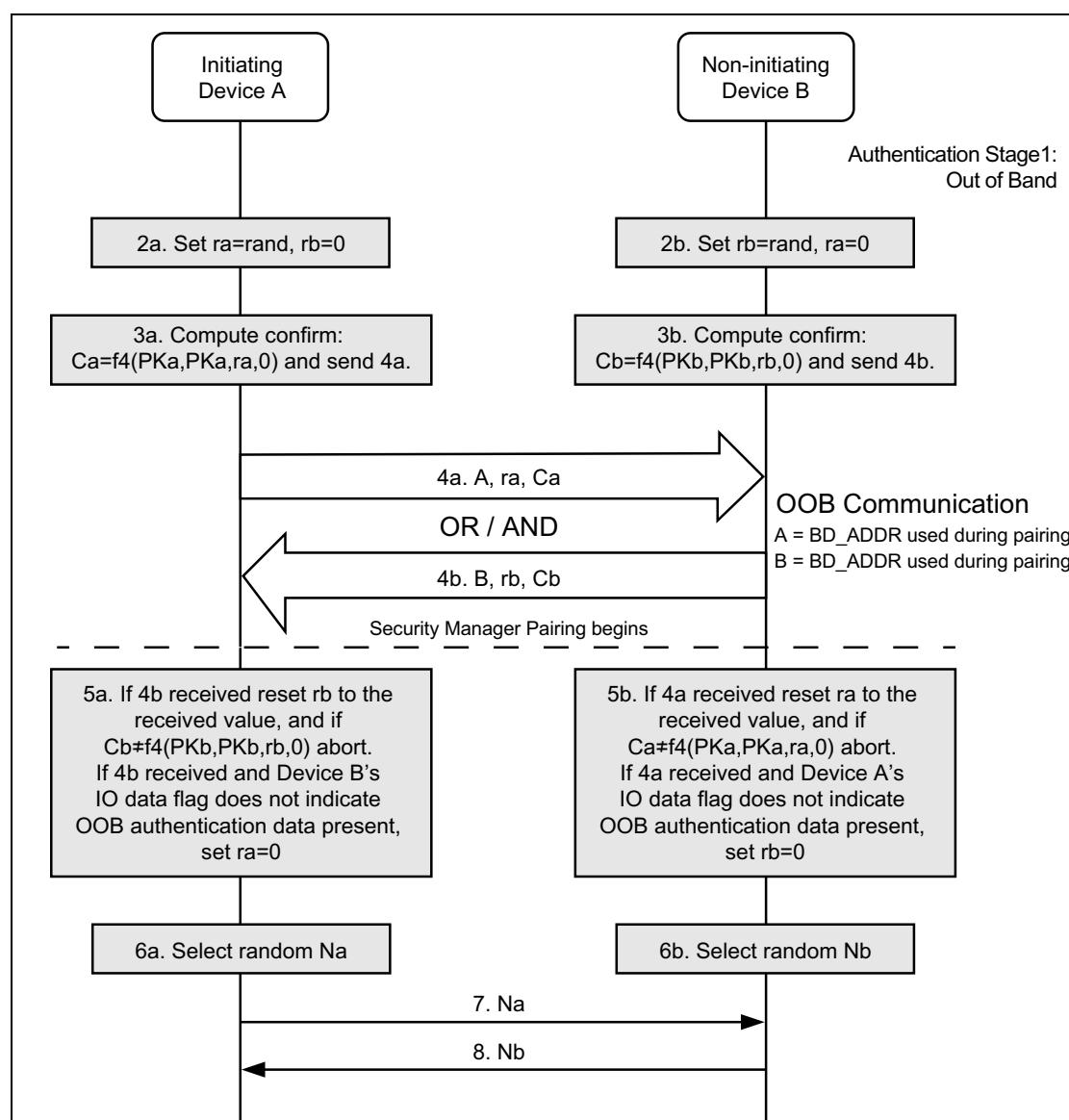


Figure 2.5: Authentication Stage 1: Out of Band, LE Secure Connections

Principle of operation. If both devices can transmit and/or receive data over an out-of-band channel, then mutual authentication will be based on the commitments of the public keys (Ca and Cb) exchanged OOB in Authentication stage 1. If OOB communication is possible only in one direction, then

authentication of the device receiving the OOB communication will be based on that device knowing a random number r sent via OOB. In this case, r must be secret: r can be created afresh every time, or access to the device sending r must be restricted. If r is not sent by a device, it is assumed to be 0 by the device receiving the OOB information in step 4a or 4b.

Roles of A and B. The OOB Authentication Stage 1 protocol is symmetric with respect to the roles of A and B. It does not require that device A always will initiate pairing and it automatically resolves asymmetry in the OOB communication.

Order of steps. The public key exchange must happen before the verification step 5. In the diagram the in-band public key exchange between the devices (step 1) is done before the OOB communication (step 4). But when the pairing is initiated by an OOB interface, public key exchange will happen after the OOB communication (step 1 will be between steps 4 and 5).

Values of ra and rb : Since the direction of the peer's OOB interface cannot be verified before the OOB communication takes place, a device should always generate and if possible transmit through its OOB interface a random number r to the peer. Each device applies the following rules locally to set the values of its own r and the value of the peer's r :

1. Initially, r of the device is set to a random number and r of the peer is set to 0 (step 2).
2. If a device has received OOB, it sets the peer's r value to what was sent by the peer (Step 5).
3. If the remote device's OOB data flag sent in the SMP Pairing Request or SMP Pairing Response is set to "OOB Authentication data not present", it sets its own r value to 0 (Step 5)

2.3.5.6.5 Authentication Stage 2 and Long Term Key Calculation

The second stage of authentication then confirms that both devices have successfully completed the exchange. This stage is identical in all three protocols.

Each device computes the MacKey and the LTK using the previously exchanged values and the newly derived shared key (step 9). Each device then computes a new confirmation value that includes the previously exchanged values and the newly derived MacKey (step 10a and 10b). The initiating device then transmits its confirmation value, which is checked by the responding device (step 11). If this check fails, it indicates that the initiating device has not confirmed the pairing, and the protocol must be aborted. The responding device then transmits its confirmation value, which is checked by the initiating device (step 12). A failure indicates that the responding device has not confirmed the pairing and the protocol should abort.

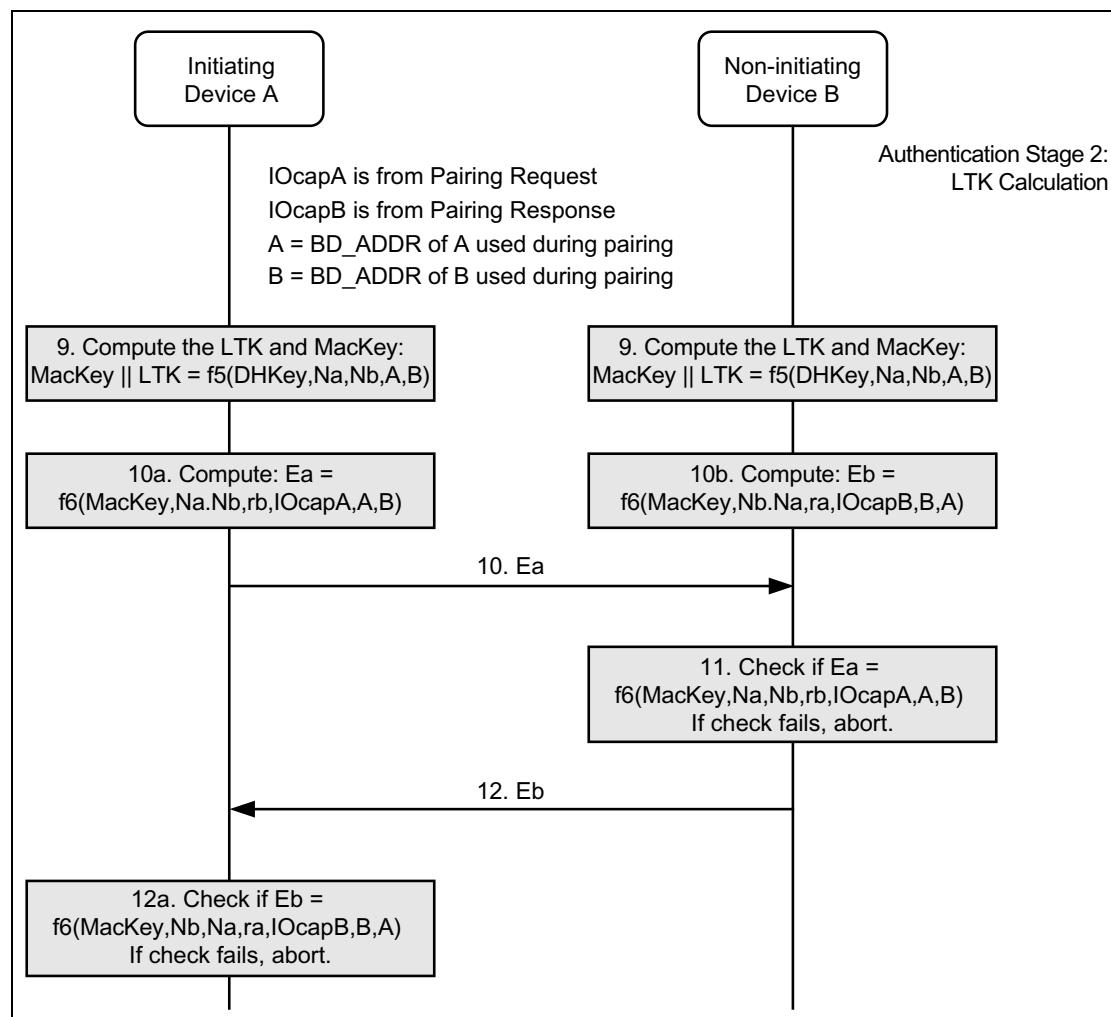


Figure 2.6: Authentication Stage 2 and Long Term Key Calculation

2.3.5.7 Cross-transport Key Derivation

When a pair of BR/EDR/LE devices support Secure Connections on a transport, the devices may optionally generate a key of identical strength for the other transport. There are two sequences:

- If Secure Connections pairing occurs first on the LE transport the procedures in [Section 2.4.2.4](#) may be used.
- If Secure connections pairing occurs first on the BR/EDR transport the procedures in [Section 2.4.2.5](#) may be used.

2.3.6 Repeated Attempts

When a pairing procedure fails a waiting interval shall pass before the verifier will initiate a new Pairing Request command or Security Request command to the same claimant, or before it will respond to a Pairing Request command or Security Request command initiated by a device claiming the same identity as the failed device. For each subsequent failure, the waiting interval shall be increased exponentially. That is, after each failure, the waiting interval before a new attempt can be made, could be for example, twice as long as the waiting interval prior to the previous attempt¹. The waiting interval should be limited to a maximum.

The maximum waiting interval depends on the implementation. The waiting time shall exponentially decrease to a minimum when no new failed attempts are made during a certain time period. This procedure prevents an intruder from repeating the pairing procedure with a large number of different keys.

To protect a device's private key, a device should implement a method to prevent an attacker from retrieving useful information about the device's private key using invalid public keys. For this purpose, a device can use one of the following methods:

- Change its private key after three failed attempts from any BD_ADDR and after 10 successful pairings from any BD_ADDR; or after a combination of these such that 3 successful pairings count as one failed pairing; or
- Verify that the received public keys from any BD_ADDR are on the correct curve; or
- Implement elliptic curve point addition and doubling using formulas that are valid only on the correct curve.

1. Another appropriate integer value larger than 1 may be used.

2.4 SECURITY IN BLUETOOTH LOW ENERGY

Security shall be initiated by the Security Manager in the device in the master role. The device in the slave role shall be the responding device. The slave device may request the master device to initiate pairing or other security procedures, see [Section 2.4.6](#).

The slave in the key distribution phase gives keys to the master so a reconnection can be encrypted, its random addresses can be resolved, or the master device can verify signed data from the slave.

The master may also provide keys to the slave device so a reconnection can be encrypted if the roles are reversed, the master's random addresses can be resolved, or the slave can verify signed data from the master.

2.4.1 Definition of Keys and Values

LE security uses the following keys and values for encryption, signing, and random addressing:

1. Identity Resolving Key (IRK) is a 128-bit key used to generate and resolve random addresses.
2. Connection Signature Resolving Key (CSRK) is a 128-bit key used to sign data and verify signatures on the receiving device.
3. Long Term Key (LTK) is a 128-bit key used to generate the contributory session key for an encrypted connection. Link Layer encryption is described in [\[Vol 6\] Part B, Section 5.1.3](#).
4. Encrypted Diversifier (EDIV) is a 16-bit stored value used to identify the LTK distributed during LE legacy pairing. A new EDIV is generated each time a unique LTK is distributed.
5. Random Number (Rand) is a 64-bit stored valued used to identify the LTK distributed during LE legacy pairing. A new Rand is generated each time a unique LTK is distributed.

2.4.2 Generation of Distributed Keys

Any method of generation of keys that are being distributed that results in the keys having 128 bits of entropy can be used, as the generation method is not visible outside the slave device (see [Section Appendix B](#)). The keys shall not be generated only from information that is distributed to the master device or only from information that is visible outside of the slave device.

2.4.2.1 Generation of IRK

The Identity Resolving Key (IRK) is used for resolvable private address construction (see [\[Part C\], Generic Access Profile, Section 10.8.2](#)). A master that has received IRK from a slave can resolve that slave's random device

addresses. A slave that has received IRK from a master can resolve that master's random device addresses. The privacy concept only protects against devices that are not part of the set to which the IRK has been given.

IRK can be assigned, or randomly generated by the device during manufacturing, or some other method could be used. If IRK is randomly generated then the requirements for random generation defined in [\[Vol 2\] Part H, Section 2](#) shall be used.

The encryption key size does not apply to IRK; therefore, its size does not need to be reduced before distribution.

2.4.2.2 Generation of CSRK

The Connection Signature Resolving Key (CSRK) is used to sign data in a connection. A device that has received CSRK can verify signatures generated by the distributing device. The signature only protects against devices that are not part of the set to which CSRK has been given.

CSRK can be assigned or randomly generated by the device during manufacturing, or some other method could be used. If CSRK is randomly generated then the requirements for random generation defined in [\[Vol 2\] Part H, Section 2](#) shall be used.

The encryption key size does not apply to CSRK, therefore its size does not need to be reduced before distribution.

2.4.2.3 LE Legacy Pairing - Generation of LTK, EDIV and Rand

Devices which support encryption in the Link Layer Connection State in the Slave Role shall be capable of generating LTK, EDIV, and Rand.

The EDIV and Rand are used by the slave device to establish a previously shared LTK in order to start an encrypted connection with a previously paired master device.

The generated LTK size must not exceed the negotiated encryption key size and its size may need to be reduced (see [Section 2.3.4](#)).

New values of LTK, EDIV, and Rand shall be generated each time they are distributed.

The slave device may store the mapping between EDIV, Rand and LTK in a security database so the correct LTK value is used when the master device requests encryption. Depending upon the LTK generation method additional information may be stored, for example the size of the distributed LTK.

The master device may also distribute EDIV, Rand, and LTK to the slave device which can be used to encrypt a reconnection if the device roles are reversed in a future connection.

2.4.2.4 Derivation of BR/EDR Link Key from LE LTK

The LTK from the LE physical transport can be converted to the BR/EDR link key for the BR/EDR transport as follows, using intermediate link key (ILK) as an intermediate value:

1. ILK = $h6(\text{LTK}, \text{"tmp1"})$
2. BR/EDR link key = $h6(\text{ILK}, \text{"lebr"})$

The string “lebr” is mapped into keyID using extended ASCII as follows:

```
keyID [0] = 0111 0010
keyID [1] = 0110 0010
keyID [2] = 0110 0101
keyID [3] = 0110 1100
keyID     = 0x6c656272
```

The string “tmp1” is mapped into keyID using extended ASCII as follows:

```
keyID [0] = 0011 0001
keyID [1] = 0111 0000
keyID [2] = 0110 1101
keyID [3] = 0111 0100
keyID     = 0x746D7031
```

2.4.2.5 Derivation of LE LTK from BR/EDR Link Key

The BR/EDR Link Key from the BR/EDR physical transport can be converted to the LTK for the LE transport as follows, using intermediate long term key (ILT) as an intermediate value:

1. ILT = $h6(\text{Link Key}, \text{"tmp2"})$
2. LTK = $h6(\text{ILT}, \text{"brle"})$

The string “brle” is mapped into keyID using extended ASCII as follows:

```
keyID [0] = 0110 0101
keyID [1] = 0110 1100
keyID [2] = 0111 0010
keyID [3] = 0110 0010
keyID     = 0x62726c65
```

The string “tmp2” is mapped into keyID using extended ASCII as follows:

```
keyID [0] = 0011 0010
keyID [1] = 0111 0000
keyID [2] = 0110 1101
keyID [3] = 0111 0100
keyID     = 0x746D7032
```

2.4.3 Distribution of Keys

Key distribution for LE Legacy Pairing and LE Secure Connections is described in the following sections.

2.4.3.1 LE Legacy Pairing Key Distribution

The slave may distribute to the master the following keys:

- LTK, EDIV, and Rand
- IRK
- CSRK

The master device may distribute to the slave the following keys:

- LTK, EDIV, and Rand
- IRK
- CSRK

The security properties of the distributed keys shall be set to the security properties of the STK that was used to distribute them. For example if STK has Unauthenticated no MITM Protection security properties then the distributed keys shall have Unauthenticated no MITM Protection security properties.

The link shall be encrypted or re-encrypted using STK generated in Phase 2 (see [Section 2.4.4.1](#)) before any keys are distributed.

Note: The distributed EDIV and Rand values are transmitted in clear text by the master device to the slave device during encrypted session setup.

The BD_ADDR that is received in the Identity Address Information command shall only be considered valid once a reconnection has occurred using the BD_ADDR and LTK distributed during that pairing. Once this is successful the BD_ADDR and the distributed keys shall be associated with that device in the security database.

A device may request encrypted session setup to use the LTK, EDIV, and Rand values distributed by the slave device when the key distribution phase has completed; however, this does not provide any additional security benefit. If an attacker has established the distributed LTK value then performing encrypted session setup to use the distributed values does not provide any protection against that attacker.

2.4.3.2 LE Secure Connections Key Distribution

The master and slave may distribute the following keys:

- IRK
- CSRK

The security properties of the distributed keys shall be set to the security properties of the LTK that was used to distribute them. For example if LTK has Unauthenticated no MITM Protection security properties then the distributed keys shall have Unauthenticated no MITM Protection security properties.

The link shall be encrypted or re-encrypted using LTK generated in Phase 2 (see [Section 2.4.4.1](#)) before any keys are distributed.

The BD_ADDR that is received in the Identity Address Information command shall only be considered valid once a reconnection has occurred using the BD_ADDR and LTK generated during that pairing. Once this is successful the BD_ADDR and the distributed keys shall be associated with that device in the security database.

2.4.4 Encrypted Session Setup

During the encryption session setup the master device sends a 16-bit Encrypted Diversifier value, *EDIV*, and a 64-bit Random Number, *Rand*, distributed by the slave device during pairing, to the slave device. The master's Host provides the Link Layer with the Long Term Key to use when setting up the encrypted session. The slave's Host receives the *EDIV* and *Rand* values and provides a Long Term Key to the slaves Link Layer to use when setting up the encrypted link.

When both devices support LE Secure Connections, the *EDIV* and *Rand* are set to zero.

2.4.4.1 Encryption Setup using STK

To distribute LTK and other keys in pairing Phase 3 an encrypted session needs to be established (see [Section 2.3.5.5](#)).

The encrypted session is setup using STK generated in Phase 2 (see [Section 2.3.5.5](#)) as the Long Term Key provided to the Link Layer, (see [\[Vol 6\] Part B, Section 5.1.3.1](#)) *EDIV*, and *Rand* values shall be set to zero.

If the link is already encrypted then the encryption pause procedure is performed using STK generated in Phase 2 as the Long Term Key provided to the Link Layer (see [\[Vol 6\] Part B, Section 5.1.3.2](#)). *EDIV* and *Rand* values shall be set to zero.

2.4.4.2 Encryption Setup using LTK

The master device must have the security information (*LTK*, *EDIV*, and *Rand*) distributed by the slave device in LE legacy pairing or the LTK generated in LE Secure Connections to setup an encrypted session.

The master initiates the encrypted session using the security information; see [\[Vol 6\] Part B, Section 5.1.3.1](#). If the link is already encrypted the encryption pause procedure is performed using the security information; see [\[Vol 6\] Part B, Section 5.1.3.2](#).

In LE legacy pairing, the *EDIV* and *Rand* values are used to establish *LTK* which is used as the Long Term Key on the slave device. If LTK cannot be established from EDIV and Rand values then the slave shall reject the request to encrypt the link and may optionally disconnect the link.

The LTK size must not exceed the negotiated encryption key size and its size may need to be reduced (see [Section 2.3.4](#)).

When the security information is stored, subsequent encryption setups may fail if the remote device has deleted the security information. [Table 2.9](#) defines what shall be done depending on the type of the security properties and whether or not bonding was performed when subsequent encryption setup fails.

Security Properties	Devices Bonded	Action to take when enabling encryption fails
Unauthenticated, no MITM protection	No	Depends on security policy of the device: • Option 1: Automatically initiate pairing • Option 2: Notify user and ask if pairing is ok. Option 1 is recommended.
Unauthenticated, no MITM protection	Yes	Notify user of security failure
Authenticated MITM protection	No	Depends on security policy of the device: • Option 1: Automatically initiate pairing • Option 2: Notify user and ask if pairing is ok. Option 2 is recommended.
Authenticated MITM protection	Yes	Notify user of security failure

Table 2.9: Action after encryption setup failure

2.4.5 Signing Algorithm

An LE device can send signed data without having to establish an encrypted session with a peer device. Data shall be signed using CSRK. A device performing signature verification must have received CSRK from the signing device. The sending device will use its CSRK to sign the transmitted data.

The following are inputs to the signing algorithm:

m is variable length

k is 128 bits

SignCounter is 32 bits

Signing shall be performed using the algorithm defined in the NIST Special Publication 800-38B (<http://csrc.nist.gov/publications/PubsSPs.html>) using AES-128 as the block cipher. NIST SP 800-38B defines the message authentication code (MAC) generation function:

$$\text{MAC} = \text{CMAC}(K, M, T\text{len})$$

The bit length of the MAC (*Tlen*) shall be 64 bits. The key used for signature generation (*k*) shall be set to CSRK.

The message to be signed (*M*) by the CMAC function is the concatenation of the variable length message to be signed (*m*) and 4 octet string representing the 32-bit counter value (*SignCounter*) least significant octet first.

$$M = m \parallel \text{SignCounter}$$

For example, if data to be signed is the 7 octet sequence ‘3456789ABCDEF1’ and *SignCounter* is set to 67653874 (0x040850F2) then *M* is the octet sequence ‘3456789ABCDEF1F2500804’. Examples of CMAC generation using AES-128 as the block cipher are included in NIST Special Publication 800-38B Appendix.

The *SignCounter* shall be initialized to zero when CSRK is generated and incremented for every message that is signed with a given CSRK.

Note: If a device generates 100,000 signed events a day, a 32-bit counter will wrap after approximately 117 years.

The 64-bit result of the CMAC function is used as the result of the signing algorithm.

To verify a signature a device computes the MAC of a received message and *SignCounter* and compares it with the received MAC. If the MAC does not match then the signature verification has failed. If the MACs match then the signature verification has succeeded.

The device performing verification should store the last verified *SignCounter* in the security database and compare it with a received *SignCounter* to prevent replay attacks. If the received *SignCounter* is greater than the stored value then the message has not been seen by the local device before and the security database can be updated.

2.4.6 Slave Security Request

The slave device may request security by transmitting a Security Request command to the master. When a master device receives a Security Request command it may encrypt the link, initiate the pairing procedure, or reject the request.

The slave shall not send the Security Request command if the pairing procedure is in progress, or if the encryption procedure is in progress.

The Security Request command includes the required security properties. A security property of MITM protection required shall only be set if the slave's IO capabilities would allow the Passkey Entry association model to be used or out of band authentication data is available.

The master shall ignore the slave's Security Request if the master has sent a Pairing Request without receiving a Pairing Response from the slave or if the master has initiated encryption mode setup.

If pairing or encryption mode is not supported or cannot be initiated at the time when the slave's Security Request Command is received, then the master shall respond with a Pairing Failed Command with the reason set to "Pairing Not Supported."

After receiving a Security Request, the master shall first check whether it has the required security information to enable encryption; see [Section 2.4.4.2](#). If this information is missing or does not meet the security properties requested by the slave, then the master shall initiate the pairing procedure. If the pairing procedure is successful, the master's security database is updated with the keys and security properties are distributed during the pairing procedure.

If the master has the required security information to enable encryption and it meets the security properties request by the slave, it shall perform encryption setup using LTK, see [Section 2.4.4.2](#).

[Figure 2.7](#) shows a summary of the actions and decisions that a master shall take when receiving a Security Request.

The slave shall check that any Pairing Request command received from the master after sending a Security Request command contains Authentication Requirements that meet the requested security properties.

If the slave requests a security property that is not Just Works and receives an encryption procedure request after sending a Security Request command then it shall check that any existing Security Information is of sufficient security properties.

