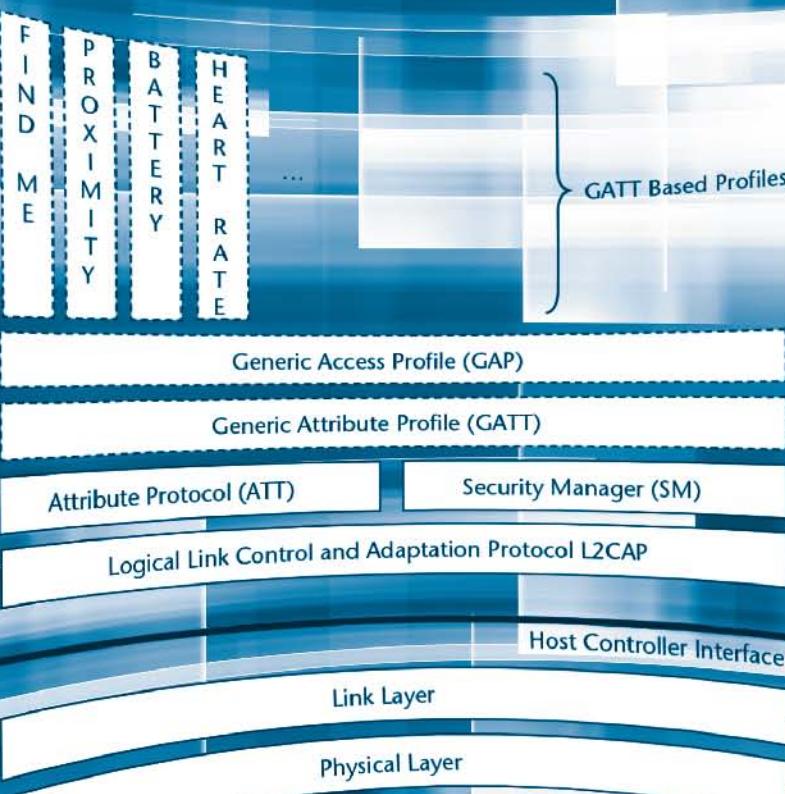




mobile communications series

Inside BLUETOOTH LOW ENERGY

Naresh Gupta



Inside Bluetooth Low Energy

For a listing of recent titles in the
Artech House Mobile Communications Series,
turn to the back of this book.

Inside Bluetooth Low Energy

Naresh Gupta



**ARTECH
H O U S E**

BOSTON | LONDON
artechhouse.com

Library of Congress Cataloging-in-Publication Data

A catalog record for this book is available from the U.S. Library of Congress.

British Library Cataloguing in Publication Data

A catalog record for this book is available from the British Library.

ISBN-13: 978-1-60807-579-9

Cover design by Vicki Kane

© 2013 Artech House

All rights reserved. Printed and bound in the United States of America. No part of this book may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without permission in writing from the publisher.

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Artech House cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

This book is in no way affiliated with SIG. The author is writing as an individual interested in the technology.

10 9 8 7 6 5 4 3 2 1

*To my respected parents, my dear wife, and my naughty kids.
You are always there for me, and have never doubted my dreams,
no matter how crazy they may sound.*

*I know I stole countless hours that I should have spent with you
in writing this book but you never complained...*

Contents

Preface	<i>xix</i>
Acknowledgments	<i>xxiii</i>
Foreword	<i>xxv</i>

CHAPTER 1

Introduction	1
1.1 Introduction to Wireless Communication	1
1.2 Data Rates and Coverage	2
1.2.1 Wide Area Network	2
1.2.2 Metropolitan Area Network	3
1.2.3 Local Area Network	3
1.2.4 Personal Area Network	3
1.2.5 Body Area Network	3
1.3 Why Have Standards?	4
1.4 Introduction to Bluetooth and Bluetooth Low Energy	6
1.5 Applications	8
1.5.1 Finding and Alerting Devices	8
1.5.2 Proximity and Presence Detection	9
1.5.3 Health Care	10
1.5.4 Sports and Fitness Equipment	10
1.5.5 Mobile Payments	11
1.6 Competing Technologies	12
1.6.1 ANT and ANT+	12
1.6.2 ZigBee	12
1.6.3 Near Field Communication (NFC)	13
1.7 Summary	14
References	14

CHAPTER 2

Background of Bluetooth	15
2.1 Introduction	15
2.2 Ad Hoc Networks—Why?	15
2.2.1 Printing Documents, Photos	15
2.2.2 Exchanging Business Cards, Photos, Music, Files	16

2.3	What is Bluetooth?	16
2.4	Bluetooth SIG	17
2.5	History of the Bluetooth Specification	19
2.6	IEEE 802.15 Family of Specifications	19
2.7	Bluetooth Basics	19
2.8	Bluetooth Architecture Overview	21
2.9	Basic Terminology	24
2.9.1	Host, Host Controller, and Host Controller Interface (HCI)	24
2.9.2	Device Address (BD_ADDR) and Device Name	26
2.9.3	Class of Device (CoD)	27
2.9.4	Bluetooth Clock	27
2.9.5	Protocol Data Unit (PDU)	28
2.10	Data Rates	28
2.11	Connection Setup and Topology	29
2.12	IEEE Word Usage	30
2.13	Summary	31
	References	32

CHAPTER 3

	Bluetooth Lower Layers	33
3.1	Introduction	33
3.2	Bluetooth Radio	33
3.2.1	Frequency Band and Hopping	34
3.3	Baseband Controller	35
3.3.1	Topology—Piconet and Scatternet	35
3.3.2	Time Division Duplex	36
3.3.3	Adaptive Frequency Hopping (AFH)	36
3.3.4	Master, Slave Roles and Role Switch	39
3.3.5	Channel, Transport and Links	39
3.3.6	Packet Format	46
3.3.7	Packet Types	47
3.3.8	Link Controller States	50
3.4	Link Manager (LM)	56
3.4.1	Connection Control	57
3.4.2	Security	58
3.5	Host Controller Interface (HCI)	60
3.5.1	HCI Packet Types	61
3.5.2	HCI Commands and Events	63
3.5.3	Buffers	65
3.5.4	HCI Flow Control	65
3.5.5	Connection Handle	68
3.5.6	HCI Transport Layer	68
3.6	Security—Secure Simple Pairing (SSP)	71
3.6.1	Passive Eavesdropping Protection	71
3.6.2	Man-in-the-Middle (MITM) Attack Protection	71
3.6.3	Association Models	72

3.7	Practical Scenarios	73
3.7.1	Inquiry	74
3.7.2	Connection Establishment	75
3.8	Summary	76
	Reference	77

CHAPTER 4

	Bluetooth Upper Layers and Profiles	79
4.1	Introduction	79
4.2	Logical Link Control and Adaptation Protocol (L2CAP)	80
4.2.1	Modes of operation	82
4.2.2	L2CAP PDUs	82
4.2.3	L2CAP Features	83
4.3.3	L2CAP Signaling	86
4.4	Service Discovery Protocol (SDP)	88
4.4.1	Service Record, Service Attributes and Service Class	89
4.4.2	Searching and Browsing Services	90
4.4.3	SDP Transactions	91
4.5	RFCOMM	93
4.6	Object Exchange Protocol (OBEX)	96
4.6.1	OBEX Operations	96
4.7	Audio/Video Control Transport Protocol (AVCTP)	97
4.8	Audio/Video Distribution Transport Protocol (AVDTP)	98
4.9	Profiles	100
4.10	Generic Access Profile (GAP)	100
4.10.1	Bluetooth Parameters Representation	102
4.10.2	Modes	103
4.10.3	Idle Mode Procedures	104
4.10.4	Established Procedures	105
4.10.5	Authentication	106
4.10.6	Security	106
4.11	Serial Port Profile (SPP)	107
4.12	Headset Profile, Hands-Free Profile	107
4.13	Generic Object Exchange Profile (GOEP)	110
4.14	Object Push Profile (OPP)	111
4.15	File Transfer Profile (FTP)	112
4.16	Generic Audio/Video Distribution Profile (GAVDP)	113
4.17	Advanced Audio Distribution Profile (A2DP)	114
4.18	Audio/Video Remote Control Profile (AVRCP)	116
4.19	Summary	117
	Bibliography	117

CHAPTER 5

	Getting the Hands Wet	119
5.1	Introduction	119
5.2	Ingredients	119

5.3	Basic Bluetooth Operations	120
5.3.1	Enabling and Disabling Bluetooth	120
5.3.2	Discovering Devices	121
5.3.3	Browsing Services	121
5.4	Real World Application—Café Bluebite	122
5.4.1	Requirements Specification	122
5.4.2	High Level Design	123
5.4.3	Code	123
5.4.4	Complete Code	128
5.5	Disclaimer	130
5.6	Summary	130
	Bibliography	130

CHAPTER 6

	Bluetooth Low Energy—Fundamentals	131
6.1	Introduction	131
6.2	Single Mode versus Dual Mode Devices	132
6.3	Bluetooth Smart Marks	133
6.3.1	Bluetooth Smart (Sensor-Type Devices)	134
6.3.2	Bluetooth Smart Ready (Hubs)	134
6.4	LE Fundamentals	135
6.4.1	Frequency Bands	135
6.4.2	Mostly Off Technology	136
6.4.3	Faster Connections	136
6.4.4	Reduced Functionality	136
6.4.5	Shorter Packets	137
6.4.6	Reduced Dynamic Memory Footprint	138
6.4.7	Optimized Power Consumption of Peripherals	139
6.4.8	No Need for Continuous Polling	139
6.4.9	Backward Compatibility With BR/EDR	139
6.5	LE Architecture	140
6.6	Comparison between BR/EDR and LE	141
6.7	Summary	141
	Bibliography	143

CHAPTER 7

	Physical Layer	145
7.1	Introduction	145
7.2	Frequency Bands	145
7.3	Transmitter Only, Receiver Only, or Both	146
7.4	Output Power	147
7.5	Range	147
7.6	Modulation Characteristics	147
7.7	LE Timeline	149
7.8	Summary	149
	Bibliography	150

CHAPTER 8

Link Layer	151
8.1 Introduction	151
8.2 Overview of Link Layer States	151
8.2.1 Standby State	151
8.2.2 Advertising State (Advertiser)	152
8.2.3 Scanning State (Scanner)	153
8.2.4 Initiating State (Initiator)	153
8.2.5 Connection State (Master or Slave)	153
8.3 Device Address	153
8.3.1 Public Device Address	153
8.3.2 Random Address	154
8.4 Physical Channel	154
8.5 Channel Map	156
8.6 Adaptive Frequency Hopping	156
8.7 Events	157
8.7.1 Advertising Events	157
8.7.2 Connection Events	158
8.8 Topology	159
8.9 Packet Format	160
8.9.1 Preamble	161
8.9.2 Access Address	161
8.9.3 CRC	161
8.9.4 PDU	161
8.10 Bit Stream Processing	165
8.11 Link Layer States	166
8.11.1 Nonconnected States	167
8.11.2 Connection State	173
8.12 Link Layer Control Procedures	176
8.12.1 Connection Update Procedure	176
8.12.2 Channel Map Update Procedure	177
8.12.3 Encryption Procedure	178
8.12.4 Feature Exchange Procedure	180
8.12.5 Version Exchange Procedure	180
8.12.6 Termination Procedure	181
8.12.7 Device Filtering and White List	181
8.13 Practical Examples	183
8.14 Summary	183
Bibliography	184

CHAPTER 9

Host Controller Interface and Commands	185
9.1 Introduction	185
9.1.1 HCI Packet Types	186
9.1.2 HCI Command Packets	186

9.1.3	HCI Event Packet	186
9.1.4	HCI ACL Data Packet	186
9.2	HCI Commands and Events	187
9.2.1	Device Setup	189
9.2.2	Controller Flow Control	189
9.2.3	Host Flow Control	191
9.2.4	Controller Information	191
9.2.5	Remote Information	192
9.2.6	Controller Configuration	192
9.2.7	Device Discovery	193
9.2.8	Connection Setup	193
9.2.9	Connection State	193
9.2.10	Physical Links	194
9.2.11	Link Information	194
9.2.12	Authentication and Encryption	194
9.2.13	Testing	195
9.2.14	Usage of White Lists	195
9.3	Practical Sequence Diagrams	196
9.3.1	Passive Scanning	197
9.3.2	Typical Sequence for Active Scanning	197
9.3.3	Connection Establishment	198
9.3.4	Setting up White list	198
9.4	Summary	199
	Bibliography	199

CHAPTER 10

	Logical Link Control and Adaptation Protocol (L2CAP)	201
10.1	Introduction	201
10.2	PDU and SDU	201
10.3	Basic Assumptions	202
10.4	Maximum Transmission Unit (MTU)	202
10.5	L2CAP Features	203
10.5.1	Fixed Channel Identifiers	203
10.5.2	Fragmentation and Defragmentation of Data	204
10.5.3	Channel Multiplexing	204
10.6	Data Packets	205
10.7	L2CAP Parameters	205
10.8	L2CAP Signaling	206
10.8.1	Command Reject	207
10.8.2	Connection Parameter Update Request	207
10.8.3	Connection Parameter Update Response	207
10.9	Practical Examples	208
10.10	Summary	209
	Bibliography	209

CHAPTER 11

Security Manager (SM)	211
11.1 Introduction	211
11.2 Security in Host Instead of Controller	211
11.3 Asymmetrical Architecture	212
11.4 Security Breaches	212
11.4.1 Passive Eavesdropping	212
11.4.2 MITM (Active Eavesdropping)	213
11.4.3 Tracking	214
11.5 Pairing Methods	214
11.5.1 Just Works	214
11.5.2 Passkey Entry	214
11.5.3 Out of Band	215
11.6 Security Properties	215
11.6.1 Authenticated MITM Protection	215
11.6.2 Unauthenticated no MITM Protection	215
11.6.3 No Security	216
11.7 Cryptographic Functions	216
11.7.1 Security Function e	216
11.7.2 Random Address Function ah	216
11.7.3 Confirm Value Generation Function c1	216
11.7.4 Key Generation Function s1	216
11.8 Pairing	217
11.8.1 Phase 1: Pairing Feature Exchange	217
11.8.2 Phase 2: Short Term Key (STK) Generation	219
11.8.3 Phase 3: Transport Specific Key Distribution	220
11.9 Security Manager Protocol	222
11.9.1 Commands Used During Phase 1 (Pairing Feature Exchange)	222
11.9.2 Commands Used During Phase 2 (STK Generation)	224
11.9.3 Commands Used During Phase 3 (Transport Specific Key Distribution)	226
11.10 Practical Examples	227
11.11 Summary	230
Bibliography	231

CHAPTER 12

Attribute Protocol (ATT)	233
12.1 Introduction	233
12.2 Attribute	233
12.2.1 Attribute Type	234
12.2.2 Attribute Handle	235
12.2.3 Attribute Permissions	235
12.2.4 Attribute Value	236
12.2.5 Control Point Attributes	236

12.2.6	Grouping of Attribute Handles	237
12.2.7	Atomic Operations	237
12.3	Attribute Protocol	237
12.3.1	PDU Format	238
12.3.2	Sequential Transactions	239
12.4	Methods	240
12.4.1	Request and Response Type Methods	240
12.4.2	Command Type Methods	250
12.4.3	Notification Type Methods	252
12.4.4	Indication and Confirmation Type Methods	252
12.5	Practical Examples	253
12.5.1	Exchange MTU	253
12.5.2	Reading Primary Services of a Device	254
12.6	Summary	257
	Bibliography	258

CHAPTER 13

Generic Attribute Profile (GATT)	259	
13.1	Introduction	259
13.1.1	Profile Dependencies	260
13.1.2	GATT-Based Profile Architecture	261
13.2	Roles	266
13.3	Attributes	267
13.3.1	Attribute Caching	269
13.3.2	Attribute Grouping	270
13.3.3	Notification and Indication	270
13.4	Service Definition	271
13.4.1	Service Declaration	271
13.4.2	Include Definition	271
13.4.3	Characteristic Definition	271
13.5	Configured Broadcast	275
13.6	GATT Features	275
13.6.1	Server Configuration	275
13.6.2	Primary Service Discovery	276
13.6.3	Relationship Discovery	280
13.6.4	Characteristic Discovery	281
13.6.5	Characteristic Descriptor Discovery	286
13.6.6	Characteristic Value Read	288
13.6.7	Characteristic Value Write	292
13.6.8	Characteristic Value Notification	297
13.6.9	Characteristic Value Indication	297
13.6.10	Characteristic Descriptors	298
13.7	Timeouts	301
13.8	GATT Service	301
13.8.1	Service Changed Characteristic	302

13.9	Security Considerations	304
13.9.1	Authentication and Authorization Requirements	304
13.10	Summary	304
	Bibliography	304

CHAPTER 14

	Generic Access Profile	305
14.1	Introduction	305
14.2	Roles	306
14.2.1	Broadcaster Role	306
14.2.2	Observer Role	306
14.2.3	Peripheral Role	306
14.2.4	Central Role	307
14.3	Representation of Bluetooth Parameters	307
14.3.1	Bluetooth Device Address	307
14.3.2	Bluetooth Device Name	308
14.3.3	Bluetooth Passkey	308
14.3.4	Bluetooth Class of Device	308
14.3.5	Pairing—Authentication and Bonding	308
14.4	Advertising and Scan Response Data Format	309
14.4.1	Local Name (AD Type = 0x08 or 0x09)	310
14.4.2	Flags (AD Type = 0x01)	310
14.4.3	Manufacturer Specific Data (AD Type = 0xFF)	310
14.5	GAP Characteristics	311
14.5.1	Device Name Characteristic	311
14.5.2	Appearance Characteristic	312
14.5.3	Peripheral Privacy Flag Characteristic	313
14.5.4	Reconnection Address Characteristic	313
14.5.5	Peripheral Preferred Connection Parameters Characteristic	314
14.6	Operational Modes and Procedures	315
14.6.1	Broadcast Mode and Observation Procedure	315
14.6.2	Discovery Modes and Procedures	316
14.6.3	Connection Modes and Procedures	319
14.6.4	Bonding Modes and Procedures	325
14.7	Security	327
14.7.1	LE Security Mode 1	328
14.7.2	LE Security Mode 2	328
14.7.3	Authentication Procedure	328
14.7.4	Authorization Procedure	329
14.7.5	Encryption Procedure	329
14.7.6	Data Signing	329
14.7.7	Privacy Feature	329
14.7.8	Random Device Address	331
14.8	Summary	331
	Bibliography	332

CHAPTER 15

GATT-Based Profiles	333
15.1 Introduction	333
15.2 Profile, Services, and Characteristics	333
15.3 Immediate Alert Service (IAS)	335
15.3.1 Service Declaration	335
15.3.2 Service Characteristics	335
15.4 Find Me Profile (FMP)	336
15.4.1 Roles	336
15.5 Link Loss Service (LLS)	337
15.5.1 Service Declaration	337
15.5.2 Service Characteristics	337
15.6 Transmit Power Service (TPS)	338
15.6.1 Service Declaration	339
15.6.2 Service Characteristics	339
15.7 Proximity Profile (PXP)	339
15.7.1 Roles	339
15.8 Battery Service (BAS)	339
15.8.1 Service Declaration	340
15.8.2 Service Characteristics	340
15.9 Device Information Service (DIS)	340
15.9.1 Service Declaration	340
15.9.2 Service Characteristics	340
15.10 Current Time Service (CTS)	341
15.10.1 Service Declaration	341
15.10.2 Service Characteristics	341
15.11 Health Thermometer Service (HTS)	341
15.11.1 Service Declaration	341
15.11.2 Service Characteristics	341
15.12 Health Thermometer Profile (HTP)	343
15.12.1 Roles	343
15.13 Blood Pressure Service (BPS)	343
15.13.1 Service Declaration	343
15.13.2 Service Characteristics	343
15.14 Blood Pressure Profile (BLP)	344
15.14.1 Roles	344
15.15 Health, Sports and Fitness Profiles	344
15.16 Other Services and Profiles	345
15.17 Practical Examples	345
15.18 Summary	348
Bibliography	348

CHAPTER 16

Developing LE Applications	349
16.1 Introduction	349

16.2	Ingredients	350
16.2.1	Installing hcidump	350
16.2.2	Basic Bluetooth operations	352
16.3	Advertising and Scanning	356
16.4	Creating a Connection	358
16.5	GATT Operations	358
16.5.1	Enable GATT Functionality on Server	358
16.5.2	Execute GATT Procedures from the Client	360
16.5.3	Reading and Writing Characteristics	366
16.6	Disconnecting	368
16.6.1	Disconnecting the GATT Connection	368
16.6.2	Disconnecting the LE Connection	369
16.7	Real-World Application—Find Lost Keys	369
16.8	Debugging LE Applications	370
16.8.1	Logging the HCI Interface	370
16.8.2	Air Sniffer	371
16.8.3	Peer Devices and Interoperability Testing	371
16.8.4	Profile Tuning Suite (PTS)	371
16.9	Disclaimer	372
16.10	Summary	372
	References	372

CHAPTER 17

Testing and Qualification	373	
17.1	Introduction	373
17.2	Need for Bluetooth Qualification	373
17.3	What Is Interoperability?	374
17.4	Development Resources and Events	374
17.4.1	UnPlugFest (UPF)	374
17.4.2	Automotive Events	375
17.4.3	SIG Device Library	375
17.4.4	Profile Tuning Suite (PTS)	376
17.5	Bluetooth Qualification Program	376
17.6	Test Categories	378
17.6.1	BQTF, BRTF	378
17.6.2	BQE	378
17.6.3	Test Documents	379
17.6.4	Prequalification	380
17.7	Summary	380
	Bibliography	380
	Glossary of Acronyms and Important Terms	381
	About the Author	385
	Index	387

Preface

The idea for writing this book was sparked by a former colleague over a cup of coffee a couple of years ago when he said, “Naresh, why don’t you write a book on Bluetooth Low Energy?” At that time I brushed off that idea, saying that I was too busy in implementing Bluetooth Low Energy features and writing a book was not a piece of cake. Later that year I kept realizing that, since the technology is very new, the amount of available reading material is very limited and it’s not too easy to understand the technology, especially for newbies. In the winter of 2011, I finally decided to try my hand at explaining this technology to people who were eager to understand but did not have a good starting point.

Objectives of This Book

This book covers the concepts of Bluetooth and Bluetooth Low Energy. It introduces the reader to the history of the technology, terminology, use cases, architecture, and details of how it works, along with some hands-on examples. It’s not intended to cover everything down to the minute detail that is written in the Bluetooth specification because that is already done well by the specification. Rather, it’s expected to be a working companion to the specification. Instead of diving directly into the full specification, the user can first understand the technology at a broad level from this book and then dive deeper into the relevant sections of the specification.

This book does not assume prior knowledge of Bluetooth or other wireless protocols, though knowing them would certainly be a plus. Most of the new terminology used in this book is also explained within this book. A lot of practical examples are provided so that the concepts can be correlated to the real-world applications. Some sample programs are also provided that can be used to gain understanding and as a starting point of a full-fledged implementation. Screenshots of air logs and message sequence charts are also provided to give a good view of how things actually work.

It will be good if the reader is familiar with the Unix or Linux operating systems to understand the sample programs provided in this book. Some basic knowledge of any scripting language would also be helpful. The commands and programs provided in this book have been written directly on the Linux shell prompt or using very basic features of Perl or C language. It’s expected that these programs could be adapted easily to any other language as well.

Intended Audience

This book is meant for engineering students, software and hardware engineers, architects, and engineering and business managers.

This book serves the purpose of introducing engineering students to the concepts of Bluetooth and Bluetooth Low Energy. It helps them to understand the “what” and “how” parts, including a broad view of the technology, the various building blocks, and how they come together. It also has practical exercises to get a first-hand feel for how the technology works. Once they understand the concepts, users can dig into the specification for an in-depth understanding of their area of interest.

For software engineers, hardware engineers, and architects, this book helps them to understand the architecture of the Bluetooth Low Energy Stack and the functionality provided by each of the layers. The book also discusses the enhancements that have been made in Bluetooth Low Energy compared to previous versions of Bluetooth so that readers can appreciate why this technology leads to such huge savings in power consumption. It guides them on setting up their own system in a quick and efficient manner with inexpensive, easily available hardware, and a couple of PCs running Linux. The sample programs help users understand the “how” part of the technology. The book then builds further on the concepts by going from simple operations and programs to more complex programs that can be used as a starting point for the reader’s own implementation.

For engineering and business managers, this book helps them to understand the “why” part. Why should they choose this particular technology? What does it offer them as a USP (Unique Selling Proposition)? Is it the right technology to solve their business problem? They may not have the time to dig into the deep technical intricacies of the technology before making a decision. This book aids them by quickly introducing them to the technology and providing them sufficient information to make an informed decision.

Prerequisite Knowledge

Bluetooth Low Energy is expected to be the next inflection point for Bluetooth technology and is expected to pave the way for billions of devices in the future. These devices will need software and hardware implementation, a variety of applications to realize the full potential of the devices, tools to support the implementation, tools to test the implementation, and several more building blocks. It is expected that many of the people who are implementing these building blocks would be exposed directly to Bluetooth Low Energy without having previously worked on Bluetooth.

This book does not assume prior knowledge of Bluetooth or any other wireless technology. Rather, it has a dedicated set of four chapters to explain Bluetooth technology before moving on to Bluetooth Low Energy technology. So it should serve both as a good starting point to people new to Bluetooth and a good refresher for people already familiar with Bluetooth who are seeking to understand what is new in Bluetooth Low Energy.

The book also contains several programming examples. All programs illustrated in this book have been tested on a Linux system using the BlueZ protocol

stack. BlueZ is the “official Linux Bluetooth protocol stack.” Support for BlueZ can be found in many Linux distributions, and in general it is compatible with any Linux system in the market. As far as possible, these examples are written using very simple commands and scripts that can be executed with the bash shell. Some familiarity with Linux and shell commands will help the reader in understanding these programs faster.

Organization of This Book

This book is organized into five parts:

Part 1 starts with the background of wireless technologies and then introduces Bluetooth and Bluetooth Low Energy. It then illustrates some of the use case scenarios of Bluetooth Low Energy and introduces some of the competing technologies. It comprises the following chapter:

Chapter 1: Introduction

Part 2 sets the ball rolling by explaining the Bluetooth technology. It serves as a good introduction for readers who are not familiar with Bluetooth. It serves as a refresher to readers who have some familiarity with Bluetooth but not a full understanding of how it works. It may be skipped by people who are Bluetooth experts. This part explains the fundamentals of Bluetooth and the architecture. It then moves on to explaining the Bluetooth protocol stack from bottom to top, including the profiles and use cases. Practical examples, message sequence charts, and air sniffer logs have been provided to give a view of how each component works and to show how the various components work together to support end-to-end real-world scenarios. This part concludes with a practical chapter on setting up the Bluetooth development environment and step-by-step development of a Bluetooth real-world application. The application, fictitiously named Café Bluebite, illustrates how simple Bluetooth commands can be used to implement very powerful uses. It comprises the following chapters:

Chapter 2: Background of Bluetooth

Chapter 3: Bluetooth Lower Layers

Chapter 4: Bluetooth Upper Layers and Profiles

Chapter 5: Getting the Hands Wet

Part 3 focuses on the lower layers of Bluetooth Low Energy. It starts with an introduction of Bluetooth Low Energy, and then explains single- and dual-mode devices followed by some of the fundamental concepts of Bluetooth Low Energy. It moves on to explain the Bluetooth Low Energy architecture. Here again a bottom to top approach is taken and the protocol stack is explained starting from the lowest layers. During the explanation of each layer, the main enhancements (compared to Bluetooth) are explained along with how the technology leads to drastic reductions in power consumption. This part contains a good number of sequence diagrams, air sniffer captures, and examples of how the various layers work together. It comprises the following chapters:

Chapter 6: Bluetooth Low Energy—Fundamentals

Chapter 7: Physical Layer

Chapter 8: Link Layer

Chapter 9: Host Controller Interface and Commands

Part 4 focuses on the upper layers of Bluetooth Low Energy. It continues where Part 3 left off and starts with a discussion of the L2CAP layer and the main differences from the L2CAP layer of prior Bluetooth versions. Security is an important part of any wireless system and can be a major criterion behind the success or failure of any wireless technology. This part explains the Security Manager in detail and how Bluetooth helps to prevent various possible security breaches. This part moves on to explaining the concept of attributes and Attribute Protocol. Attribute Protocol provides the building blocks for profiles and services that are explained in the chapter related to Generic Attribute Profile. This part then explains the Generic Access Profile. It is one of the basic profiles in the Bluetooth world and provides services to all other profiles. Bluetooth Low Energy defines a very simple architecture for GATT-based profiles. This part explains the GATT-based profiles in detail along with an end-to-end explanation of what happens behind the scenes when a profile is active. It finally includes a chapter on how to develop basic Bluetooth Low Energy applications. This chapter starts with some basic LE operations and then goes on to discuss the development of interesting and useful programs to understand the different concepts that were explained earlier. Finally, it provides some tips and tricks on how to debug LE applications. It comprises the following chapters:

Chapter 10: Logical Link Control and Adaptation Protocol (L2CAP)

Chapter 11: Security Manager (SM)

Chapter 12: Attribute Protocol (ATT)

Chapter 13: Generic Attribute Profile (GATT)

Chapter 14: Generic Access Profile (GAP)

Chapter 15: GATT-Based Profiles

Chapter 16: Developing LE Applications

Part 5 provides an introduction to the Bluetooth testing and qualification process. It introduces the various tools that are available from the Bluetooth SIG for testing. It also gives an overview of the various events organized from time to time by the Bluetooth SIG and other organizations where engineers get an opportunity to test their implementations. Such events are very useful from an interoperability perspective to ensure that the device works seamlessly with other devices that are currently available in the market as well as with future devices. The overview of qualification helps the reader to understand the mandatory requirements that any Bluetooth device has to comply with before it can be sold in the market. It comprises the following chapter:

Chapter 17: Testing and Qualification

In addition a complete glossary of acronyms and important terms is provided to serve as a quick reference.

Acknowledgments

First and foremost, I would like to thank my former colleagues at ST-Ericsson. It was a great opportunity to learn, experiment, and innovate during almost 9 years with ST-Ericsson. Along with my team members, I would like to thank my former supervisors, Davy Jacops and Dr. Aloknath De. Under their able guidance I was able to grow my team from a handful of people to a reasonably large team working on the latest evolutions in various connectivity areas.

I would like to thank my former colleague Balvinder Pal Singh for being a critical reviewer of my work. His eye for detail and out-of-the-box thinking always amazed me. He gave several excellent suggestions to make this book much more useful to newbies as well as Bluetooth experts.

Next, I would like to thank Frontline, in particular Tomas O'Raghallaigh, for allowing me to use the screenshots captured from the Frontline ComProbe Protocol Analysis System software and sharing some of the sample air logs for Bluetooth transactions. Information about Frontline products related to Bluetooth Low Energy may be found at <http://www.fte.com/lowenergy>.

I would also like to thank Artech House for their support during the whole process. Special thanks to Aileen Storry for constantly reminding me of the schedule and responding so promptly to my e-mails.

On the personal side, I would like to thank Ekta, my wife, for her unwavering support. Almost all my weekends and vacation time for more than a year went into writing, rewriting, revising, and then re-revising the book. She was always understanding and the one who would quietly set the 4:00 AM alarm without telling me so that I could finish the book as soon as possible.

Thanks as well to Vishesh and Twinkle. I never realized before I started this book that they have grown old enough to actually edit and put final touches to some of the figures in this book. Thanks to them, I had to spend less time on refining the figures and could focus on getting the material organized. Thanks as well to sweet little Onashi for not pressing the only button on my laptop that she likes—the shutdown button.

My father, Mr. A. K. Gupta, is an avid computer and Bluetooth user but not a wireless expert. For him technologies like Bluetooth should just work intuitively without knowing the technical jargon. He read the book cover-to-cover several times. He helped me to add sufficient background information for people new to

wireless technologies and Bluetooth to help them understand the technical jargon and what is happening behind the scenes when they trigger a Bluetooth operation. I would like to thank my father for his very useful comments on how to make this book more readable and understandable for all.

Foreword

Bluetooth (BT) technology has become all-pervasive, with attach rates close to one hundred percent for mobiles and laptops. Bluetooth Low Energy (BLE) is the next growth area that leverages on the success of BT but caters to the applications where frequent battery charging is not an option. The lower power consumption in BLE is not achieved by the nature of the active radio transport, but by the design of the protocol to allow low duty cycles with burst transmission and by the use cases envisaged.

Many features of classic BT are inherited in BLE technology, including the broad architecture of the protocol stack. Data transfer rates for BLE technology are below 100 kbps, and also many profiles (including object exchange and audio/video distribution) are not offered in BLE in order to keep the power consumptions low. On the positive side, a master device could support a large number of slave devices, and the connection setup is pretty fast. Because a BLE device is in sleep mode the majority of the time and the communication is “bursty,” the average power consumption is reduced to significantly low levels.

This book unravels the beauty and subtlety of BLE technology and contrasts this technology vis-a-vis classic BT technology. Naresh has been working on BT and short-range connectivity area for close to ten years; his vast developmental experience has a footprint throughout the book. He has a focused approach in discussing BLE profiles, ATT, GATT, GAP, security aspects, HCI commands, development tools, and debugging and testing mechanisms. At the same time, he has captured well the BLE architectural aspects and reference design relating to standard specifications.

The book serves as a practical guide to promising BLE technology that is well-suited for sensors, actuators, and other small devices with ultra low power consumption. BT 4.0 with low energy technique paves the way for BT connected devices and the Internet of Things. Opening a garage wirelessly, receiving alerts to watch for incoming calls, or finding lost keys (and even cats)—any of these applications could easily be built with BLE technology. The book provides many such application examples and the underlying working principles so that practicing engineers could learn how to build innovative applications on BT-smart and smart-ready devices.

It has been a pleasure for me to pen this foreword to this book on BLE authored by Naresh. This is particularly so, as I hired Naresh to build the Bluetooth team, and the team increased both in mandate and strength under my guidance in ST-Ericsson, where I was serving as Country Director until recently. Naresh and his colleagues have filed three patents from their inventive work. A good deal of this material has also been presented as classroom lectures to Masters degree students at Jaypee University, as well as at the Indian Institute of Technology-Delhi, where I served as Adjunct Professor. I sincerely believe that this timely book will immensely help students and engineers alike to get a holistic view of this promising technology.

*Aloknath De, Ph.D.
Fellow-Indian National Academy of Engineering*

Introduction

1.1 Introduction to Wireless Communication

Wireless communication involves any form of communication without using wires. Information may be transferred between two or more points that are not physically connected. The distance between the points could be as short as a few centimeters or meters like in the case of near field communication and remote controls to as long as thousands of kilometers in the case of Global Positioning System (GPS) and deep space communications.

Wireless communication has existed for ages. In early days people used smoke signals to communicate over long distances and pass encoded messages. Carrier pigeons were used to deliver written messages. These were also forms of wireless communication and were used much before wired communication was invented.

In modern times, wireless communication is found almost everywhere. We are surrounded by devices like mobile phones, remote controls, wireless keyboards, headsets, FM radios, satellite television, broadcast television, cordless phones, sports equipment, wireless toys, GPS units, key fobs, smartcards, and many more. All these devices use wireless communication to exchange data.

Wireless communication has changed the dynamics of how people work and how they communicate. People no longer needed to be constrained by wires to exchange information. Some of the major benefits that wireless communications provide include the following:

- *Mobility:* The communication is not restricted by the length of the wires. Wireless devices can be easily moved around offering users the flexibility to move while still being connected.
- *Convenience:* It is much more convenient to use devices like remote controls, cordless phones, and keyboards compared to their wired counterparts. Users are freed up from clumsy wires that hamper movement.
- *Almost zero setup time:* In most cases, setting up a wireless device involves just switching it on and using it. There are no cables to set up.

- *Reduced cost:* No cost for setting up expensive cabled infrastructure. For example, in many countries the mobile connections have now surpassed the landline connections because of reduced infrastructure costs.
- *Connectivity in remote areas:* It is much easier to provide connectivity in remote areas like deserts, oceans, forests, and remote villages using wireless technologies compared to building wired infrastructures. In many cases, wired infrastructures may not be even feasible.
- *Enhanced productivity:* Wireless communication improves productivity for a mobile workforce by providing the possibility to stay connected to the workplaces while on the move. Workers, students, and professionals can stay connected and work at their convenience thus enhancing productivity.
- *Never possibilities:* Wireless communication makes possible scenarios like deep space communication, satellite navigation, location-based services, and communication in remote locations, which are not possible with wired communication.
- *Safety and security:* Wireless communications make emergency services like E911 possible where users can seek help from any remote location and can be tracked and assisted in case of emergency.
- *No compromises on speed:* Wireless communications offer almost equivalent and sometimes higher speeds compared to the wired counterparts.

1.2 Data Rates and Coverage

Wireless technologies find applications in various scenarios involving data transfer rates ranging from a few kilobits per second (Kbps) to several gigabits per second (Gbps). At the lower end of the data rate spectrum are technologies like NFC, Bluetooth Low Energy, Zigbee, radio frequency identification (RFID), and ANT that offer data rates in the range of few hundred Kbps or less. At the higher end there are technologies like wireless USB, WiFi, and ultrawideband (UWB) that may offer data rates in the range of several hundred Mbps to even few Gbps.

Figure 1.1 shows the typical data range and coverage of various wireless technologies. Each technology is suited to a particular set of applications that need specific data rates and coverage areas.

Wireless (and wired) technologies can be classified into several categories depending on the distances over which these are used. Some of these categories are outlined below.

1.2.1 Wide Area Network

A wide area network (WAN) covers a broad geographical distance that can range from a few kilometers to thousands of kilometers. This includes technologies such as GSM (originally Groupe Spécial Mobile, also referred to as Global System for Mobile Communications), general packet radio service (GPRS), 3G, and code division multiple access (CDMA).

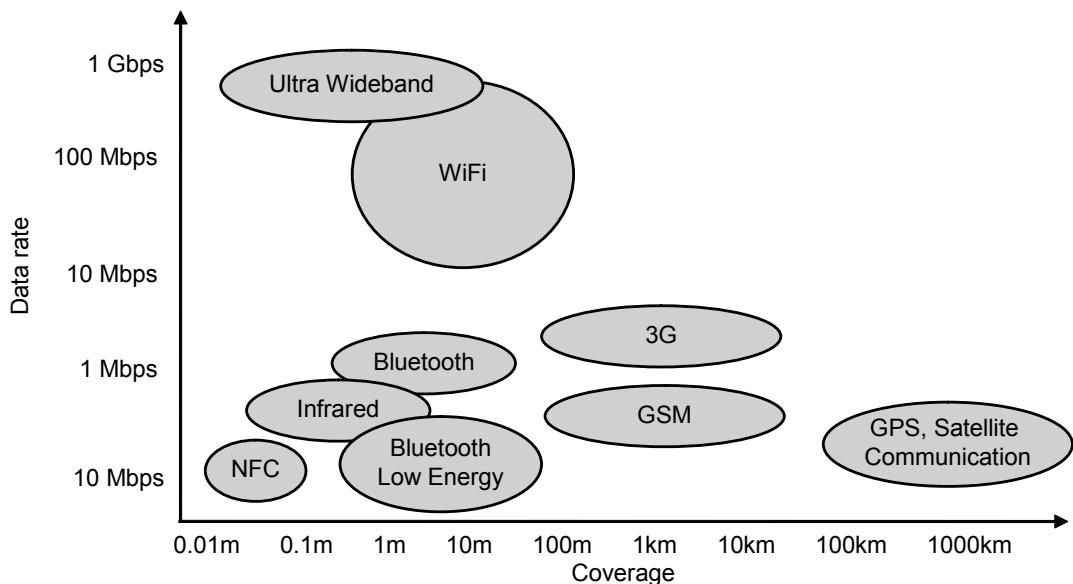


Figure 1.1 Typical data rates and coverage of various wireless technologies.

1.2.2 Metropolitan Area Network

A metropolitan area network (MAN) covers a smaller area than a WAN. Typically this could be a small town or a metropolis. This includes technologies like Worldwide Interoperability for Microwave Access (WiMAX) that cover several kilometers.

1.2.3 Local Area Network

A local area network (LAN) typically covers distances in the range of a few hundred meters. This can be a building, a campus, and so forth. This includes technologies such as WiFi.

1.2.4 Personal Area Network

A personal area network (PAN) encompasses communication between personal devices like mobile phones, PDAs, remote controls, keyboards, printers, and cameras. The range of PANs is typically a few meters. This includes technologies like Infrared Data Association (IrDA), Bluetooth, Wireless Universal Serial Bus (USB), Near Field Communication (NFC), and Zigbee.

1.2.5 Body Area Network

A body area network (BAN) encompasses communication between devices that are supposed to be carried or worn by people. These include devices like mobile phones, headsets, watches, and fitness equipment like blood pressure and heart rate sensors. This includes technologies like Bluetooth, RFID, NFC, and ANT.

1.3 Why Have Standards?

Industry standards play a vital role in research and development, worldwide adoption, standardization of products, ensuring interoperability of products from one vendor to another and protection from patent infringement. Several organizations have come together to form task forces and standards bodies to contribute to the development of the standards and ensure their wide acceptance.

Standards help the consumer by driving down prices by helping to build an ecosystem of companies that contribute to the innovation, mass manufacturing of the products, and a fair competition among suppliers. Consumers don't get bound to one single supplier and can have flexibility to choose a vendor based on price versus feature comparison.

Standards also help in ensuring worldwide regulatory compliance. This is especially true in the wireless world since wireless products may contain transmitters and there might be regulatory restrictions on the frequencies on which transmissions are allowed.

There are several organizations which are working on defining standards and regulations. Some of these organizations are mentioned below. The information presented here about these organizations has been picked up from the Web site of the respective organizations. For details, see the references.

3rd Generation Partnership Program (3GPP) [1]

- 3GPP unites six telecommunications standard development organizations: Association of Radio Industries and Businesses (ARIB); Alliance for Telecommunications Industry Solutions (ATIS); China Communications Standards Association (CCSA); European Telecommunications Standards Institute (ETSI); Telecommunications Technology Association (TTA); and Telecommunication Technology Committee (TTC). These organizational partners provide their members with a stable environment to produce the highly successful "Reports and Specifications" that define 3GPP technologies.

Bluetooth Special Interest Group (SIG) [2]

- Bluetooth SIG is a privately held, not-for-profit trade association. The main tasks for Bluetooth SIG are to publish Bluetooth specifications, administer the qualification program, protect the Bluetooth trademarks, and evangelize Bluetooth wireless technology.

European Commission for Standardization (CEN) [3]

- CEN was officially created as an international nonprofit association based in Brussels on October 30, 1975. CEN is a business facilitator in Europe, removing trade barriers for European industry and consumers. Its mission is to foster the European economy in global trading, the welfare of European citizens, and the environment. Through its services it provides a platform for the development of European Standards and other technical specifications.

European Telecommunications Standard Issue (ETSI) [4]

- ETSI produces globally applicable standards for information and communications technologies (ICT), including fixed, mobile, radio, converged, broadcast, and Internet technologies.

Federal Communications Commission (FCC) [5]

- The FCC regulates interstate and international communications by radio, television, wire, satellite, and cable in all 50 states, the District of Columbia and U.S. territories.

International Electrotechnical Commission (IEC) [6]

- The IEC is the world's leading organization that prepares and publishes international standards for all electrical, electronic, and related technologies. Over 10,000 experts from industry, commerce, government, test and research labs, academia, and consumer groups participate in IEC standardization work.

Institute of Electrical and Electronics Engineers (IEEE) [7]

- IEEE is the world's largest professional association dedicated to advancing technological innovation and excellence for the benefit of humanity. IEEE and its members inspire a global community through IEEE's highly cited publications, conferences, technology standards, and professional and educational activities.

International Organization for Standardization (ISO) [8]

- The ISO is the world's largest developer of voluntary international standards. International standards give state-of-the-art specifications for products, services, and good practice, helping to make industry more efficient and effective. Developed through global consensus, they help to break down barriers to international trade.

International Telecommunications Union (ITU) [9]

- The ITU is the United Nations specialized agency for information and communication technologies (ICTs). They allocate global radio spectrum and satellite orbits, develop the technical standards that ensure networks and technologies seamlessly interconnect, and strive to improve access to ICTs to underserved communities worldwide.

Wi-Fi Alliance [10]

- The Wi-Fi Alliance is a global nonprofit organization with the goal of driving adoption of high-speed wireless local area networking.

Near Field Communication (NFC) Forum [11]

- The Near Field Communication Forum was formed to advance the use of NFC technology by developing specifications, ensuring interoperability among devices and services, and educating the market about NFC technology.

ZigBee Alliance [12]

- ZigBee Alliance is an open, nonprofit association of members that has created a thriving global ecosystem of a growing family of innovative, reliable, and easy-to-use ZigBee standards.

1.4 Introduction to Bluetooth and Bluetooth Low Energy

Bluetooth is a global standard for short-range, low-power, low-cost, small-form-factor wireless technology that allows devices to communicate with each other over radio links. It originated as a cable replacement technology mainly to replace the serial data cables that connect various devices. Over the years, the use cases have grown to exchanging files between PCs, mobiles, listening to music, printing documents, browsing, taking mobile calls on Bluetooth headsets and car kits, and several more.

Today the attach rate of Bluetooth is almost 100% for mobile phones, tablets, and laptops. Bluetooth is also widely used in wireless headsets, speakers, cameras, car kits, gaming consoles, and peripherals like keyboards, mouses, printers, and scanners.

Bluetooth Low Energy (also referred to as LE) is one of the latest enhancements to the Bluetooth technology that was added as a part of Bluetooth 4.0 specification. As the name suggests, it is aimed at “ultra” low power devices. This technology was known as Wibree and Ultra Low Power (ULP) in the past. The terms Bluetooth Low Energy and LE will be used interchangeably in this book to refer to Bluetooth Low Energy.

Bluetooth Low Energy extends the low power feature of Bluetooth even further. Devices compliant with this standard are expected to consume very low power so that they can operate for months or even years on coin cell or smaller batteries without the need of recharging or replacing batteries. This is very useful in applications where it may be difficult to recharge frequently and longer battery life is important. Data communication is generally in short bursts that do not need to be very frequent. It is best suited for devices that do not require high throughput or streaming of data.

As per ABI research [16]:

... In the first 10 years of its life (up to 2010) cumulative shipments of Bluetooth enabled devices reached 5 billion. Growth has been largely driven by its use in mobile phones and accessories. This market is still growing overall but it will start to plateau out over the next five years...

...The introduction of Bluetooth v4.0, with low energy as its pivotal enabler, will drive a second wave of Bluetooth enabled device shipment growth. The market is expected to achieve cumulative shipments over 20 billion by 2017...

Some of the key features of Bluetooth Low Energy are:

- Ultra low power, which enables months or even years of operation on coin cell or smaller batteries;
- Small size;
- Low cost;
- Short range;
- Faster connections (link setup time reduced to 3 ms);
- Small chunks of data;
- Infrequent transfers;
- Secure;
- Interoperable.

Bluetooth Low Energy finds a wide variety of applications including the following:

- Internet of Things;
- Health care devices such as thermometers, blood pressure monitors, and glucose meters;
- Sports and fitness equipment such as smart watches, pedometers, GPS locators, and heart rate monitors;
- Home automation;
- Home entertainment, remote controls, wireless keyboards;
- Smart energy (meters and displays);
- Advertisements;
- Mobile payments;
- Automotive devices such as vehicle tire pressure sensors, motion sensors, temperature sensors, and pollution sensors;
- Security.

What is the Internet of Things?

More and more devices are being embedded with sensors and have the ability to communicate. This enables a network of devices that can identify themselves, collect data, and also communicate with each other. The Internet of Things is the next radical transformation in the communications era where the gadgets talk to each other without any need for human intervention.

Some examples are smart energy meters that report electricity and gas usage to the utility company, vending machines with cellular modems to update inventory and pricing, vehicle fleet management systems that report location of all vehicles in the fleet to a central server, and so forth.

One of the key advantages that LE has over other similar technologies is that it will be able to build onto the existing Bluetooth infrastructure, especially smartphones, tablets, and laptops. At present the attach rate of Bluetooth to these devices is almost 100%. Once these devices get upgraded to use LE chips, they can act as gateways to the LE sensors. A wide variety of use cases can be enabled with a very low incremental cost for the end user. Therefore, LE is expected to be a significant contributor to the overall wireless sensor market.

The Bluetooth 4.0 core specification [13] that introduced enhancements related to LE was released in July 2010. Since then several devices have hit the market and several more are already being developed and announced. This has also opened up several requirements for both hardware and software developers. Newer devices and uses are being developed that will need both hardware implementation and software applications.