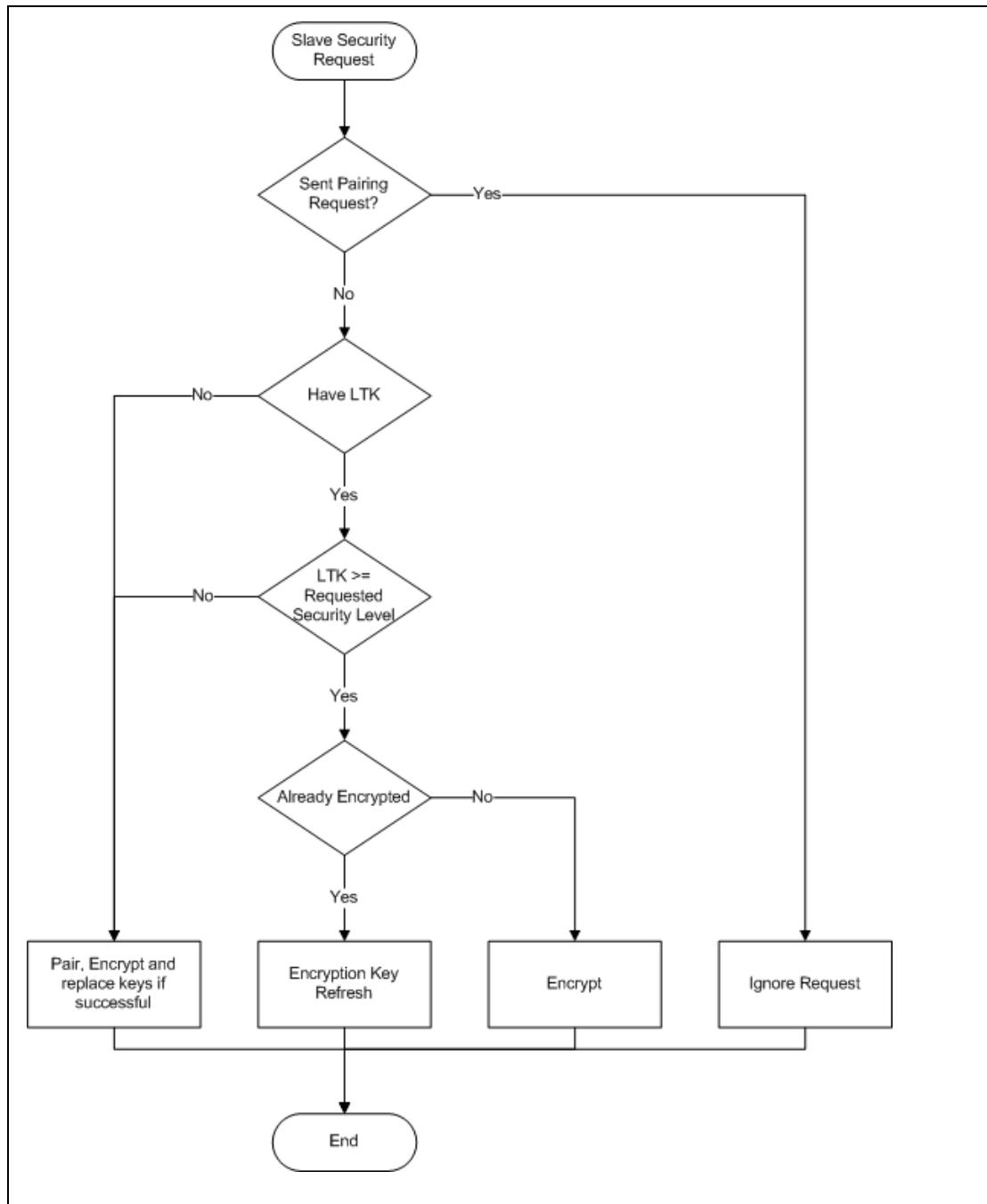


Figure 2.7: Master actions after receiving Security Request

3 SECURITY MANAGER PROTOCOL

3.1 INTRODUCTION

The Security Manager Protocol (SMP) is used for pairing and transport specific key distribution.

3.2 SECURITY MANAGER CHANNEL OVER L2CAP

All SMP commands are sent over the Security Manager Channel which is an L2CAP fixed channel (see [\[Vol 3\] Part A, Section 2.1](#)). The configuration parameters for the Security Manager Channel when LE Secure Connections is not supported shall be as shown below in [Table 3.1](#).

Parameter	Value
MTU	23
Flush Timeout	0xFFFF (Infinite)
QoS	Best Effort
Mode	Basic Mode

Table 3.1: Security Manager Channel Configuration Parameters without LE Secure Connections

The configuration parameters for the Security Manager Channel when LE Secure Connections is supported shall be as shown below in [Table 3.2](#).

Parameter	Value
MTU	65
Flush Timeout	0xFFFF (Infinite)
QoS	Best Effort
Mode	Basic Mode

Table 3.2: Security Manager Channel Configuration Parameters with LE Secure Connections

3.3 COMMAND FORMAT

The general format for all SMP commands is shown in [Figure 3.1](#).

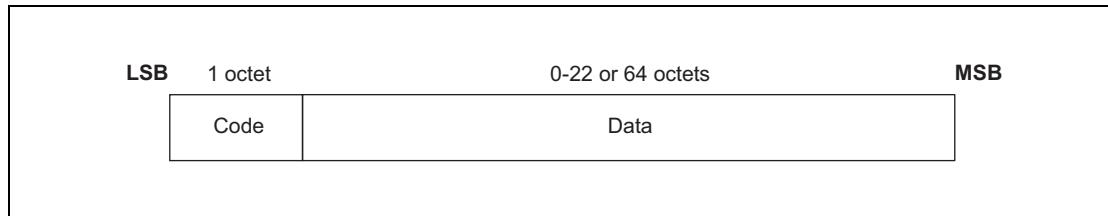


Figure 3.1: SMP Command Format

The following are the fields shown:

- **Code (1 octet)**

The Code field is one octet long and identifies the type of command. [Table 3.3](#) lists the codes defined by this document. If a packet is received with a reserved Code it shall be ignored.

Code	Description	Logical Link Supported
0x00	Reserved	
0x01	Pairing Request	LE-U, ACL-U
0x02	Pairing Response	LE-U, ACL-U
0x03	Pairing Confirm	LE-U
0x04	Pairing Random	LE-U
0x05	Pairing Failed	LE-U, ACL-U
0x06	Encryption Information	LE-U
0x07	Master Identification	LE-U
0x08	Identity Information	LE-U, ACL-U
0x09	Identity Address Information	LE-U, ACL-U
0x0A	Signing Information	LE-U, ACL-U
0x0B	Security Request	LE-U
0x0C	Pairing Public Key	LE-U
0x0D	Pairing DHKey Check	LE-U
0x0E	Pairing Keypress Notification	LE-U
0x0F – 0xFF	Reserved	

Table 3.3: SMP Command Codes

- *Data (0 or more octets)*

The Data field is variable in length. The Code field determines the format of the Data field.

If a device does not support pairing then it shall respond with a Pairing Failed command with the reason set to “Pairing Not Supported” (see [Section 3.5.5](#)) when any command is received. If pairing is supported then all commands shall be supported.

3.4 SMP TIMEOUT

To protect the Security Manager protocol from stalling, a Security Manager Timer is used. Upon transmission of the Pairing Request command or reception of the Pairing Request command, the Security Manager Timer shall be reset and started.

The Security Manager Timer shall be reset when an L2CAP SMP command is queued for transmission.

When SMP completes, the Security Manager Timer shall be stopped.

If the Security Manager Timer reaches 30 seconds, the procedure shall be considered to have failed, and the local higher layer shall be notified. No further SMP commands shall be sent over the L2CAP Security Manager Channel. A new SM procedure shall only be performed when a new physical link has been established.

3.5 PAIRING METHODS

The SMP commands defined in this section are used to perform Pairing Feature Exchange and key generation (see [Section 2.1](#)).

3.5.1 Pairing Request

The initiator starts the Pairing Feature Exchange by sending a Pairing Request command to the responding device. The Pairing Request command is defined in [Figure 3.2](#).

The rules for handing a collision between a pairing procedure on the LE transport and a pairing procedure on the BR/EDR transport are defined in [\[Vol 3\] Part C, Section 14.2](#).

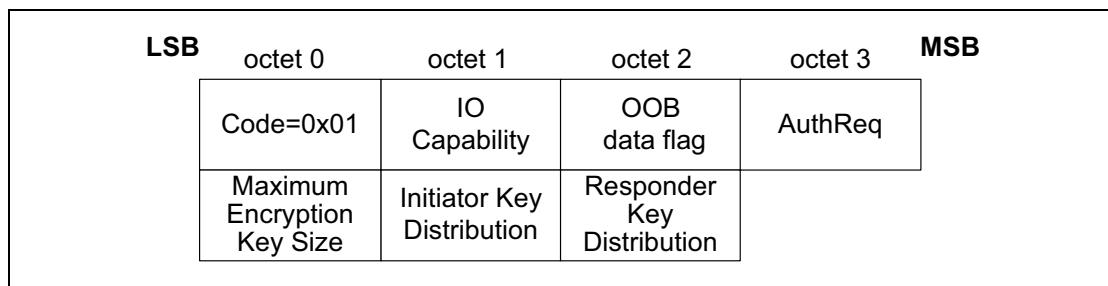


Figure 3.2: Pairing Request Packet

The following data fields are used:

- *IO Capability (1 octet)*

[Table 3.4](#) defines the values which are used when exchanging IO capabilities (see [Section 2.3.2](#)).

Value	Description
0x00	DisplayOnly
0x01	DisplayYesNo
0x02	KeyboardOnly
0x03	NoInputNoOutput
0x04	KeyboardDisplay
0x05-0xFF	Reserved

Table 3.4: IO Capability Values

- *OOB data flag (1 octet)*

[Table 3.5](#) defines the values which are used when indicating whether OOB authentication data is available (see [Section 2.3.3](#)).

Value	Description
0x00	OOB Authentication data not present
0x01	OOB Authentication data from remote device present
0x02-0xFF	Reserved

Table 3.5: OOB Data Present Values

- *AuthReq (1 octet)*

The AuthReq field is a bit field that indicates the requested security properties (see [Section 2.3.1](#)) for the STK and LTK and GAP bonding information (see [Part C, Section 9.4](#)).

[Figure 3.3](#) defines the authentication requirements bit field.

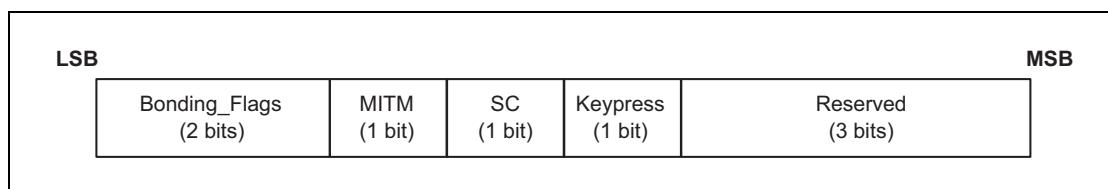


Figure 3.3: Authentication Requirements Flags

The Bonding_Flags field is a 2-bit field that indicates the type of bonding being requested by the initiating device as defined in [Table 3.6](#).

Bonding_Flags b_1b_0	Bonding Type
00	No Bonding
01	Bonding
10	Reserved
11	Reserved

Table 3.6: Bonding Flags

The MITM field is a 1-bit flag that is set to one if the device is requesting MITM protection, otherwise it shall be set to 0. A device sets the MITM flag to one to request an Authenticated security property for the STK when using LE legacy pairing and the LTK when using LE Secure Connections.

The SC field is a 1-bit flag that is set to one to request LE Secure Connection pairing, otherwise it shall be set to 0 based on the supported features of the initiator and responder, the possible resulting pairing mechanisms are: if both devices support LE Secure Connections, use LE Secure Connections; otherwise use LE legacy pairing.

The keypress field is a 1-bit flag that is used only in the Passkey Entry protocol and is ignored in other protocols. When both sides set that field to one, Keypress notifications shall be generated and sent using SMP Pairing Keypress Notification PDUs.

The reserved 3-bit field shall be set to zero and ignored upon reception.

- *Maximum Encryption Key Size (1 octet)*

This value defines the maximum encryption key size in octets that the device can support. The maximum key size shall be in the range 7 to 16 octets.

- *Initiator Key Distribution / Generation (1 octet)*

The Initiator Key Distribution / Generation field indicates which keys the initiator is requesting to distribute / generate or use during the Transport Specific Key Distribution phase (see [Section 2.4.3](#)). The Initiator Key Distribution / Generation field format and usage is defined in [Section 3.6.1](#).

- *Responder Key Distribution / Generation (1 octet)*

The Responder Key Distribution / Generation field indicates which keys the initiator is requesting the responder to distribute / generate or use during the Transport Specific Key Distribution phase (see [Section 2.4.3](#)). The Responder Key Distribution / Generation field format and usage is defined in [Section 3.6.1](#).

If Secure Connections pairing has been initiated over BR/EDR, the IO Capability, OOB data flag and Auth Req fields of the SM Pairing Request PDU shall be set to zero on transmission, and ignored on reception.

3.5.2 Pairing Response

This command is used by the responding device to complete the Pairing Feature Exchange after it has received a Pairing Request command from the initiating device, if the responding device allows pairing. The Pairing Response command is defined in [Figure 3.4](#).

The rules for handing a collision between a pairing procedure on the LE transport and a pairing procedure on the BR/EDR transport are defined in [\[Vol 3\] Part C, Section 14.2](#).

If a Pairing Request is received over the BR/EDR transport when either cross-transport key derivation/generation is not supported or the BR/EDR transport is not encrypted using a Link Key generated using P256, a Pairing Failed shall be sent with the error code "Cross-transport Key Derivation/Generation not allowed" (0x0E).

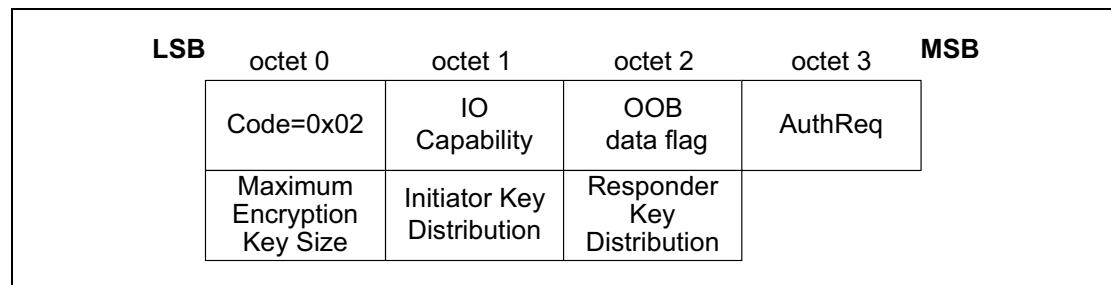


Figure 3.4: Pairing Response Packet

The following data fields are used:

- *IO Capability (1 octet)*
[Table 3.4](#) defines the values which are used when exchanging IO capabilities (see [Section 2.3.2](#)).
- *OOB data flag (1 octet)*
[Table 3.5](#) defines the values which are used when indicating whether OOB authentication data is available (see [Section 2.3.3](#)).
- *AuthReq (1 octet)*

The AuthReq field is a bit field that indicates the requested security properties (see [Section 2.3.1](#)) for the STK or LTK and GAP bonding information (see [Part C, Section 9.4](#)).

[Figure 3.3](#) defines the authentication requirements bit field.

The Bonding_Flags field is a 2-bit field that indicates the type of bonding being requested by the responding device as defined in [Table 3.6](#).

The MITM field is a 1-bit flag that is set to one if the device is requesting MITM protection, otherwise it shall be set to 0. A device sets the MITM flag to one to request an Authenticated security property for the STK when using LE legacy pairing and the LTK when using LE Secure Connections.

The SC field is a 1-bit flag that is set to one to request LE Secure Connection pairing, otherwise it shall be set to 0 based on the supported features of the initiator and responder, the possible resulting pairing mechanisms are: if both devices support LE Secure Connections, use LE Secure Connections; otherwise use LE legacy pairing.

The keypress field is a 1-bit flag that is used only in the Passkey Entry protocol and is ignored in other protocols. When both sides set that field to one, Keypress notifications shall be generated and sent using SMP Pairing Keypress Notification PDUs.

The reserved 3-bit field shall be set to zero and ignored upon reception.

- *Maximum Encryption Key Size (1 octet)*

This value defines the maximum encryption key size in octets that the device can support. The maximum key size shall be in the range 7 to 16 octets.

- *Initiator Key Distribution (1 octet)*

The Initiator Key Distribution field defines which keys the initiator shall distribute and use during the Transport Specific Key Distribution phase (see [Section 2.4.3](#)). The Initiator Key Distribution field format and usage are defined in [Section 3.6.1](#).

- *Responder Key Distribution (1 octet)*

The Responder Key Distribution field defines which keys the responder shall distribute and use during the Transport Specific Key Distribution phase (see [Section 2.4.3](#)). The Responder Key Distribution field format and usage are defined in [Section 3.6.1](#).

If Secure Connections pairing has been initiated over BR/EDR, the IO Capability, OOB data flag and Auth_Req fields of the SM Pairing Response PDU shall be set to zero on transmission, and ignored on reception.

3.5.3 Pairing Confirm

This is used following a successful Pairing Feature Exchange to start STK Generation for LE legacy pairing and LTK Generation for LE Secure Connections pairing. The Pairing Confirm command is defined in [Figure 3.5](#).

This command is used by both devices to send the confirm value to the peer device, see [Section 2.3.5.5](#) for LE legacy pairing and [Section 2.3.5.6](#) for LE Secure Connections pairing.

The initiating device starts key generation by sending the Pairing Confirm command to the responding device. If the initiating device wants to abort pairing it can transmit a Pairing Failed command instead.

The responding device sends the Pairing Confirm command after it has received a Pairing Confirm command from the initiating device.

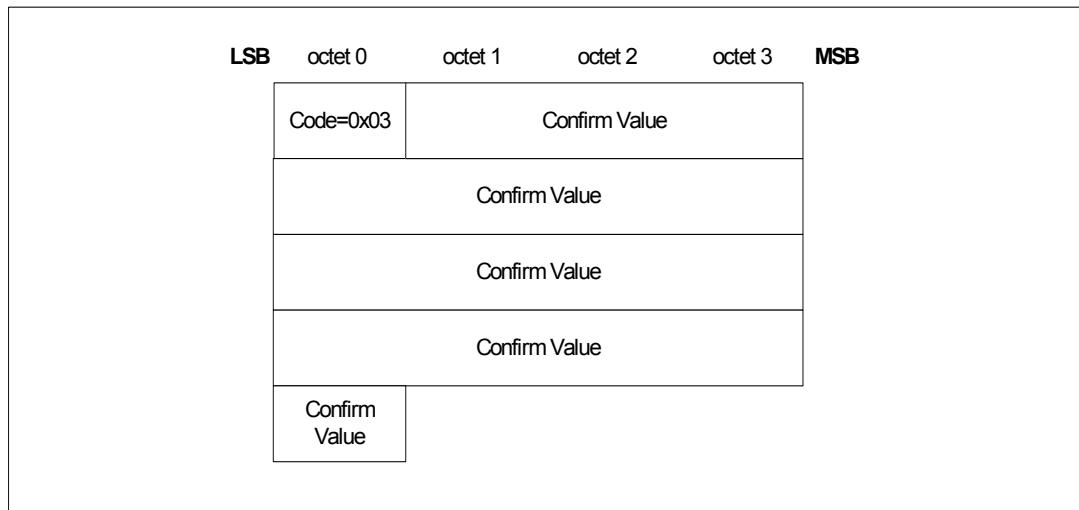


Figure 3.5: Pairing Confirm Packet

The following data field is used:

- *Confirm value (16 octets)*

In LE legacy pairing, the initiating device sends *Mconfirm* and the responding device sends *Sconfirm* as defined in [Section 2.3.5.5](#).

In LE Secure Connections, *Ca* and *Cb* are defined in [Section 2.2.6](#).

3.5.4 Pairing Random

This command is used by the initiating and responding device to send the random number used to calculate the Confirm value sent in the Pairing Confirm command. The Pairing Random command is defined in [Figure 3.6](#).

The initiating device sends a Pairing Random command after it has received a Pairing Confirm command from the responding device.

In LE legacy pairing, the responding device shall send a Pairing Random command after it has received a Pairing Random command from the initiating device if the Confirm value calculated on the responding device matches the Confirm value received from the initiating device. If the calculated Confirm

value does not match then the responding device shall respond with the Pairing Failed command.

In LE Secure Connections, the responding device shall send a Pairing Random command after it has received a Pairing Random command from the initiating device. If the calculated Confirm value does not match then the responding device shall respond with the Pairing Failed command.

The initiating device shall encrypt the link using the generated key (STK in LE legacy pairing or LTK in LE Secure Connections) if the Confirm value calculated on the initiating device matches the Confirm value received from the responding device. The successful encryption or re-encryption of the link is the signal to the responding device that key generation has completed successfully. If the calculated Confirm value does not match then the initiating device shall respond with the Pairing Failed command.

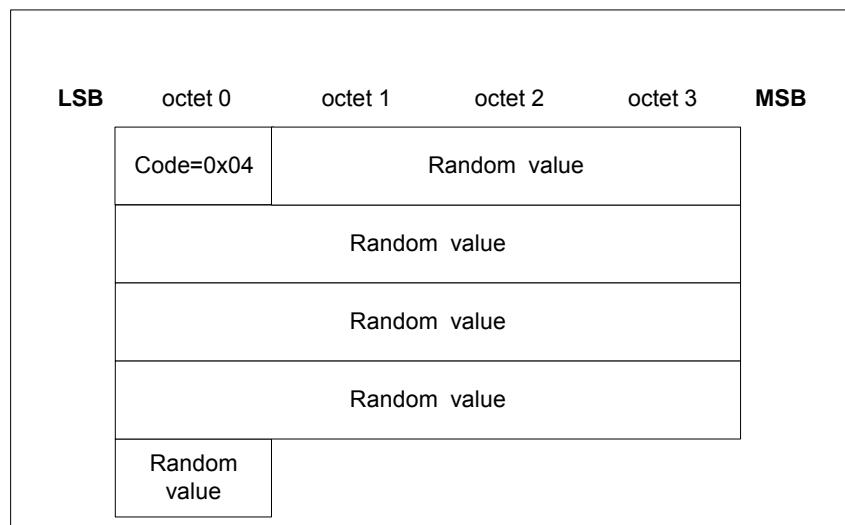


Figure 3.6: Pairing Random Packet

The following are the data fields:

- *Random value (16 octets)*

In LE legacy pairing, the initiating device sends *Mrand* and the responding device sends *Srand* as defined in [Section 2.3.5.5](#).

In LE Secure Connections, the initiating device sends *Na* and the responding device sends *Nb*.

3.5.5 Pairing Failed

This is used when there has been a failure during pairing and reports that the pairing procedure has been stopped and no further communication for the current pairing procedure is to occur. The Pairing Failed command is defined in [Figure 3.7](#).

Any subsequent pairing procedure shall restart from the Pairing Feature Exchange phase.

This command may be sent at any time during the pairing process by either device in response to a message from the remote device.

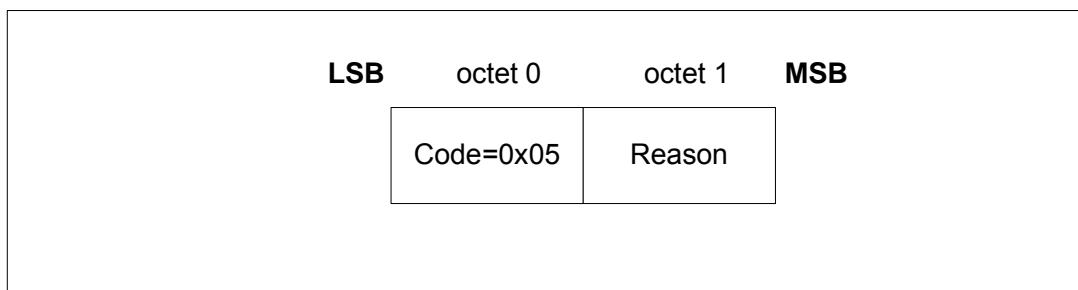


Figure 3.7: Pairing Failed Packet

The following data field is used:

- *Reason (1 octets)*

The Reason field indicates why the pairing failed. The reason codes are defined in [Table 3.7](#).

Value	Name	Description
0x00	Reserved	Reserved for future use
0x01	Passkey Entry Failed	The user input of passkey failed, for example, the user cancelled the operation
0x02	OOB Not Available	The OOB data is not available
0x03	Authentication Requirements	The pairing procedure cannot be performed as authentication requirements cannot be met due to IO capabilities of one or both devices
0x04	Confirm Value Failed	The confirm value does not match the calculated compare value
0x05	Pairing Not Supported	Pairing is not supported by the device
0x06	Encryption Key Size	The resultant encryption key size is insufficient for the security requirements of this device

Table 3.7: Pairing Failed Reason Codes

Value	Name	Description
0x07	Command Not Supported	The SMP command received is not supported on this device
0x08	Unspecified Reason	Pairing failed due to an unspecified reason
0x09	Repeated Attempts	Pairing or authentication procedure is disallowed because too little time has elapsed since last pairing request or security request
0x0A	Invalid Parameters	The Invalid Parameters error code indicates that the command length is invalid or that a parameter is outside of the specified range.
0x0B	DHKey Check Failed	Indicates to the remote device that the DHKey Check value received doesn't match the one calculated by the local device.
0x0C	Numeric Comparison Failed	Indicates that the confirm values in the numeric comparison protocol do not match.
0x0D	BR/EDR pairing in progress	Indicates that the pairing over the LE transport failed due to a Pairing Request sent over the BR/EDR transport in process.
0x0E	Cross-transport Key Derivation/Generation not allowed	Indicates that the BR/EDR Link Key generated on the BR/EDR transport cannot be used to derive and distribute keys for the LE transport.
0x0F - 0xFF	Reserved	Reserved for future use

Table 3.7: Pairing Failed Reason Codes

3.5.6 Pairing Public Key

This message is used to transfer the device's local public key (X and Y co-ordinates) to the remote device. This message is used by both the initiator and responder. This PDU is only used for Secure Connections.