1.5 Applications

This section describes some possible real-world uses of LE.

1.5.1 Finding and Alerting Devices

LE can be used to find misplaced devices. Some example scenarios are:

1. If the remote control of a home appliance or car keys is misplaced, they can be found from a mobile device. The user will press a button on the mobile and the misplaced device will start giving an audio or visual alert. For example it may start beeping or an LED on it may start flashing.
2. The microwave oven or washing machine will send an alert to the mobile phone or wrist watch once the food is cooked or the clothes are washed.

This is shown in Figure 1.2.

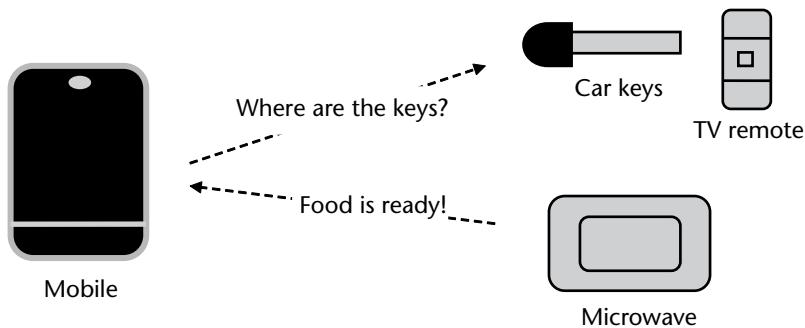


Figure 1.2 Finding and alerting devices.

1.5.2 Proximity and Presence Detection

LE can be used to detect the presence or absence of a device and take actions accordingly. It can also be used to detect when the device is moving away from a predefined range. Some example scenarios are (see Figure 1.3):

1. As soon as a car comes close to the garage the car keys will automatically instruct the garage door to open without the need to press any buttons.
2. While a person parks the car, the devices in the home will detect his or her presence and automatically take some action. The lights could be switched on and temperature control (air conditioner or heater) could be activated.
3. As soon as a person leaves the home the lights would switch off, the doors would lock, and the temperature control would adjust automatically.
4. If a person is in a crowded place and his or her mobile is stolen, as the thief runs away with the mobile device, the distance from the owner would

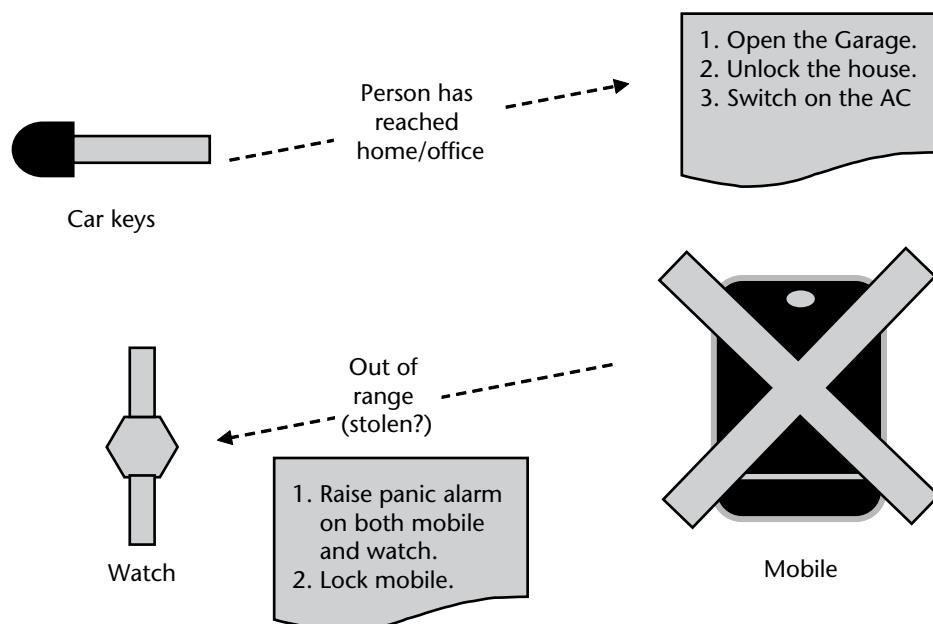


Figure 1.3 Proximity and presence detection.

increase. In this situation, both the owner's watch and the mobile would start raising an alarm and the mobile would lock itself. If the owner is lucky, the thief may panic, throw the mobile down, and run away! At least automatically locking the mobile will ensure that the sensitive data stored on the mobile is not misused.

1.5.3 Health Care

One of the major markets intended for LE is the health care domain. Some applications in this area are (see Figure 1.4):

1. Health care devices like thermometers, heart rate monitors, glucose monitors, and scales can take measurements and send the data to the smartphone or laptop. The smartphone or laptop can perform some analysis of the data like historical trends and alert the user if any parameters are beyond the prescribed limits. It may also send the data through Internet to the doctor.
2. The health care devices can be programmed to take measurements at various intervals depending on the condition of the patient. These can also be programmed to report data in various units (like Celsius or Fahrenheit).

1.5.4 Sports and Fitness Equipment

LE can be used in several sports and fitness applications. Some example applications are (see Figure 1.5):

1. Wearable heart rate monitors to monitor the heart rate during cardio exercises.
2. Pedometers and speedometers to track the exercise done.

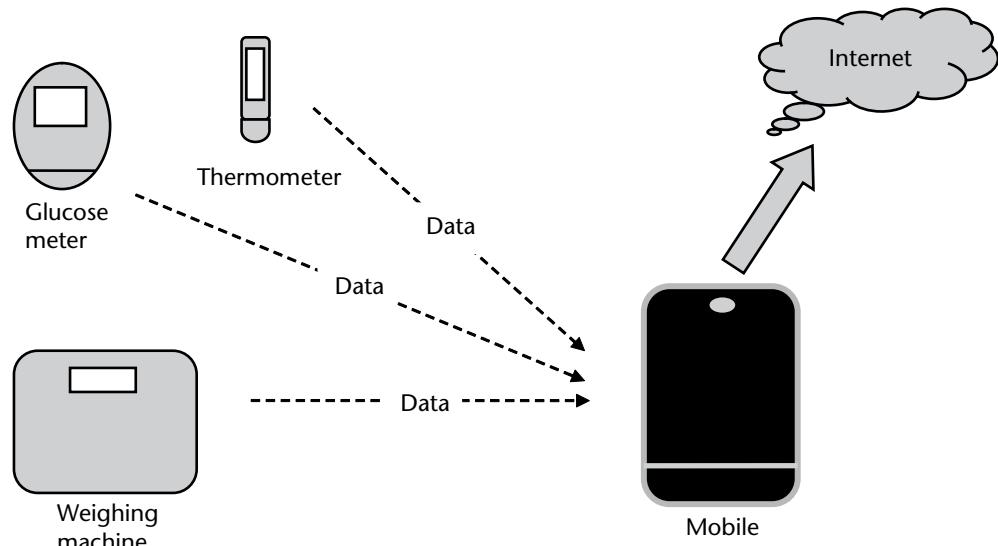


Figure 1.4 Health care.

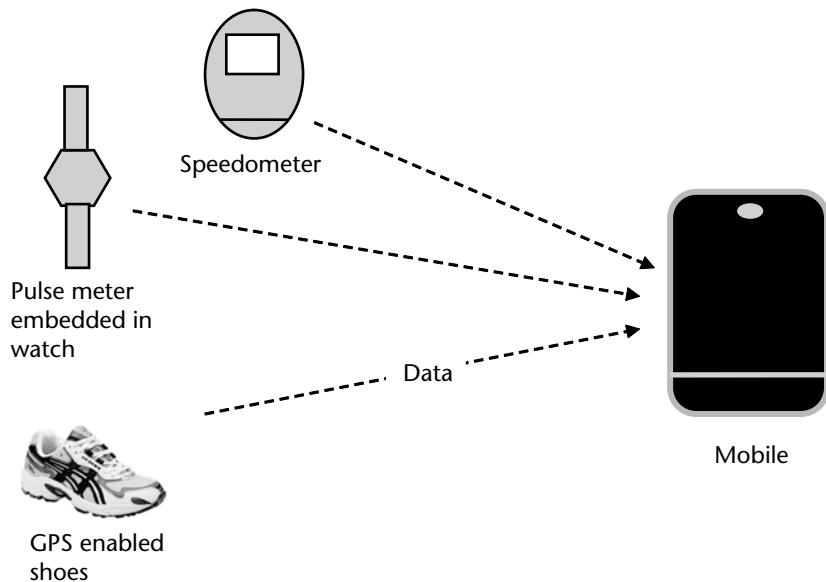


Figure 1.5 Sports and fitness equipment.

3. GPS locators enabled with LE sensors to provide information on the speed, distance traveled, current location, etc.

1.5.5 Mobile Payments

LE can be used in electronic wallet applications for making mobile payments. Some example scenarios are (see Figure 1.6):

1. The user could walk into a store and select merchandise to buy. Instead of using a credit card to make payments, the user could use the LE-enabled mobile phone. The mobile phone would establish a secure link with the retail terminal in the shop for PIN verification, and so forth. The verification and acceptance of the financial transaction could be cloud-based.

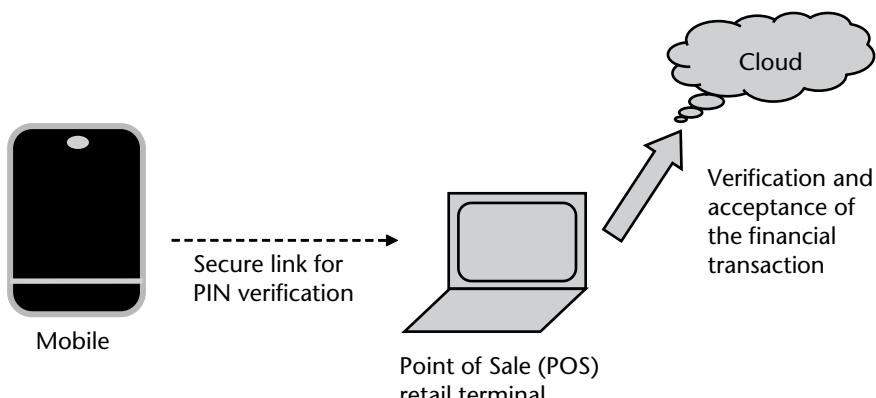


Figure 1.6 Mobile payments.

1.6 Competing Technologies

There are many other wireless technologies that are competing in the same space as Bluetooth Low Energy. Each of these technologies has its own pros and cons and some of the uses overlap with those of Bluetooth Low Energy. Some of these technologies are briefly mentioned here. This information has been picked up from the respective Web sites of the relevant technologies (see the references).

The intention here is not to compare technologies but to introduce these technologies to the reader so that details of these technologies can be looked up in the references. There are already some good resources on the Internet that provide an in-depth comparison of these technologies both from a technical perspective as well as from a commercial perspective.

1.6.1 ANT and ANT+

ANT™ [16] is a proven ultra-low power (ULP) wireless protocol that is responsible for sending information wirelessly from one device to another in a robust and flexible manner. With millions of deployed nodes, ANT is perfectly suited for any kind of low data rate sensor network topologies—from peer-to-peer or star, to practical mesh—in personal area networks (PANs) that are well-suited for sports, fitness, wellness, and home health applications. ANT is a practical solution for local area networks (LANs) in homes and industrial automation applications.

ANT is specifically designed for wireless sensor networks (WSNs) that require:

- Ultra low power (runs on a coin cell for years of operation);
- Highly resource optimized (fits into a compact-sized memory);
- Network flexibility and scalability (self-adaptive and able to do practical mesh network);
- Easy to use with low system cost (operates independently with a single chip).

ANT+ represents an interoperability function that can be added to the base ANT protocol. This facilitates collection, automatic transfer, and tracking of sensor data. ANT+ is finding applications in sports and wellness equipment like heart rate monitors, sports watches, bikes, temperature sensors, and so forth.

1.6.2 ZigBee

ZigBee is a standards-based wireless technology designed to address the unique needs of low-cost, low-power wireless sensor and control networks in just about any market. Since ZigBee can be used almost anywhere, is easy to implement, and needs little power to operate, the opportunity for growth into new markets, as well as innovation in existing markets, is limitless. Here are some facts about ZigBee from the ZigBee Web site:

- With hundreds of members around the globe, ZigBee uses the 2.4-GHz radio frequency to deliver a variety of reliable and easy-to-use standards anywhere in the world.
- Consumer, business, government, and industrial users rely on a variety of smart and easy-to-use ZigBee standards to gain greater control of everyday activities.
- With reliable wireless performance and battery operation, ZigBee gives people the freedom and flexibility to do more.
- ZigBee offers a variety of innovative standards smartly designed to help people be green and save money.

1.6.3 Near Field Communication (NFC)

NFC technology makes life easier and more convenient for consumers around the world by making it simpler to make transactions, exchange digital content, and connect electronic devices with a touch. A standards-based connectivity technology, NFC harmonizes today's diverse contactless technologies, enabling current and future solutions in areas such as:

- Access control;
- Consumer electronics;
- Health care;
- Information collection and exchange;
- Loyalty and coupons;
- Payments;
- Transport.

NFC provides a range of benefits to consumers and businesses, such as:

- *Intuitive*: NFC interactions require no more than a simple touch.
- *Versatile*: NFC is ideally suited to the broadest range of industries, environments, and uses.
- *Open and standards-based*: The underlying layers of NFC technology follow universally implemented ISO, ECMA, and ETSI standards.
- *Technology-enabling*: NFC facilitates fast and simple setup of wireless technologies such as Bluetooth and Wi-Fi.
- *Inherently secure*: NFC transmissions are short-range (from a touch to a few centimeters).
- *Interoperable*: NFC works with existing contactless card technologies.
- *Security-ready*: NFC has built-in capabilities to support secure applications.

1.7 Summary

Wireless technologies have become an important part of our lives. Depending on the application requirements, wireless technologies offer ample choice in terms of data rates and coverage. There are several standard bodies that help in developing specifications for these technologies.

Bluetooth Low Energy finds applications in several fields. It builds onto the existing ecosystem of Bluetooth devices and addresses use cases where ultra-low power consumption is needed. These include scenarios where it may be difficult to replace or recharge the batteries. LE devices are expected to work for several months to several years without need of a recharge.

References

- [1] 3GPP Web site, <http://3gpp.org>.
- [2] Bluetooth SIG Web site, <http://www.bluetooth.org>.
- [3] CEN Web site, <http://www.cen.eu>.
- [4] ETSI Web site, <http://www.etsi.org>.
- [5] FCC Web site, <http://www.fcc.gov>.
- [6] IEC Web site, <http://www.iec.ch/>.
- [7] IEEE Web site, <http://www.ieee.org>.
- [8] ISO Web site, <http://www.iso.org>.
- [9] ITU Web site, <http://www.itu.int>.
- [10] WiFi Alliance Web site, <http://wi-fi.org>.
- [11] NFC Forum Web site, <http://www.nfc-forum.org>.
- [12] ZigBee Web site, <http://www.zigbee.org>.
- [13] Bluetooth Core Specification 4.0, <http://www.bluetooth.org>.
- [14] Bluetooth Profiles Specifications from the Bluetooth Web site.
- [15] ANT Web site, <http://www.thisisant.com>.
- [16] “Bluetooth Smart Will Drive Cumulative Bluetooth Enabled Device Shipments to 20 Billion by 2017,” ABI Research Web site, www.abiresearch.com.

Background of Bluetooth

2.1 Introduction

This chapter provides a brief background of the Bluetooth technology and explains some of the commonly used concepts and terminology. It may be skipped if the reader is well versed with the Bluetooth technology.

2.2 Ad Hoc Networks—Why?

In today's world, with the advent of the “always connected” devices and “on the move” users, users expect devices to seamlessly connect to each other and exchange information without the need to install drivers, upgrade software, figure out cable connections, lookup configurations on centralized servers or even read user manuals. Information is the key in this connected world and the ability to quickly exchange that information across various kinds of devices has become more of a necessity than a luxury.

There are many applications where the user wants to bring devices close to each other and expects them to exchange data conveniently without going through lengthy setup procedures. The faster the setup→data exchange→tear down cycle, the better is the user experience.

A wireless ad hoc network may be defined as follows:

A wireless ad hoc network is a decentralized type of wireless network. The network is termed as ad hoc because it does not rely on a preexisting infrastructure, such as routers in wired networks or access points in wireless networks like WiFi.

Some of the real world scenarios of an ad hoc network are described below.

2.2.1 Printing Documents, Photos

Consider the scenario where a user took some photos using a camera and wants to get them printed. The most convenient scenario would be to just select the printer and send the document or photo to it without the need of connecting any cables or installing any printer drivers. This is shown in Figure 2.1.

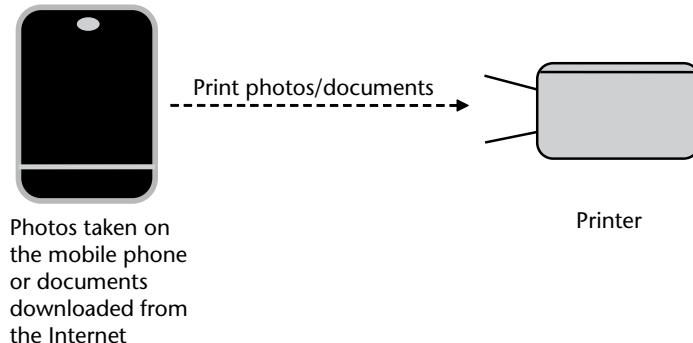


Figure 2.1 Printing documents, photos.

2.2.2 Exchanging Business Cards, Photos, Music, Files

Suppose a person meets a friend at a train station and they need to exchange phone numbers, e-mail addresses and other contact information. They may also decide to exchange some other things like photos, music, or some other files before their trains arrive and they go their respective ways. In such a scenario, an ad hoc technology is needed to quickly establish a connection and send information at a high data rate. Besides high data rate, the ad hoc technology also needs to be secure since they are using it in a public place and they want to ensure that their sensitive information is not received by others. This is shown in Figure 2.2.

2.3 What is Bluetooth?

Bluetooth is a global standard for short range, low power, low cost, small form factor wireless technology that allows devices to communicate with each other over radio links. It originated as a cable replacement technology mainly to replace the serial data cables that connect various devices. Over the years its uses have grown to exchanging files between PCs, mobile devices, listening to music, printing documents, browsing, taking mobile calls on Bluetooth headsets and car kits, and several more.

Today the attach rate of Bluetooth is almost 100% for mobile phones, tablet and laptops. Bluetooth is also widely used in wireless headsets, speakers, cam-

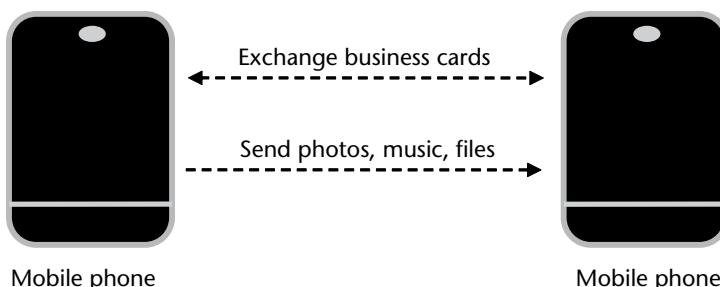


Figure 2.2 Exchanging business cards, photos, music, files.

eras, car kits, gaming consoles, and peripherals like keyboard, mouse, printer and scanner

Some of the key features of Bluetooth are:

- Ad hoc;
- Small Size;
- Low Cost;
- Low Power;
- Short Range;
- Secure;
- Interoperable;
- Global Standard;
- Ease-of-use;
- Does not require line of sight;
- Does not suffer from interference from obstacles like walls;
- Can co-exist with other wireless technologies;
- Big set of profiles already available to address real world scenarios.

Bluetooth is also termed as a Wireless Personal Area Network (WPAN) technology. As explained in Chapter 1, a Personal Area Network (PAN) is used for communication among personal devices like laptops, PCs and mobile phones. The communication may be done over wired buses like USB, Firewire, or plain old serial cables. A WPAN uses a wireless media to connect the personal devices such as mobile phones, PDAs, tablets and accessories. There are several wireless technologies like IrDA, Bluetooth, Wireless USB, NFC, and ZigBee which are used for WPANs.

2.4 Bluetooth SIG

The Bluetooth Special Interest Group (SIG) is a privately held, not-for-profit association. Bluetooth SIG does not make, manufacture, or sell Bluetooth products. The main tasks of the Bluetooth SIG are:

- Publish Bluetooth specifications.
- Administer the qualification program.
- Evangelize Bluetooth wireless technology.

The Bluetooth technology originated at Ericsson labs in Sweden in 1994. At that time Ericsson began a study to examine alternatives to cables that linked mobile phones with accessories.

The Bluetooth SIG was formally established by the following five founding members in February 1998:

- Ericsson;
- Intel;
- IBM;
- Nokia;
- Toshiba.

Over the years, several companies have signed up for the Bluetooth SIG membership. Currently the Bluetooth SIG membership surpasses 16,000 companies. The membership is open to all companies that wish to develop, market, and promote Bluetooth products.

The Bluetooth SIG offers two levels of membership to new member companies, plus promoter membership for companies represented on the Board of Directors.

- Adopter membership (Free): This provides access to Bluetooth resources and specifications to build Bluetooth products and license to use the Bluetooth word mark and logos.
- Associate membership (Annual fee): This provides early access to Bluetooth specifications which are still under development along with the opportunity to contribute to the specifications by joining working groups and committees. This membership also provides discounts on qualification fees, tools, trainings and more.

The Bluetooth SIG currently has seven promoter companies:

- Ericsson;
- Intel;
- Lenovo;
- Microsoft;
- Motorola Mobility;
- Nokia;
- Toshiba.

The Bluetooth SIG holds regular round tables and hosts mailing lists, working groups, discussion forums, and UnPlugFests where the members can contribute to the specifications, submit errata, and test their devices for any interoperability issues.

The UnPlugFests (UPFs) are events organized by the Bluetooth SIG three times a year where the members can register, bring their products, and test them against products from other vendors.

Besides Bluetooth specifications, the SIG also provides Test Specifications which include test cases to test the Bluetooth stack and profiles. It also provides a tool called PTS (Profile Tuning Suite) which can be used to test the Bluetooth implementation in an automatic fashion. The test results of this tool can be used as direct evidence of compliance when the product is to be qualified.

The Bluetooth SIG also has a full-fledged qualification program which can be used to qualify a product before going to the market.

The Bluetooth Testing and Qualification will be covered in detail in Chapter 17.

2.5 History of the Bluetooth Specification

The word Bluetooth came from a Danish Viking and King, Harald Blåtand (Bluetooth in English). He united and controlled large parts of Norway and Denmark into one single kingdom during the 10th century. This probably provided the inspiration of the name Bluetooth—uniting a wide variety of devices from computing and communications domains.

The Bluetooth specification has evolved significantly in the last several years. Some of the major enhancements that each version added are mentioned in Table 2.1.

2.6 IEEE 802.15 Family of Specifications

IEEE 802.15 is a working group of the IEEE 802 standards committee that specifies WPAN standards. It includes seven Task Groups (TGs):

1. TG 1: Bluetooth.
2. TG 2: Coexistence for WPANs.
3. TG 3: High Rate WPAN: 11 Mbps to 55 Mbps.
4. TG 4: Low Rate WPAN: 20 kbps to 250 kbps.
5. TG 5: Mesh Networking.
6. TG 6: BAN (Body Area Networks).
7. TG 7: Visible Light Communication.

Versions 1.1 and 1.2 of Bluetooth were ratified as IEEE802.15.1-2002 and IEEE802.15.2-2005 specifications. Since then, the Bluetooth standard has evolved independently.

2.7 Bluetooth Basics

One of the first uses of Bluetooth was to replace cables between devices such as mobile phones, laptops, headphones, printers, fax machines, keyboard, mouse, and a host of other devices. Besides providing data channels, Bluetooth also provides voice channels allowing wireless connections between the mobile phones and headsets and car kits.

Bluetooth supports ad hoc networks. This means that it does not rely on any pre-existing infrastructure such as routers in wired networks or access points in wireless networks. The devices can dynamically come close to each other and exchange data and go out of range.

Table 2.1 Evolution of Bluetooth Specification

<i>Specification Version</i>	<i>Release Date</i>	<i>Key features of the version</i>
1.0 and 1.0a	Jul 1999	These were the very first versions of the Bluetooth specification. The primary objective was to replace the serial cables with a wireless link.
1.0b	Dec 1999	This version added minor updates to fix some of issues.
1.1	Feb 2001	Bluetooth was ratified as IEEE 802.15.1-2002 standard.
1.2	Nov 2003	This release of the Bluetooth standard added new facilities including the following: <ul style="list-style-type: none"> • Adaptive Frequency Hopping (AFH) was introduced to provide better resistance to interference in noisy environments. • Extended Synchronous Connection Oriented (eSCO) links were added to provide better voice quality. This was also ratified as IEEE 802.15.1-2005. This was the last version issued by IEEE and after that Bluetooth technology evolved independently.
2.0 + EDR	Nov 2004	This release of the Bluetooth standard introduced enhancements to the throughput using Enhanced Data Rates (EDR). The previous versions of the standard supported a throughput up to 721 kbps. This version increased it to 2.1 Mbps. This made it more suitable for applications that required fast data transfers like file transfer, browsing, printing, etc.
2.1 + EDR	Jul 2007	This version brought in several enhancements and added SSP (Secure Simple Pairing) to both simplify the pairing mechanism and to improve security.
3.0 + HS	Apr 2009	This version provided a significant increase in throughput by introducing the support for multiple radios. This was referred to as Alternate MAC/PHY (AMP). The supported maximum throughput went up to 24 Mbps. The rationale, very briefly, was that several devices like Laptops, Mobile phones and Tablets have both Bluetooth and 802.11 chips on them. This version of the specification allowed connection using Bluetooth and then moving on to the 802.11 chip to achieve high speed data transfers.
4.0	Jun 2010	This version went into a completely different direction compared to the previous versions. While in the previous versions the main focus was to introduce new features and enhance the throughput, this version addressed the markets where the need was not of high throughput but of ultra-low power. This was referred to as Bluetooth Low Energy (LE).

Bluetooth supports a maximum distance of 100 meters though typically it is used for much shorter distances. The specification provides support for different power levels for Bluetooth radios so that the appropriate combination of power consumption and distance can be selected based on the application that the device is intended for.

Originally Bluetooth supported a maximum data rate of 721 kbps. This is referred to as Basic Rate (BR). The Bluetooth 2.0+EDR specification added support for data rates up to 2.1 Mbps. This is referred to as Enhanced Data Rate (EDR). The Bluetooth 3.0+HS (High Speed) specification enhanced it even further to 24 Mbps.

The Bluetooth technology uses the license free 2.4 GHz ISM band for its radio signals. ISM stands for Industrial, Scientific, and Medical radio band. This band

is globally unlicensed and can be used in any country without asking for prior permissions.

Since the ISM radio band is an unlicensed band, it is used by other devices including WiFi devices, and remote control toys. There is also a possibility of interference from devices like microwaves in this band. To combat interference, Bluetooth uses a Frequency Hopping Spread Spectrum (FHSS) mechanism. Instead of using a constant frequency to send and receive data, the communicating devices use a set of frequencies and hop rapidly from one frequency to another using a pseudo random pattern.

Bluetooth SIG mandates the devices to undergo qualification before being launched in the market. The purpose of the Bluetooth qualification program is to promote interoperability and enforce compliance to the specification. It is a necessary prerequisite to obtain the Bluetooth intellectual property license and use the Bluetooth logos. This program is one of the key reasons behind the success of the Bluetooth technology. When a user buys a device from one vendor, he or she is assured that the device will work well with devices from other vendors.

A summary of the key features of Bluetooth are provided in Table 2.2.

2.8 Bluetooth Architecture Overview

Bluetooth has a layered architecture. The high-level architecture of Bluetooth is shown in Figure 2.3. At a very broad level, the Bluetooth architecture comprises the following components going from bottom to top:

- *Lower Layers or Controller:* These are the layers responsible for performing low level operations like discovering devices in the vicinity, making

Table 2.2 Summary of Key Features

Connection Type	Frequency Hopping Spread Spectrum
Spectrum	2.4 GHz ISM Band. Regulatory range: 2400–2483.5 MHz.
Frequency Hopping	1600 hops per second across 79 RF channels. The channels are separated by 1 MHz.
Modulation	Gaussian Frequency Shift Keying (GFSK).
Maximum Output Power	1 mW to 100 mW.
Transmit Power	Nominal = 0dBm. Goes up to 20 dBm with power control.
Receiver Sensitivity	-70 dBm at 0.1% Bit Error Rate
Maximum Data Rate	721.2 kbps for Basic Rate. 2.1 Mbps with Enhanced Data Rate (BT Spec 2.0+EDR). 24 Mbps with High Speed (BT Spec 3.0+HS).
Typical Range	10 m to 100 m.
Topology	Up to 8 devices in a piconet including 1 Master and up to 7 Slaves.
Voice Channels	3
Data Security: Authentication Key	128 bit key.
Data Security: Encryption Key	8-128 bits (configurable).
Applicability	Does not require line of sight. Intended to work anywhere in the world since it uses unlicensed band.

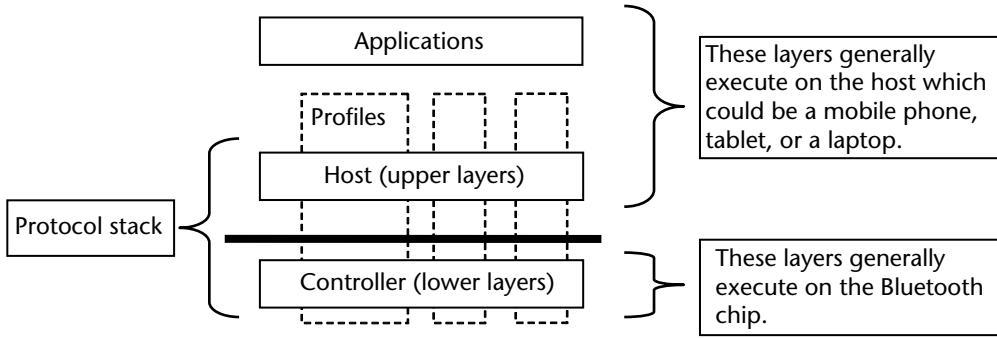


Figure 2.3 High-level Bluetooth architecture.

connections, exchanging data packets, security, low power modes, etc. This functionality is generally implemented in a Bluetooth chip and it is also referred to as a Bluetooth Controller.

- *Upper Layers:* These layers make use of the functionality provided by the lower layers to provide more complex functionality like serial port emulation, transferring big chunks of data by splitting them into smaller pieces and reassembling them, streaming music, etc.
- *Profiles:* The profiles can be considered to be vertical slices through the protocol stack. They provide information on how each of the protocol layers come together to implement a specific usage model. Profiles help to guarantee that an implementation from one vendor works properly with implementation from another vendor. So they form the basis for interoperability and logo requirements. The profiles need to be tested and certified before a device can be sold in the market. A device can support one or more profiles at the same time.
- *Bluetooth Application:* This is the entity that generally performs the tasks of an MMI (Man Machine Interface) so that the user can make use of the Bluetooth functionality. Some examples of this are:
 - Selecting a file and then transferring it on Bluetooth;
 - Searching for Bluetooth devices in the vicinity and displaying the results;
 - Pressing a button on the headset to make a connection;
 - Browsing the files and folders of a remote device on Bluetooth.

One of the strong points of the Bluetooth technology is that it tries to reuse things that are already available instead of specifying everything from scratch. It picks up some components from existing standards, adapts them as needed, and then defines only the core components that are needed for Bluetooth operations. The protocols can be broadly grouped into two categories: Core Protocols and Adopted Protocols.

The protocols that are defined from scratch by the Bluetooth SIG are referred to as core protocols. Some examples are L2CAP, Link Manager and SDP (these will be explained in detail later).

The protocols that are adopted from other standard bodies are referred to as adopted protocols. Some examples are:

- *RFCOMM*: This protocol is adopted from European Telecommunications Standards Institute (ETSI) standard 07.10.
- *OBEX*: This protocol is adopted from the IrOBEX protocol which is defined by the Infrared Data Association (IrDA).
- *TCS-BIN*: Used for the Telephony protocols.
- *IEEE 11073-20601*: Used for the MCAP protocol for health applications.
- *IrMC*: Used for SYNC profile.
- *HID profile*: This is adopted from USB HID (used in wired keyboard, mice and gaming devices).

The adopted protocols help to re-use existing specifications instead of reinventing the wheel. The existing specifications are established and are proven to work well. In many cases, even the existing software can be re-used to a large extent instead of writing the software from scratch. This helps in speedier implementations.

The detailed Bluetooth architecture is shown in Figure 2.4. The adopted protocols are shown with shaded rectangles and core protocols are shown with plain rectangles.

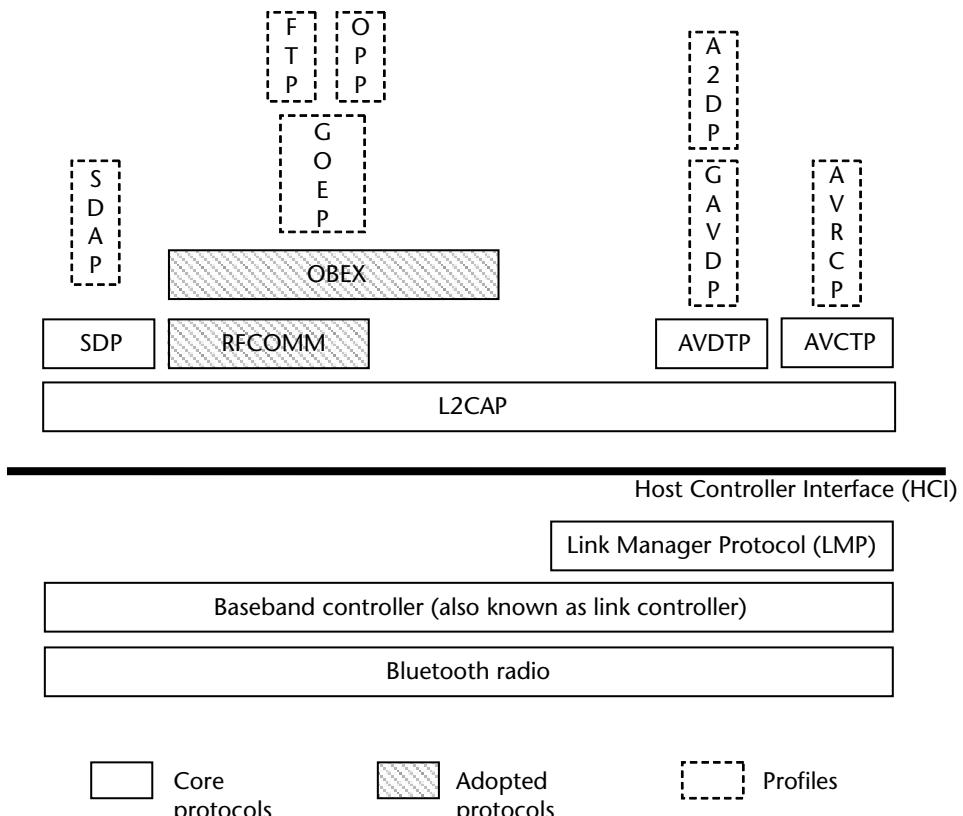


Figure 2.4 Detailed Bluetooth architecture.

rectangles. The profiles are shown by rectangles with dashed boundaries. The protocol stacks and profiles will be covered in further details in the next two chapters.

2.9 Basic Terminology

This section introduces some of the basic terminology used in Bluetooth. Familiarity with these terms will help in a better understanding of the further chapters.

2.9.1 Host, Host Controller, and Host Controller Interface (HCI)

Based on the device on which the Bluetooth functionality is implemented, the Bluetooth architecture can be considered to be split into two broad logical parts as shown in Figure 2.5.

The Host is a logical entity that executes the upper layers of the Bluetooth protocol stack along with the profiles and applications. It includes the following:

1. The Protocol stack layers above the Host Controller Interface—L2CAP, RFCOMM, SDP, AVDTP, AVCTP, and BNEP layers.
2. The profiles like GAP, SDAP, GOEP, OPP, FTP, A2DP, AVRCP, and HF.
3. The MMI applications which interact with the user.

Typically the Host software executes on an application processor or microcontroller.

The Controller is a logical entity that executes the lower layers of the protocol stack. It includes all layers below the Host Controller Interface—Radio, Link Manager, and Baseband. Typically the Controller functionality is embedded in a Bluetooth chip that is attached to the Host.

The Host Controller Interface (HCI) provides a communication interface between the Host and the Controller. Physically this may run on top of an interface like UART, RS-232, USB or SD. The set of packets that can be exchanged on this interface is defined by the Bluetooth specification. One of the strong points of the Bluetooth specification as compared to few other standards is a well-defined interface layer between the host and the wireless controller. It allows independent and parallel development of the host and controller and ensures compatibility of a host from one vendor with a controller from a different vendor.

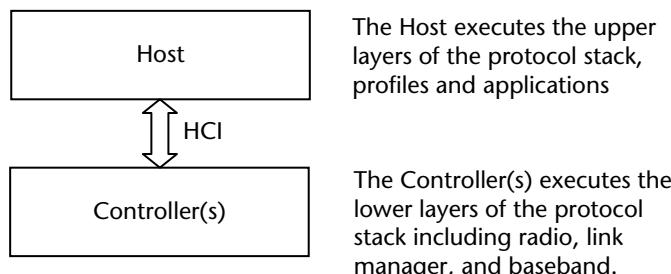


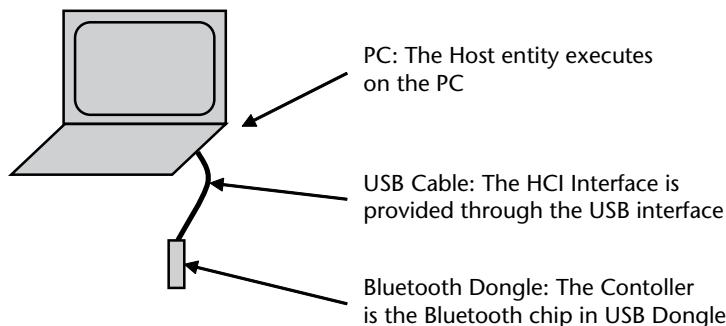
Figure 2.5 Host, Host Controller and Host Controller Interface.

The HCI interface is optional and may be omitted in implementations where the host and the controller are tightly coupled with each other and run on the same processor. If this interface is omitted, then the upper layers interact directly with the lower layers.

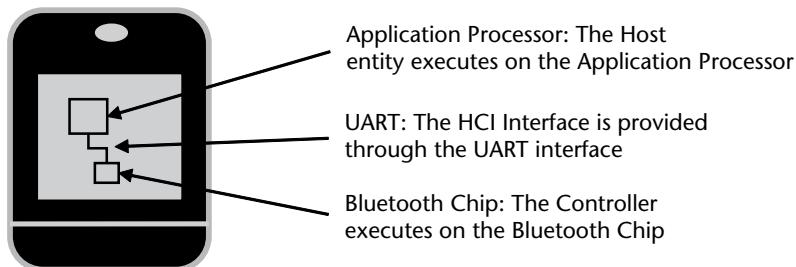
Let us consider a few scenarios to understand this well. These scenarios are shown in Figure 2.6.

Scenario 1: PC attached with a Bluetooth USB dongle

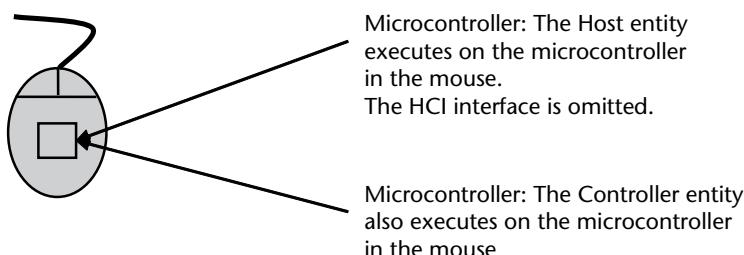
In this scenario:



Scenario 1: PC attached with a Bluetooth USB dongle



Scenario 2: Smart phone or Tablet



Scenario 3: Bluetooth mouse or audio headset

Figure 2.6 Scenarios depicting Host, Controller and HCI.

- The Host entity executes on the PC (The Bluetooth software running on the PC Operating System).
- The Controller entity executes on the Bluetooth chip that resides inside the Bluetooth USB Dongle.
- The Host Controller interface is provided through the USB interface.

Scenario 2: Smart phone or Tablet

In this scenario:

- The Host entity executes on the Phone's application processor.
- The Controller entity executes on the Bluetooth chip that is mounted on the Phone's PCB.
- The Host Controller interface is provided through the UART (or some other) connection on the PCB between the application processor and Bluetooth chip.

Scenario 3: Bluetooth mouse or audio headset

In this scenario:

- Both the Host and the Controller entities run on a single microcontroller.
- The Host Controller interface is omitted and the upper layers of the stack interact directly with lower layers of the stack.

Prior to the Bluetooth 3.0 + HS specification, a Bluetooth system could have only one Host and one Controller. From the Bluetooth 3.0 + HS specification onwards, a system can have one host and multiple controllers. Two types of controllers are defined by Bluetooth 3.0 + HS specification: Primary controller and Secondary controller. A system can have only one Primary controller and may have zero or more Secondary controllers.

The Primary controller may support BR/EDR only, LE only or a combination of BR/EDR + LE functionality.

The Secondary controllers support one or more Alternate MAC/PHY (AMP) controllers. These AMP controllers help in increasing the throughput up to 24 Mbps by using the 802.11 transport layer instead of the classic Bluetooth transport layer for high speed data transfers.

2.9.2 Device Address (BD_ADDR) and Device Name

Each BR/EDR controller has a globally unique 48-bit Bluetooth Device Address, also referred to as BD_ADDR. This address is used to identify the device. It is similar to an Ethernet MAC address, and is in fact, administered by the same organization, IEEE.

BD_ADDR consists of two fields:

1. 24-bit company id assigned by IEEE Registration authority. This is called Organizationally Unique Identifier (OUI) [24 most significant bits].
2. 24-bit unique number assigned by the company to each controller. [24 least significant bits].

The Bluetooth device name is a friendly name that can be assigned to a device. Unlike the BD_ADDR, this can be changed by the user or application and provides an easy mechanism to identify and remember a device. It is possible (though not desirable) for several devices to have the same name. This device name is generally fetched from the remote device to identify it.

2.9.3 Class of Device (CoD)

The Class of Device is a 24-bit value used to indicate the capabilities of the local device to remote devices. This is returned as one of the parameters when searching for Bluetooth devices in the vicinity.

The CoD field has a variable format. The format is indicated using the *Format Type field* within the CoD. The length of the Format Type field is variable and it ends with two bits different from '11'.

In format #1 of the CoD, the bits are assigned as follows:

- Format Type field: Two bits: 00.
- 11-bits to indicate a high level generic category of service class. These bits are assigned as a bit mask so that more than one service class category bit can be enabled at the same time. Currently 7 categories are defined.
- 11 bits to define the device type category and other device specific characteristics. This is further divided into:
 - 5 bits to denote the Major class.
 - 6 bits to denote the Minor class.

The details on the Class of Device can be looked up in the Assigned Numbers document on the Bluetooth SIG website in the baseband section. Some of the commonly used values of Class of Device are shown in Figure 2.7.

2.9.4 Bluetooth Clock

The Bluetooth clock refers to the native clock of the Bluetooth device that is derived from the free running system clock. Each Bluetooth controller has its own clock that is started when the device is powered on.

The Bluetooth clock is used to synchronize with other devices. Since each device may have a different absolute value of the clock, the *clock offset* values are used for synchronization instead of the absolute value of the clock. When these offsets are added to the native clock on each of the Bluetooth devices, mutually synchronized clocks are obtained.

Clock offsets are very important in a Bluetooth network since all timings are based on these clocks. The worst case accuracy required for the reference crystal

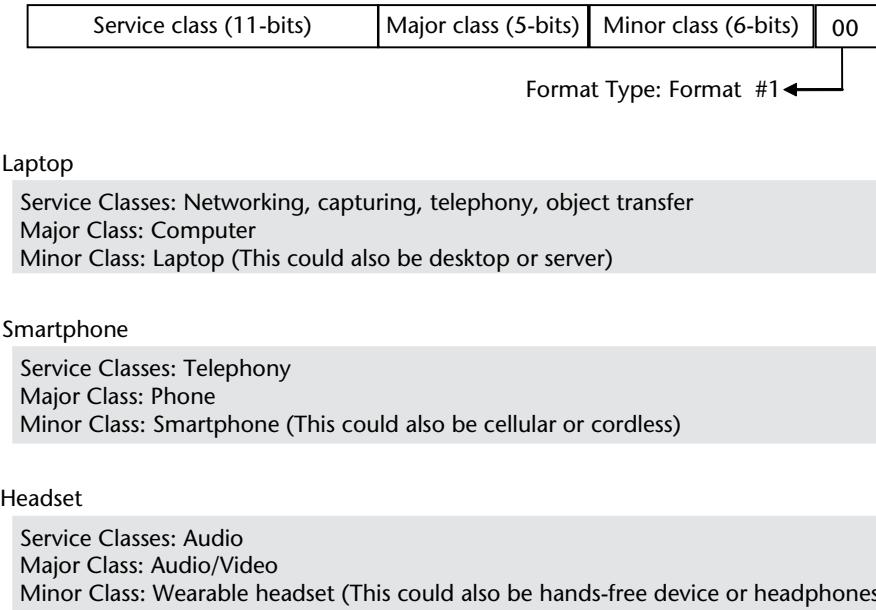


Figure 2.7 Class of Device.

oscillator that drives the Bluetooth clock is ± 20 ppm (Parts Per Million). This means that in a million ticks, the clock cannot drift by more than 20 ticks compared with a fully accurate reference clock.

The Bluetooth clock is different from a Real Time Clock (RTC) which is used to maintain the time of the day. The RTC generally runs even when the system is switched off. For example the RTC is used to provide the current time in laptops, PCs, and mobile phones.

2.9.5 Protocol Data Unit (PDU)

The term PDU is used to refer to information that is exchanged between two entities in a network. This information is in the form of data packets. Some examples of PDUs are:

1. The packets that are sent by the lower layers of one Bluetooth device to the lower layers of the peer Bluetooth device.
2. The packets that are sent by the upper layers (For example L2CAP) of the protocol stack to the corresponding layers of protocol stack on the peer Bluetooth device.

2.10 Data Rates

The data rates supported by Bluetooth have been enhanced in a major way along with evolutions of the specification. Prior to version 2.0, data rates of up to 721.2 kbps were supported. This is referred to as Basic Rate (BR).

The Bluetooth 2.0+EDR specification added support for enhanced data rates which enhanced the throughput to 2.1 Mbps. This is referred to as Enhanced Data Rate (EDR).

Several different packet types are defined by the specification with each packet type providing support for different data rates. These will be discussed in detail in the next chapter.

The term BR/EDR is often used together to refer to a system supporting Basic and Enhanced Data Rates and to distinguish it from a system that supports LE (Bluetooth Low Energy). This term is used extensively in the specification and will also be used in this book.

The Bluetooth 3.0+HS specification added support for using Alternate MAC/PHY (AMP) with which the maximum throughput was increased to 24 Mbps.

The Bluetooth 4.0 (LE) specification did not add any further data rates. Rather the focus of this specification was on reducing the current consumption to enable Low Energy devices. In fact the LE system is designed for uses and applications for lower data rates and smaller packet sizes. The maximum throughput in LE is 305 Kbps though in practice the applications send data at a much lower rate to conserve battery power.

2.11 Connection Setup and Topology

This section briefly introduces the connection setup procedure and Bluetooth topologies. These will be explained in detail in subsequent chapters.

If two devices need to communicate, then they may take the following steps:

1. One of the devices, say device B, needs to be in a mode where it can be “seen” or discovered by other devices. It is said to become *discoverable*.
2. The second device, say device A can search for devices in the vicinity. This is called *inquiry*. During the inquiry process, it will locate device B.
3. In order to make a connection, the device B needs to allow other devices to connect to it. So it needs to become *connectable*.
4. Device A, can now create a *connection* to device B. This process is called *paging*.
5. Once the connection is created, device A is said to become the *Master* and device B is said to become the *Slave* and the devices are said to be *connected*.
 - a. *Connection* means that device A is able to receive the packets sent by device B and vice versa. This means that they are synchronized on the frequencies that they will use for transmission and reception.
6. When the devices don’t need the connection any more, they *disconnect*. Either the Master or the Slave can initiate the *disconnection*.