LSB	Octet 0	Octet 1	Octet 2	Octet 3	MSB
	Code = 0x0C		Public Key X [0-2]		
			Public Key X [3-6]		
			Public Key X [7-10]		
			Public Key X [11-14]		
			Public Key X [15-18]		
			Public Key X [19-22]		
			Public Key X [23-26]		
			Public Key X [27-30]		
	Public Key X [31]		Public Key Y [0-2]		
			Public Key Y [3-6]		
			Public Key Y [7-10]		
			Public Key Y [11-14]		
			Public Key Y [15-18]		
			Public Key Y [19-22]		
			Public Key Y [23-26]		
			Public Key Y [27-30]		
	Public Key Y [31]				

Figure 3.8: Pairing Public Key PDU

3.5.7 Pairing DHKey Check

This message is used to transmit the 128-bit DHKey Check values (Ea/Eb) generated using f6. This message is used by both initiator and responder. This PDU is only used for LE Secure Connections.

LSB	Octet 0	Octet 1	Octet 2	Octet 3	MSB
	Code = 0x0D		DHKey Check (E) [0-2]		
			DHKey Check (E) [3-6]		
			DHKey Check (E) [7-10]		
			DHKey Check (E) [11-14]		
	DHKey check (E) [15]				

Figure 3.9: Pairing DHKey Check PDU

3.5.8 Keypress Notification

This message is used during the Passkey Entry protocol by a device with KeyboardOnly IO capabilities to inform the remote device when keys have been entered or erased.

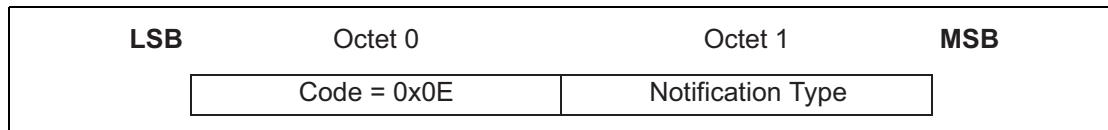


Figure 3.10: Pairing Keypress Notification PDU

Notification Type can take one of the following values:

Value	Parameter Description
0	Passkey entry started
1	Passkey digit entered
2	Passkey digit erased
3	Passkey cleared
4	Passkey entry completed
5-255	Reserved for future use

Table 3.8: Notification Type

3.6 SECURITY IN BLUETOOTH LOW ENERGY

3.6.1 Key Distribution and Generation

Bluetooth low energy devices can distribute keys from the slave to the master and from the master to the slave device. When using LE legacy pairing, the following keys may be distributed from the slave to the master:

- LTK using Encryption Information command
- EDIV and Rand using Master Identification command
- IRK using Identity Information command
- Public device or static random address using Identity Address Information command
- CSRK using Signing Information command

When using LE Secure Connections, the following keys may be distributed from the slave to the master:

- IRK using Identity Information command
- Public device or static random address using Identity Address Information command
- CSRK using Signing Information command

When using LE legacy pairing, the master may distribute to the slave the following key:

- LTK using Encryption Information command
- EDIV and Rand using Master Identification command
- IRK using Identity Information command
- Public device or static random address using Identity Address Information command
- CSRK using Signing Information command

When using LE Secure Connections, the master may distribute to the slave the following key:

- IRK using Identity Information command
- Public device or static random address using Identity Address Information command
- CSRK using Signing Information command

The keys which are to be distributed in the Transport Specific Key Distribution phase are indicated in the Key Distribution field of the Pairing Request and Pairing Response commands see [Section 3.5.1](#) and [Section 3.5.2](#).

The format of the Initiator Key Distribution / Generation field and Responder Key Distribution / Generation field in the Pairing Request and Pairing Response commands for LE is defined in [Figure 3.11](#).

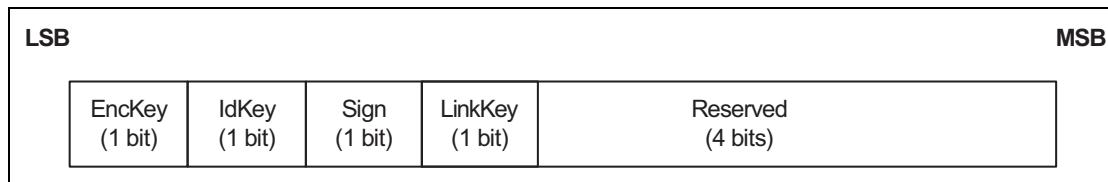


Figure 3.11: LE Key Distribution Format

The Key Distribution / Generation field has the following flags:

- In LE legacy pairing, EncKey is a 1-bit field that is set to one to indicate that the device shall distribute LTK using the Encryption Information command followed by EDIV and Rand using the Master Identification command.

When the SC bit is set to 1 by both devices and if SMP is running on the LE transport, then the EncKey field is ignored. EDIV and Rand shall be set to zero and shall not be distributed.

When SMP is running on the BR/EDR transport, the EncKey field is set to one to indicate that the device would like to derive the LTK from the BR/EDR Link Key. When EncKey is set to 1 by both devices in the initiator and responder Key Distribution / Generation fields, the procedures for calculating the LTK from the BR/EDR Link Key shall be used.

- IdKey is a 1-bit field that is set to one to indicate that the device shall distribute IRK using the Identity Information command followed by its public device or static random address using Identity Address Information.
- Sign is a 1-bit field that is set to one to indicate that the device shall distribute CSRK using the Signing Information command.
- LinkKey is a 1-bit field. When SMP is running on the LE transport, the LinkKey field is set to one to indicate that the device would like to derive the Link Key from the LTK. When LinkKey is set to 1 by both devices in the initiator and responder Key Distribution / Generation fields, the procedures for calculating the BR/EDR link key from the LTK shall be used. Devices not supporting LE Secure Connections shall set this bit to zero and ignore it on reception. When SMP is running on the BR/EDR transport, the LinkKey field shall be set to zero and ignored on reception.
- Reserved is a 4-bit field that shall be set to zero and ignored on reception.

The Initiator Key Distribution / Generation field in the Pairing Request command is used by the master to request which keys are distributed or generated by the initiator to the responder. The Responder Key Distribution / Generation field in the Pairing Request command is used by the master to request which keys are distributed or generated by the responder to the initiator. The Initiator Key Distribution / Generation field in the Pairing Response command from the slave defines the keys that shall be distributed or

generated by the initiator to the responder. The Responder Key Distribution / Generation field in the Pairing Response command from the slave defines the keys that shall be distributed or generated by the responder to the initiator. The slave shall not set to one any flag in the Initiator Key Distribution / Generation or Responder Key Distribution / Generation field of the Pairing Response command that the master has set to zero in the Initiator Key Distribution / Generation and Responder Key Distribution / Generation fields of the Pairing Request command.

When using LE legacy pairing, the keys shall be distributed in the following order:

1. LTK by the slave
2. EDIV and Rand by the slave
3. IRK by the slave
4. BD ADDR by the slave
5. CSRK by the slave
6. LTK by the master
7. EDIV and Rand by the master
8. IRK by the master
9. BD_ADDR by the master
10. CSRK by the master

When using LE Secure Connections, the keys shall be distributed in the following order:

1. IRK by the slave
2. BD ADDR by the slave
3. CSRK by the slave
4. IRK by the master
5. BD_ADDR by the master
6. CSRK by the master

If a key is not being distributed then the command to distribute that key shall not be sent.

Note: If a key is not distributed, then the capabilities that use this key will not be available. For example, if a LTK is not distributed from the slave to the master, then the master cannot encrypt a future link with that slave, therefore pairing would have to be performed again.

Note: The initiator should determine the keys needed based on the capabilities that are required by higher layer specifications. For example, if the initiator determines that encryption is required in a future link with that slave, then the initiator must request that slave's LTK is distributed by setting the EncKey bit to

one in the Responder Key Distribution / Generation field of the Pairing Request command.

If EncKey, IdKey, and Sign are set to zero in the Initiator Key Distribution / Generation and Responder Key Distribution / Generation fields, then no keys shall be distributed or generated and the link will be encrypted using the generated STK when using LE legacy pairing and LTK when using LE Secure Connections pairing.

Key distribution is complete in the device sending the final key when it receives the baseband acknowledgement for that key and is complete in the receiving device when it receives the final key being distributed.

3.6.2 Encryption Information

Encryption Information is used in the LE legacy pairing Transport Specific Key Distribution to distribute LTK that is used when encrypting future connections. The Encryption Information command is defined in [Figure 3.12](#).

The Encryption Information command shall only be sent when the link has been encrypted or re-encrypted using the generated STK.

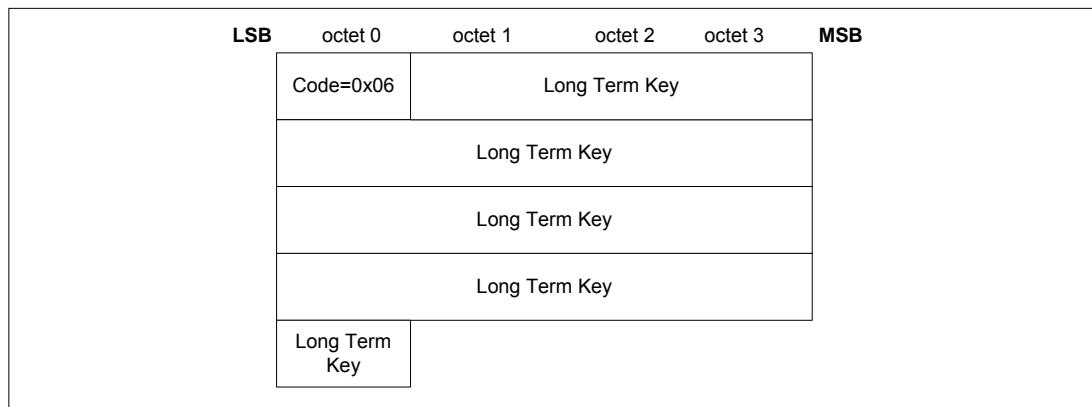


Figure 3.12: Encryption Information Packet

The following is the data field:

- *Long Term Key (16 octets)*

The generated LTK value being distributed, see [Section 2.4.2.3](#).

3.6.3 Master Identification

Master Identification is used in the LE legacy pairing Transport Specific Key Distribution phase to distribute EDIV and Rand which are used when encrypting future connections. The Master Identification command is defined in [Figure 3.13](#).

The Master Identification command shall only be sent when the link has been encrypted or re-encrypted using the generated STK.

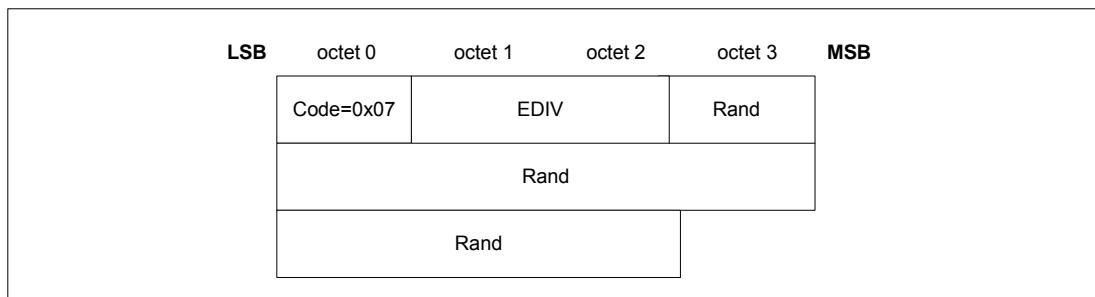


Figure 3.13: Master Identification Packet

The following data fields are used:

- *EDIV (2 octets)*
The EDIV value being distributed (see [Section 2.4.2.3](#)).
- *Rand (8 octets)*
64-bit Rand value being distributed (see [Section 2.4.2.3](#)).

3.6.4 Identity Information

Identity Information is used in the Transport Specific Key Distribution phase to distribute the IRK. The Identity Information command is defined in [Figure 3.14](#).

The Identity Information command shall only be sent when the link has been encrypted or re-encrypted using the generated key.

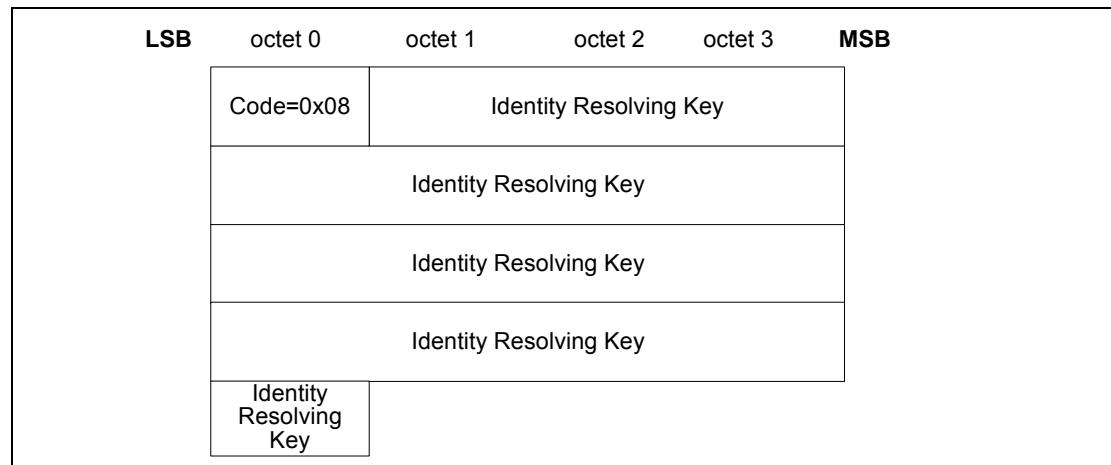


Figure 3.14: Identity Information Packet

The following are the data fields:

- *Identity Resolving Key (16 octets)*
128-bit IRK value being distributed (see [Section 2.4.2.1](#)).

Note: An all zero Identity Resolving Key data field indicates that a device does not have a valid resolvable private address.

3.6.5 Identity Address Information

Identity Address Information is used in the Transport Specific Key Distribution phase to distribute its public device address or static random address. The Identity Address Information command is defined in [Figure 3.15](#).

The Identity Address Information command shall only be sent when the link has been encrypted or re-encrypted using the generated key.

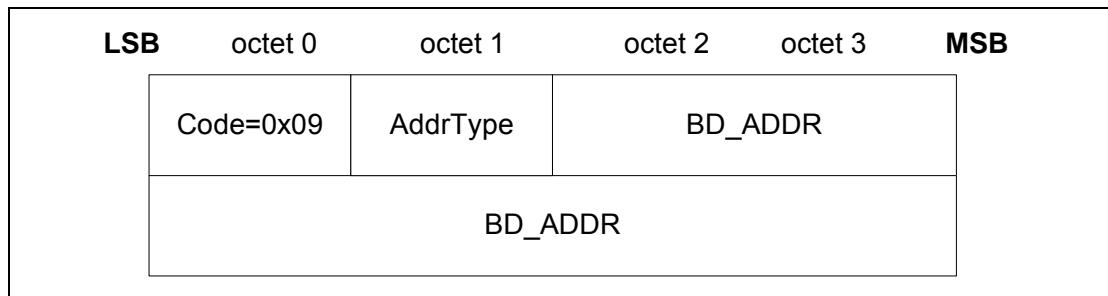


Figure 3.15: Identity Address Information Packet

The data fields are:

- *AddrType (1 octet)*
If BD_ADDR is a public device address, then AddrType shall be set to 0x00.
If BD_ADDR is a static random device address then AddrType shall be set to 0x01.
- *BD_ADDR (6 octets)*
This field is set to the distributing device's public device address or static random address.

3.6.6 Signing Information

Signing Information is used in the Transport Specific Key Distribution to distribute the CSRK which a device uses to sign data. The Signing Information command is defined in [Figure 3.16](#).

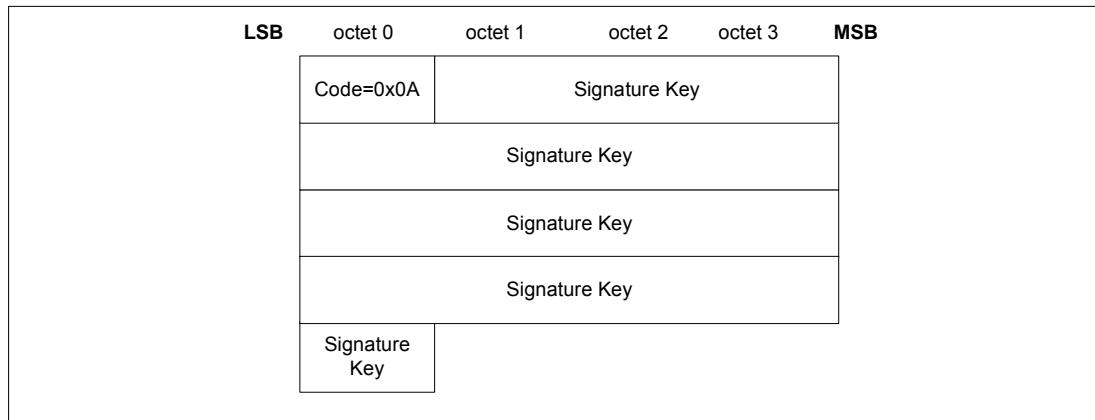


Figure 3.16: Signing Information Packet

The following data field is used:

- *Signature Key (16 octets)*
128-bit CSRK that is being distributed; see [Section 2.4.2.2](#).

3.6.7 Security Request

The Security Request command is used by the slave to request that the master initiates security with the requested security properties, see [Section 2.4.6](#). The Security Request command is defined in [Figure 3.17](#).

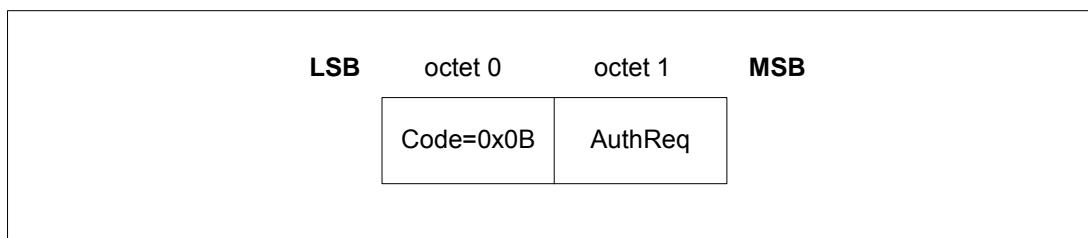


Figure 3.17: Security Request Packet

The following data field is used:

- *AuthReq (1 octet)*
The AuthReq field is a bit field that indicates the requested security properties (see [Section 2.3.1](#)) for the STK or LTK and GAP bonding information (see [Part C, Section 9.4](#)).

[Figure 3.3](#) defines the authentication requirements bit field.

The Bonding_Flags field is a 2-bit field that indicates the type of bonding being requested by the responding device as defined in [Table 3.6](#).

The MITM field is a 1-bit flag that is set to one if the device is requesting MITM protection, otherwise it shall be set to 0. A device sets the MITM flag to one to request an Authenticated security property for the STK when using LE legacy pairing and the LTK when using LE Secure Connections.

The SC field is a 1-bit flag that is set to one to request LE Secure Connection pairing, otherwise it shall be set to 0 based on the supported features of the initiator and responder, the possible resulting pairing mechanisms are: if both devices support LE Secure Connections, use LE Secure Connections; otherwise use LE legacy pairing.

The keypress field is a 1-bit flag that is used only in the Passkey Entry protocol and is ignored in other protocols. When both sides set that field to one, Keypress notifications shall be generated and sent using SMP Pairing Keypress Notification PDUs.

The reserved 3-bit field shall be set to zero and ignored upon reception.

APPENDIX A EDIV AND RAND GENERATION

EDIV and Rand are used by the responding device to identify an initiator and recover LTK. This section provides an example of how the distributed EDIV value is a masked version of the real value (DIV) which is used to recover LTK. Other methods can be used that provide equal or higher levels of confidentiality for DIV.

A.1 EDIV MASKING

The masking process uses a Diversifier Hiding Key (DHK) which is a 128-bit key that is never distributed.

DHK can be assigned, randomly generated by the device during manufacturing, part of a key hierarchy (see [Appendix B, Section B.2.3](#)) or some other method could be used, that results in DHK having 128 bits of entropy. If DHK is randomly generated then the requirements for random generation defined in [\[Vol 2\] Part H, Section 2](#) shall be used.

If DHK is changed then DIV values cannot be recovered from previously distributed EDIV values.

[Section A.1.1](#) defines a cryptographic function that is used by the responding device when generating EDIV and recovering DIV.

[Section A.1.2](#) describes how a responding device generates an EDIV value to be distributed to an initiating device and [Section A.1.3](#) describes how the responding device recovers DIV from a distributed EDIV value.

A.1.1 DIV Mask generation function *dm*

DIV is masked before distribution and unmasked during the encryption session setup using the output of the DIV mask generation function *dm*.

The following are inputs to the DIV mask generation function *dm*:

k is 128 bits

r is 64 bits

padding is 64 bits

r is concatenated with padding to generate *r'* which is used as the 128-bit input parameter *plaintextData* to security function *e*:

$$r' = \text{padding} \parallel r$$

The least significant octet of *r* becomes the least significant octet of *r'* and the most significant octet of *padding* becomes the most significant octet of *r'*.

For example, if the 64-bit value r is 0x123456789ABCDEF0 then r' is 0x0000000000000000123456789ABCDEF0.

The output of the DIV mask generation function dm is

$$dm(k, r) = e(k, r') \bmod 2^{16}$$

The output of the security function e is then truncated to 16 bits by taking the least significant 16 bits of the output of e as the result of dm .

A.1.2 EDIV Generation

The responding device generates a 64-bit random value, $Rand$. The $Rand$ value is used to generate 16-bit Y using the DIV mask generation function dm with the input parameter k set to DHK and the input parameter r set to $Rand$.

$$Y = dm(DHK, Rand)$$

The responding device then masks the DIV value to be distributed by bitwise XORing it with Y to generate EDIV.

$$EDIV = Y \text{ xor } DIV$$

EDIV and Rand are distributed to an initiating device during the transport specific key distribution phase using the Master Identification command.

A.1.3 DIV Recovery

When the responding device receives a request to encrypt a session it calculates Y using the DIV mask generation function dm with the input parameter k set to DHK and the input parameter r set to $Rand$. The Y value is bitwise XORed with $EDIV$ from the initiator to recover DIV.

$$DIV = Y \text{ xor } EDIV$$

The recovered DIV value can then be used to recover LTK which is used to enable encryption on the link.

APPENDIX B KEY MANAGEMENT

The security provided by different methods can vary and care should be taken to ensure that a chosen method is suitable for a device's requirements.

[Section B.1](#) uses a database for managing the keys. [Section B.2](#) uses a key hierarchy to manage the keys.

B.1 DATABASE LOOKUP

The LTK which is distributed is a 128-bit random number which is stored in a database, using EDIV as an index. There is no direct relationship between LTK and EDIV.

The requirements for random generation defined in [\[Vol 2\] Part H, Section 2](#) shall be used when generating LTK. This method provides an LTK with 128 bits of entropy.

CSRK, IRK, and other keys shall also be stored in the database. There is no relationship between the keys stored in the database or distributed LTKs, EDIVs, or Rands.

If the example EDIV and Rand generation method described in [Appendix A](#), [Section A.1](#) is used then the database shall be used to store DHK. DIV should be used as the index to recover LTK.

B.2 KEY HIERARCHY

A key hierarchy can be used to generate the keys.

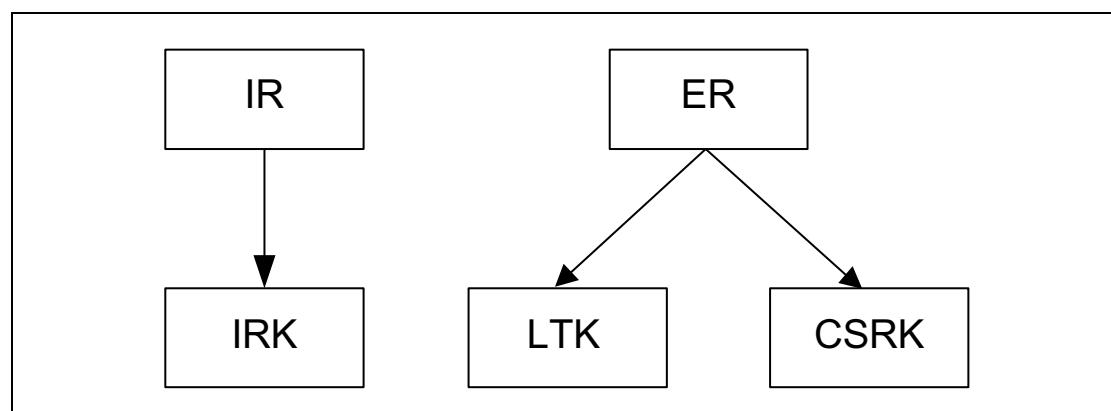


Figure B.1: Example Key Hierarchy

Figure B.1 is an example key hierarchy where LTK and CSRK are generated from a common ER key, and IRK is generated from a common IR key.