These steps are shown in Figure 2.8.

Bluetooth communication is based on the following two network topologies

1. Piconet;
2. Scatternet (Combination of two or more piconets).

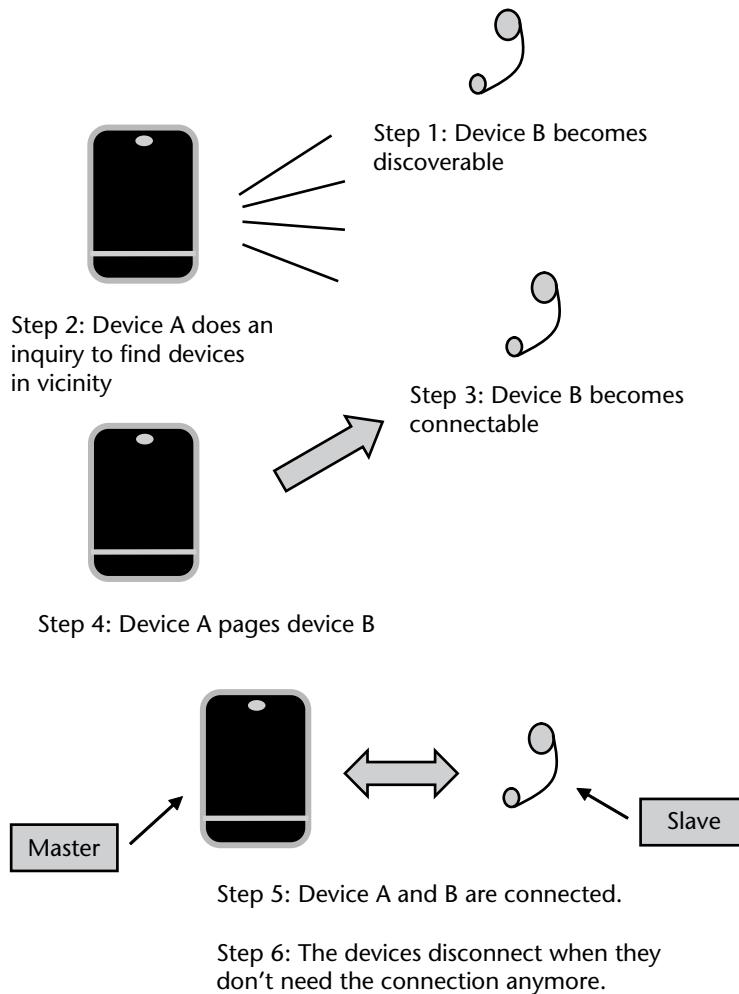


Figure 2.8 Steps during making a connection.

A piconet is the smallest unit of Bluetooth communication. It consists of one Master and up to seven Slaves. So in Figure 2.8, device A and device B are said to form a piconet at step 4. Another example of piconet is shown in Figure 2.9.

A scatternet is formed when two or more piconets come together by sharing a device. Scatternets help to extend the number of Bluetooth devices that can communicate with each other. They allow more than seven devices to communicate with each other. A scatternet is shown in Figure 2.10 where S7 is shared between the two piconets.

2.12 IEEE Word Usage

The Bluetooth SIG has adopted section 13.1 of the IEEE Standard Style Manual (Ref [3]). This dictates the use of words like “shall”, “should”, “may”, and “can” in the documentation. Understanding of this usage is very important when reading

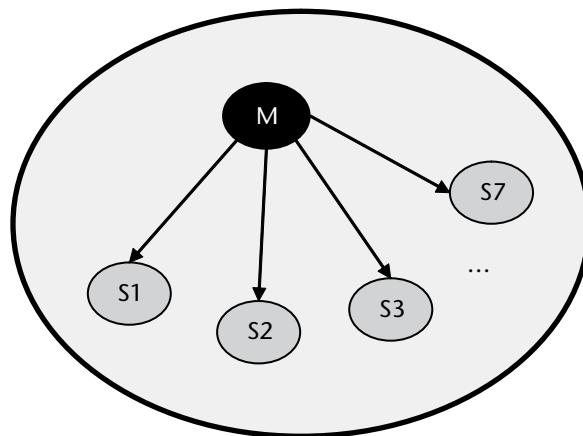


Figure 2.9 Piconet.

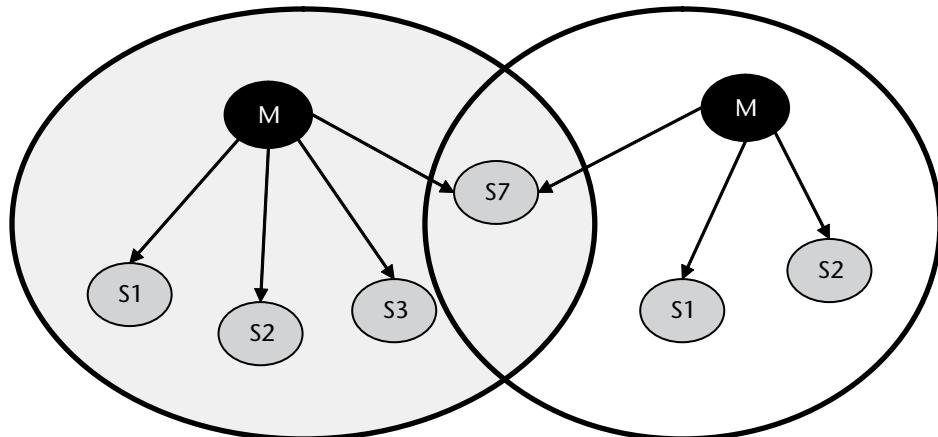


Figure 2.10 Scatternet.

the specifications documents. A brief overview of the IEEE word usage is shown in Table 2.3

2.13 Summary

Bluetooth technology has come a long way since it originated in the Ericsson labs as a cable replacement technology. While previous evolutions of this technology focused more on feature and throughput enhancements, the latest version, LE, focuses on drastic reductions in the power consumption.

The Bluetooth protocol stack follows a layered architecture. It borrows some of the protocols that are already available and only defines the protocols related to core Bluetooth functionality.

Bluetooth supports ad hoc networking where up to seven devices can simultaneously talk to each other. These devices are said to form a piconet. Bluetooth also

Table 2.3 IEEE Word Usage (Adopted from Bluetooth Core Specification)

Shall	<i>is required to</i> – used to define requirements.
	The word <i>shall</i> is used to indicate mandatory requirements that shall be followed in order to conform to the specification and from which no deviation is permitted.
Must	<i>is a natural consequence of</i> – used only to describe unavoidable situations
	The word <i>must</i> <u>shall</u> not be used when stating mandatory requirements. <i>Must</i> is used only to describe unavoidable situations and is seldom appropriate for the text of a specification.
Will	<i>it is true that</i> – only used in statements of fact
	The word <i>will</i> <u>shall</u> not be used when stating mandatory requirements. The term <i>will</i> is only used in statements of fact.
Should	<i>is recommended that</i> – used to indicate that among several possibilities one is recommended as particularly suitable, but not required.
	The word <i>should</i> is used to indicate that among several possibilities one is recommended as particularly suitable without mentioning or excluding others.
May	<i>is permitted to</i> – used to allow options
	The word <i>may</i> is used to indicate a course of action that is permissible within the limitations of the specification. This is generally used when there is a single, optional attribute described, but multiple alternatives <u>may</u> be cited.
Can	<i>is able to</i> – used to relate statements in a causal fashion
	The term <i>can</i> is used for statements of possibility and capability, whether material, physical, or casual. The term <i>can</i> equals <i>is able to</i> .
Is	<i>is defined as</i> – used to further explain elements that are previously required or allowed
Note	<informational text ONLY>

provides flexibility to increase this number by forming a scatternet which combines two piconets.

This chapter introduced some of the basic concepts of Bluetooth. The next two chapters will cover the Bluetooth lower layers and Bluetooth upper layers.

References

- [1] Bluetooth Core Specification 4.0 <http://www.bluetooth.org>.
- [2] Assigned Numbers Specification: <https://www.bluetooth.org/Technical/AssignedNumbers/home.htm>
- [3] IEEE Standards Style Manual (<http://standards.ieee.org/guides/style/>)

Bluetooth Lower Layers

3.1 Introduction

The layered architecture of the Bluetooth protocol stack was introduced in the previous chapter. This chapter covers the lower layers of the Bluetooth protocol stack including the Bluetooth Radio, Baseband Controller, Link Manager and Host Controller Interface. These are shown in the bottom half of Figure 3.1.

The Host Controller Interface specification is common for BR/EDR and LE. So the Host Controller Interface will be explained in detail in this chapter for both BR/EDR and LE. Only a few LE specific parts will be explained in Chapter 9.

Some scenarios on how these layers come together to implement certain practical uses are shown toward the end of this chapter.

3.2 Bluetooth Radio

Bluetooth Radio operates in the 2.4 GHz ISM band. It uses a frequency hopping mechanism with 79 channels to combat interference. A Time Division Duplex (TDD) scheme is used for full duplex transmission.

This layer is responsible for the following primary tasks:

1. Transmission and Reception of packets: This includes modulation and de-modulation of the packets. Two modulations modes are defined:
 - a. Basic Rate: This mode uses a shaped binary FM modulation mechanism and is designed to minimize complexity of the transceiver. It provides a gross air data rate of 1 Mbps.
 - b. Enhanced Data Rate: This mode uses Phase Shift Keying (PSK) Modulation and supports higher data rates. The gross air data rate supported is 2 Mbps or 3 Mbps.
2. Support appropriate power class: Three power classes are defined by the Bluetooth specification based on the maximum output power. A higher value of maximum output power leads to a longer range. The device can support the power class most appropriate to its intended use:
 - a. Power Class 1: Maximum output power of 100 mW.

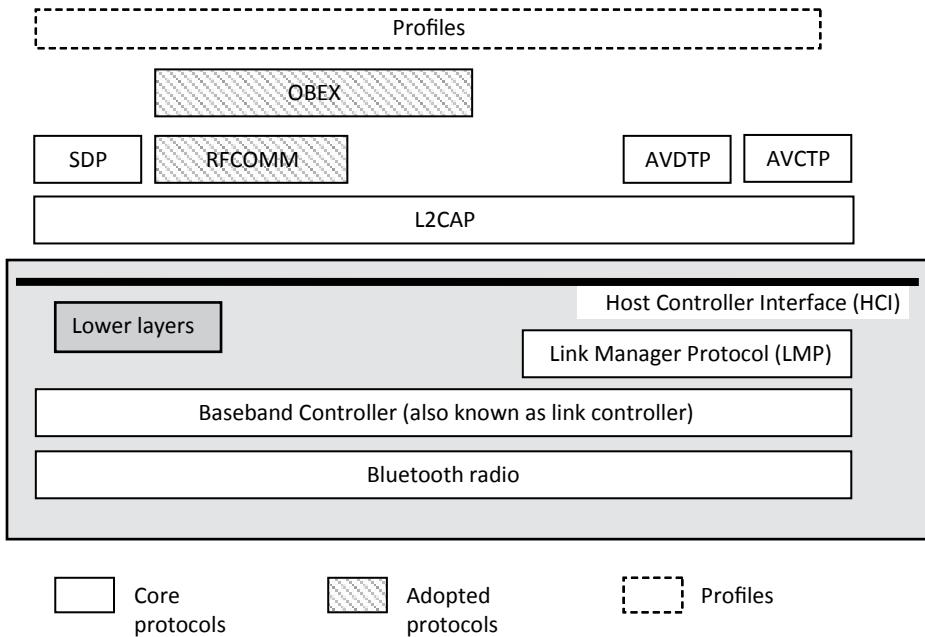


Figure 3.1 Lower layers in Bluetooth protocol stack.

- b. Power Class 2: Maximum output power of 2.4 mW.
- c. Power Class 3: Maximum output power of 1 mW.

3.2.1 Frequency Band and Hopping

Bluetooth uses the globally unlicensed 2.4 GHz ISM band for communication.

Bluetooth divides the frequency band into 79 channels. Each channel is 1 MHz wide. To combat interference, the Bluetooth devices change channels up to 1600 times per second. So even if there is noise on one channel, the next transmission is on a different channel which may be noise free.

What is ISM?

ISM stands for Industrial, Scientific and Medical radio band. Besides 2.4 GHz, these bands include frequencies in the range of 13.560 MHz, 27.120 MHz, 5.8 GHz, 24.125 GHz and several more. These bands are reserved internationally and do not require any special license to operate.

ISM bands are shared by several devices including Remote control toys, cordless phones, near field communication, wireless LAN, etc. Some microwave ovens also generate interference in these bands. Since these bands may be shared by many devices, different wireless technologies employ different mechanisms to combat interference.

The pattern of changing the channels is pseudo-random so that all devices which are communicating with each other know which frequency to hop to next.

The frequencies of various channels are derived from the formula:

$$f(k) = 2402 + k \text{ MHz}, k = 0, \dots, 78$$

This is called *frequency hopping*. The set of devices that communicate with each other follow the same hopping pattern so that they can listen to data sent by the other devices. This set of devices is referred to as a piconet which is the fundamental unit of communication in Bluetooth. Piconet will be explained in detail in the next section.

3.3 Baseband Controller

The Baseband controller (also referred to as the Link Controller) performs the following major functions:

1. Management of physical channels and links for single or multiple links.
2. Selection of the next hopping frequency for transmitting and receiving packets.
3. Formation of piconet and scatternet.
4. Formation of packets and then giving them to the Bluetooth radio for transmission.
5. Inquiry and Inquiry Scan.
6. Connection and Page Scan.
7. Security (including data encryption).
8. Power management (including low power modes).

3.3.1 Topology—Piconet and Scatternet

Bluetooth supports both point-to-point connections and point-to-multi-point connections. In point-to-point connection, the physical channel is shared by two devices while in point-to-multi-point connection it is shared by multiple devices.

The piconet is the smallest unit of communication in Bluetooth. Two or more devices sharing the physical channel are said to form a piconet. It is characterized by one Master and up to seven active Slaves. All the devices are synchronized to each other in terms of clock and frequency hopping pattern. This common piconet clock is the same as Master's clock and frequency hopping pattern is determined by Bluetooth device address (BD_ADDR) and clock of the Master.

Several piconets may co-exist in proximity with each other without interfering with each other. This is because each piconet will have its own Master and thus its own frequency hopping pattern. The chances of two pseudo random frequency hopping patterns which are hopping on 79 different frequencies to select the same frequency for next transmission are quite remote.

A scatternet is formed when two piconets share a device. The shared device participates in the two piconets in a Time Division Multiplexing manner. So it

participates in the first piconet for some time and then participates in the second piconet for the remaining time. Before moving to the second piconet, the device puts itself in low power mode in the first piconet so that the other devices of the first piconet are aware of its temporary absence. This can be extended further to any number of piconets.

Figure 3.2 shows three piconets joined together to form a scatternet. Two scenarios are shown:

- Device M2 is Master in one piconet and Slave in the second piconet.
- Device S7 is Slave in two piconets.

It's not possible for a device to be in a Master in two piconets. Why?

This is because the piconet is defined by the Master's BD_ADDR and clock. So, in effect, all devices that are synchronized with that Master form one single piconet and not two different piconets.

3.3.2 Time Division Duplex

The physical channel is divided into slots in the time domain. Each slot is 625 microseconds and a packet may be sent in 1, 3, or 5 slots depending on the length of the packet. The Master and Slave send the packet alternately in slot pairs. A slot pair starts with a Master transmitting a packet to one of the Slaves. The packet is 1, 3, or 5 slots in length. The response to that packet is received from the Slave (to which the Master had sent the packet) in the next slot. The response packet may also be 1, 3 or 5 slots in length.

The Master can send packets to a Slave in only EVEN Slots. The Slave can send packets to Master only in ODD slots. This is illustrated in Figure 3.3.

3.3.3 Adaptive Frequency Hopping (AFH)

Normal frequency hopping provides some level of protection to interference from other devices in the sense that if another device (whether Bluetooth or not) is using the same RF channel and the data is corrupted due to interference, then the data would be retransmitted next time on the next pseudo random channel. It is possible that the next pseudo random channel is noise free and the packet gets received successfully. This will still lead to loss of throughput since there are retransmissions in case of interference and retransmissions need bandwidth.

The Bluetooth 1.2 specification introduced AFH which helps to increase the immunity of Bluetooth devices against interference generated by other systems in the ISM band.

Also, AFH serves another purpose of reducing the interference caused by Bluetooth on other devices in the ISM band.

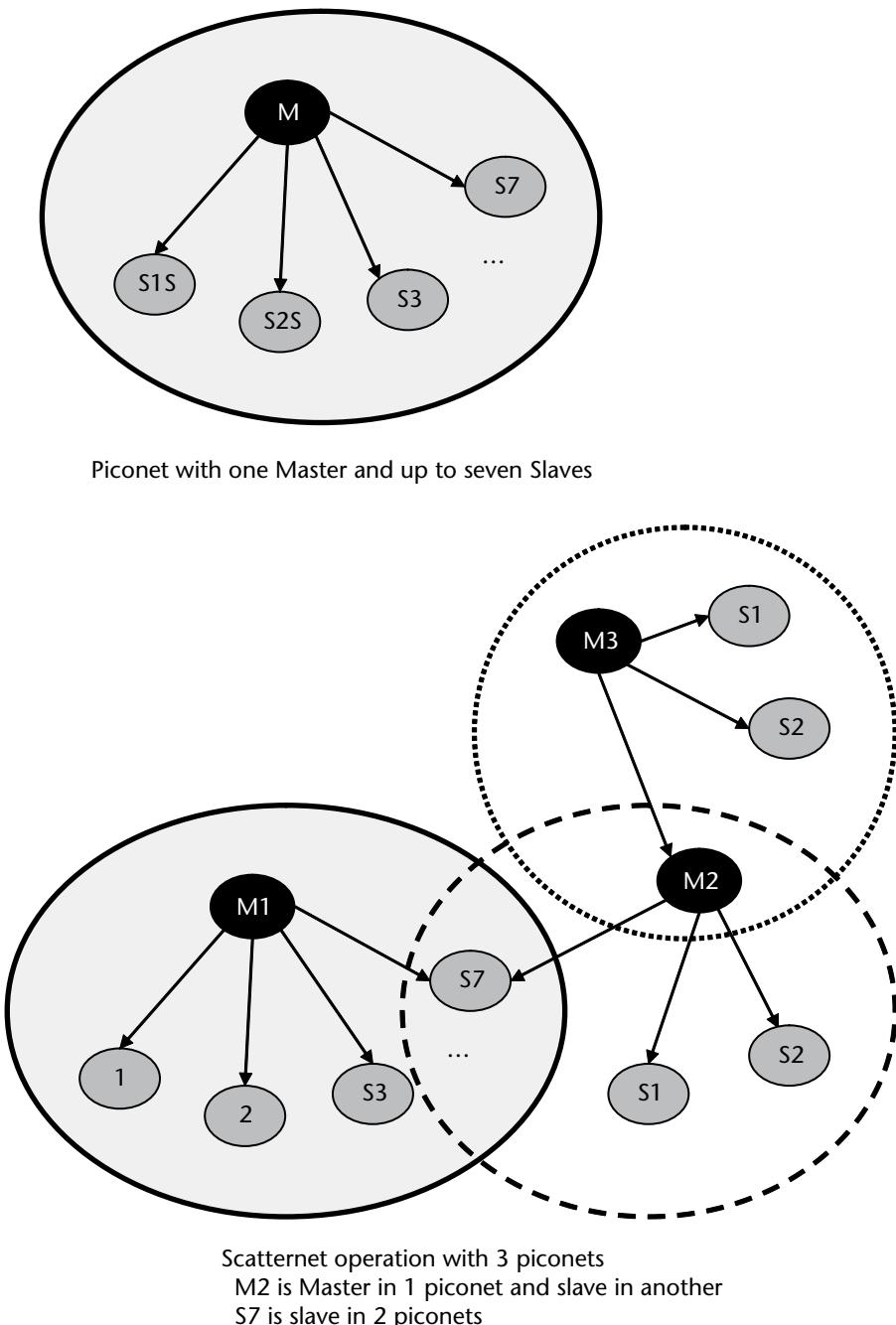


Figure 3.2 Bluetooth topology.

When AFH is enabled, the channels which have interference are marked as unused. The Master informs the unused channels to the Slaves and these channels are excluded from the frequency hopping pattern. So, even though the number of channels on which the frequency hopping occurs decreases, there is no decrease in the throughput. If AFH were not enabled, the frequency hopping would have occurred

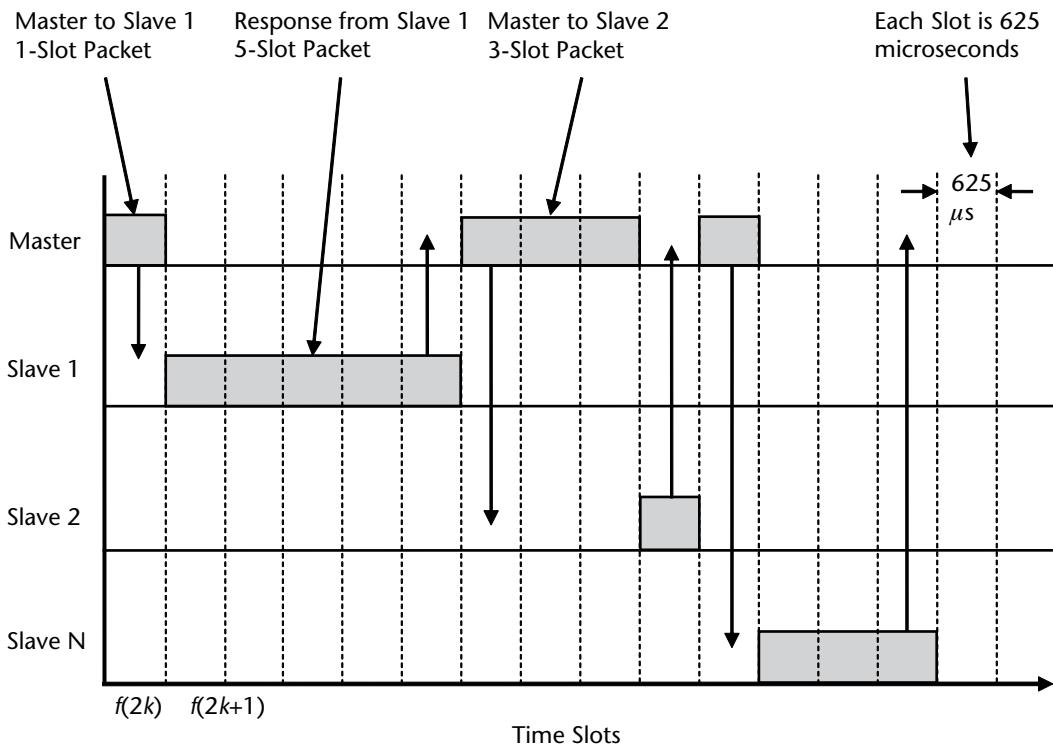


Figure 3.3 Time Division Duplex.

on some of the bad channels as well and it would have resulted in retransmissions. The retransmissions would have led to decrease in the throughput.

With AFH, the number of channels in use can be reduced until it reaches 20. Remember that without AFH the number of channels used by Bluetooth is 79.

To use AFH, the Master maintains a channel map in which it classifies a channel as used or unused. It keeps on updating the channel map based on information that it gathers about whether the packets could be transmitted successfully or not on a particular frequency. It can gather this information based on whether it received an acknowledgement of the packet that it sent or not.

The Slaves also help the Master by providing information on whether the channels are good or bad. Based on this information, the Master can mark the channel as used or unused.

AFH is enabled by the Master after a connection is made.

The Master may also request Channel Classification information from the Slave using LMP_channel_classification_req PDU. (The LMP PDUs are exchanged between the link manager of the Master and the link manager of the Slave. These will be explained in detail later). A Slave that supports this feature periodically provides the information on whether the channels are Good, Bad or Unknown using the LMP_channel_classification PDU.

The Master can use this information to update its channel map. It sends the updated channel map to the Slave using the LMP_set_AFH PDU.

The updated channel map is sent to Slaves periodically so that only the channels marked as *used* are used during frequency hopping.

In a piconet it's possible that a Master enables AFH with connection to some Slaves but disables it for some other Slaves. This will especially be the case if some Slaves support Bluetooth 1.2 or newer specification while others support an older version.

3.3.4 Master, Slave Roles and Role Switch

As mentioned earlier, a piconet consists of one Master and up to seven Slaves. The devices in the piconet are synchronized to a common clock and frequency hopping pattern. This synchronization reference is provided by the Master.

When a device decides to connect to another device (maybe after doing an inquiry), it initiates the connection procedure. By default, the device which initiates the connection becomes the Master. It is also possible for the other device (which is acting as Slave) to become the Master by doing a role switch. It can do so either during the connection establishment itself or any time after the connection is established.

The procedure to swap the roles of two devices connected in a piconet is called role switch. It is initiated using the HCI_Switch_Role command. (HCI commands are sent by the host to the controller on the HCI interface to request the controller to take specific action. These will be explained in detail later).

Either of the two devices (Master or the Slave) can initiate a role switch.

One interesting point to note at this stage is that LE does not allow Role Switch in order to keep things simple. We'll come to that later when we dig deeper into LE.

3.3.5 Channel, Transport and Links

The Bluetooth data transport system follows a layered architecture. This is shown in Figure 3.4. The lowest layer comprises the physical channel that provides the communication mechanism between the devices on a Time Division Duplex channel. At the top, L2CAP channels provide communication channels for upper layers of the protocol stack to exchange data. Each of the layers is explained in the following sections going from bottom to top.

3.3.5.1 Physical Channel

The physical channel is the lowest architectural layer for communication in Bluetooth. It is characterized by the pseudo-random frequency hopping sequence, the specific slot timing of the transmissions, the access code and the packet header encoding. This means that for two or more devices to communicate with each other, their radios must be tuned to the same RF frequency at the same time. Besides this, the radios must be in range to listen to each other's transmissions.

When a device is synchronized to the timing, frequency and access code of a physical channel it is said to be connected to that channel.

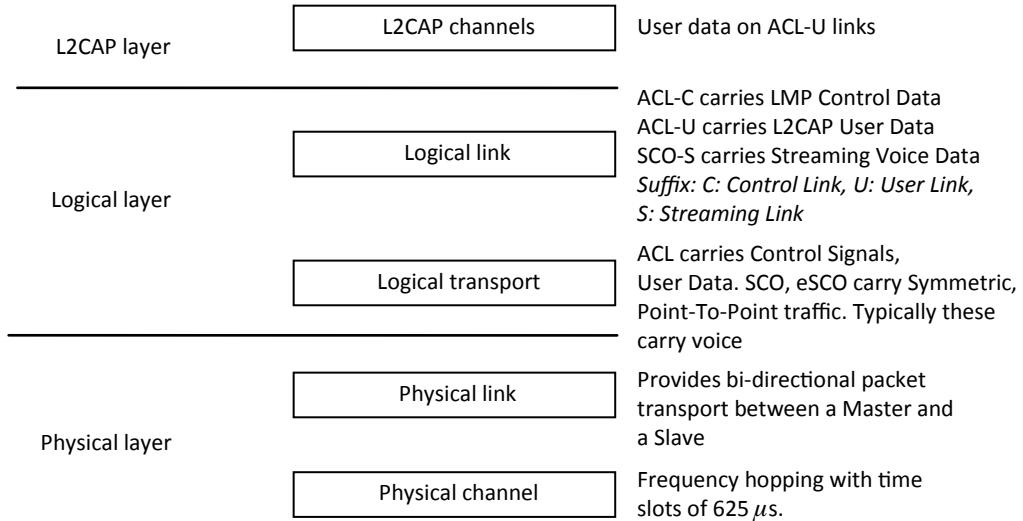


Figure 3.4 Data transport architecture.

The physical channel is subdivided into time units called slots. Each slot is 625 μ s and full duplex transmission is achieved using Time Division Duplex (TDD) scheme. A packet may occupy 1, 3, or 5 slots depending on the packet type. As mentioned earlier a frequency hopping scheme is used and each packet is transmitted on the next pseudo-random frequency. The frequency hop occurs once per packet and not once per time slot.

Four physical channels are defined:

- *Basic Piconet Channel:* This channel is used for communication between devices in a piconet.
- *Adapted Piconet Channel:* This is similar to basic piconet channel and is used with devices that have AFH enabled.
- *Inquiry Scan Channel:* This channel is used for discovering the devices.
- *Page Scan Channel:* This channel is used for creating connections with the devices.

A device can use only one physical channel at a time and has to use time division multiplexing between the channels to support other channels if it has to support multiple concurrent operations.

For example, if a device is already connected to another device but still needs to be in discoverable mode so that other devices can discover it. It already has a physical channel established with another device which is either the basic piconet channel or adapted piconet channel depending on whether AFH is enabled on that channel. To support discoverable mode along with the physical channel, it has to use time division multiplexing to support the inquiry scan channel. This feature would be useful when forming a scatternet or adding more devices to the piconet.

3.3.5.2 Physical Link

A physical link defines the baseband to baseband connection at the lowest level. The physical link is what actually carries the data physically over the channel. There can be only one physical link between two Bluetooth devices. This means that if we consider two devices—one Master and the other Slave—there will be only one physical link between these two devices although at a logical level there could be two or more links on top of it. (Logical links are explained in the next section.)

Within a physical channel, a physical link is established between any two devices that transmit packets. In a piconet, a physical link is formed between the Master and each Slave. It is not permitted to form a physical link between two Slaves.

It is worthwhile to note that Bluetooth does not support direct communication between two Slaves. If the two Slaves need to talk, then they should disconnect from their respective piconets and form a separate piconet with one of them acting as the Master and the other acting as a Slave.

A physical link is always associated with exactly one physical channel. The physical link supports link supervision so that a connection loss can be detected. This could happen, for example, when the device moves out of range or one of the devices is powered off. The physical link also supports encryption of the packets that are transmitted.

3.3.5.3 Logical Transport

A logical transport is formed on a physical link to transfer control signals and synchronous or asynchronous user data. The following three Logical Transports are most commonly used.

1. Asynchronous Connection Oriented (ACL) Logical Transport.
2. Synchronous Connection Oriented (SCO) Logical Transport.
3. Extended Synchronous Connection Oriented (eSCO) Logical Transport.

Asynchronous Connection Oriented (ACL) Logical Transport

The Asynchronous Connection Oriented (ACL) transport is used to carry control signals, user data and broadcast traffic. It provides a packet-switched connection between the Master and all Slaves that are active in the piconet. The default ACL connection is created when a device joins a piconet. This transport is used to send data in bursts whenever data is ready and whenever a slot is available and not reserved for the synchronous connection oriented transport.

Between a Master and a Slave only one single ACL logical transport is created. The higher layer protocols multiplex their data on top of this logical transport. Packet retransmissions are supported on the ACL transport to ensure data integrity when the packets are delivered to the application on the receiver side.

If a SCO or eSCO transport exists between the Master and the Slave, then the time slots are first allocated to the SCO or eSCO transport. (This is because SCO and eSCO slots carry time bounded data like voice which need guaranteed bandwidth.) The remaining slots are reserved for the ACL transport. (The SCO and eSCO transports are covered in the next section.)

You might be wondering why the acronym is ACL and not ACO?

This is for historical reasons. In previous versions of the Bluetooth specification (Prior to Core Spec 1.2) this link was called Asynchronous Connection Less and abbreviated as ACL. That abbreviation still continues...

Synchronous Connection Oriented (SCO) Logical Transport

The Synchronous Connection Oriented (SCO) transport provides support for continuous transfer of data. It reserves time slots on the physical channel at the time of establishment of the connection so it can be considered as a circuit-switched connection. It is a symmetric point-to-point link between a Master and a particular Slave. SCO links carry 64 kb/s of information in both directions. Typically this is useful in transporting voice streams where timing of the data is as important as the data content.

If a data packet is corrupted during transmission it is not retransmitted. This is suitable for voice data since if there are retransmissions, these would introduce delays. Hence the retransmitted packets may arrive too late to be played. On the other hand if the average number of dropped packets is not too high, it may not lead to noticeable degradation in the user experience.

Unlike ACL logical transport, the SCO (and eSCO) logical transports do not support multiplexed logical links on top of them. So there is no further layering of any stack components above SCO (and eSCO) logical transport.

A Master may support up to three SCO links to the same Slave or to different Slaves. A Slave may support up to three SCO links from the same Master or two SCO links if the links originate from different Masters. (SCO links can originate from different Masters in the scatternet topology where a Slave is connected to two Masters in two different piconets.)

A Master can support an ACL transport and a SCO transport simultaneously with the Slave. One practical example of this is when connecting to a mono headset from a mobile phone. The mobile phone establishes both the ACL transport as well as the SCO transport with the headset. The ACL transport is used to carry commands to create a connection to the headset, increase/decrease volume, disconnect, etc. while the SCO transport is used to carry the voice traffic.

In theory the Master can support up to 3 SCO transports simultaneously with an ACL transport to the Slave. In practice this is not used much. Most of the practical implementations use only one SCO link to transfer voice data along with ACL link to transfer control data.

Extended Synchronous Connection Oriented (eSCO) Logical Transport

The extended synchronous connection oriented (eSCO) link is a symmetric or asymmetric point-to-point link between a Master and a particular Slave. This is in contrast to SCO link which supports only symmetric traffic. The eSCO logical transport also reserves slots similar to SCO logical transport and can be considered as a circuit-switched connection.

The support for eSCO Logical Transport was added from version 1.2 of the Bluetooth specification as an enhancement to the functionality of the SCO Logical Transport.

It offers the following extensions over the SCO link:

- The transmission may be symmetric or asymmetric in the case of eSCO while it's only symmetric in the case of SCO. This means that in the case of eSCO it's possible to use 3-slot packets in one direction and 1-slot packet in the reverse direction.
- eSCO logical transport provides better reliability (and voice quality) by providing a limited number of retransmissions of packets that get corrupted. The retransmission slots are allocated right after the reserved slots and are used only if needed. This offers better voice quality as compared to SCO transport. The number of retransmission slots is limited to ensure that the timing constraints of voice data are still met in case of retransmissions and that packets don't arrive too late to be played.
- eSCO provides higher data rates as compared to SCO links. This is achieved by introducing additional eSCO packet types related to Enhanced Data Rate.
- eSCO is used to transfer not only 64 kb/s voice packets, but also any other types of packets which require constant traffic. For example the eSCO Logical Transport is used to transfer *Wide Band Speech* data.

What is Wide Band Speech (WBS)?

The bandwidth of the sound signals used in telephony is limited to about 200–3400 Hz. This is referred to as Narrow Band Speech. As per Nyquist sampling theorem, it is sampled at the rate of 8 KHz. (The Nyquist sampling theorem is beyond the scope of this book. In brief it states that if a signal is band limited at B Hz, then it can be perfectly reconstructed from a sequence of samples if the sampling frequency is greater than $2 * B$ samples per second).

The 200–3400 Hz limitation on the bandwidth imposes a limit on the communication quality in Narrow Band Speech.

Most of the frequencies in speech signals are present below 7000 Hz. So increasing this bandwidth to 50–7000 Hz increases the naturalness of speech and gives the feeling of face-to-face communication. This is referred to as Wide Band Speech and is quite frequently used in cellular systems, voice over IP, etc. The signal is sampled at the rate of 16 KHz in the case of Wide Band Speech.

The 3G systems support Wide Band Speech and since the audio may be routed to a Bluetooth headset, the Bluetooth headsets also need to support Wide Band Speech to ensure that the voice quality is not degraded when routing the call over a Bluetooth link.

3.3.5.4 Logical Links

The logical links are supported on top of the logical transport. Five types of logical links are defined. These are described in Table 3.1.

The LC (link control) Logical link is carried in the packet headers. It contains low level information like acknowledgment/repeat request (ARQ), flow control and payload characterization. All other links are carried in the packet payloads.

The control ACL (ACL-C) logical link carries the control information between the link managers and the Master and Slave(s). It has a higher priority than ACL-U logical link.

The user ACL link (ACL-U) is used to carry user data. This link is used by the L2CAP layer. For example this link is used for transferring a file during a file transfer using FTP profile. (We will discuss L2CAP layer and FTP profile in the next chapter.)

The SCO-S link is supported on top of the SCO logical transport. Each SCO-S link is supported by a single SCO logical transport. Same is the case for eSCO-S logical link which is supported on top of the eSCO logical transport.

Most of the time the C, U, or S suffix is dropped and the links are just referred to as ACL, SCO, and eSCO, links. This does not lead to any ambiguity since the type of links being used is clear from the context. For example, if link manager data is being transferred then ACL would mean ACL-C logical link. Similarly if user data (L2CAP) is being transferred, ACL would mean ACL-U logical link. Some practical scenarios are explained in the Figure 3.5 for a better understanding of these links.

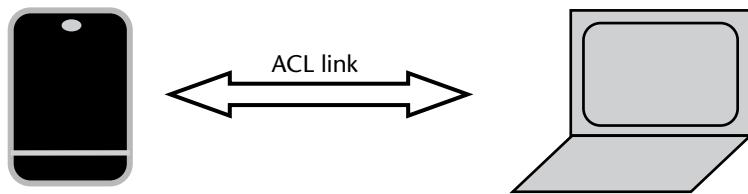
Scenario 1 shows a file transfer between a mobile phone and laptop. In this scenario, an ACL link is established between these two devices and the file transfer happens on this link.

Scenario 2 is a bit more complex. It shows a connection between a mobile phone and mono headset and routing of an audio call (from the cellular network) on the Bluetooth link to the headset. In this scenario, the initial connection establishment happens on the ACL link. Once the two devices are ready to exchange audio (For example, when the user presses the button on the headset to accept incoming call), the SCO or eSCO link is established.

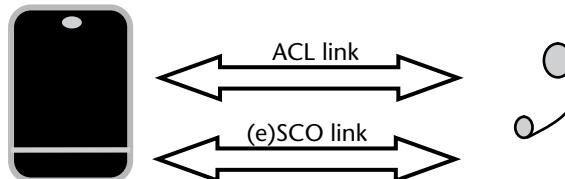
Scenario 3 shows the case where a stereo headset is connected to a mobile phone and a music file (MP3) is streamed over the Bluetooth link. The MP3 file is

Table 3.1 Types of Logical Links

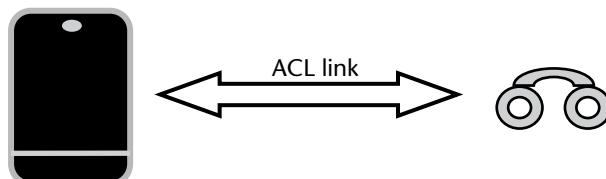
Link Type	Traffic Type	Carrier
Link Control (LC)	Low level link control information.	Mapped in the packet header and carried in every packet. (Except identity packet since it does not have a packet header).
ACL Control (ACL-C)	Control information between the link manager layers of Master and Slave(s).	Mostly ACL logical transport. May also be carried in data part of DV packets on SCO. (DV will be discussed later).
User Asynchronous/Isochronous (ACL-U)	Asynchronous or isochronous user information between L2CAP layers	
User Synchronous (SCO-S)	Synchronous user information	SCO logical transport
User Extended Synchronous (eSCO-S)	Synchronous user information	eSCO logical transport



Scenario 1: File transfer between a mobile phone and a laptop



Scenario 2: Routing a voice call from mobile phone to Bluetooth mono headset



Scenario 3: Listening to music on a stereo Bluetooth headset

Figure 3.5 Usage of different link types.

generally encoded in a different format by the mobile phone (by default Bluetooth specification specifies the usage of an SBC codec) and then streamed on the ACL link.

The link to transfer *music* should not be confused with the link to transfer *voice*. Music links require much higher bandwidth as compared to voice links. This is because voice is sampled at 8 KHz (or 16 KHz for Wideband speech) while music is sampled at (up to) 48 KHz.

The SCO link has bandwidth to carry only 8 KHz voice. This can be enhanced to 16 KHz by using eSCO links. This is still not sufficient for music links.

So music is transferred over ACL links that provide a much higher bandwidth. In fact even in the case of ACL links, the bandwidth is not sufficient to transfer raw music. So the music file is first encoded and then transmitted. It is decoded after reception and then played back.

If this is not clear at this stage, just hang on and we will revisit the subject when we discuss Bluetooth profiles and in particular the A2DP profile in the next chapter.

3.3.6 Packet Format

The format of BR and EDR packets is shown in Figure 3.6.

All transmissions begin with an access code. This is the only mandatory part of a packet; all other parts are optional. It is used for synchronization, DC Offset compensation, and identification. All packets sent on the same physical channel are preceded by the same access code. It indicates the arrival of a packet on the receiver side.

Three different access codes are defined:

- *Device Access Code (DAC)*: This is used during pre-connection phase when the devices are trying to connect to each other.
- *Channel Access Code (CAC)*: This is used when the devices are connected and is prefixed to all packets that are exchanged between the devices in a piconet. The receiver uses this code to check if the packet belongs to the piconet that it is in or not.
- *Inquiry Access Code (IAC)*: This is used during the inquiry phase when the devices try to find out other devices in the Bluetooth vicinity.

The packet header contains the following information:

- *Logical Transport Address (LT_ADDR)*: Indicates the logical transport address of the destination device.
- *Packet Type (TYPE)*: Various packet types are defined for ACL, SCO and eSCO logical transports. These will be explained in detail in the next section.
- *Flow control (FLOW)*: To start and stop the flow of packets depending on whether the receiver is capable of receiving more packets or not. (For example if the buffers in the receiver are full, then this bit will be used to indicate to the transmitter to stop sending further packets)
- *Acknowledgment (ARQN)*: Positive or negative acknowledgment to indicate the transmitter of a successful transfer of payload data after checking the CRC.
- *Sequence Number (SEQN)*: Sequential numbering of packets to ensure that packets are received in the correct order.
- *Header Error Check (HEC)*: To check the header integrity on reception. If the HEC is incorrect on the receiver side, the entire packet is discarded.

ACCESS CODE	HEADER	PAYLOAD
-------------	--------	---------

Standard Basic Rate (BR) packet format

ACCESS CODE	HEADER	GUARD	SYNC	PAYLOAD	TRAILER
-------------	--------	-------	------	---------	---------

Standard Enhanced Data Rate (EDR) packet format

Figure 3.6 BR and EDR Packet Formats.

The payload contains the data that is to be transferred along with the CRC. The data can have asynchronous data field, synchronous data field or both. The ACL data packets have the asynchronous data field and the SCO/eSCO packets have the synchronous data field. There are a special DV packet types which are defined as a part of SCO packet types. These have both asynchronous and synchronous data field.

For EDR packets, GUARD and SYNC fields are placed before the payload and a TRAILER field is placed after the payload. This is because EDR packets use a different modulation scheme for the payload to achieve higher data rates. These fields are used to facilitate the change in the modulation scheme just before payload data is transmitted.

3.3.7 Packet Types

Different logical transports use different packet types. As mentioned in the previous section, the TYPE field within the packet header is used to indicate the various packet types. Before getting into the details of the packet types, it's important to understand the nomenclature. Figure 3.7 shows the significance of each of the alphabets in the packet types 3DH5 and 2EV3.

The Packet types can be classified into 3 broad categories:

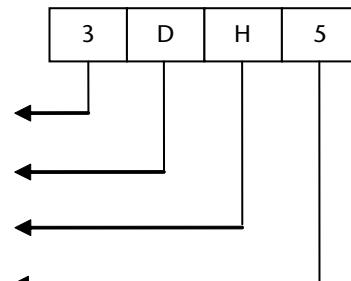
1. Link Control Packets.

EDR Packets are prefixed with 2 or 3 to indicate the type of modulation.
For BR packets this field is absent

D indicates ACL Data Packet

High Rate Packet

The packet occupies 5 Slots.

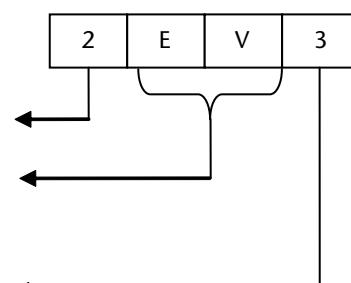


Asynchronous packets

EDR Packets are prefixed with 2 or 3 to indicate the type of modulation.
For BR packets this field is absent.

EV: Enhanced Voice: eSCO packet.
HV, DV: Voice or Data-Voice (SCO Packet)

The packet occupies 3 Slots.



Synchronous packets

Figure 3.7 Packet Nomenclature.

2. ACL Packets.
3. Synchronous Packets.

These are briefly described in the following sections.

3.3.7.1 Link Control Packet Types

There are 5 link control packet types:

- ID;
- NULL;
- POLL;
- FHS;
- DM1.

The ID (Identity) packet is used before connection establishment. It's a very robust packet and contains the device access code (DAC) or inquiry access code (IAC). It does not have a packet header or payload.

The NULL packet has no payload. It is generally used to indicate success of the previous transmission or status of receive buffer. Typically this is used when a Slave receives a packet from the Master and it has to acknowledge that packet but it has no data to send back to the Master. So the Slave uses the NULL packet to acknowledge. The Slave also uses the NULL packet to indicate to the Master if its receive buffers are full. The Master will then halt transmissions till the time the Slave empties its buffers to receive further packets. The default packet type for data is NULL. This is used if there is no data to be sent. The NULL packet itself is not required to be acknowledged by the Master.

The POLL packet is quite similar to the NULL packet. It also does not have a payload. It is sent only by the Master. The Master generally uses this packet to poll the Slaves to first detect whether they are still present and secondly to check if they have any data to send. Unlike the NULL packet, the POLL packet must be

Table 3.2 Summary of Link Control Packets

Type	Remarks
ID	Identity (ID) packet. This packet is used before connection establishment to pass on an address.
NULL	The NULL packet has no payload. It is generally used to indicate success of previous transmission or status of Rx buffer.
POLL	The POLL packet has no payload. It is generally used by the Master to poll the Slaves. The Slave responds to this with a packet. If it doesn't have any information to send, it responds with a NULL packet.
FHS	The Frequency Hop Synchronization (FHS) contains real time clock information. It is used for frequency hop synchronization before the piconet channel has been established or when existing piconet changes to a new piconet (by means of a role switch).
DM1	The DM1 (Data Medium Rate 1-slot) packet type is used to carry control packets and data packets.

acknowledged by the Slave. If the Slave does not have any information to send, it responds to the Master with a NULL packet.

The FHS (Frequency Hop Synchronization) is a special control packet containing real time information. It contains the BD_ADDR and clock information of the sender. The clock information is updated before each retransmission (since it is real time and the clock would have changed since the last transmission) for this particular packet type. This packet is used for frequency hop synchronization before the piconet channel has been established or when existing piconet changes to a new piconet (by means of a role switch).

The DM1 (Data Medium Rate 1-slot) packet is used by the Link Manager and Link Controller to exchange control packets. Besides control packets, this packet can also be used to carry regular data. The details for this packet type are provided in the ACL Packet Types section.

The scenarios in which these packet types are used are illustrated later in Figure 3.10.

3.3.7.2 ACL Packet Types

The different types of ACL packets are explained in Table 3.3. Different packet types are suitable for different scenarios. For example when high throughput is desired in only one direction, 3-DH5 packets are the most suitable. This is the case when transferring files, downloading data from the internet etc. In such cases the majority of the data is transmitted in one direction and there are only a few bytes that need to be transmitted in the reverse direction.

The information about supported ACL packet types is provided at the time of ACL connection creation. The link managers running on the two devices negotiate the packet types that will be used subsequently based on this information. Only the packet types that are successfully negotiated are used for subsequent transmissions.

Table 3.3 ACL Packet Types

Type	User Payload (bytes)	Symmetric Max Rate (kb/s)	Asymmetric Max Rate (kb/s)		Remarks
			Forward	Reverse	
DM1	0-17	108.8	108.8	108.8	Data – Medium Rate, 1 Slot
DH1	0-27	172.8	172.8	172.8	Data – High Rate, 1 Slot
DM3	0-121	258.1	387.2	54.4	
DH3	0-183	390.4	585.6	86.4	
DM5	0-224	286.7	477.8	36.3	
DH5	0-339	433.9	723.2	57.6	This packet is used for maximum throughput in one direction for Basic Rate (BR) links.
AUX1	0-29	185.6	185.6	185.6	
2-DH1	0-54	345.6	345.6	345.6	
2-DH3	0-367	782.9	1174.4	172.8	
2-DH5	0-679	869.1	1448.5	115.2	
3-DH1	0-83	531.2	531.2	531.2	
3-DH3	0-552	1177.6	1766.4	235.6	
3-DH5	0-1021	1306.9	2178.1	177.1	This packet is used for maximum throughput in one direction for EDR links.