1. ER is a 128-bit key generated for each LE device that supports encrypted connections. It is used to generate LTK using EDIV; see [Section B.2.2](#).
2. IR is a 128-bit key generated for each LE device that supports encrypted connections, uses random addresses or signing data. IR is used to generate IRK and CSRK, see [Section B.2.3](#). It can also be used to generate DHK; see [Appendix A](#).

New LTK, EDIV, Rand, and CSRK values shall be generated each time they are distributed. If ER is changed then any previously distributed LTK or CSRK keys will no longer be valid.

The distributed IRK shall be the same for all devices it is distributed to. If IR is changed then any previously distributed IRK keys will no longer be valid.

The distributing device only needs to store IR and ER. LTK, IRK, and CSRK can be regenerated when they are required. This reduces the storage requirements on the distributing device.

[Appendix B.2.1](#) defines an example of the key diversifying function which can be used to generate LTK, IRK, CSRK, and other keys. Other implementations of this function can be used depending upon the exact security requirements of the device.

The NIST Special Publication 800-108 (<http://csrc.nist.gov/publications/PubsSPs.html>) defines key derivation functions which could be used instead of the example diversifying function *d1*.

B.2.1 Diversifying function *d1*

Diversified keys are generated with function *d1*. The diversifying function *d1* makes use of the security function *e*.

The following are inputs to diversifying function *d1*:

- k is 128 bits
- d is 16 bits
- r is 16 bits
- padding is 96 bits

d is concatenated with *r* and *padding* to generate *d'*, which is used as the 128-bit input parameter *plaintextData* to security function *e*:

$$d' = \text{padding} \parallel r \parallel d$$

The least significant octet of *d* becomes the least significant octet of *d'* and the most significant octet of *padding* becomes the most significant octet of *d'*.

For example, if the 16-bit value d is 0x1234 and the 16-bit value r is 0xabcd, then d' is 0x00000000000000000000abcd1234.

The output diversifying function $d1$ is:

$$d1(k, d, r) = e(k, d')$$

The 128-bit output of the security function e is used as the result of diversifying function $d1$.

B.2.2 Generating Keys from ER

ER is used to generate LTK and CSRK. ER can be assigned, randomly generated by the device during manufacturing or some other method could be used, that results in ER having 128 bits of entropy. If ER is randomly generated then the requirements for random generation defined in [\[Vol 2\] Part H, Section 2](#) shall be used.

The EDIV and Rand generation method described in [Appendix A](#) shall be used. LTK is the result of the diversifying function $d1$ with the ER as the input parameter k, the DIV as the input parameter d, and the value 0 as the input parameter r; see [Section B.2.1](#).

$$\text{LTK} = d1(\text{ER}, \text{DIV}, 0)$$

LTK can be recovered from ER and DIV by repeating the calculation when LTK is required.

CSRK is the result of the diversifying function $d1$ with the ER as the input parameter k, the DIV as the input parameter d, and the value 1 as the input parameter r; see [Section B.2.1](#).

$$\text{CSRK} = d1(\text{ER}, \text{DIV}, 1)$$

CSRK can be recovered from ER and DIV by repeating the calculation when CSRK is required.

This method provides an LTK and CSRK with limited amount of entropy because LTK and CSRK are directly related to EDIV and may not be as secure as other generation methods.

To reduce the probability of the same LTK or CSRK value being generated, the DIV values must be unique for each CSRK, LTK, EDIV, and Rand set that is distributed.

A method for preventing a malicious device from repeatedly pairing and collecting CSRK, LTK and DIV information, which could be used in a known plain text attack in ER, should be implemented.

B.2.3 Generating Keys from IR

IR can be used to generate IRK and other required keys. IR can be assigned, randomly generated by the device during manufacturing or some other method could be used, that results in IR having 128 bits of entropy. If IR is randomly generated then the requirements for random generation defined in [\[Vol 2\] Part H, Section 2](#) shall be used.

IRK is the result of the diversifying function $d1$ with IR as the input parameter k and the value 1 as the input parameter d and the value 0 as the input parameter r ; see [Section B.2.1](#).

$$\text{IRK} = d1(\text{IR}, 1, 0)$$

If the example EDIV and Rand generation method described in [Appendix A, Section A.1](#) is used then DHK can be the result of diversifying function $d1$ with IR as the input parameter k and the value 3 as the input parameter d and the value 0 as the input parameter r .

$$\text{DHK} = d1(\text{IR}, 3, 0)$$

Other keys can be generated by using different values for k as the input to the diversifying function $d1$. If the value of k is reused for a given IR then the resulting key will be the same.

APPENDIX C MESSAGE SEQUENCE CHARTS

This section is informative and illustrates only the most common scenarios; it does not cover all possible alternatives. Furthermore, the message sequence charts do not consider errors over the air interface or host interface.

A flow diagram of pairing is shown in [Figure C.1](#). The process has 4 steps. Step 2 has a number of different options.

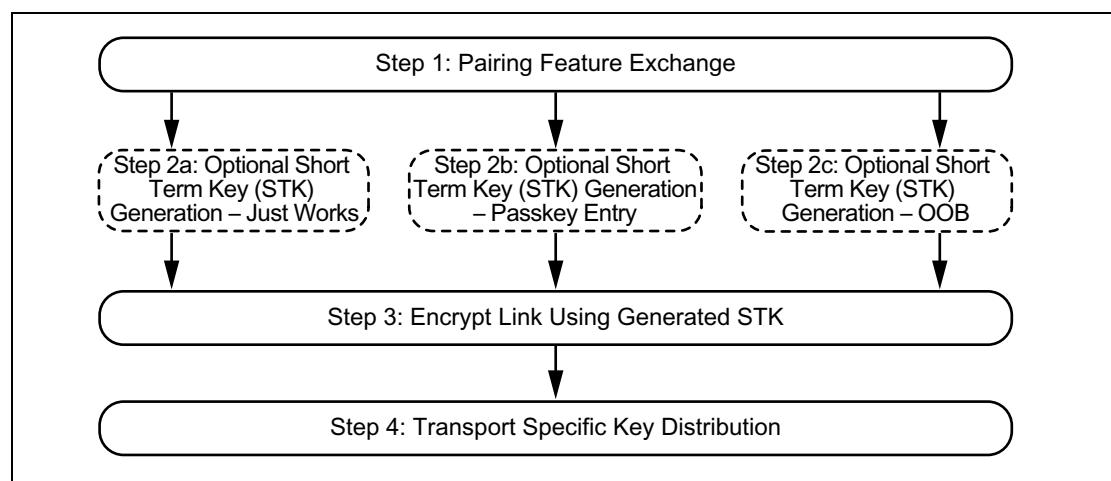


Figure C.1: Pairing Process Overview

Note: In all the MSCs, the Master device is also the Initiating Device and the Slave device is also the Responding (non-Initiating) Device.

C.1 PHASE 1: PAIRING FEATURE EXCHANGE

The master initiates the pairing procedure using Pairing Request command as shown in [Figure C.2](#).

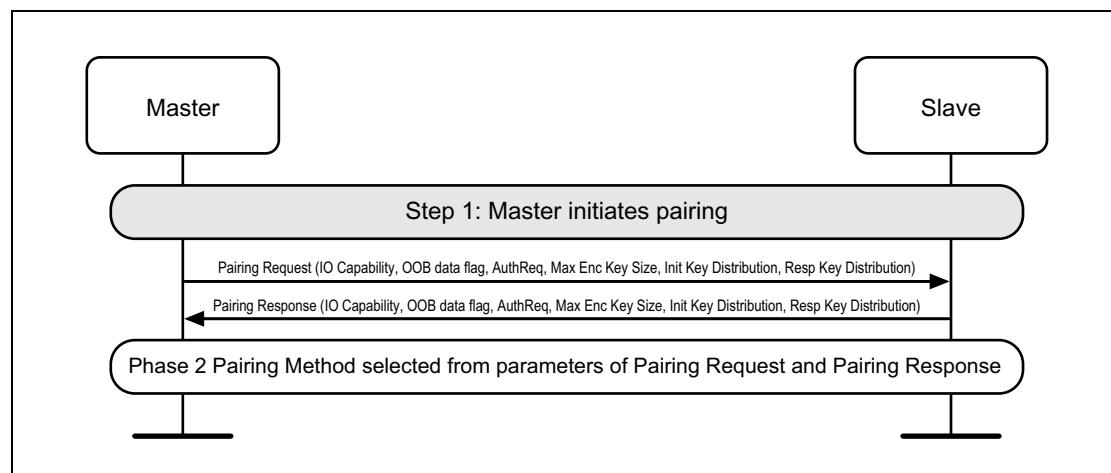


Figure C.2: Pairing initiated by master

C.1.1 Slave Security Request – Master Requests Pairing

The slave may request the master initiates security procedures. Figure C.3 shows an example where the slave requests security and the master initiates pairing in response.

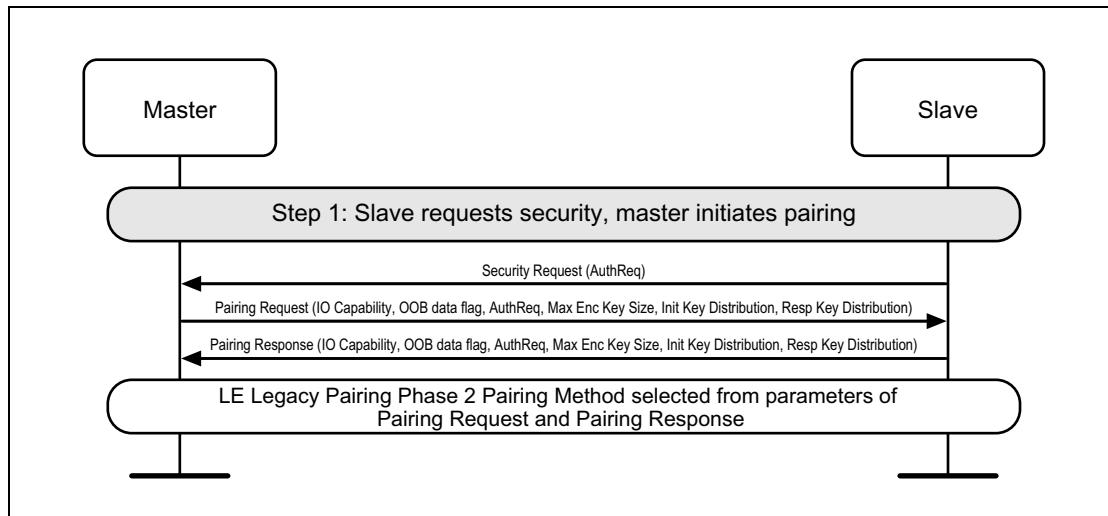


Figure C.3: Slave security request, master initiated pairing

C.2 PHASE 2: AUTHENTICATING AND ENCRYPTING

After Pairing Feature Exchange has completed a pairing method is selected one of the possible short term key generation sequences are used. This can be Just Works, Passkey Entry or Out of Band pairing method.

C.2.1 LE Legacy Pairing

The following sub-sections include message sequence charts for LE legacy pairing.

C.2.1.1 Legacy Phase 2: Short Term Key Generation – Just Works

After Pairing Feature Exchange has completed a pairing method is selected. [Figure C.4](#) shows the Just Works pairing method.

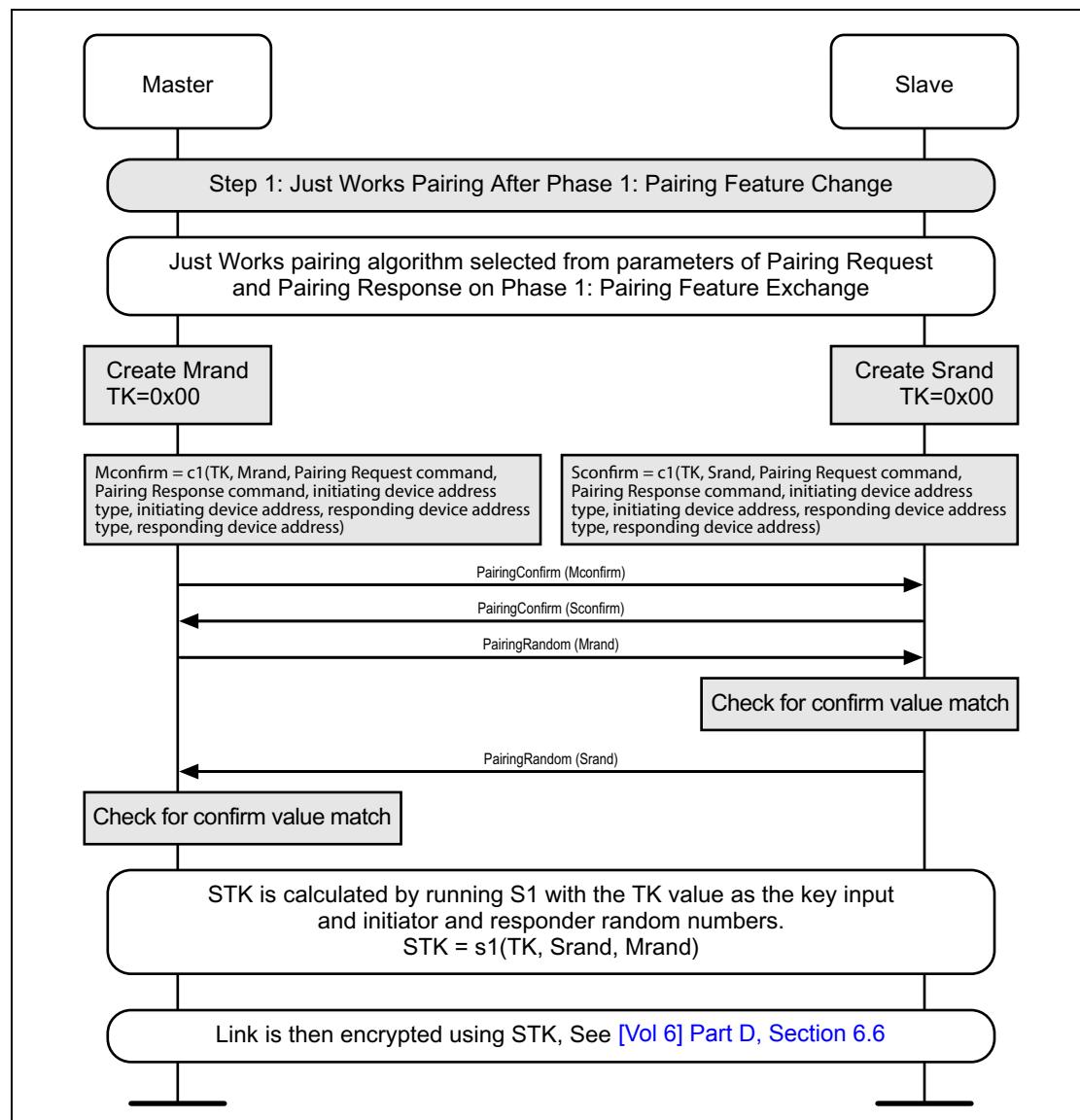


Figure C.4: Legacy Just Works Pairing Method

C.2.1.2 Legacy Phase 2: Short Term Key Generation – Passkey Entry

After Pairing Feature Exchange has completed, a pairing method is selected. Figure C.5 shows the Passkey Entry pairing method.

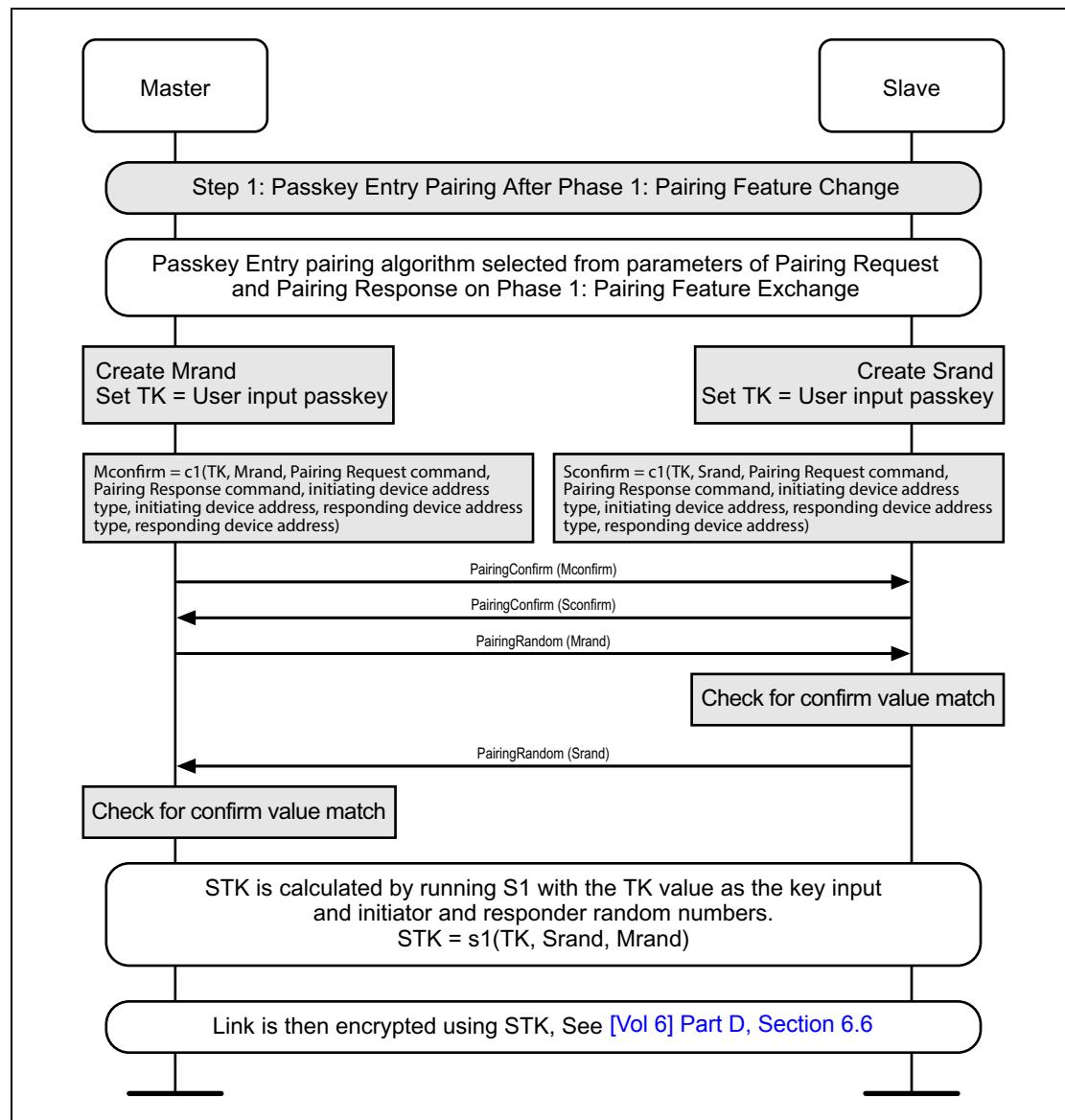


Figure C.5: Legacy Passkey Entry Pairing Method

C.2.1.3 Legacy Phase 2: Short Term Key Generation – Out of Band

After Pairing Feature Exchange has completed, a pairing method is selected. Figure C.6 shows the Out of Band pairing method.

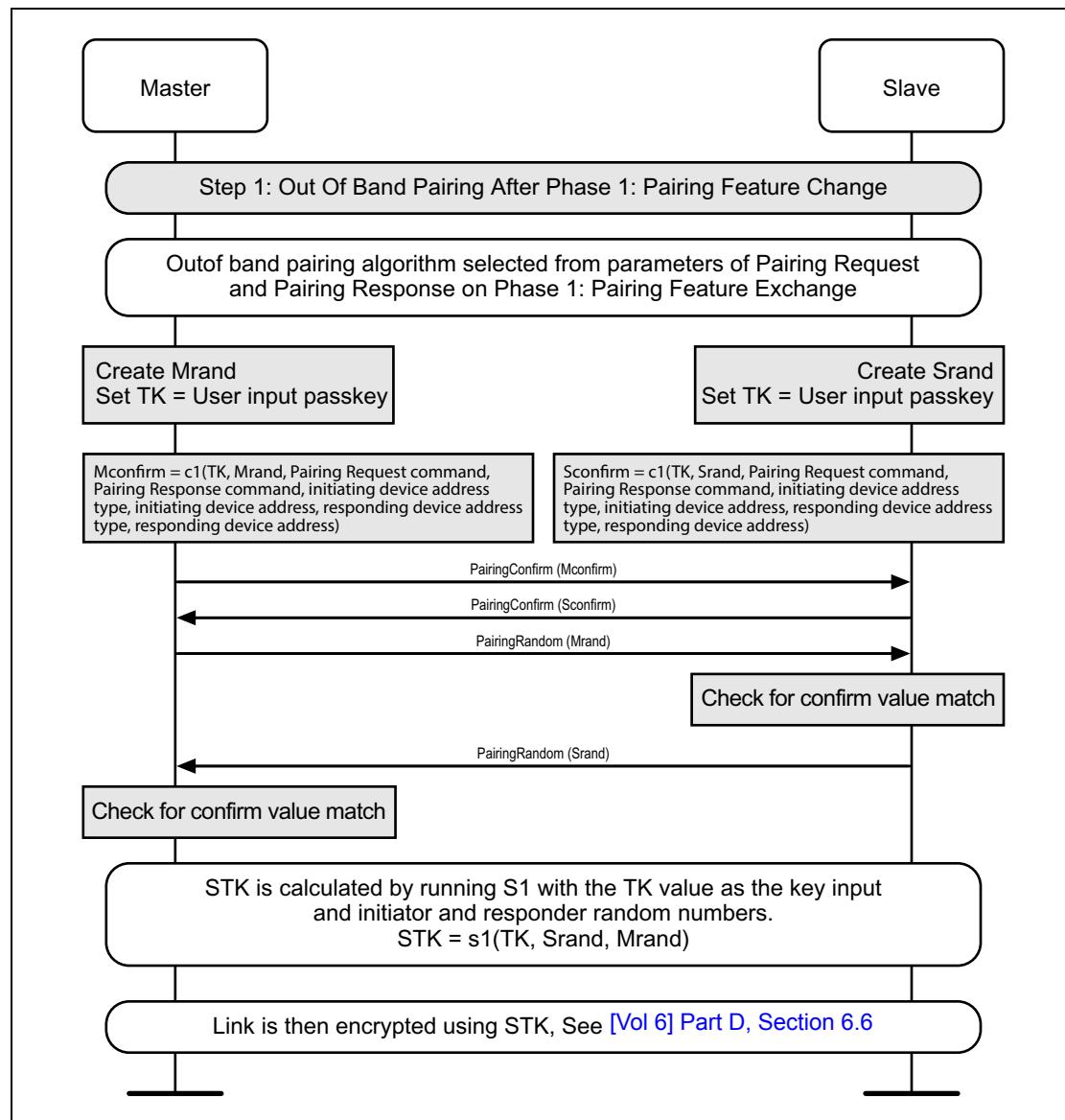


Figure C.6: Legacy OOB Pairing Method

C.2.2 LE Secure Connections

The following sub-sections include message sequence charts for LE Secure Connections.

C.2.2.1 Public Key Exchange

In Step 1a and 1b, the two devices exchange public keys. The master sends its public key to the slave followed by slave sending its public key to the master.

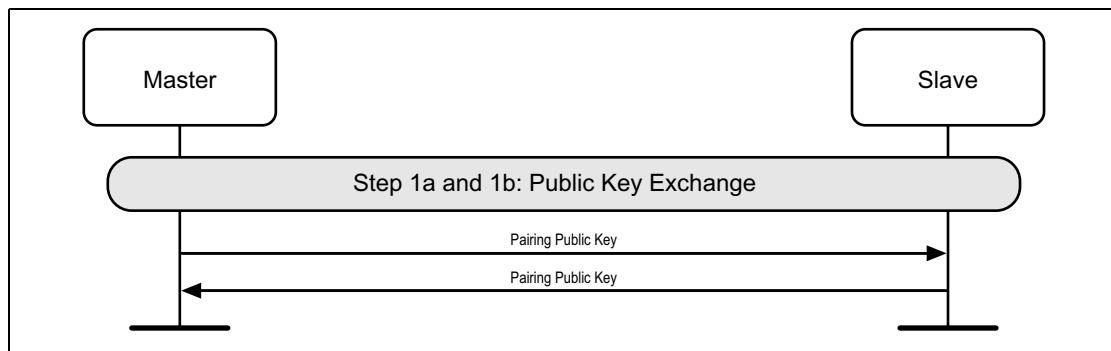


Figure C.7: Pairing Phase 2 - Public Key Exchange

C.2.2.2 Authentication Stage 1

The following sub-sections include message sequence charts for the success and failure cases in each association model.

C.2.2.2.1 Successful Numeric Comparison (or Just Works)

The numeric comparison step will be done when both devices have output capabilities, or if one of the devices has no input or output capabilities. If both devices have output capabilities, this step requires the displaying of a user confirmation value. This value should be displayed until the end of step 2. If one or both devices do not have output capabilities, the same protocol is used but the Hosts will skip the step asking for the user confirmation.

Note: The sequence for Just Works is identical to that of Numeric Comparison with the exception that the Host will not show the numbers to the user.

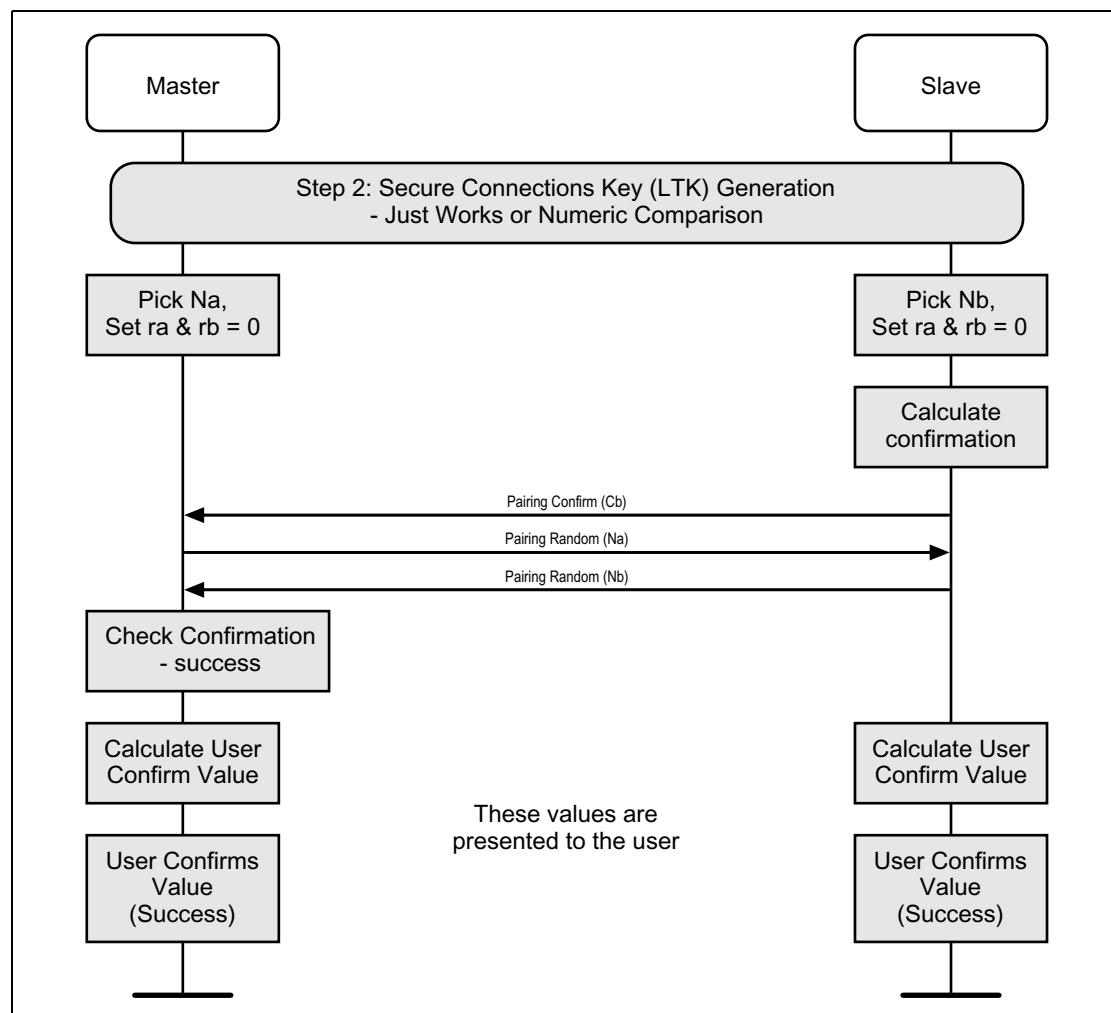


Figure C.8: Pairing Phase 2, Authentication Stage 1, Successful Numeric Comparison

C.2.2.2.2 Numeric Comparison – Confirm Check failure on the Initiator side

If the Confirm value calculated by the Initiator is not equal to the Commitment value received from the Responder, the Initiator will abort Pairing process by sending Pairing Failed with reason “Confirm Value Failed”.

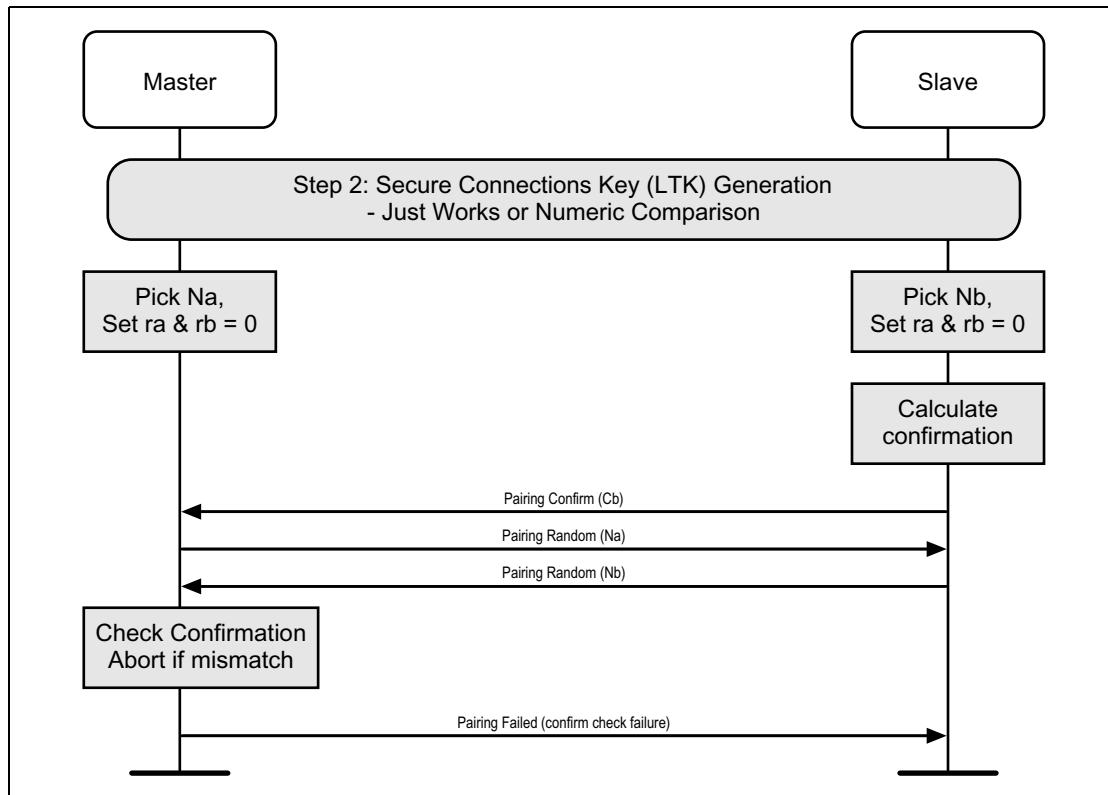


Figure C.9: Pairing Phase 2, Authentication Stage 1, Numeric Comparison – Confirm Check failure on Initiator side

C.2.2.2.3 Numeric Comparison Failure on Initiator Side

If the numeric comparison fails on the initiating side due to the user indicating that the confirmation values do not match, Pairing is terminated.

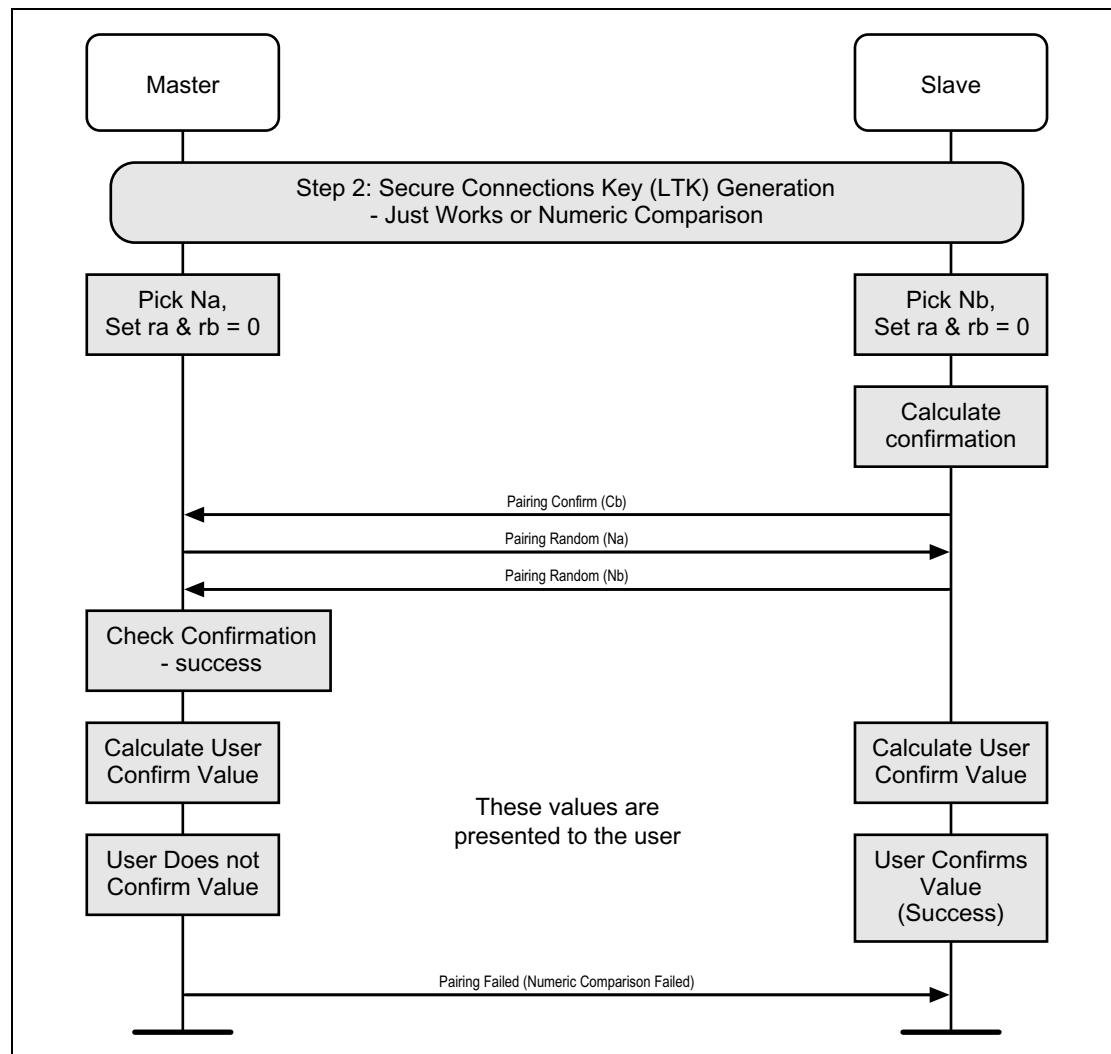


Figure C.10: Pairing Phase 2, Authentication Stage 1, Numeric Comparison Failure on Initiator Side

C.2.2.2.4 Numeric Comparison Failure on Responding Side

If the numeric comparison fails on the responding side due to the user indicating that the confirmation values do not match, Pairing is terminated.

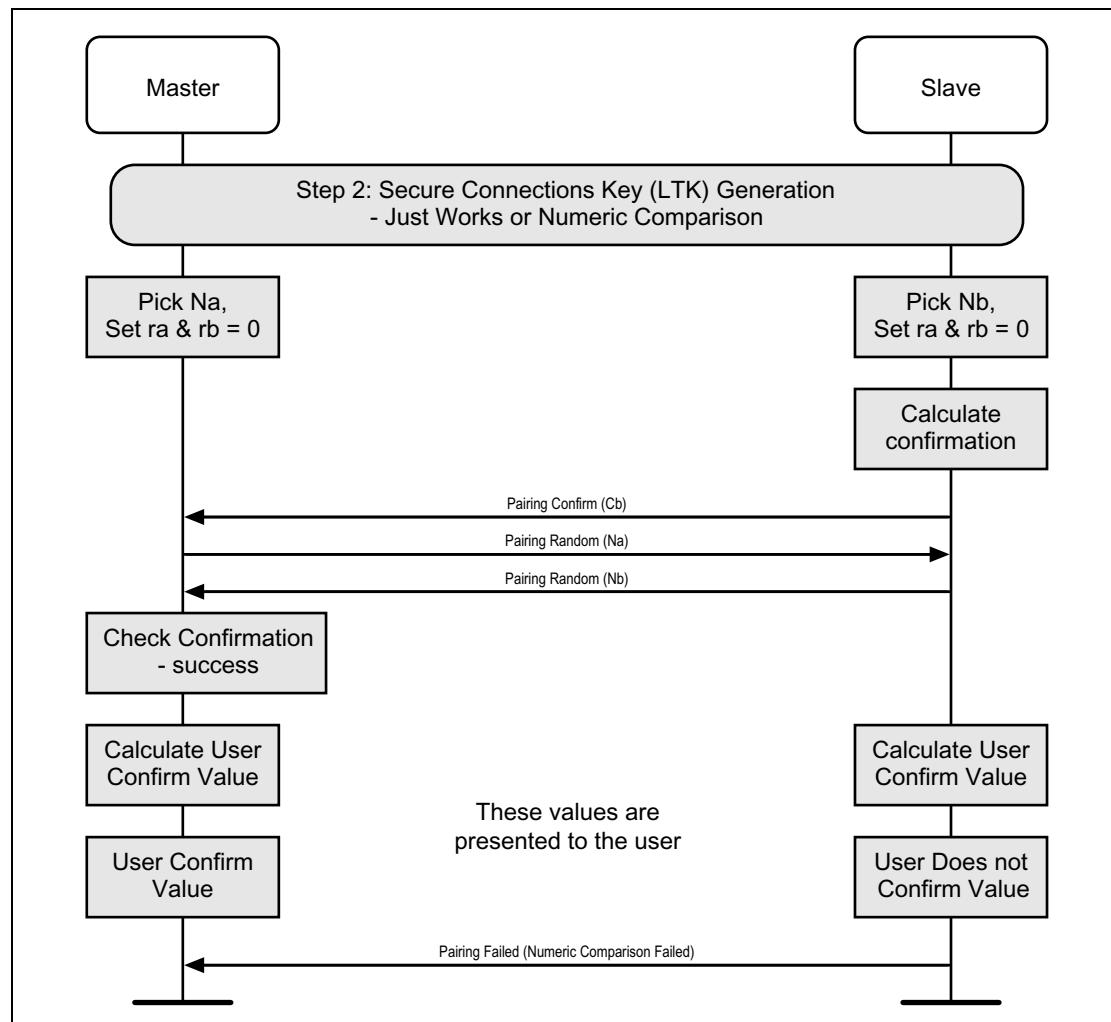


Figure C.11: Pairing Phase 2, Authentication Stage 1, Numeric Comparison Failure on Responding Side

C.2.2.2.5 Successful Passkey Entry

The Passkey Entry step is used in two cases: when one device has numeric input only and the other device has either a display or numeric input capability. In this step, one device displays a number to be entered by the other device or the user enters a number on both devices. Key press notification messages are shown during the user input phase.

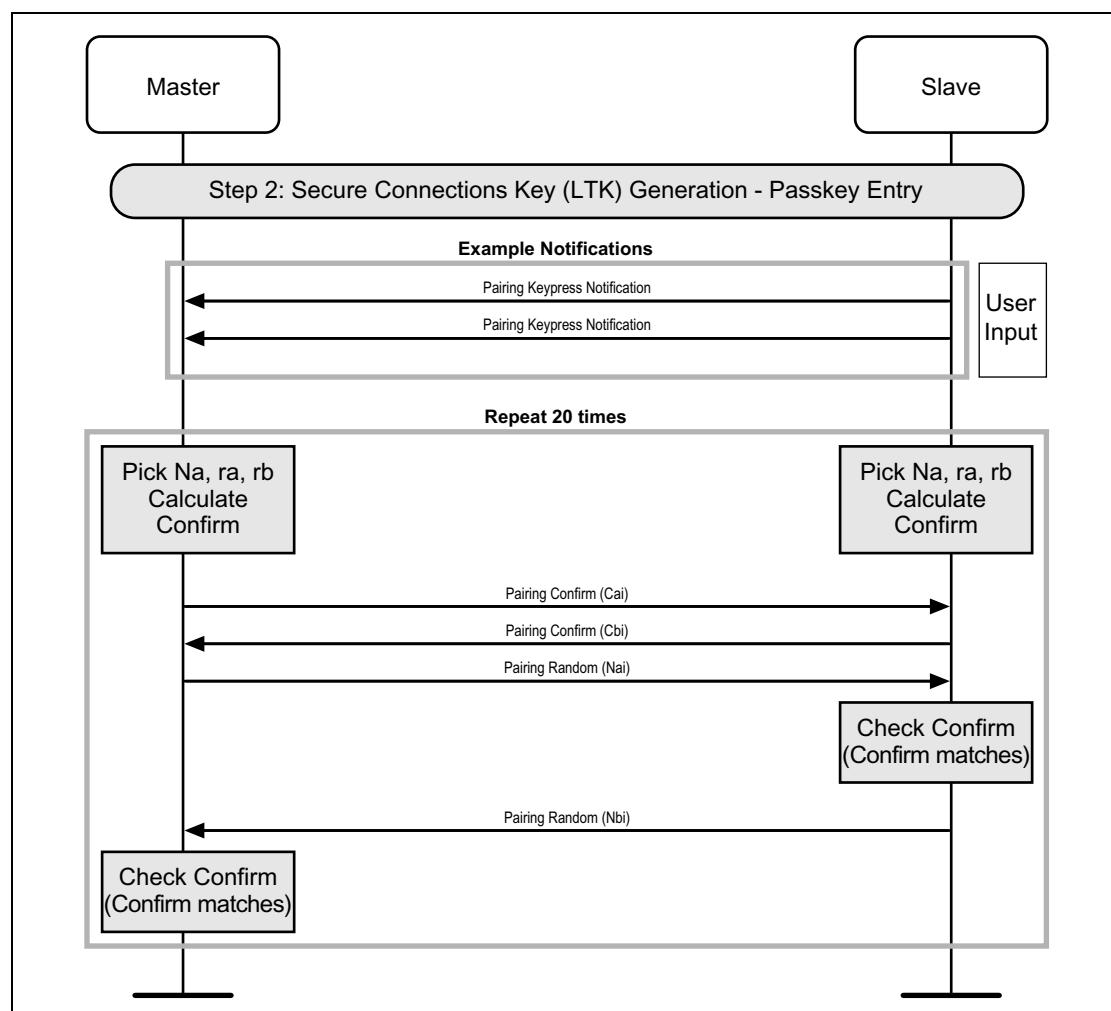


Figure C.12: Pairing Phase 2. Authentication Stage 1. Successful Passkey Entry

Note: Passkey Entry may prolong pairing experience because of the time required to execute 20 repetitions over SMP.

C.2.2.2.6 Passkey Entry – Confirm Check failure on the Responder side

If during one of the 20 repetitions, the Confirm calculated by the Responder is not equal to the one received from the Initiator, the Responder will abort the Pairing process by sending Pairing Failed with reason “Confirm Value Failed”.

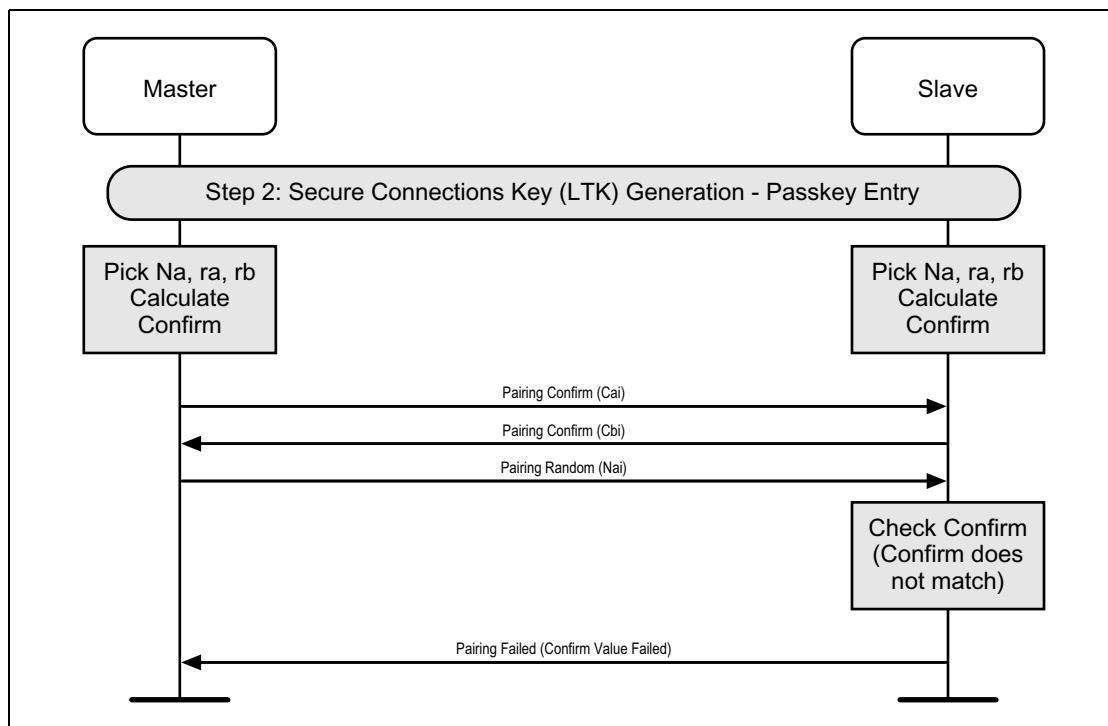


Figure C.13: Pairing Phase 2, Authentication Stage 1, Passkey Entry – Confirm Check failure on Responder side

C.2.2.2.7 Passkey Entry – Confirm Check failure on the Initiator side

If during one of the 20 repetitions, the Confirm calculated by the Initiator is not equal to the one received from the Responder, the Initiator will abort the Pairing process by sending Pairing Failed with reason “Confirm Value Failed”.

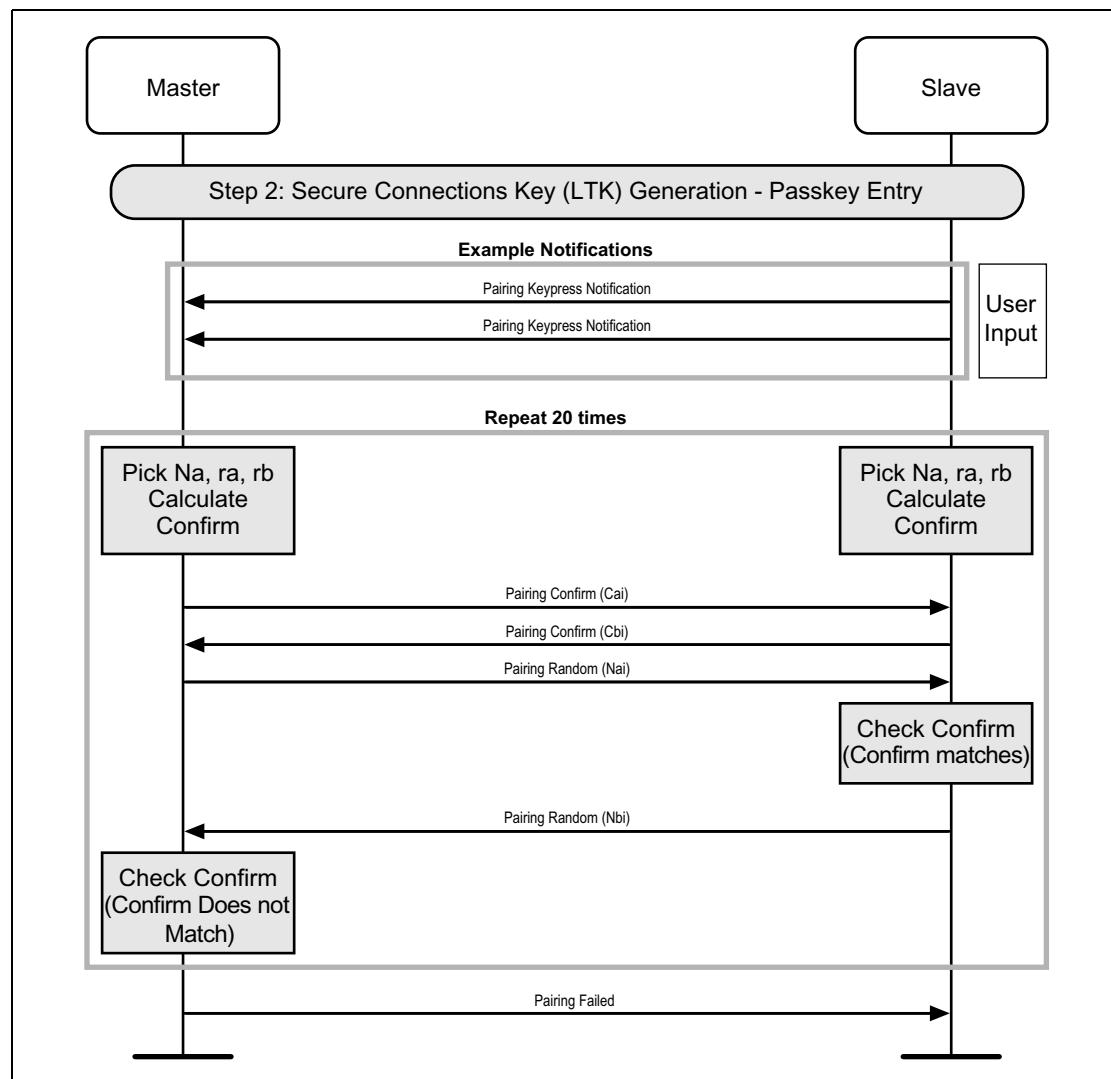


Figure C.14: Pairing Phase 2, Authentication Stage 1, Passkey Entry – Confirm Check failure on Initiator side

C.2.2.2.8 Passkey Entry Failure on Responding Side

If the passkey entry fails on the responding side, Pairing is terminated.