As shown in Table 3.3, the maximum throughput is achieved when using asymmetric data transfer using DH5 packets for BR and 3-DH5 packets for EDR.

3.3.7.3 Synchronous Packet Types

HV (High Quality Voice) and DV (Data-Voice) packets are used for SCO transmissions. The HV packets do not include a CRC and thus these are not retransmitted. The DV is a combined data-voice packet and it has separate sections for voice and data. These packets do not include a CRC on the voice section, but include a CRC on the data section. So in case the data section is not received properly at the remote side, it is retransmitted. The voice section however is not retransmitted and contains the next speech information.

eSCO uses EV packets. These packets include a CRC and retransmission is done if an acknowledgment is not received from the remote side within the retransmission window. EV3, EV4 and EV5 packets are used for Basic Rate (BR) operations. 2-EV3, 2-EV5, 3-EV3, 3-EV5 packets are used for Enhanced Data Rate (EDR) transmissions. As mentioned previously eSCO packets can be used for both 64kb/s speech as well as data at the rates mentioned below.

3.3.8 Link Controller States

There are three major states used in the link controller:

- **STANDBY:** This is the default state of the device. In this state, the device may also enter low power mode to save power. The link controller may leave this state to scan for inquiry or page messages from other devices or to itself inquire or page.
- **CONNECTION:** Once a device knows the address of the device to connect to (using the inquiry procedure), it can create a connection to it. This procedure is known as paging. The device that initiates paging becomes the Master

Table 3.4 Synchronous Packet Types

Type	Symmetric		
	User Payload (bytes)	Max Rate (kb/s)	Remarks
HV1	10	64	
HV2	20	64	
HV3	30	64	
DV	10 + (0–9) D	64.0+57.6 D	Data Voice. This packet type can support both Data and Voice simultaneously. D indicates the Data Payload.
EV3	1-30	96	
EV4	1-120	192	
EV5	1-180	288	
2-EV3	1-60	192	
2-EV5	1-360	576	
3-EV3	1-90	288	
3-EV5	1-540	864	

after the connection is established. In this state, packets can be exchanged between the Master and the Slave(s). This state is left through a detach or reset command.

- **PARK:** A Slave can enter the park state if it does not need to participate in the piconet but still needs to remain synchronized with it. This mode helps conserve power. The Slave wakes up periodically to resynchronize and listen to the channel to decide whether it should come back to active state or not.

Besides this, there are seven substates. These are interim states that are used for discovering devices in the vicinity and establishing a connection. The seven substates are divided into two categories:

Device Discovery Substates

1. *Inquiry scan:* In this substate, the device listens for incoming inquiry requests.
2. *Inquiry:* This substate is used to discover devices in the vicinity.
3. *Inquiry response:* After receiving the inquiry message, the Slave enters this state and transmits an inquiry response message.

Connection Establishment Substates

4. *Page scan:* In this substate, the device listens for incoming connection requests.
5. *Page:* This substate is used by a device to connect to another device. The device that initiates the paging ends up being a Master after the connection is established and the device that was in page scan mode ends up being Slave.
6. *Slave response:* After receiving the page message, the Slave enters this state and transmits a Slave response message.
7. *Master response:* The Master enters this state after it receives the Slave response message.

The different states along with the permitted transitions from one state to another are illustrated in Figure 3.8. As an example, the transition of states and substates during inquiry, connection and disconnection is illustrated in Figure 3.9 and Figure 3.10.

3.3.8.1 Connection State

During the connection state, the packets can be exchanged between the Master and the Slave. The first few packets contain control messages that are exchanged between the link managers of the devices to setup the link and negotiate link parameters. If the connection is no longer needed, the connection state may be left by either a detach or reset command.

During the connection state, the device may be in one of the following three modes:

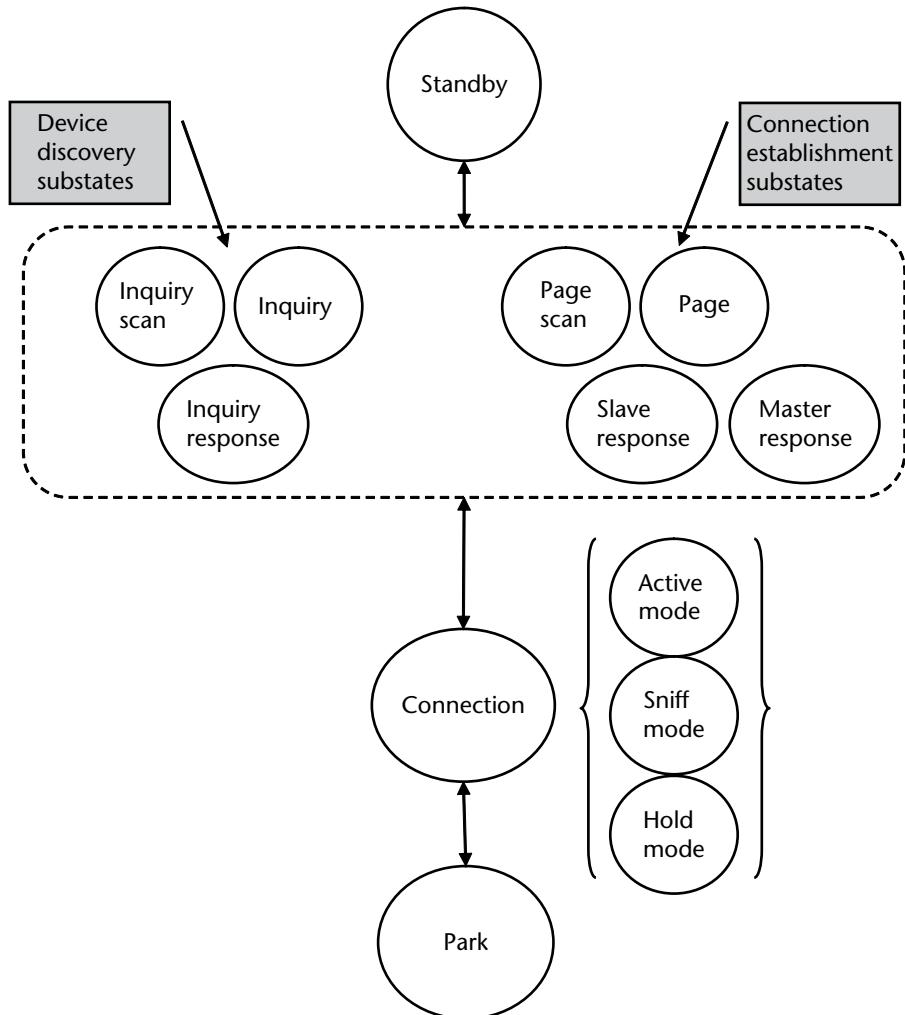


Figure 3.8 Link Controller States.

- Active Mode.
- Hold Mode.
- Sniff Mode.

Active Mode

In the active mode, one Master and up to seven Slaves can be active at any time in the piconet. The Master schedules the transmissions in different slots based on the SCO/eSCO connections that are established and the traffic requirements from different Slaves. Even if the Master does not have any data to send, it keeps on regularly transmitting POLL packets to the Slaves to keep them synchronized to the channel. If the Slave has data to send, it sends the appropriate data packet. Else it responds with a NULL packet to acknowledge the POLL packet.

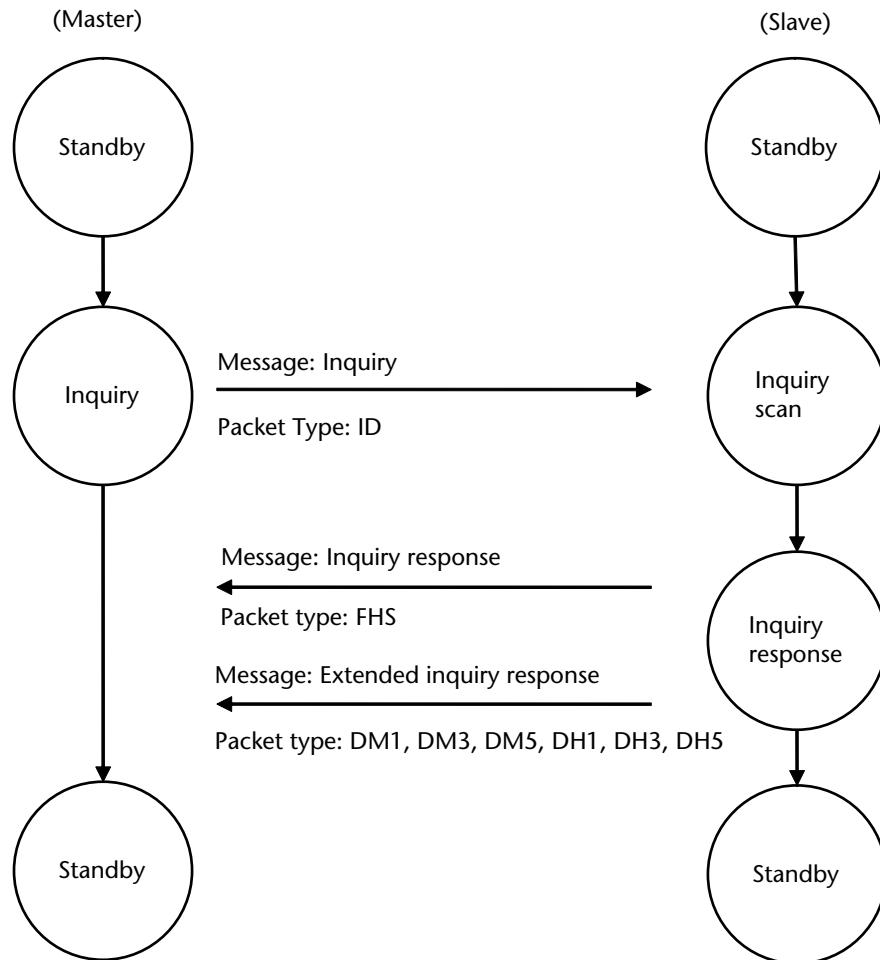


Figure 3.9 Link Controller Messages and States during inquiry.

Each Slave is assigned a 3-bit address called Active Member Address (AM_ADDR). The Master uses this address in the packets to identify the Slave for which those packets are meant.

Hold Mode

During the connection state, the ACL logical transport to the Slave can be set to hold mode. In this mode, the ACL packets are not exchanged. The voice links (SCO, eSCO) are still supported.

Before entering the hold mode, the Master and the Slave agree on the time duration for which the Slave will remain in hold mode. During this time duration, the Slave can either go to low power mode or do other activities like scanning, paging, inquiring or participating in another piconet.

On expiration of the time duration of the hold mode, the Slave wakes up and listens for transmissions from the Master.

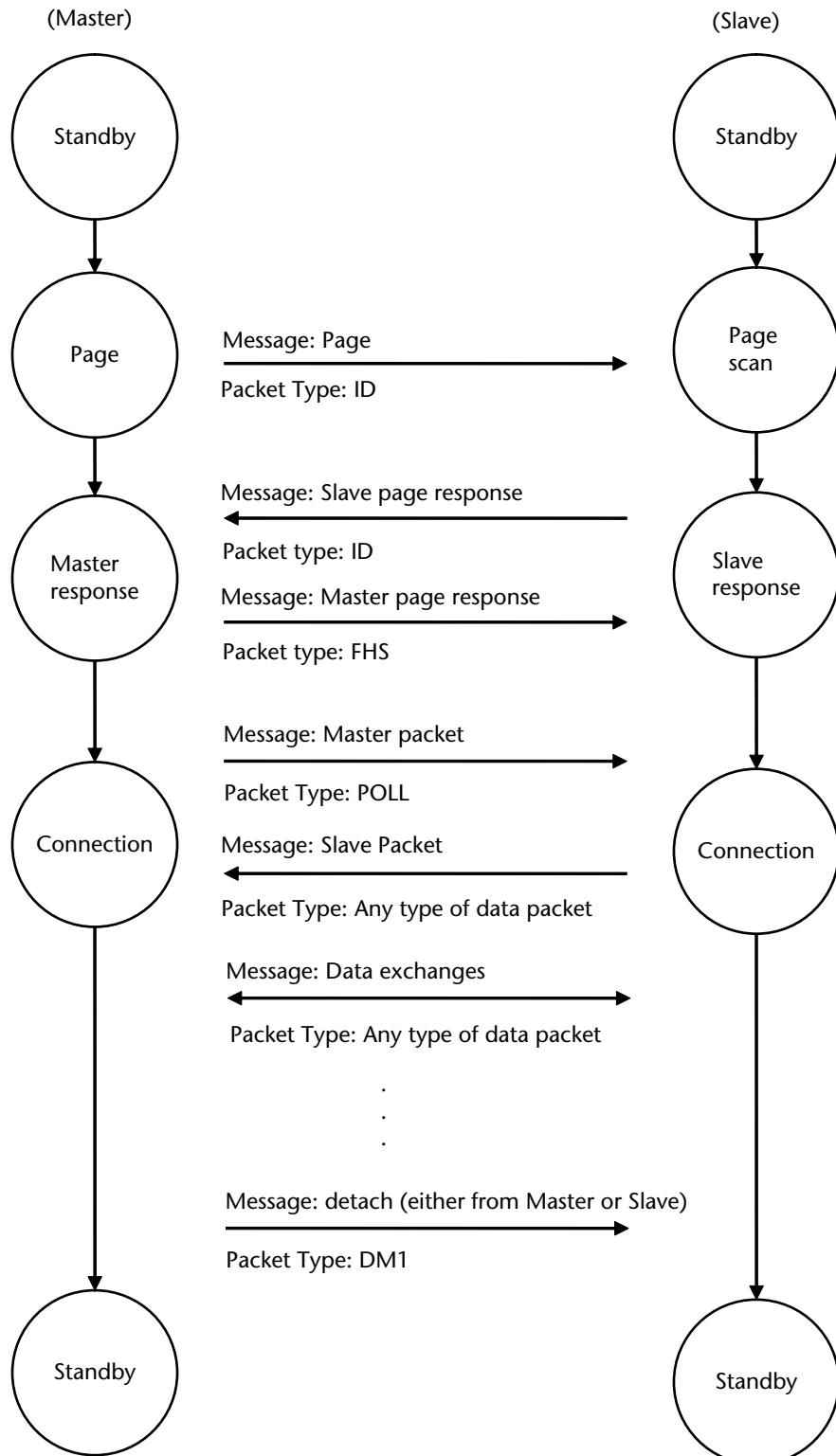


Figure 3.10 Link Controller Messages and States during connection and disconnection.

Sniff Mode

The sniff mode is the most commonly used low power mode. It affects only the ACL logical transport and does not affect the SCO or eSCO logical transports. This means that when this mode is activated, the transmissions on ACL logical transport are reduced, while the transmissions on SCO or eSCO logical transport continue as usual.

In the sniff mode, the device can become absent from the piconet for a certain period of time. So a duty cycle is defined consisting of the duration for which the device will be present and active in the piconet versus the time it will be absent.

A practical use of the sniff mode is when a connection is made between a mobile phone and a mono headset. The two devices can continue to be connected for several days even when there is no active voice call going on. This ensures that as soon as there is an incoming call, time is not wasted in first creating an ACL connection and then creating a SCO connection to route the call. So, to conserve power, the ACL connection between the mobile phone and the headset is put in sniff mode. Most Bluetooth headsets implement the sniff mode.

Another use of the sniff mode is when a Bluetooth keyboard is attached to a PC or laptop. Since the user may not be actively typing all the time, the ACL link is put into sniff mode after an inactivity timer. As soon as the user types a key, the link is brought back into the active mode.

Besides saving power, the sniff mode may also be useful when the device needs to enter into a scatternet scenario. In this scenario, the device may put the ACL link of the current piconet into sniff mode and then become active on another ACL link in the second piconet. So by tuning the sniff parameters it may have periods in which it is absent in first piconet and present in the second piconet and vice versa.

This mode requires a good bit of attention at the link level, especially when there is a connection in sniff mode for a long duration. Any clock drifts on either side while in sniff mode for a long time, could lead to the loss of packets or even loss of the link. Generally controllers wake up time to time in between to synchronize with the Master and go back to sniff mode again to avoid such problems.

3.3.8.2 Park State

A Slave can enter park state if it does not need to participate in the piconet channel but still needs to be synchronized with it. It's a very low power mode with very little activity. In this mode both ACL and SCO or eSCO traffic is stopped. The Slave still remains synchronized to the channel.

The parked Slave wakes up at regular intervals to listen to the channel to check whether there are any messages for it. The Master informs the parked Slaves about any messages for them through broadcast messages.

When a device is put into park state, a different address known as Parked Member Address (PM_ADDR) is assigned to it by the Master during the parking procedure. The Active Member Address (AM_ADDR) that was earlier used by the Slave in active mode is freed up so that it can be re-used by the Master to make a connection with some other device.

Park state is also used when more than seven Slaves need to be connected to a single Master. At a time only seven Slaves can be active. The remaining Slaves are

put into park state. If a parked Slave needs to be unparked then it can be swapped with another active Slave that will be put to park state.

3.4 Link Manager (LM)

The Link Manager performs the functions of link setup and control. The Link Manager Protocol (LMP) is used for communication between the Link Managers of the two devices. The communication messages between the devices are carried over the ACL-C logical link. (As mentioned in the previous section, ACL-C logical link carries control data while the ACL-U logical link carries the user data).

The LMP messages are transmitted using DM1 packets. If HV1 SCO link has been established between the two devices and the length of the payload is less than 9 bytes, then DV packets may also be used. In practice, DV packets are seldom used.

The packets transmitted by LM are referred to as PDUs. Each PDU contains the following fields:

- Transaction ID (TID);
- OpCode;
- Payload.

A transaction is a set of messages that are transmitted to achieve a particular purpose. It may have more than one PDU and all PDUs contain the same TID. The OpCode is used to uniquely identify different type of PDUs. The LMP messages are denoted as *LMP_message_name*. For example, the message to establish a connection is denoted as *LMP_host_connection_req*. A response PDU to this could be *LMP_accepted* or *LMP_not_accepted*.

The functionality supported by LMP includes the following:

- *Connection Control*: This includes procedures for creation and removal of a connection as well as controlling all aspects of a connection.
- *Security*: This includes procedures for authentication, pairing, and encryption.
- *Informational Requests*: This includes various PDUs that the LMs exchange with each other to get the characteristics of the remote device. For example, the LM may find out the version of the remote device using the *LMP_version_req* PDU. Similarly it may find out the list of features supported by the remote device by sending the *LMP_features_req* PDU. The remote side would respond using an *LMP_features_res* PDU.
- *Role switch*: If the Slave device wants to reverse the roles and become the Master, then these procedures are used. A practical example of this would be if a Bluetooth mouse establishes a connection with a laptop. Since the connection establishment is initiated by the mouse, it becomes the Master of the connection. After the connection is established the laptop may prefer to become the Master of the connection so that it can control other peripherals as well. So it can invoke the role switch procedures to become the Master.

- *Modes of operation:* In the baseband section, hold mode, sniff mode, and park state were explained. These different modes of operations are invoked by the link manager. Besides this the link managers of the two devices negotiate the parameters for these modes.
- *Logical transports:* The LM invokes the procedures to create and remove the SCO and eSCO logical transports. The parameters of these transports are also negotiated by the LMs of the two devices. These can also be renegotiated at any time during the connection. For example an eSCO link is established by sending an LMP_eSCO_link_req and removed by sending the LMP_remove_eSCO_link_req.
- *Test Mode:* This includes various PDUs for certification and compliance testing of the Bluetooth radio and baseband.

The connection control and security functionality of link manager is explained in further detail below.

3.4.1 Connection Control

The connection functionality includes all procedures related to creation and removal of a connection and controlling all aspects of the operation of that connection.

Some of the major functions include the following:

- *Connection establishment:* This includes the procedures for establishing a connection with a remote device. If security on the connection is required, then security procedures are also invoked. At the end of this procedure, the device which initiated the connection establishment becomes the Master and the other device becomes the Slave. It's also possible to switch the roles by invoking the role switch procedure during connection establishment. In that scenario, the device which initiated the connection becomes the Slave and the other device becomes the Master.
- *Detaching a connection:* This includes the procedures for detaching a connection. It may be started by either the Master or the Slave.
- *Link supervision:* This feature is used to detect loss of a physical link, for example, when the devices move away from range or one of the devices loses battery power. The LMs of both devices use a supervision timer to detect if the link has been lost. If the link is lost, then the LM reports the link loss to the host. One of the sections earlier explained the POLL and NULL packets. These packets are like a heartbeat between the pair of devices in a piconet. These ensure that the link is still active and that the other device has not gone out of range or has been reset. Hence there is a timer (timeout is negotiated right at the link establishment time) which is fired at both sides for detecting an inactivity on these heartbeat signals. Once there is an inactivity detected for the stipulated timeslots, the link managers on both sides assume that there is something wrong on the link, or the devices have moved out of range. Hence they indicate a link supervision timeout to the upper layers.

- *Enable, disable AFH and update of channel map:* Adaptive Frequency Hopping (AFH) was explained in previous sections. This feature helps in improving the performance in case of interference by removing the channels which have interference from the frequency hopping pattern. The LM of the Master enables or disables this feature and provides the updated channel map to each Slave. The LM of the Slave receives these PDUs and updates its channel map accordingly. The LM of the Master may also request the Slave for information about the quality of channels. This is an optional feature and if the Slave supports it then it provides a channel map to the Master indicating whether the channels are good, bad, or unknown.
- *Control of the transmit power level:* If the receiver observes that the characteristics of the received signal differ too much from preferred values, it may ask the transmitter to increase or decrease the power level of the transmitting device. This feature is optional and it allows the devices to use the most optimal power levels for communication. As an example, as soon as the devices move away from each other, the receiver may observe degradation of the received signal. So it may request the transmitter to increase the transmit power level. If the devices move closer, the receiver may again request the transmitter to reduce the transmit power level. This will allow more efficient use of battery power.
- *Quality of Service:* This feature is used for bandwidth allocation on ACL logical transport so that the requested amount of bandwidth can be reserved on the ACL logical transport.

An example of air logs of different LMP transactions is shown in Figure 3.11. Some of the points worth observing are:

1. There are two different columns: *Role* and *Initiated By*
 - a. Role indicates which device sent the current packet.
 - b. Initiated By indicates which device originally initiated this transaction. For example if the transaction was initiated by the Master and the Slave is now responding, then Role will indicate Slave and Initiated By will indicate Master.
2. Frame #1: Connection request initiated by the Master.
3. Frame #9: Connection completed.
4. Frame #2 to #8: Master and Slave exchange information about supported features and version.
5. Frame #14: SET_AFH command sent by Master to Slave to enable Adaptive Frequency Hopping.
6. Frame #292: detach command sent by Master to disconnect the connection.

3.4.2 Security

The security procedures include the following:

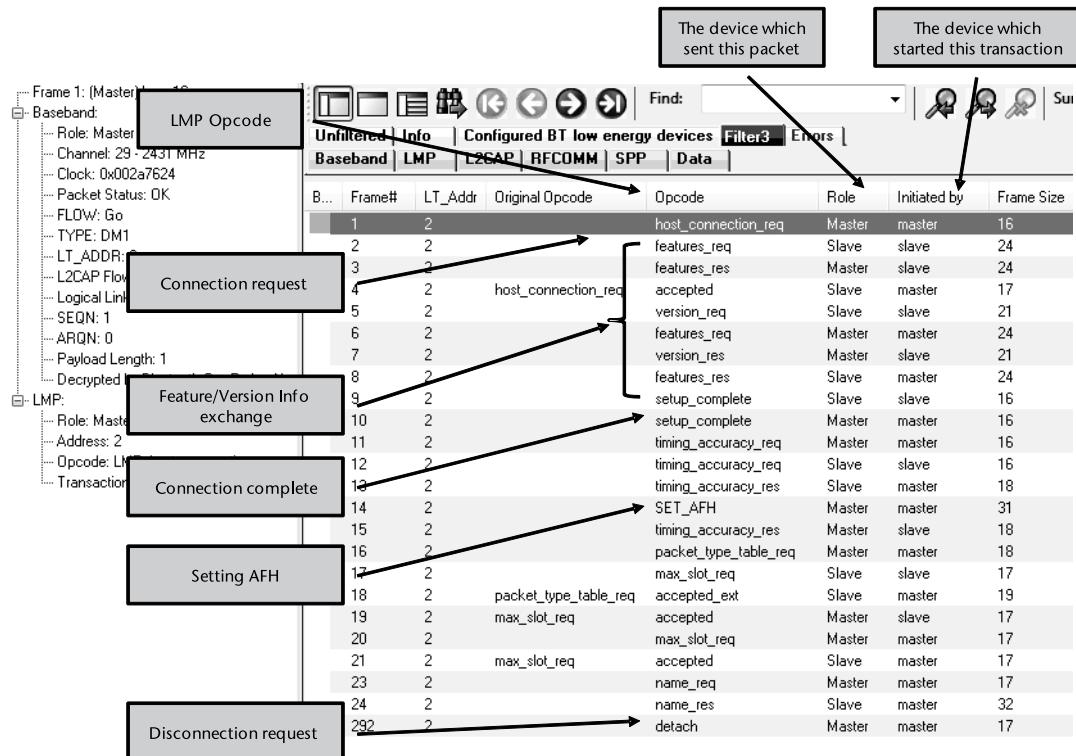


Figure 3.11 Example of LMP transactions.