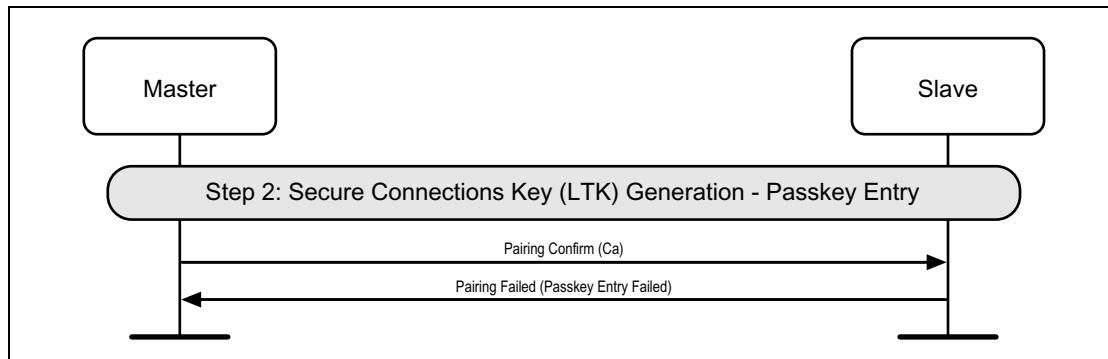


Figure C.15: Pairing Phase 2, Authentication Stage 1, Passkey Entry Failure on Responding Side

C.2.2.2.9 Passkey Entry Failure on Initiator Side

If the passkey entry fails on the initiating side, Pairing is terminated.

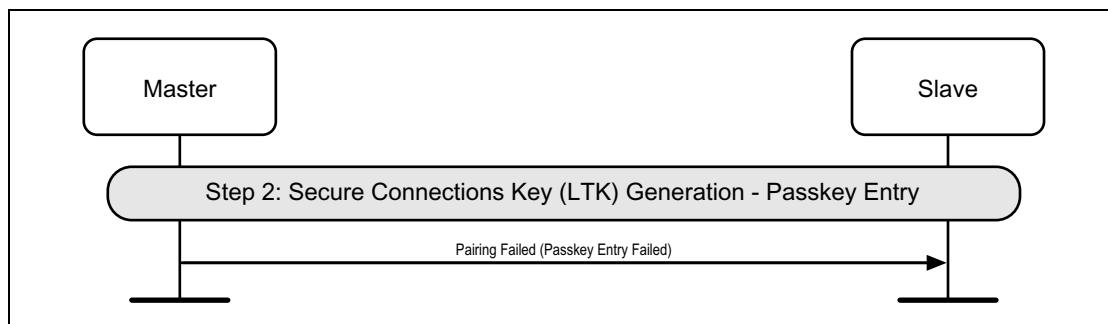


Figure C.16: Pairing Phase 2, Authentication Stage 1, Passkey Entry Failure on Initiator Side

C.2.2.2.10 Successful Out of Band

The OOB authentication will only be done when at least one device has some OOB information to use. This step requires no user interaction.

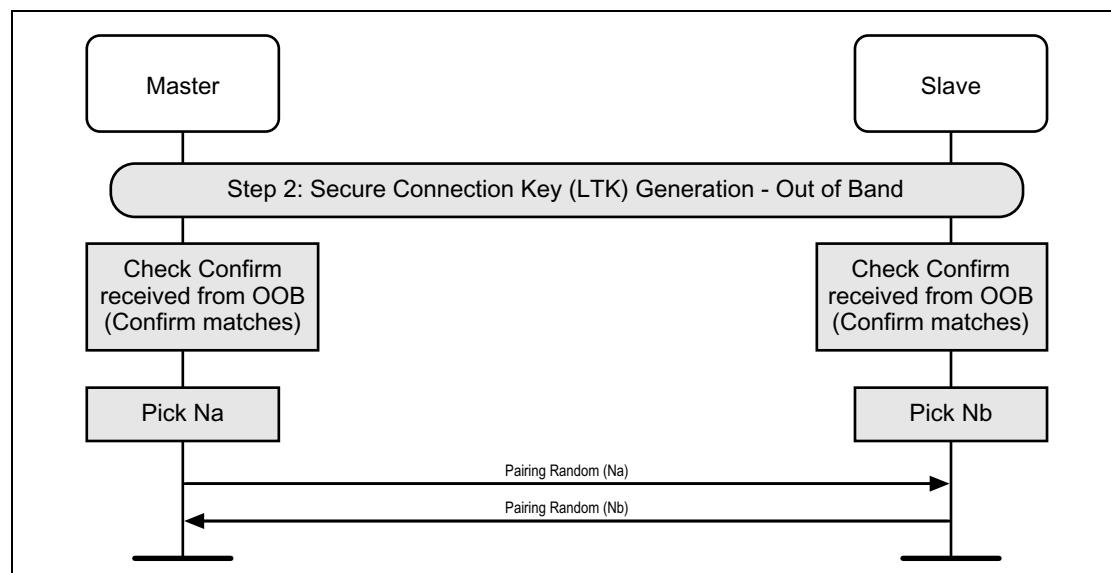


Figure C.17: Pairing Phase 2, Authentication Stage 1, Successful Out of Band

C.2.2.2.11 Out of Band – Confirm Check failure on the Responder side

If the Confirm value received from OOB is not equal to the calculated Confirm value, the Responder will abort the Pairing process by sending Pairing Failed with reason “Confirm Value Failed”.

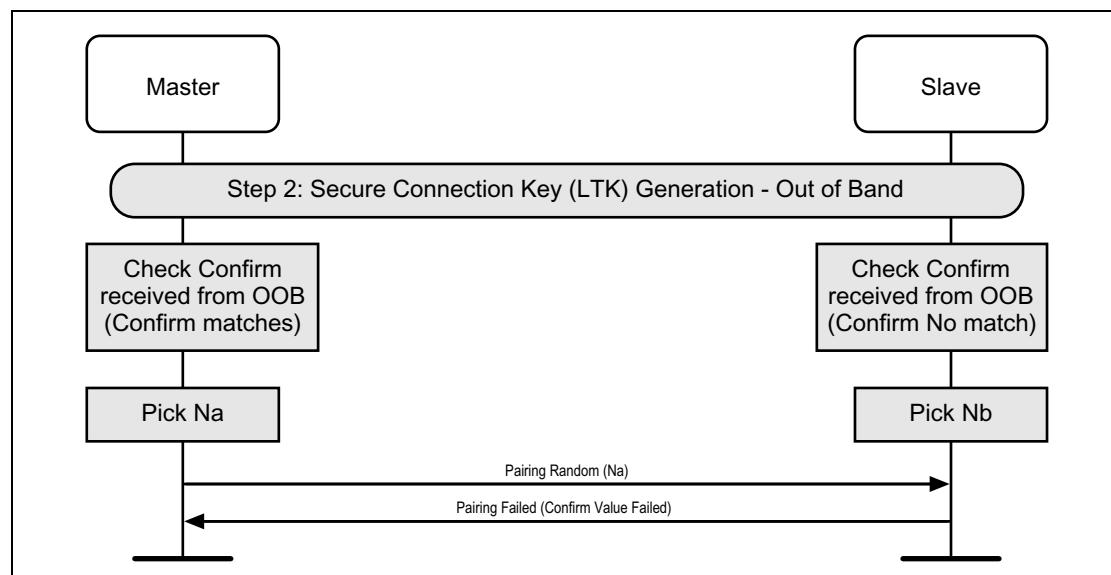


Figure C.18: Pairing Phase 2, Authentication Stage 1, Out of Band – Confirm Check failure on Responder side

C.2.2.2.12 Out of Band - Confirm Check failure on the Initiating side

If the Confirm value received from OOB is not equal to the calculated Confirm value, the Initiator will abort the Pairing process by sending Pairing Failed with reason "Confirm Value Failed".

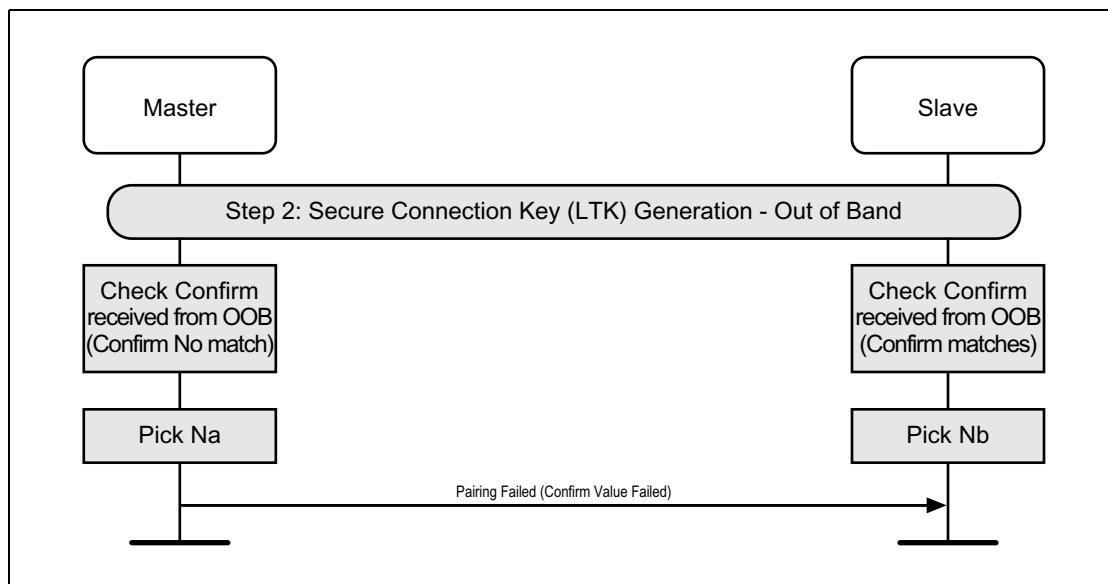


Figure C.19: Pairing Phase 2: Authentication Stage 1, Out of Band - Confirm Check failure on Initiator side"

C.2.2.2.13 Out of Band Failure on Initiator side (OOB information not available)

If the initiating side does not have the responder's OOB information, Pairing is terminated.

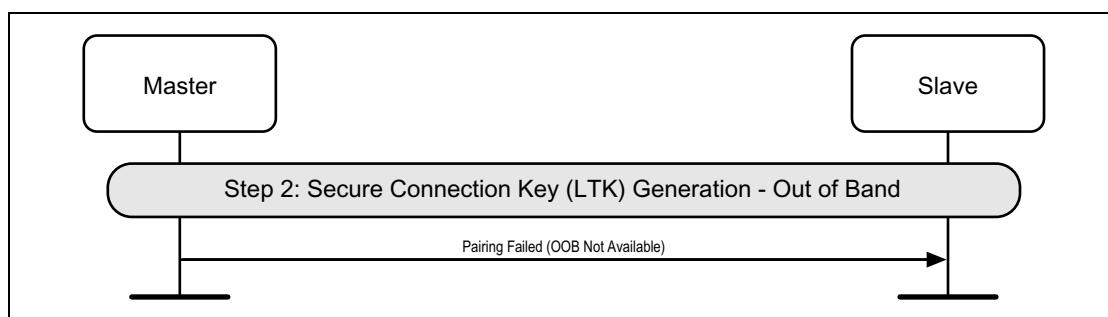


Figure C.20: Pairing Phase 2, Authentication Stage 1, Out of Band Failure on Initiator Side

C.2.2.2.14 Out of Band Failure on Responding side (OOB information not available)

If the responding side does not have the initiator's OOB information, Pairing is terminated.

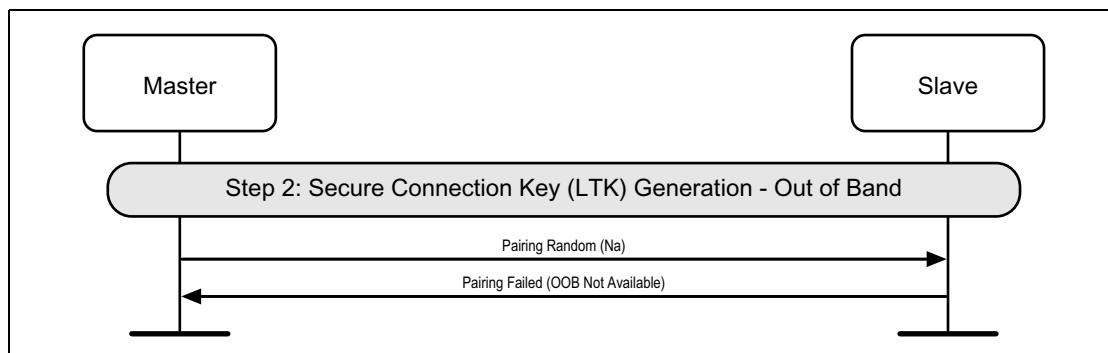


Figure C.21: Pairing Phase 2, Authentication Stage 1, Out of Band Failure on Responding Side

C.2.2.3 Long Term Key Calculation

Once the DHKey generation is complete, the Long Term Key (LTK) is calculated from the DHKey.

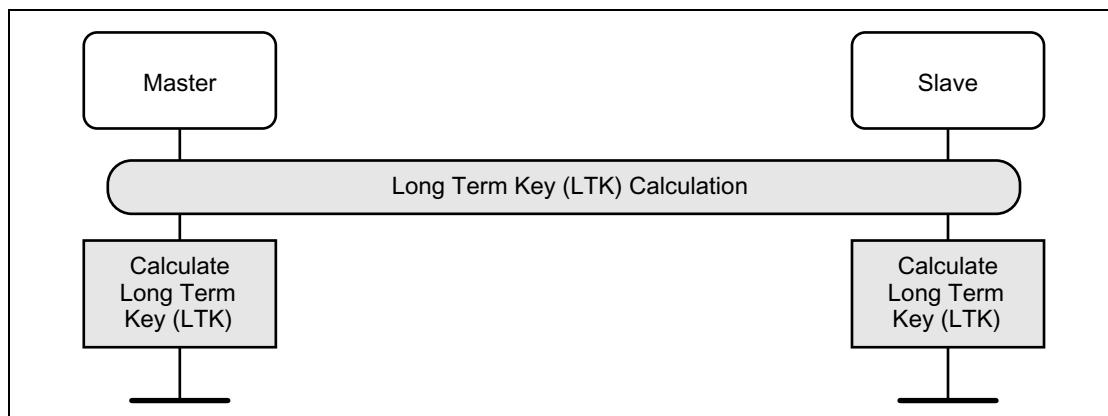


Figure C.22: Long Term Key Calculation

C.2.2.4 Authentication Stage 2 (DHKey checks)

Once the LTK calculation has completed, the DHKey value generated is checked. If this succeeds, then both devices would have finished displaying information to the user about the process, and therefore the host can stop displaying this information.

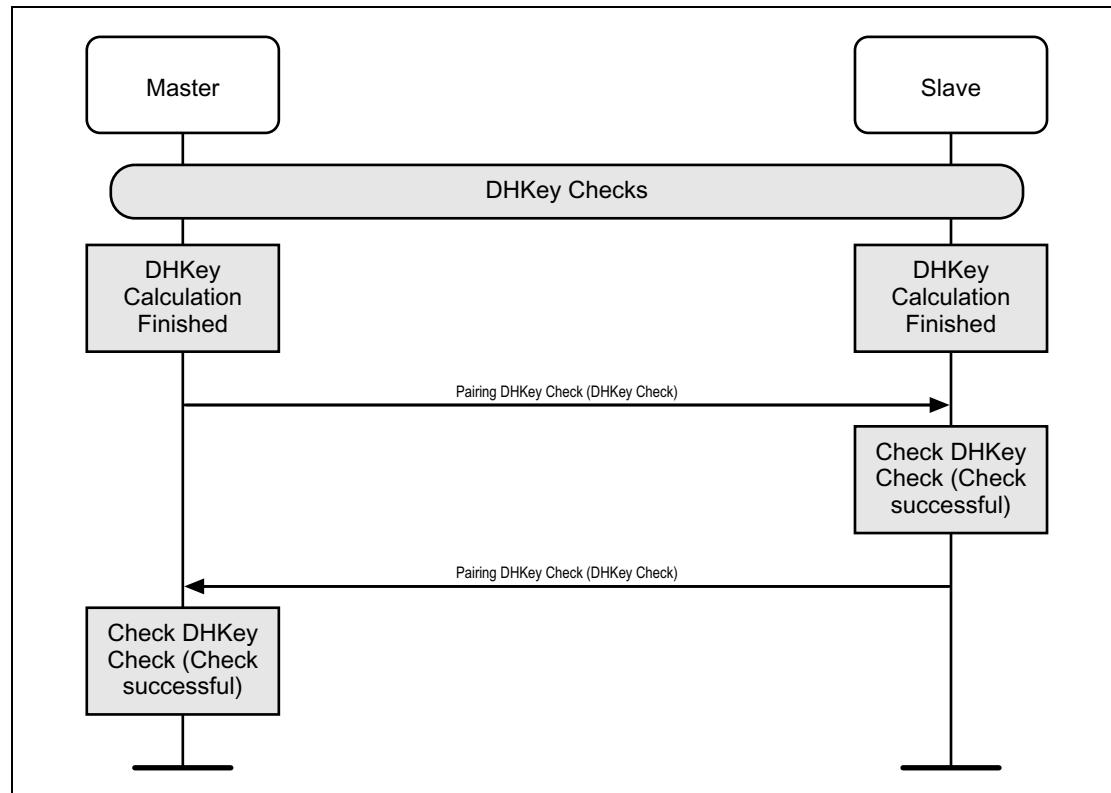


Figure C.23: Pairing Phase 2, Authentication Stage 2, DHKey checks

C.3 PHASE 3: TRANSPORT SPECIFIC KEY DISTRIBUTION

After short term key generation and the link has been encrypted, transport specific keys are distributed. [Figure C.24](#) shows an example of all keys and values being distributed by Master and Slave.

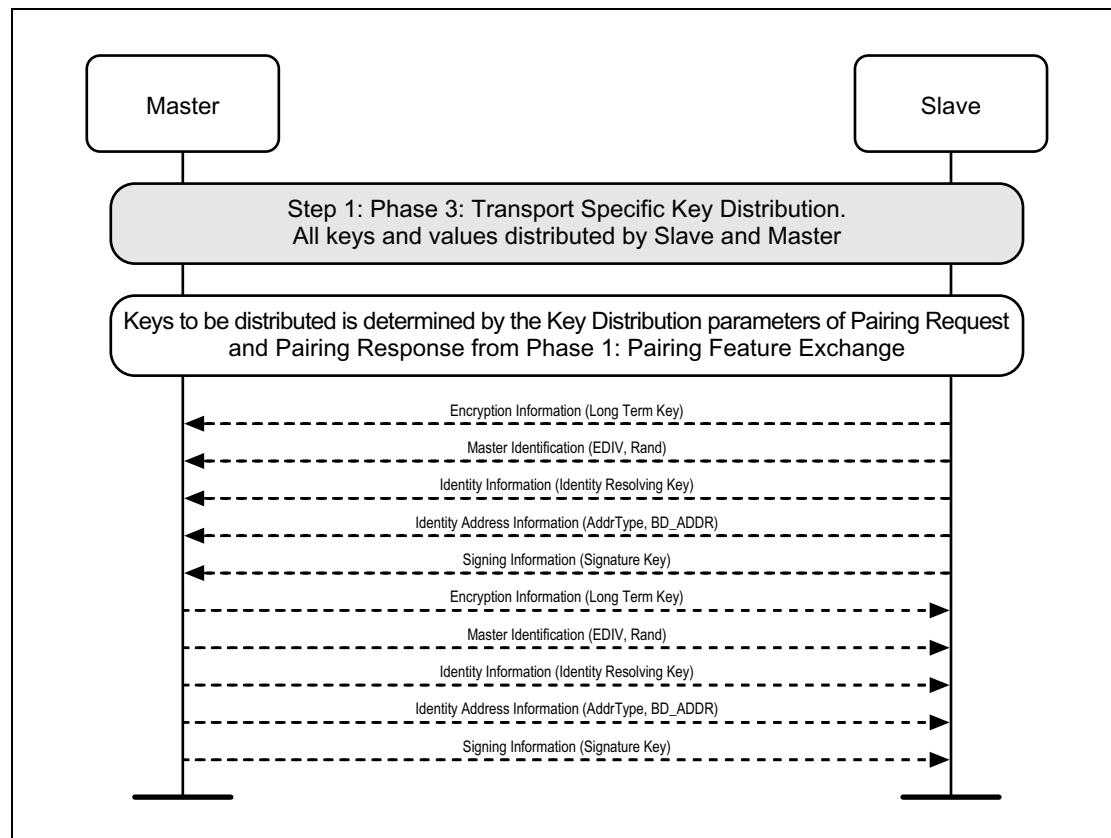


Figure C.24: Transport Specific Key Distribution

C.4 SECURITY RE-ESTABLISHED USING PREVIOUSLY DISTRIBUTED LTK

Devices may re-establish security using a previously distributed LTK. The master device always initiates the encryption procedures, and therefore there are two possible sequences: master initiated and slave requested.

C.4.1 Master Initiated Security - Master Initiated Link Layer Encryption

The master initiates encryption procedures. There is no SM signaling to enable this; the master initiates Link Layer encryption only. See [\[Vol 6\] Part D, Section 6.6](#).

C.4.2 Slave Security Request - Master Initiated Link Layer Encryption

The slave may request the master initiates security procedures. [Figure C.25](#) shows an example where the slave requests security and the master initiates Link Layer encryption.

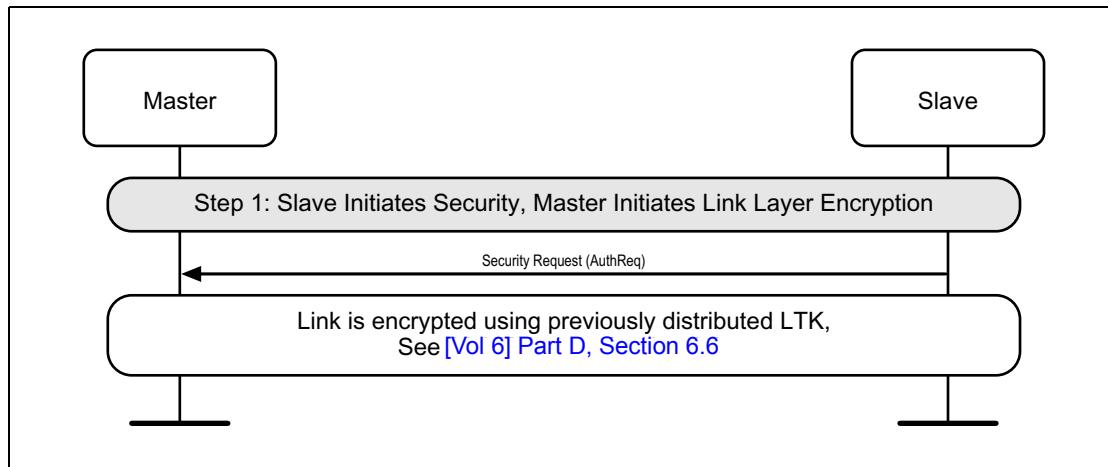


Figure C.25: Slave security request, master initiates Link Layer encryption

C.5 FAILURE CONDITIONS

The following sequences show possible failure conditions and their associated signaling.

C.5.1 Pairing Not Supported by Slave

If the slave device does not support pairing or pairing cannot be performed the slave can reject the request from the master. [Figure C.26](#) shows the slave rejecting a Pairing Request command from the master.

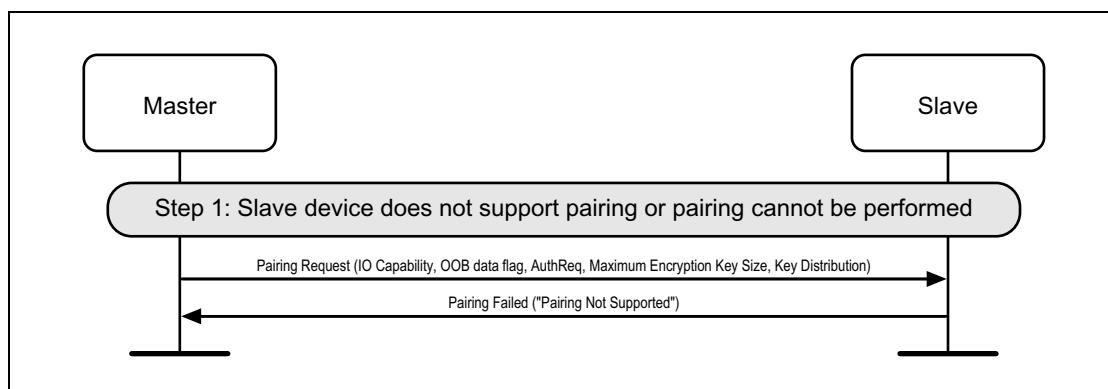


Figure C.26: Slave rejects pairing attempt

C.5.2 Master Rejects Pairing Because of Key Size

During Pairing Feature Exchange the size of the Encryption Key is negotiated. [Figure C.27](#) shows an example where the master terminates the pairing procedure because the resulting key size is not acceptable.

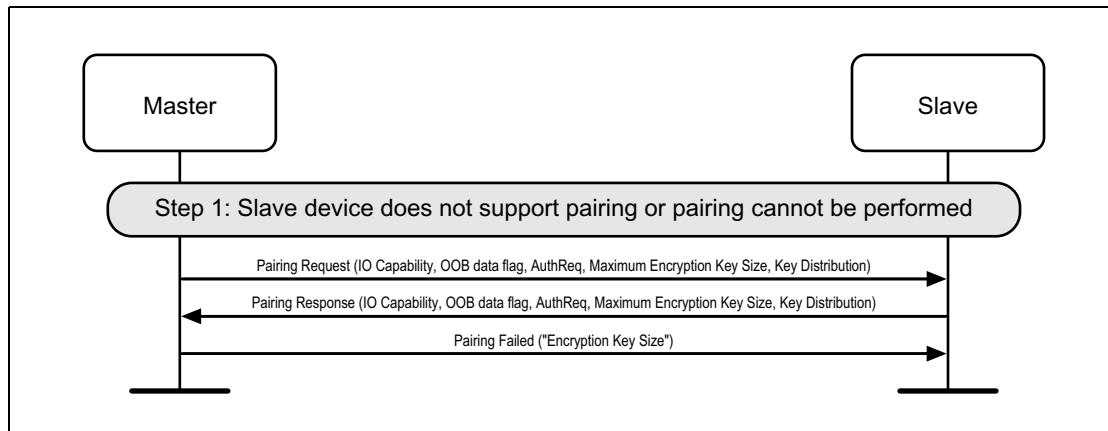


Figure C.27: Master rejects pairing because of key size

C.5.3 Slave Rejects Pairing Because of Key Size

During Pairing Feature Exchange the size of the Encryption Key is negotiated. [Figure C.28](#) shows an example where the slave terminates the pairing procedure because the resulting key size is not acceptable.

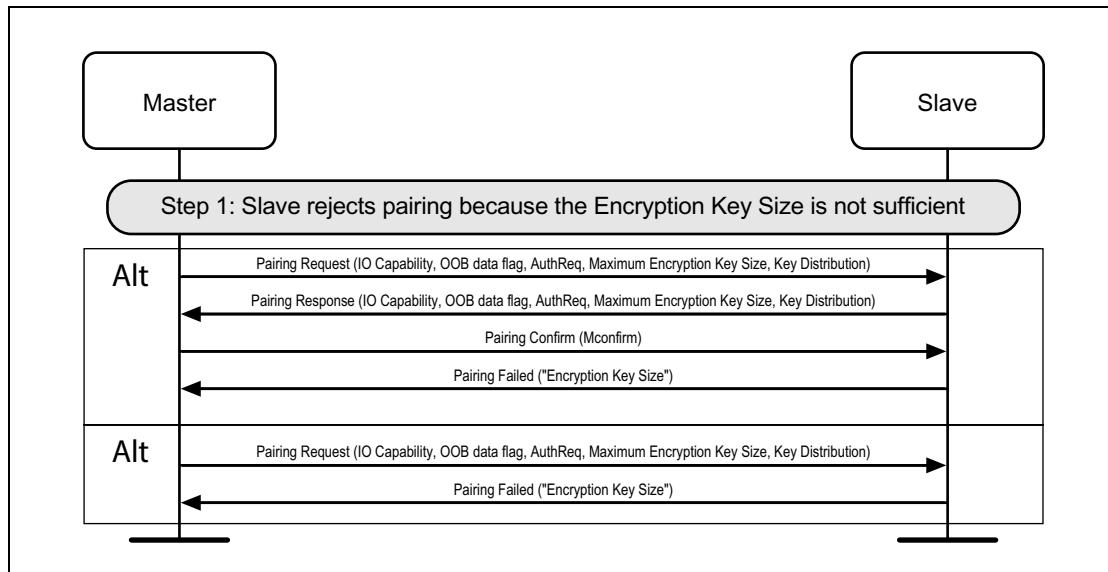


Figure C.28: Slave rejects pairing because of key size

C.5.4 Passkey Entry Failure on Master

During Passkey Entry pairing the user enters a passkey on both devices. [Figure C.29](#) shows an example where the passkey entry fails on the master device.

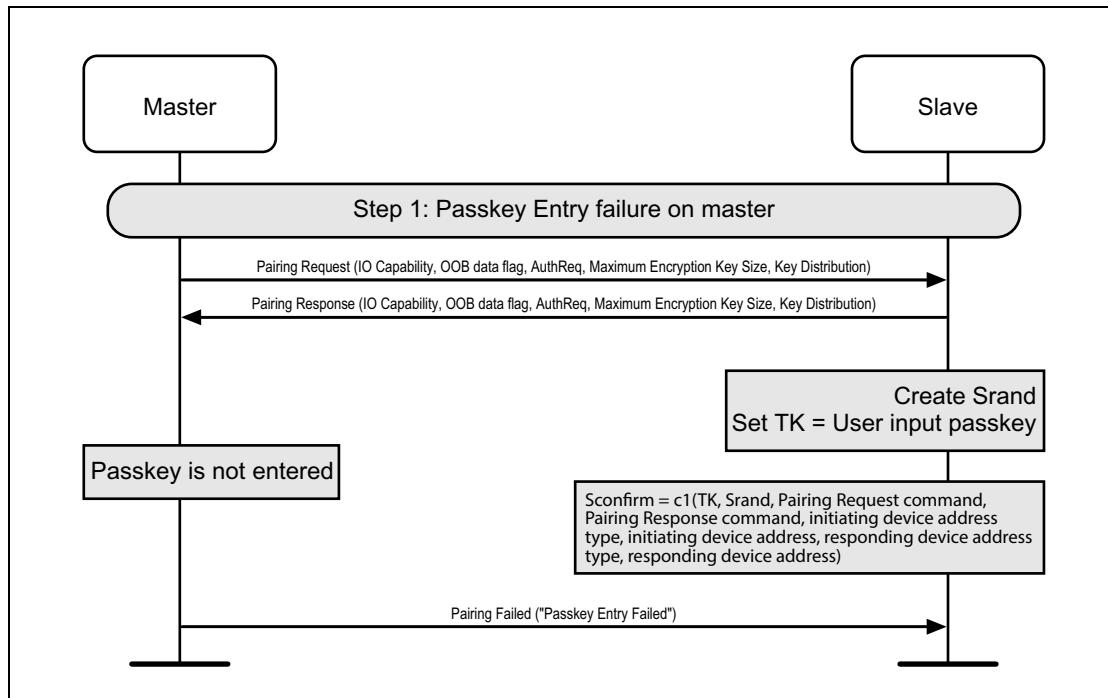


Figure C.29: Passkey Entry failure on master

C.5.5 Passkey Entry Failure on Slave

During Passkey Entry pairing the user enters a passkey on both devices. [Figure C.30](#) shows an example where the passkey entry fails on the slave device.

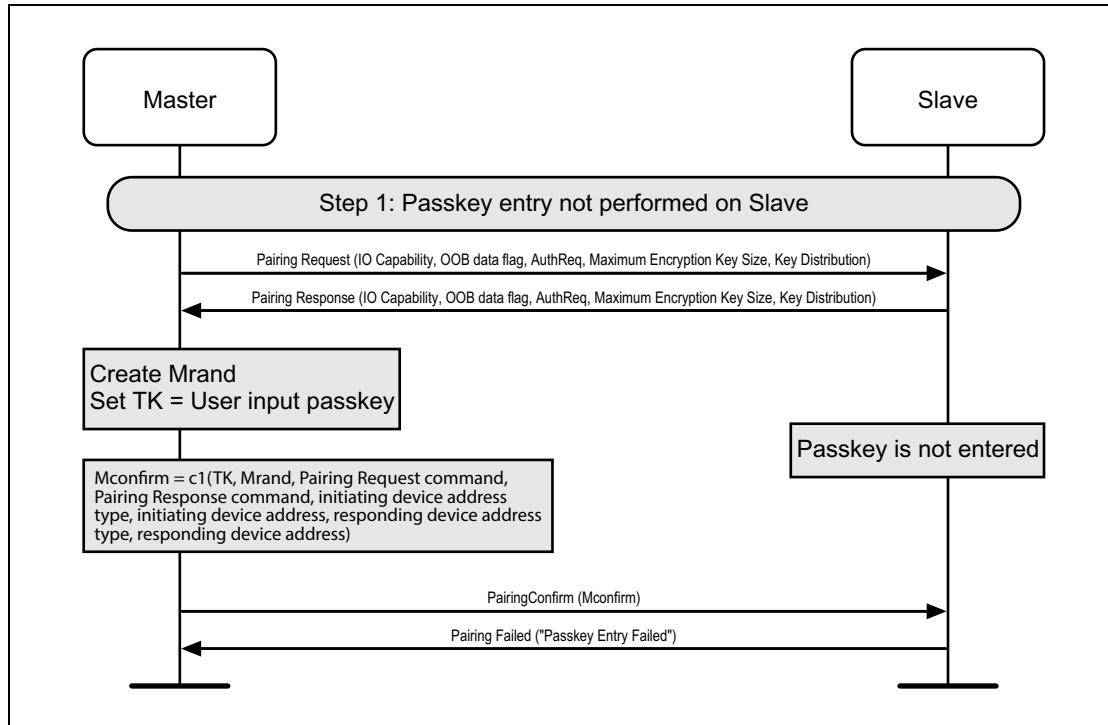


Figure C.30: Passkey Entry failure on slave

C.5.6 Slave Rejects Master's Confirm Value

During Passkey Entry pairing the user enters a passkey on both devices. [Figure C.31](#) shows an example where a different passkey is entered on both devices. This sequence could also occur if any of the inputs to c1 (Passkey, Mrand, Srand, Pairing Request command, Pairing Response command, address types or addresses) are incorrect or altered.

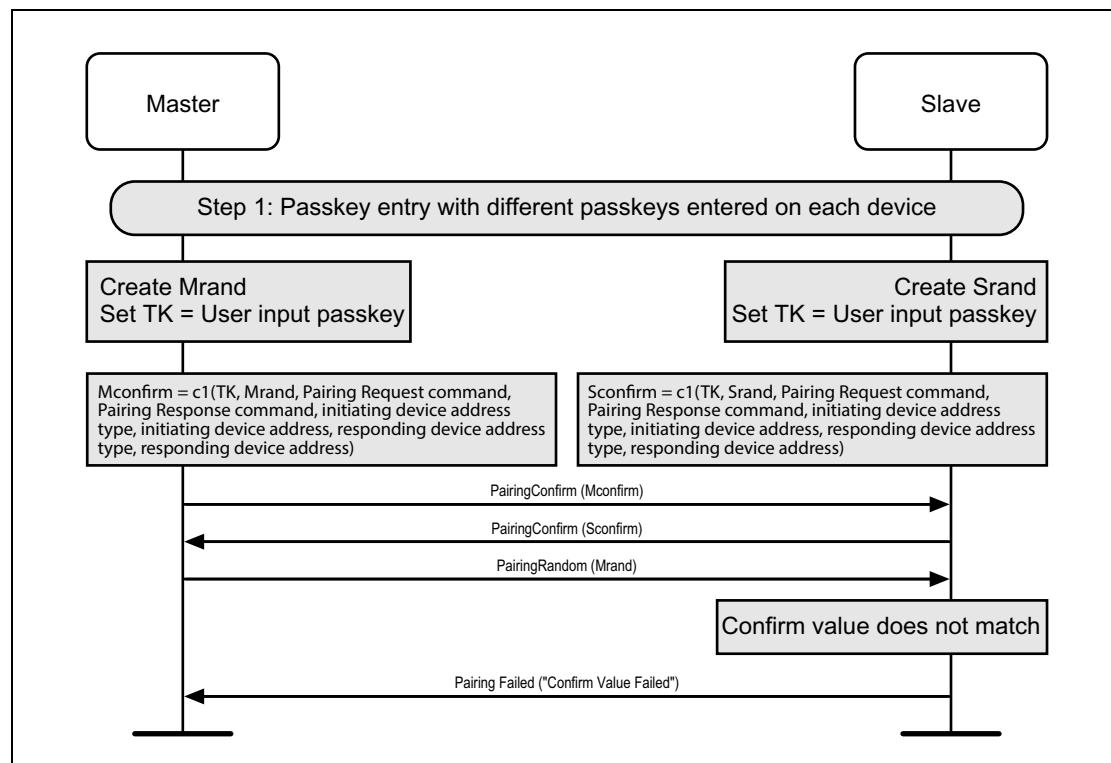


Figure C.31: Different Passkeys entered

C.5.7 Master Rejects Slaves Confirm Value

During all the pairing methods the master and slave device send random numbers and confirm values. Figure C.32 shows an example where the master device rejects pairing because it cannot verify the confirm value from the slave because any of the inputs to c1 (Passkey, Mrand, Srand, Pairing Request command, Pairing Response command, address types or addresses) are incorrect or altered.

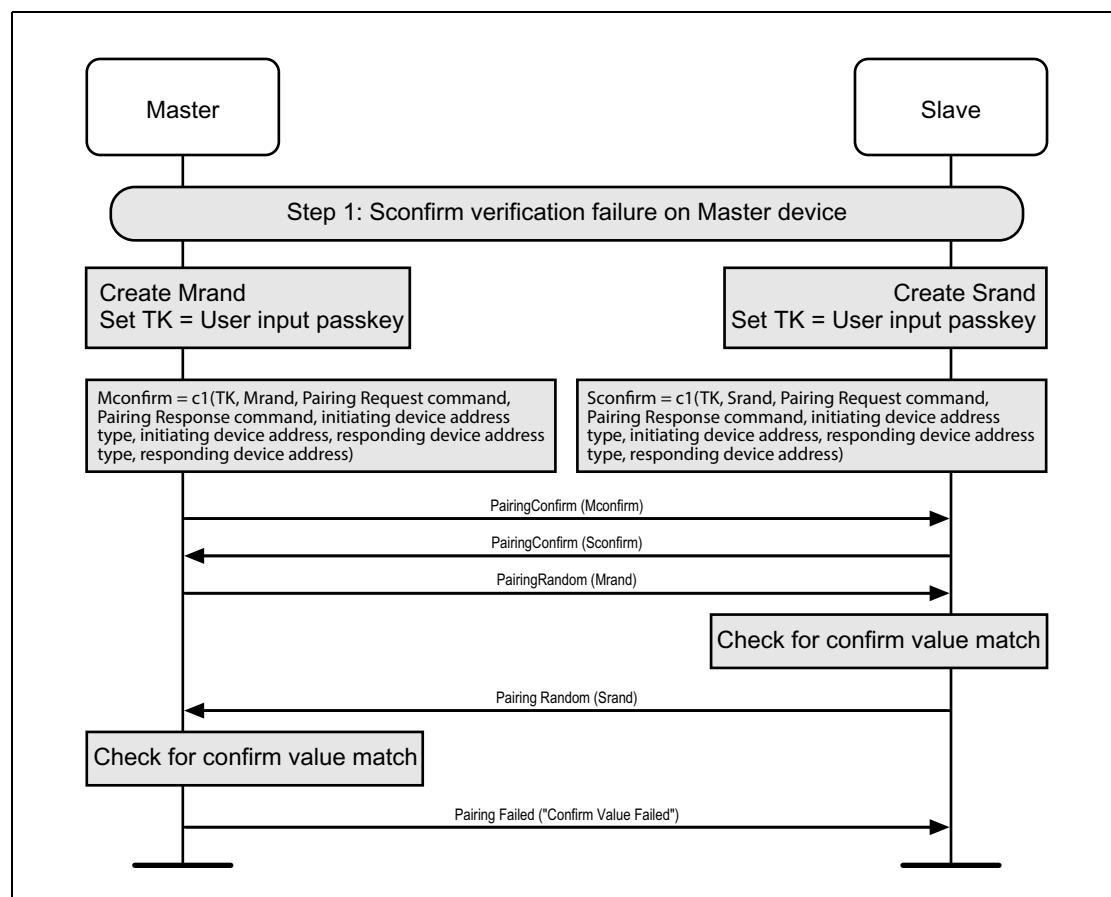


Figure C.32: Master rejects Sconfirm value from slave

APPENDIX D SAMPLE DATA

In each data set in this section, the bytes are ordered from most significant on the left to least significant on the right. 'M' represents the message byte array for which the AES CMAC is calculated.

D.1 AES-CMAC RFC4493 TEST VECTORS

The following test vectors are referenced from RFC4493.

K	2b7e1516 28aed2a6 abf71588 09cf4f3c
Subkey Generation	
AES_128 (key,0)	7df76b0c 1ab899b3 3e42f047 b91b546f
K1	fbeed618 35713366 7c85e08f 7236a8de
K2	f7ddac30 6ae266cc f90bc11e e46d513b

D.1.1 Example 1: Len = 0

M	<empty string>
AES_CMAC	bb1d6929 e9593728 7fa37d12 9b756746

D.1.2 Example 2: Len = 16

M	6bc1bee2 2e409f96 e93d7e11 7393172a
AES_CMAC	070a16b4 6b4d4144 f79bdd9d d04a287c

D.1.3 Example 3: Len = 40

M0	6bc1bee2 2e409f96 e93d7e11 7393172a
M1	ae2d8a57 1e03ac9c 9eb76fac 45af8e51
M2	30c81c46 a35ce411
AES_CMAC	dfa66747 de9ae630 30ca3261 1497c827

D.1.4 Example 4: Len = 64

M0	6bc1bee2 2e409f96 e93d7e11 7393172a
M1	ae2d8a57 1e03ac9c 9eb76fac 45af8e51
M2	30c81c46 a35ce411 e5fbcb19 1a0a52ef
M3	f69f2445 df4f9b17 ad2b417b e66c3710
AES_CMAC	51f0beb7 7e3b9d92 fc497417 79363cfe

D.2 f4 LE SC CONFIRM VALUE GENERATION FUNCTION

U	20b003d2 f297be2c 5e2c83a7 e9f9a5b9 eff49111 acf4fddb cc030148 0e359de6
V	55188b3d 32f6bb9a 900afc9fb eed4e72a 59cb9ac2 f19d7cfb 6b4fdd49 f47fc5fd
X	d5cb8454 d177733e fffffb2ec 712baeab
Z	0x00
M0	20b003d2 f297be2c 5e2c83a7 e9f9a5b9 eff49111 acf4fddb cc030148 0e359de6
M1	55188b3d 32f6bb9a 900afc9fb eed4e72a 59cb9ac2 f19d7cfb 6b4fdd49 f47fc5fd
M2	00
M3	AES_CMAC f2c916f1 07a9bd1c f1eda1be a974872d

D.3 f5 LE SC KEY GENERATION FUNCTION

DHKey (W)	ec0234a3 57c8ad05 341010a6 0a397d9b 99796b13 b4f866f1 868d34f3 73bfa698
T	3c128f20 de883288 97624bdb 8dac6989
keyID	62746c65
N1	d5cb8454 d177733e fffffb2ec 712baeab
N2	a6e8e7cc 25a75f6e 216583f7 ff3dc4cf
A1	00561237 37bfce
A2	00a71370 2dcfc1
Length (LTK)	0100
M0	0162746c 65d5cb84 54d17773 3efffffb2
M1	ec712bae aba6e8e7 cc25a75f 6e216583
M2	f7ff3dc4 cf005612 3737bfce 00a71370
M3	2dcfc101 00
AES_CMAC (MacKey)	69867911 69d7cd23 980522b5 94750a38
M0	0062746c 65d5cb84 54d17773 3efffffb2
M1	ec712bae aba6e8e7 cc25a75f 6e216583
M2	f7ff3dc4 cf005612 3737bfce 00a71370
M3	2dcfc101 00
AES_CMAC	2965f176 a1084a02 fd3f6a20 ce636e20

D.4 f6 LE SC CHECK VALUE GENERATION FUNCTION

N1	d5cb8454 d177733e fffffb2ec 712baeab
N2	a6e8e7cc 25a75f6e 216583f7 ff3dc4cf
MacKey	2965f176 a1084a02 fd3f6a20 ce636e20
R	12a3343b b453bb54 08da42d2 0c2d0fc8
IOcap	010102
A1	00561237 37bfce
A2	00a71370 2dcfc1
M0	d5cb8454 d177733e fffffb2ec 712baeab
M1	a6e8e7cc 25a75f6e 216583f7 ff3dc4cf
M2	12a3343b b453bb54 08da42d2 0c2d0fc8
M3	01010200 56123737 bfce00a7 13702dcf
M4	c1
AES_CMAC	e3c47398 9cd0e8c5 d26c0b09 da958f61

D.5 *g2* LE SC NUMERIC COMPARISON GENERATION FUNCTION

U	20b003d2 f297be2c 5e2c83a7 e9f9a5b9
	eff49111 acf4fddb cc030148 0e359de6
V	55188b3d 32f6bb9a 900afc9fb eed4e72a
	59cb9ac2 f19d7cfb 6b4fdd49 f47fc5fd
X	d5cb8454 d177733e fffffb2ec 712baeab
Y	a6e8e7cc 25a75f6e 216583f7 ff3dc4cf
M0	20b003d2 f297be2c 5e2c83a7 e9f9a5b9
M1	eff49111 acf4fddb cc030148 0e359de6
M2	55188b3d 32f6bb9a 900afc9fb eed4e72a
M3	59cb9ac2 f19d7cfb 6b4fdd49 f47fc5fd
M4	a6e8e7cc 25a75f6e 216583f7 ff3dc4cf
AES_CMAC	1536d18d e3d20df9 9b7044c1 2f9ed5ba
g2	2f9ed5ba

D.6 *h6* LE SC LINK KEY CONVERSION FUNCTION

Key	ec0234a3 57c8ad05 341010a6 0a397d9b
keyID	6c656272
M	6c656272
AES_CMAC	2d9ae102 e76dc91c e8d3a9e2 80b16399

D.7 *ah* RANDOM ADDRESS HASH FUNCTIONS

IRK	ec0234a3 57c8ad05 341010a6 0a397d9b
prand	00000000 00000000 00000000 00708194
M	00000000 00000000 00000000 00708194
AES_128	159d5fb7 2ebe2311 a48c1bdc c40dfbaa
ah	0dfbaa



LIST OF FIGURES (ALL PARTS)

Part A

LOGICAL LINK CONTROL AND ADAPTATION PROTOCOL SPECIFICATION

Figure 1.1:	L2CAP architectural blocks	30
Figure 2.1:	Dynamically allocated CID assignments.....	38
Figure 2.2:	Channels between devices.....	40
Figure 2.3:	L2CAP transaction model.....	41
Figure 3.1:	L2CAP PDU format in Basic L2CAP mode on connection-oriented channels (field sizes in bits).....	45
Figure 3.2:	L2CAP PDU format on the Connectionless channel	46
Figure 3.3:	L2CAP PDU formats in Flow Control and Retransmission Modes	47
Figure 3.4:	The LFSR circuit generating the FCS.....	52
Figure 3.5:	Initial state of the FCS generating circuit.....	52
Figure 3.6:	L2CAP PDU format in LE Credit Based Flow Control Mode	55
Figure 4.1:	L2CAP PDU format on a signaling channel.....	57
Figure 4.2:	Command format.....	58
Figure 4.3:	Command Reject Packet.....	60
Figure 4.4:	Connection Request Packet	61
Figure 4.5:	Connection Response Packet	63
Figure 4.6:	Configuration Request Packet.....	65
Figure 4.7:	Configuration Request Flags field format	65
Figure 4.8:	Configuration Response Packet	67
Figure 4.9:	Configuration Response Flags field format.....	67
Figure 4.10:	Disconnection Request Packet.....	69
Figure 4.11:	Disconnection Response Packet	70
Figure 4.12:	Echo Request Packet	70
Figure 4.13:	Echo Response Packet	71
Figure 4.14:	Information Request Packet	71
Figure 4.15:	Information Response Packet	72
Figure 4.16:	Create Channel Request Packet	76
Figure 4.17:	Create Channel Response Packet	77
Figure 4.18:	Move Channel Request Packet	78
Figure 4.19:	Move Channel Response Packet	80
Figure 4.20:	Move Channel Confirmation Packet	81
Figure 4.21:	Move Channel Confirmation Response Packet.....	82
Figure 4.22:	Connection Parameters Update Request Packet	83
Figure 4.23:	Connection Parameters Update Response Packet	84
Figure 4.24:	LE Credit Based Connection Request Packet.....	85
Figure 4.25:	LE Credit Based Connection Response Packet	86



Figure 4.26: LE Flow Control Credit Packet	88
Figure 5.1: Configuration option format	89
Figure 5.2: MTU Option Format.....	90
Figure 5.3: Flush Timeout option format	91
Figure 5.4: Quality of Service (QoS) option format containing Flow Specification	93
Figure 5.5: Retransmission and Flow Control option format.....	96
Figure 5.6: FCS option format	102
Figure 5.7: Extended Flow Specification option format.....	103
Figure 5.8: Extended Window Size option format.....	108
Figure 6.1: States and transitions	128
Figure 6.2: Configuration states and transitions	129
Figure 6.3: States and transitions—AMP enabled operation.....	130
Figure 7.1: L2CAP fragmentation in a BR/EDR Controller	138
Figure 7.2: Example of fragmentation processes in a device with a BR/EDR Controller and USB HCI transport	139
Figure 7.3: Segmentation and fragmentation of an SDU in a BR/EDR Controller	140
Figure 7.4: Example of segmentation and fragment processes in a device with a BR/EDR Controller and USB HCI Transport	141
Figure 8.1: Example of the transmitter side	152
Figure 8.2: Example of the receiver side	153
Figure 8.3: Overview of the receiver side when operating in flow control mode.....	161
Figure 8.4: Receiver state machine	171
Figure 9.1: Creation of First Best Effort L2CAP channel or a Guaranteed L2CAP channel.....	201
Figure 9.2: Creation of Subsequent Best Effort L2CAP channel	202
Figure 9.3: Move of a Best Effort L2CAP channel Enhanced Retransmission mode enabled to an AMP with no existing best effort logical link or move of any Guaranteed L2CAP channel	209
Figure 9.4: Move of a Best Effort L2CAP channel with Enhanced Retransmission mode enabled to an AMP where a best effort logical link exists	210
Figure A.1: Basic MTU exchange	212
Figure A.2: Dealing with Unknown Options.....	213
Figure A.3: Unsuccessful Configuration Request	214

Part B

SERVICE DISCOVERY PROTOCOL (SDP) SPECIFICATION

Figure 2.1: SDP Client-Server Interaction	222
Figure 2.2: Simplified SDP Client-Server Interaction.....	222
Figure 2.3: Service Record	224
Figure 2.4: Service Attribute	225



Figure 2.5:	Service Browsing Hierarchy.....	229
Figure 3.1:	Data Element	233
Figure 4.1:	Protocol Data Unit Format	234
Figure 4.2:	Continuation State Format.....	236
Figure 4.3:	Error Handling	237
Figure 4.4:	ServiceSearch Transaction.....	238
Figure 4.5:	ServiceAttribute Transaction.....	241
Figure 4.6:	ServiceSearchAttribute Transaction	244

Part C

GENERIC ACCESS PROFILE

Figure 1.1:	Arrows used in signaling diagrams	288
Figure 2.1:	Relationship of GAP with lower layers of the Bluetooth architecture	289
Figure 2.2:	This profile covers procedures initiated by one device (A) towards another device (B) that may or may not have an existing Bluetooth link active.....	290
Figure 5.1:	Definition of the generic authentication procedure	309
Figure 5.2:	Channel establishment with security	310
Figure 5.3:	Security mode 3 with a legacy remote device	312
Figure 5.4:	Channel establishment using security mode 4 for initiating side	316
Figure 5.5:	Channel establishment using security mode 4 for the responding side	319
Figure 5.6:	OOB Data Block Format.....	324
Figure 6.1:	General inquiry, where B is a device in non-discoverable mode, B' is a device in limited discoverable mode and B'' is a device in general discoverable mode. (Note that all discoverable devices are discovered using general inquiry, independent of which discoverable mode they are in.)	329
Figure 6.2:	Limited inquiry where B is a device in non-discoverable mode, B' is a device in limited discoverable mode and B'' is a device in general discoverable mode. (Note that only limited discoverable devices can be discovered using limited inquiry.).....	330
Figure 6.3:	Name request procedure.....	331
Figure 6.4:	Name discovery procedure.....	332
Figure 6.5:	Device discovery procedure	333
Figure 6.6:	General description of general bonding as being the link establishment procedure executed under specific conditions on both devices, followed by authentication and an optional higher layer initialization process.....	335
Figure 6.7:	Dedicated Bonding as performed when the purpose of the procedure is only to create and exchange a link key between two Bluetooth devices	336



Figure 7.1:	Link establishment procedure when the paging device (A) is in security mode 3 and the paged device (B) is in security mode 1, 2, or 4	338
Figure 7.2:	Link establishment procedure when both the paging device (A) and the paged device (B) are in security mode 3	339
Figure 7.3:	Channel establishment procedure when the initiator (A) is in security mode 3 and the acceptor (B) is in security mode 2 or 4	341
Figure 7.4:	Channel establishment procedure when the initiator (A) is in security mode 3 and the acceptor (B) is in security mode 1 or 3	341
Figure 7.5:	Connection establishment procedure when the initiator (A) is in security mode 3 and the acceptor (B) is in security mode 2 or 4	342
Figure 7.6:	Connection establishment procedure when the initiator (A) is in security mode 3 and the acceptor (B) is in security mode 1 or 3	343
Figure 7.7:	Synchronization establishment procedure.....	344
Figure 8.1:	Extended Inquiry Response data format	346
Figure 9.1:	A Central performing limited discovery procedure discovering Peripherals in the limited discoverable mode	354
Figure 9.2:	A Central performing general discovery procedure discovering Peripherals in the limited discoverable mode and general discoverable mode	355
Figure 9.3:	Flow chart for a device performing the auto connection establishment procedure	360
Figure 9.4:	Flow chart for a device performing the general connection establishment procedure	361
Figure 9.5:	Flow chart for a device performing the selective connection establishment procedure	363
Figure 9.6:	Flow chart for a device performing the directed connection establishment procedure	364
Figure 9.7:	The Bonding procedure	370
Figure 10.1:	Flow chart for a local device handling a service request from a remote device	377
Figure 10.2:	Flow chart for a local device issuing a service request to a remote device	380
Figure 10.3:	Generic format of signed data	381
Figure 11.1:	Advertising and Scan Response data format	389
Figure B.1:	LMP-authentication as defined by [2]	408
Figure B.2:	LMP-pairing as defined in [2]	409
Figure B.3:	Service discovery procedure	410



Part D

TEST SUPPORT

Figure 1.1:	Setup for Test Mode	414
Figure 1.2:	Timing for Transmitter Test	415
Figure 1.3:	General Format of TX Packet.....	416
Figure 1.4:	Use of whitening in Transmitter mode	417
Figure 1.5:	Linear Feedback Shift Register for Generation of the PRBS sequence	417
Figure 1.6:	DUT Packet Handling in Loop Back Test.....	421
Figure 1.7:	Payload & ARQN handling in normal loopback.	422
Figure 1.8:	Payload & ARQN handling in delayed loopback - start.	422
Figure 1.9:	Payload & ARQN handling in delayed loopback - end.	422
Figure 1.10:	Test Architecture.....	424
Figure 1.11:	L2CAP test command/event formats	427
Figure 2.1:	General test set-up for RF qualification	435
Figure 2.2:	General test set-up for profile qualification	436
Figure 2.3:	BB and LM qualification with TCI physical bearer	438
Figure 2.4:	BB and LM qualification with “software bearer”	439
Figure 2.5:	General test set-up for HCI qualification.....	440

Part E

AMP MANAGER PROTOCOL SPECIFICATION

Figure 2.1:	Overview of the Lower Software Layers	444
Figure 2.2:	AMP Manager Peers	445
Figure 3.1:	AMP Manager Protocol Packet Format	449
Figure 3.2:	AMP Command Reject Packet	451
Figure 3.3:	AMP Discover Request Packet	452
Figure 3.4:	Example A2MP Extended Feature Mask.....	453
Figure 3.5:	AMP Discover Response Packet.....	453
Figure 3.6:	Controller list format	454
Figure 3.7:	AMP Change Notify Packet	457
Figure 3.8:	AMP Change Response Packet.....	457
Figure 3.9:	AMP Get Info Request Packet.....	458
Figure 3.10:	AMP Get Info Response Packet.....	458
Figure 3.11:	AMP Get AMP Assoc Request Packet	460
Figure 3.12:	AMP Get AMP Assoc Response Packet	460
Figure 3.13:	AMP Create Physical Link Request Packet.....	462
Figure 3.14:	AMP Create Physical Link Response Packet.....	463
Figure 3.15:	AMP Disconnect Physical Link Request Packet.....	465
Figure 3.16:	AMP Disconnect Physical Link Response Packet.....	466

**Part F****ATTRIBUTE PROTOCOL (ATT)**

Figure 3.1: MTU request and response exchange 484

Part G**GENERIC ATTRIBUTE PROFILE (GATT)**

Figure 1.1:	Profile dependencies	519
Figure 2.1:	Protocol model.....	521
Figure 2.2:	Examples of configuration	522
Figure 2.3:	Attribute Protocol PDU	523
Figure 2.4:	Logical Attribute Representation.....	523
Figure 2.5:	GATT Profile hierarchy	527
Figure 4.1:	Exchange MTU	546
Figure 4.2:	Discover All Primary Services example	548
Figure 4.3:	Discover Primary Service by Service UUID example	549
Figure 4.4:	Find Included Services example.....	551
Figure 4.5:	Discover All Characteristics of a Service example	552
Figure 4.6:	Discover Characteristics by UUID example	554
Figure 4.7:	Discover All Characteristic Descriptors example	555
Figure 4.8:	Read Characteristic Value example.....	556
Figure 4.9:	Read Using Characteristic UUID example.....	557
Figure 4.10:	Read Long Characteristic Values example.....	558
Figure 4.11:	Read Multiple Characteristic Values example	559
Figure 4.12:	Write Without Response example	559
Figure 4.13:	Signed Write Without Response example	560
Figure 4.14:	Write Characteristic Value example	561
Figure 4.15:	Write Long Characteristic Values example	562
Figure 4.16:	Reliable Writes example	564
Figure 4.17:	Notifications example	565
Figure 4.18:	Indications example	565
Figure 4.19:	Read Characteristic Descriptors example	566
Figure 4.20:	Read Long Characteristic Descriptors example	567
Figure 4.21:	Write Characteristic Descriptors example.....	568
Figure 4.22:	Write Long Characteristic Descriptors example....	569



Part H

SECURITY MANAGER SPECIFICATION

Figure 1.1:	Relationship of the Security Manager to the rest of the LE Bluetooth architecture.....	591
Figure 2.1:	LE Pairing Phases	593
Figure 2.2:	Public Key Exchange.....	615
Figure 2.3:	"Authentication Stage 1: Just Works or Numeric Comparison, LE Secure Connections	616
Figure 2.4:	Authentication Stage 1: Passkey Entry, LE Secure Connections.....	618
Figure 2.5:	Authentication Stage 1: Out of Band, LE Secure Connections	620
Figure 2.6:	Authentication Stage 2 and Long Term Key Calculation.....	622
Figure 2.7:	Master actions after receiving Security Request	632
Figure 3.1:	SMP Command Format.....	634
Figure 3.2:	Pairing Request Packet	636
Figure 3.3:	Authentication Requirements Flags.....	637
Figure 3.4:	Pairing Response Packet	638
Figure 3.5:	Pairing Confirm Packet.....	640
Figure 3.6:	Pairing Random Packet.....	641
Figure 3.7:	Pairing Failed Packet.....	642
Figure 3.8:	Pairing Public Key PDU	644
Figure 3.9:	Pairing DHKey Check PDU	644
Figure 3.10:	Pairing Keypress Notification PDU	645
Figure 3.11:	LE Key Distribution Format.....	647
Figure 3.12:	Encryption Information Packet.....	649
Figure 3.13:	Master Identification Packet	650
Figure 3.14:	Identity Information Packet	650
Figure 3.15:	Identity Address Information Packet.....	651
Figure 3.16:	Signing Information Packet.....	652
Figure 3.17:	Security Request Packet	652
Figure B.1:	Example Key Hierarchy	656
Figure C.1:	Pairing Process Overview	660
Figure C.2:	Pairing initiated by master	660
Figure C.3:	Slave security request, master initiated pairing	661
Figure C.4:	Legacy Just Works Pairing Method	662
Figure C.5:	Legacy Passkey Entry Pairing Method	663
Figure C.6:	Legacy OOB Pairing Method	664
Figure C.7:	Pairing Phase 2 - Public Key Exchange	665
Figure C.8:	Pairing Phase 2, Authentication Stage 1, Successful Numeric Comparison	666
Figure C.9:	Pairing Phase 2, Authentication Stage 1, Numeric Comparison – Confirm Check failure on Initiator side.....	667



Figure C.10: Pairing Phase 2, Authentication Stage 1, Numeric Comparison Failure on Initiator Side.....	668
Figure C.11: Pairing Phase 2, Authentication Stage 1, Numeric Comparison Failure on Responding Side	669
Figure C.12: Pairing Phase 2, Authentication Stage 1, Successful Passkey Entry	670
Figure C.13: Pairing Phase 2, Authentication Stage 1, Passkey Entry – Confirm Check failure on Responder side	671
Figure C.14: Pairing Phase 2, Authentication Stage 1, Passkey Entry – Confirm Check failure on Initiator side.....	672
Figure C.15: Pairing Phase 2, Authentication Stage 1, Passkey Entry Failure on Responding Side	673
Figure C.16: Pairing Phase 2, Authentication Stage 1, Passkey Entry Failure on Initiator Side.....	673
Figure C.17: Pairing Phase 2, Authentication Stage 1, Successful Out of Band	674
Figure C.18: Pairing Phase 2, Authentication Stage 1, Out of Band – Confirm Check failure on Responder side	674
Figure C.19: Pairing Phase 2: Authentication Stage 1, Out of Band - Confirm Check failure on Initiator side"	675
Figure C.20: Pairing Phase 2, Authentication Stage 1, Out of Band Failure on Initiator Side.....	675
Figure C.21: Pairing Phase 2, Authentication Stage 1, Out of Band Failure on Responding Side	676
Figure C.22: Long Term Key Calculation.....	676
Figure C.23: Pairing Phase 2, Authentication Stage 2, DHKey checks	677
Figure C.24: Transport Specific Key Distribution	678
Figure C.25: Slave security request, master initiates Link Layer encryption	679
Figure C.26: Slave rejects pairing attempt	679
Figure C.27: Master rejects pairing because of key size.....	680
Figure C.28: Slave rejects pairing because of key size	680
Figure C.29: Passkey Entry failure on master	681
Figure C.30: Passkey Entry failure on slave	682
Figure C.31: Different Passkeys entered.....	683
Figure C.32: Master rejects Sconfirm value from slave	684



LIST OF TABLES (ALL PARTS)

Part A:

LOGICAL LINK CONTROL AND ADAPTATION PROTOCOL SPECIFICATION

Table 1.1: Terminology	33
Table 2.1: CID name space on ACL-U logical link.....	38
Table 2.2: CID name space on LE-U logical link	39
Table 2.3: Types of Channel Identifiers	41
Table 3.1: Standard Control Field formats.....	49
Table 3.2: Enhanced Control Field formats	50
Table 3.3: Extended Control Field formats	50
Table 3.4: SAR control element format	51
Table 3.5: S control element format: type of S-frame	51
Table 4.1: Minimum Signaling MTU.....	57
Table 4.2: Signaling Command Codes	58
Table 4.3: Reason Code Descriptions.....	60
Table 4.4: Reason Data values	61
Table 4.5: PSM ranges and usage	62
Table 4.6: Result values	63
Table 4.7: Status values	64
Table 4.8: Configuration Response Result codes	68
Table 4.9: InfoType definitions.....	71
Table 4.10: Information Response Result values	72
Table 4.11: Information Response Data fields.....	73
Table 4.12: Extended feature mask.....	74
Table 4.13: Fixed Channels Supported bit mask.....	75
Table 4.14: Result values	77
Table 4.15: Status values	78
Table 4.16: Result values	80
Table 4.17: Result values	81
Table 4.18: Result values	84
Table 4.19: LE Credit Based Connection Request LE_PSM ranges.....	85
Table 4.20: Result values for LE Credit Based Connection Response	87
Table 5.1: Service type definitions.....	93
Table 5.2: Mode definitions	96
Table 5.3: FCS types	102
Table 5.4: Default Values for Extended Flow Specification	103
Table 5.5: Traffic parameters for L2CAP QoS configuration	103
Table 5.6: Service Type definitions.....	105
Table 5.7: Extended Window Size values	108
Table 6.1: CLOSED state event table.....	112



Table 6.2:	WAIT_CONNECT_RSP state event table	113
Table 6.3:	WAIT_CONNECT state event table.....	114
Table 6.4:	CONFIG state event table	116
Table 6.5:	CONFIG state/substates event table: non-success transitions within configuration process	118
Table 6.6:	CONFIG state/substates event table: events not related to configuration process	118
Table 6.7:	OPEN state event table	120
Table 6.8:	WAIT_DISCONNECT state event table.....	121
Table 6.9:	WAIT_CREATE_RSP state event table.....	122
Table 6.10:	WAIT_CREATE state event table	122
Table 6.11:	WAIT_MOVE_RSP state event table.....	123
Table 6.12:	WAIT_MOVE state event table	124
Table 6.13:	WAIT_MOVE_CONFIRM state event table	124
Table 6.14:	WAIT_CONFIRM_RSP state event table	125
Table 7.1:	Parameters allowed in Request.....	132
Table 7.2:	Parameters allowed in Response.....	133
Table 7.3:	Permitted parameter in L2CAP Configuration Response for Service Type “Best Effort”	135
Table 7.4:	Permitted parameter changes in L2CAP Configuration Response for Service Type “Guaranteed”	136
Table 8.1:	Summary of actions when the RetransmissionTimer is in use and R=0	155
Table 8.2:	AMP Controller timeout values	169
Table 8.3:	Operators, connectives and statements used with variables...	172
Table 8.4:	XMIT state table.....	184
Table 8.5:	WAIT_F State Table	185
Table 8.6:	RECV_State table.....	186
Table 8.7:	REJ_SENT State table	189
Table 8.8:	SREJ_SENT State Table	193

Part B:

SERVICE DISCOVERY PROTOCOL (SDP) SPECIFICATION

Table 2.1:	Service Browsing Hierarchy.....	230
Table 3.1:	Data Element.....	231
Table 3.2:	Data Element Size	232

Part C:

GENERIC ACCESS PROFILE

Table 1.1:	Device Types	286
Table 1.2:	Requirements for device types	286
Table 2.1:	Physical Layer and Link Layer functionality for GAP roles when operating over an LE physical transport	292



Table 4.1:	Conformance requirements related to modes defined in this section	300
Table 4.2:	Page scan parameters for connection speed scenarios.....	305
Table 5.1:	Conformance requirements related to the generic authentication procedure and the security modes defined in this section.....	308
Table 5.2:	Simple Pairing after Authentication Failure.....	320
Table 5.3:	User Input Capabilities.....	321
Table 5.4:	User Output Capabilities.....	321
Table 5.5:	IO Capability mapping	321
Table 5.6:	IO and OOB capability mapping	322
Table 5.7:	IO Capability Mapping to Authentication Stage 1	323
Table 5.8:	Security Level mapping to link key requirements	326
Table 7.1:	Establishment procedures	337
Table 9.1:	Broadcast mode and Observation procedure Requirements...	348
Table 9.2:	Device discovery requirements.....	350
Table 9.3:	Connection modes and procedures requirements.....	357
Table 9.4:	Bonding Compliance Requirements	368
Table 10.1:	Requirements related to security modes and procedures	371
Table 10.2:	Local device responds to a service request	376
Table 10.3:	Requirements related to privacy feature.....	384
Table 12.1:	GAP Service Requirements	390
Table 12.2:	Requirements related to GAP Service characteristics.....	390
Table 12.3:	Device Name characteristic	391
Table 12.4:	Appearance characteristic	391
Table 12.5:	Peripheral Preferred Connection Parameters characteristic ...	392
Table 12.6:	Format of the preferred connection parameters structured data ...	392
Table 12.7:	Central Address Resolution Characteristic.....	393
Table 13.1:	Requirements for the modes of a BR/EDR/LE device type	394
Table 13.2:	Requirements for the bonding of a BR/EDR/LE device type ...	395
Table 14.1:	Requirements for the security aspects of a BR/EDR/LE device type	396
Table 15.1:	Requirements based on GAP Roles Supported	397
Table 15.2:	Requirements based on GAP Roles Supported	398
Table 15.3:	SDP Record for the Generic Access Profile	398
Table A.1:	Defined GAP timers.....	404

Part D:

TEST SUPPORT

Table 1.1:	L2CAP valid test protocol IDs	427
Table 1.2:	AMP Test Opcodes.....	429
Table 1.3:	AMP Command reasons for rejection.....	430

**Part E:****AMP MANAGER PROTOCOL SPECIFICATION**

Table 2.1:	AMP Manager Channel configuration parameters.....	445
Table 2.2:	AMP Manager Channel Enhanced Retransmission Mode parameters.....	445
Table 3.1:	Codes	449
Table 3.2:	Identifier	450
Table 3.3:	AMP Command Reject reasons	451
Table 3.4:	A2MP Extended Feature Mask.....	452
Table 3.5:	Controller Status	455
Table 3.6:	AMP Get Info Response Status values.....	459
Table 3.7:	AMP Get AMP Assoc Response Status values	461
Table 3.8:	AMP Create Physical Link Response Status values.....	463
Table 3.9:	AMP Disconnect Physical Link Response Status values.....	466

Part F:**ATTRIBUTE PROTOCOL (ATT)**

Table 3.1:	Format of Attribute PDU	478
Table 3.2:	Format of Error Response	481
Table 3.3:	Error Codes	481
Table 3.4:	Format of Exchange MTU Request	483
Table 3.5:	Format of Exchange MTU Response	483
Table 3.6:	Format of Find Information Request.....	485
Table 3.7:	Format of Find Information Response	485
Table 3.8:	Format field values	486
Table 3.9:	Format 0x01 - handle and 16-bit Bluetooth UUIDs.....	486
Table 3.10:	Format 0x02 - handle and 128-bit UUIDs.....	486
Table 3.11:	Format of Find By Type Value Request.....	487
Table 3.12:	Format of Find By Type Value Response	488
Table 3.13:	Format of the Handles Information	488
Table 3.14:	Format of Read By Type Request	489
Table 3.15:	Format of Read By Type Response	491
Table 3.16:	Format of the Attribute Data	491
Table 3.17:	Format of Read Request	492
Table 3.18:	Format of Read Response	492
Table 3.19:	Format of Read Blob Request	493
Table 3.20:	Format of Read Blob Response	494
Table 3.21:	Format of Read Multiple Request.....	495
Table 3.22:	Format of Read Multiple Response	496
Table 3.23:	Format of Read By Group Type Request	496
Table 3.24:	Format of Read By Group Type Response	498
Table 3.25:	Format of the Attribute Data	499



Table 3.26:	Format of Write Request.....	499
Table 3.27:	Format of Write Response.....	500
Table 3.28:	Format of Write Command	501
Table 3.29:	Format of Signed Write Command	502
Table 3.30:	Format of Prepare Write Request.....	503
Table 3.31:	Format of Prepare Write Response	505
Table 3.32:	Format of Execute Write Request.....	505
Table 3.33:	Format of Execute Write Response.....	506
Table 3.34:	Format of Handle Value Notification	507
Table 3.35:	Format of Handle Value Indication.....	507
Table 3.36:	Format of Handle Value Confirmation.....	508
Table 3.37:	Attribute Protocol Summary.....	509
Table 3.38:	Attribute Request and Response Summary	511

Part G:

GENERIC ATTRIBUTE PROFILE (GATT)

Table 3.1:	Service Declaration	530
Table 3.2:	Include Declaration	531
Table 3.3:	Characteristic declaration	532
Table 3.4:	Attribute Value Field in characteristic declaration	532
Table 3.5:	Characteristic Properties bit field	533
Table 3.6:	Characteristic Value declaration	534
Table 3.7:	Characteristic Extended Properties declaration.....	535
Table 3.8:	Characteristic Extended Properties bit field.....	535
Table 3.9:	Characteristic User Description declaration.....	536
Table 3.10:	Client Characteristic Configuration declaration.....	537
Table 3.11:	Client Characteristic Configuration bit field definition	537
Table 3.12:	Server Characteristic Configuration declaration	538
Table 3.13:	Server Characteristic Configuration bit field definition	538
Table 3.14:	Characteristic Format declaration.....	539
Table 3.15:	Characteristic Format Value definition	539
Table 3.16:	Characteristic Format types	540
Table 3.17:	Characteristic Aggregate Format declaration	542
Table 3.18:	Summary of GATT Profile Attribute types	543
Table 4.1:	GATT feature mapping to procedures	544
Table 4.2:	GATT Procedure mapping to ATT protocol opcodes	569
Table 5.1:	LE L2CAP ATT_MTU.....	573
Table 5.2:	Attribute Protocol Fixed channel configuration parameters	574
Table 7.1:	GATT Profile service characteristic support.....	577
Table 7.2:	Service Changed Characteristic declaration.....	577
Table 7.3:	Service Changed Characteristic Value declaration.....	578
Table 9.1:	SDP Record for the Generic Attribute Profile	581



Table A.1:	Example Attribute Server Attributes	583
------------	---	-----

Part H:

SECURITY MANAGER SPECIFICATION

Table 2.1:	Inputs to f4 for the different protocols	600
Table 2.2:	Inputs to f6 for the different protocols	602
Table 2.3:	User Input Capabilities.....	607
Table 2.4:	User Output Capabilities.....	607
Table 2.5:	I/O Capabilities Mapping.....	607
Table 2.6:	Rules for using Out-of-Band and MITM flags for LE legacy pairing	609
Table 2.7:	Rules for using Out-of-Band and MITM flags for LE Secure Connections Pairing	610
Table 2.8:	Mapping of IO Capabilities to Key Generation Method.....	610
Table 2.9:	Action after encryption setup failure	629
Table 3.1:	Security Manager Channel Configuration Parameters without LE Secure Connections	633
Table 3.2:	Security Manager Channel Configuration Parameters with LE Secure Connections	633
Table 3.3:	SMP Command Codes.....	634
Table 3.4:	IO Capability Values	636
Table 3.5:	OOB Data Present Values	636
Table 3.6:	Bonding Flags.....	637
Table 3.7:	Pairing Failed Reason Codes	642
Table 3.8:	Notification Type	645