- Pairing;
- Authentication;
- Encryption;
- Secure Simple Pairing.

3.4.2.1 Pairing

Pairing is the process of associating two devices with each other. When the two devices agree to communicate with each other, they exchange a passkey. This passkey can be considered to be similar to a password that is shared between the two devices. The passkey is also referred to as the Bluetooth PIN and is generally entered on the UI. For example when connecting a mobile phone to a laptop, the user will have to enter identical PIN codes on both the laptop and the mobile phone.

Pairing can be done by using older legacy pairing procedures or by *secure simple pairing* procedures. (Secure simple pairing was introduced in version 2.1 + EDR of the Bluetooth specification).

The Bluetooth PIN that is entered on the UI is used along with a random number and BD_ADDR to create a link key. This link key is used for authentication between the two devices for all subsequent connections. This key is stored in the devices so that when the next time the devices are connected, the user does not need to enter the PIN again.

3.4.2.2 Authentication

Authentication is the process of verifying *who* is at the other end of the link. The authentication process starts when the two devices initiate a connection establishment. It is based on a challenge-response scheme. The verifier sends a challenge to the other device that contains a random number (the challenge). The other device calculates a response that is the function of the challenge, its own BD_ADDR and a secret key. The response is sent back to the verifier that checks whether it is correct or not.

The success of the authentication procedure requires that the two devices share a secret key. This key was generated during the pairing process. If they do not share a secret key, then the pairing procedure is initiated.

3.4.2.3 Encryption

This is an optional procedure and the Master and Slave must agree whether to use encryption or not. To use encryption, it's mandatory to perform authentication.

If encryption is enabled, then all data exchanged on the link is encrypted using an encryption key. The encryption key can be from 8-bits to 128-bits.

3.4.2.4 Secure Simple Pairing

Secure simple pairing was introduced in version 2.1 + EDR of the Bluetooth specification to both simplify the pairing mechanism and improve security. This is explained in further detail later in this chapter.

3.5 Host Controller Interface (HCI)

One of the strengths of Bluetooth specification is that it provides a uniform method for accessing the controller's capabilities. This has several advantages. First the development of the software for controller and host can go on independently. Secondly a host from one vendor can work easily with a controller from another vendor. For example a Bluetooth dongle from any vendor can be plugged into a PC to use Bluetooth functionality.

The Host Controller Interface (HCI) provides this uniform method for accessing the controller's capabilities. The host interfaces with the controller using this layer. It is optional and is required only in implementations where the software for controller and host run on different processors. It may be skipped in implementations where the software for controller and host run on the same processor.

The communication over the HCI interface happens in form of packets. The host sends HCI command packets to the controller and is asynchronously notified by the controller using HCI events.

The commands and events for LE controllers are also exchanged over the HCI interface. The LE controllers use a reduced set of HCI commands. Some commands and events are re-used for multiple controller types.

There are four possible types of controllers:

1. *BR/EDR Controller*: Supports only BR/EDR functionality.
2. *BR/EDR/LE Controller*: Supports both BR/EDR and LE functionality.
3. *LE Controller*: Supports only LE functionality.
4. *AMP Controller*: Supports functionality of AMP Protocol Adaptation Layer (PAL) (BT 3.0 + HS).

3.5.1 HCI Packet Types

There are four types of packets that can be sent on the HCI Interface.

1. HCI Command Packet.
2. HCI Asynchronous (ACL) Data Packet.
3. HCI Synchronous (SCO/eSCO) Data Packet.
4. HCI Event Packet.

These are shown in Figure 3.12.

3.5.1.1 HCI Command Packet

The HCI Command Packet is used by the host to send commands to the controller. Each command is assigned a 2 byte unique OpCode. The OpCode is further divided into two fields:

1. *OpCode Group Field (OGF)* (upper 6 bits): This field is used to group similar OpCodes together.
2. *OpCode Command Field (OCF)* (lower 10 bits): This field is used to identify a particular command in the OpCode group.

The HCI command packets are denoted as HCI_xxx where *xxx* is the command that is given by the host to the controller. For example the command to reset the controller is denoted as HCI_Reset.

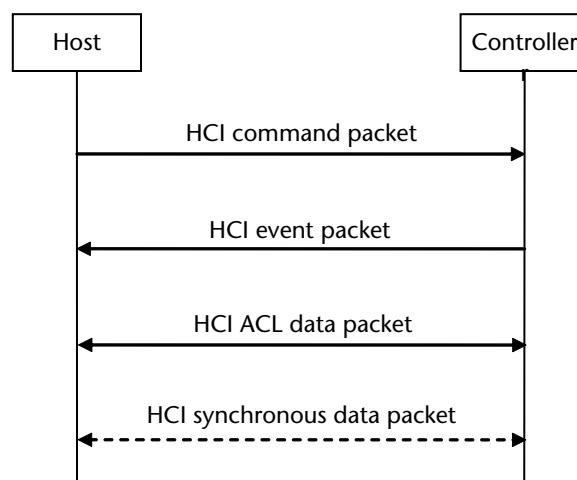


Figure 3.12 HCI Packet Types.

The format of HCI command packet is shown in Figure 3.13. It consists of a 16-bit OpCode followed by an 8-bit *parameter total length* field. The parameter total length field specifies the total length of all parameters that are contained in the remaining packet measured in octets. (An octet denotes an 8-bit value). This is followed by the command parameters.

3.5.1.2 HCI Event Packet

The HCI Event Packet is used by the controller to notify the host when an event occurs. This may be in response to an HCI command that was sent earlier (For example a command to create a connection), or due to any other event (For example loss of connection or incoming connection request). Errors are also indicated using HCI Event Packets.

The format of HCI event packet is shown in Figure 3.14. It consists of an 8-bit Event Code followed by an 8-bit parameter total length field. The parameter total length field specifies the total length of all parameters that are contained in the remaining packet (measured in octets). This is followed by the event parameters.

3.5.1.3 HCI ACL Data Packet

The HCI ACL Data Packet is used to exchange data between the host and the controller. ACL Data packets can only be exchanged after a connection has been established.

The format of HCI ACL data packet is shows in Figure 3.15. It consists of a 12-bit Handle followed by 2-bit Packet Boundary (PB) and 2-bit Broadcast (BC) flags. This is followed by 16-bit Data Total Length field which specifies the length of the following data in octets. This is followed by the data.

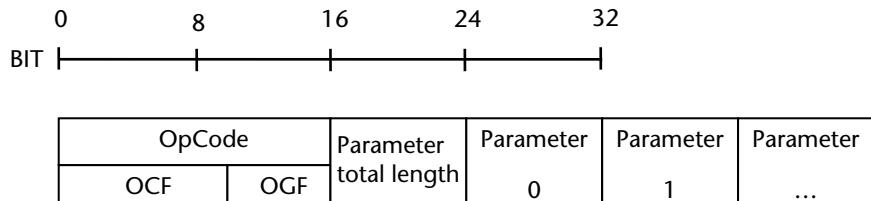


Figure 3.13 HCI Command Packet Format.

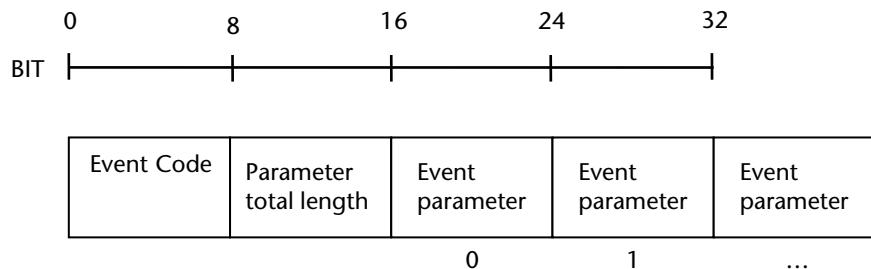


Figure 3.14 HCI event packet format.

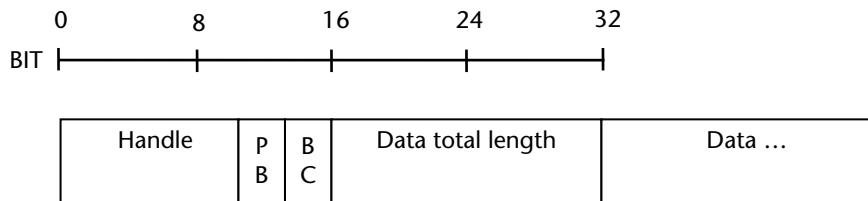


Figure 3.15 HCI ACL data packet format.

The Handle specifies the Connection Handle on which data is to be sent. Each connection is specified by a unique connection handle.

The *packet boundary flag* is useful when a higher layer protocol packet is fragmented into multiple ACL data packets. It indicates whether this packet is the first packet in the fragment or a continuing packet. This field also indicates the flushable characteristics of the packet—whether it is automatically flushable or not. Automatically flushable packets are automatically flushed based on a timer if they cannot be transmitted by the time the timer expires.

The *broadcast flag* indicates whether this is a point-to-point packet or a broadcast packet.

3.5.1.4 HCI Synchronous Data Packet

The HCI Synchronous Data Packet is used to exchange synchronous data between the host and the controller.

The format of HCI synchronous data packet is shown in Figure 3.16. It consists of a 12-bit Connection Handle followed by 2-bit Packet Status (PS) flag and 2-bit reserved (RES) field. This is followed by 16-bit Data Total Length field which specifies the length of the following data in octets. This is followed by the data.

The Connection Handle specifies the Connection Handle on which data is to be sent. Each SCO or eSCO connection is specified by a unique connection handle.

The Packet Status flag gives an indication whether the packet was correctly received or not.

3.5.2 HCI Commands and Events

The HCI commands are grouped into several categories. Some examples of the HCI commands in various groups are provided below.

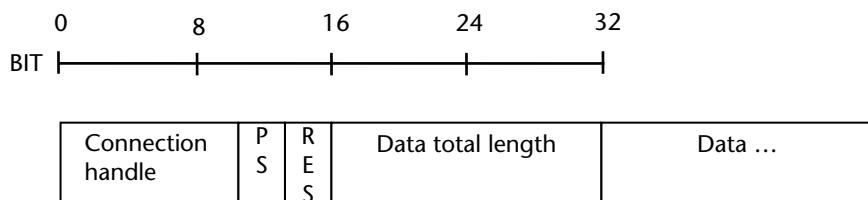


Figure 3.16 HCI synchronous data packet format.

- *Link Control Commands:* These allow a controller to control connection to other controllers.
 - HCI_Inquiry—Discover devices in the vicinity.
 - HCI_Inquiry_Cancel—Cancel Inquiry.
 - HCI_Create_Connection—Create a connection to remote device.
 - HCI_Disconnect—Terminate an existing connection.
 - HCI_Remote_Name_Request—Obtain user friendly name of another controller.
- *Link Policy Commands:* These are used to control how Bluetooth piconets and scatternets are established and maintained.
 - HCI_Hold_Mode—Put the connection in hold mode.
 - HCI_Sniff_Mode—Put the connection in sniff mode.
 - HCI_Switch_Role—Switch the role from Master to Slave and vice versa.
- *Controller and Baseband Commands:* These provide access to various capabilities of Bluetooth hardware.
 - HCI_Set_Event_Mask—Control which events can be generated by the controller to be sent to the host.
 - HCI_Reset—Reset the controller.
 - HCI_Write_Local_Name—Modify the user friendly name of the local device.
 - HCI_Write_Class_of_Device—Write the Class of Device parameter.
- *Informational Parameters:* These provide information about the capabilities of the controller.
 - HCI_Read_Local_Version_Information—Version, Manufacturer name.
 - HCI_Read_BD_ADDR—Reads the BD_ADDR of the controller.
- *Status Parameters:* These provide information about the current state of the controller.
 - HCI_Read_Link_Quality_Information about the link quality.
- *LE Controller Commands:* These are used for LE controllers.
 - HCI_LE_Set_Event_Mask—Controls which LE Events are generated by the controller to be sent to host.
 - HCI_LE_Read_Buffer_Size—Read the maximum size of data packets that can be sent to controller and the total number of buffers.
 - HCI_LE_Create_Connection—Create an LE connection.

Some examples of HCI events are provided below.

- *Inquiry Complete:* Is used to indicate completion of inquiry.
- *Inquiry Result:* Is used to provide information about remote devices found during inquiry.
- *Connection Complete:* Indicates that a connection has been established.
- *Disconnection Complete:* Indicates that a connection has been terminated.

- *Hardware Error:* Indicates some hardware failure.
- *LE Meta Event:* Is used to encapsulate all LE events. A sub-event code indicates the exact event that has occurred. It could be one of the following:
 - LE Connection Complete—Indicates that an LE connection has been established.
 - LE Advertising Report—Indicates that an advertising report has been received.
 - LE Connection Update Complete Event—Indicates that the process to update the connection parameters has been completed.

The following two HCI events are used quite frequently:

- *Command Complete:* This event is used by the controller for most commands to transmit the return status of the command as well as the return parameters associated with the command.
- *Command Status:* This event is used by the controller to indicate that it has received the command and has started performing the task for the command though the task may not have completed yet. This event is needed to provide a mechanism of asynchronous operation so that the host can continue doing other operations while the controller performs the tasks requested by the host. The host is notified by another event when the task is completed by the controller.

An example of the HCI commands and events exchanged during inquiry is shown in Figure 3.17.

3.5.3 Buffers

The controller has buffers to receive the commands and ACL/Synchronous data sent by the host.

- *Command Buffers:* The BR/EDR/LE controller uses shared buffers for the commands.
- *Data Buffers:* The controller may either have shared or separate data buffers for BR/EDR and LE.

Whether the controller has separate or shared data buffers for LE can be determined using the HCI_Read_Buffer_Size command. This will be explained in a later section.

3.5.4 HCI Flow Control

In general the controller has much less buffers and processing power as compared to the host. This means that if the host sends a few packets to the controller to process in fast succession, then it is possible that the buffers of the controller get full.

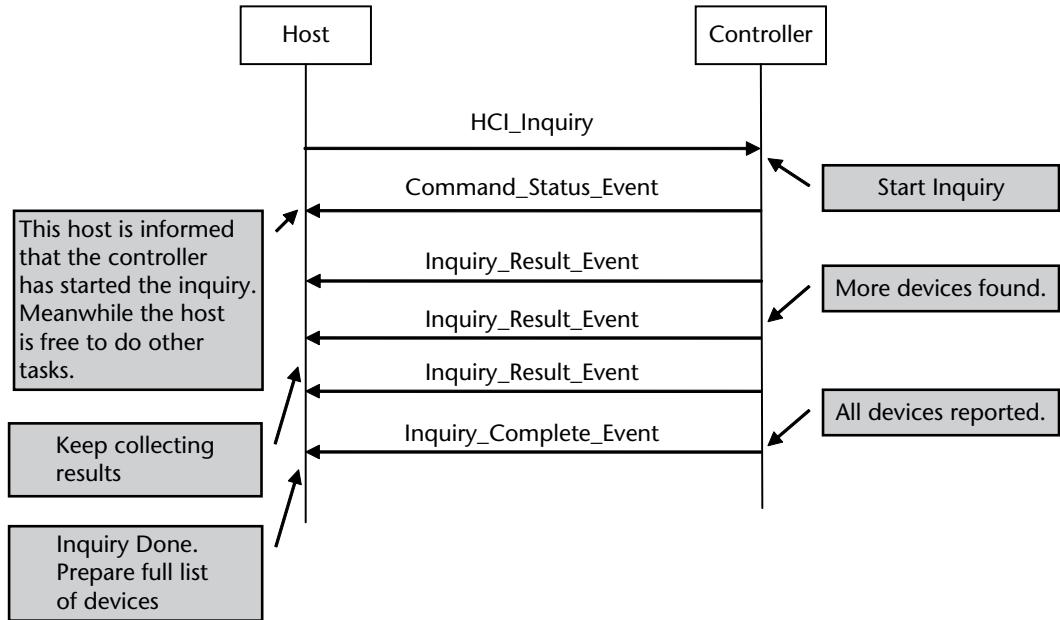


Figure 3.17 HCI commands and events for inquiry.

Flow control mechanism is needed in such a scenario to ensure that the host does not send any packets further till the time the previous ones are processed.

The HCI interface provides separate flow control mechanisms for the following:

1. *Host to Controller Data Flow Control*: ACL Data Packets from host to controller.
2. *Controller to Host Data Flow Control*: ACL Data Packets from controller to host.
3. *Command Flow Control*: HCI Command Packets from host to controller.

Out of these, Controller to Host Data Flow Control is not much widely used since typically the host has more resources than the controller. So most of the time, the controller can send data to the host without any flow control.

3.5.4.1 Host to Controller Data Flow Control

The Host to Controller Data flow control is used to regulate the flow of data from the host to the controller. This is needed because the controller may have much lesser buffers as compared to the host. Besides this the controller also needs time to process (transmit/re-transmit) those packets. During that time, the flow control ensures that the host sends only as many packets as the number of available buffers in the controller.

There are two mechanisms:

1. Packet Based Flow Control.
2. Data-Block-Based Flow Control.

The Packet Based Flow Control is used quite widely and this is explained in detail in this section. It is shown in Figure 3.18.

During initialization the host sends the Read Buffer Size command to get information about the buffers in the controller. The response to the Read Buffer Size contains the following:

1. Number of ACL buffers.
2. Size of each ACL buffer.
3. Number of SCO buffers.
4. Size of each SCO buffer.

If the controller supports LE, then it is possible for the controller to either have separate LE buffers or share the same buffers. The host can send the LE Read Buffer Size command to find this out. If the response to this indicates zero as the length of the buffers, then it means that the same buffers are shared between BR/EDR and LE. Otherwise the response would indicate the number of LE buffers present in the controller and the size of those buffers.

Once the host has information on the number of ACL buffers, it knows that at any given time, it can have a maximum of that many packets outstanding (to be processed) on the controller side. For example if the Number of ACL buffers was four, then the host can have a maximum of four packets outstanding on the controller side at any given time.

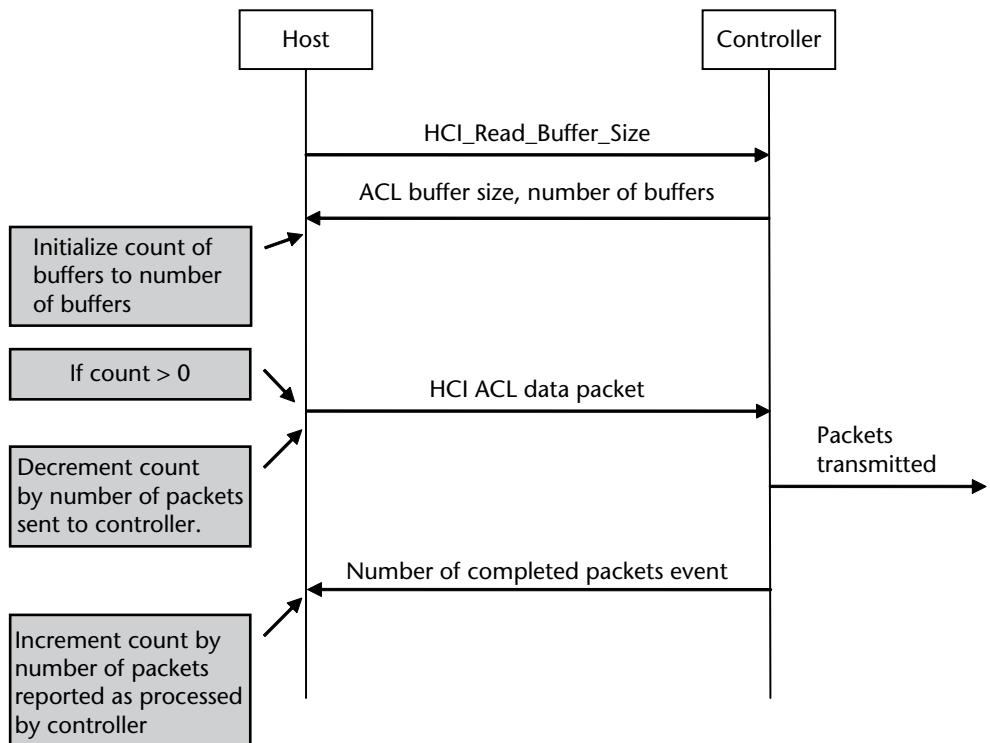


Figure 3.18 Packet based Host to Controller Data Flow Control.

The host maintains a count of the number of ACL buffers available in the controller. It initializes this value to the number of ACL buffers it received in the response to Read Buffer Size command. After that every time it sends a packet to the controller, it decreases this count by one.

Once the controller has completed processing one or more packets, it sends that count of processed packets in the Number of Completed Packets event. The host knows that some additional buffers have been freed up on the controller side and it increments its count by the number of packets reported in the Number of Completed Packets event.

3.5.4.2 Command Flow Control

The flow control used for commands is very simple. The command complete and command status events that are sent by the controller to the host have a field which indicates how many more commands can be sent to the controller. The host can send up to that many commands to the controller before waiting for the next command status or command complete event.

In general, if the controller is not able to complete a command right away, it sends a command status event. Later on when the command is completed it either sends a command complete event or another relevant event depending on the command that was sent. For example if the host requests the controller to start an inquiry, the controller starts the inquiry and returns a command status event. This indicates to the host that the controller has started processing of the inquiry command. Once the inquiry is complete, the controller sends an inquiry complete event to indicate that the inquiry command has been completed. This was shown in Figure 3.17.

3.5.5 Connection Handle

The HCI interface assigns a Connection Handle when a new logical link is created. The value of this connection handle is returned in the Connection_Complete_Event. The host uses this connection handle to manage this connection, send data and finally disconnect the connection.

3.5.6 HCI Transport Layer

The Host Controller Interface defines 4 transport layers that can be used:

1. UART Transport Layer.
2. USB Transport Layer.
3. Secure Digital (SD) Transport Layer.
4. Three-Wire UART Transport Layer.

The UART Transport Layer is described in detail here. This interface is commonly used in systems where the Host and Controller are on the same PCB (Printed Circuit Board). Some examples are Smartphones, Tablets and Printers.

Each packet that is exchanged between the Host and the Controller is prefixed with a packet indicator immediately before the HCI Packet. This allows the differentiation of various packets.

The following Packet Indicators are used on the HCI UART Interface:

1. HCI Command Packet: Packet Indicator 0x01.
2. HCI Asynchronous (ACL) Data Packet: Packet Indicator 0x02.
3. HCI Synchronous (SCO/eSCO) Data Packet: Packet Indicator 0x03.
4. HCI Event Packet: Packet Indicator 0x04.

Figure 3.19 indicates the direction in which these packets are sent along with the packet indicators. (The arrows indicate the direction in which packets are sent.)

In some implementations (e.g., some Smartphones), the HCI Synchronous Data packets are not sent on the UART Transport Layer. Rather these packets are directly sent by the host's application processor or modem to the Bluetooth chip on a different interface. For example a PCM/I2S interface may be used to send voice packets between the Application Processor and Bluetooth Chip. This is shown in Figure 3.20.

3.5.6.1 Decoding HCI Packets

Figure 3.21 shows an example of how to decode the packets on the UART transport layer. This is quite useful when debugging the HCI transactions happening on the UART. Similar method can be used to also debug packets on other transport layers.

In general the UART Transport Layer assumes that the line is free from line errors. (Why? - Since they are closely located on the same PCB so it's assumed that there will not be any errors). In case there is a synchronization loss in the communication from the Host to the Controller, the Controller sends a Hardware Error Event to the Host. Once the Host is aware of the synchronization loss, it has to terminate any pending Bluetooth activity and then reset the Controller to regain

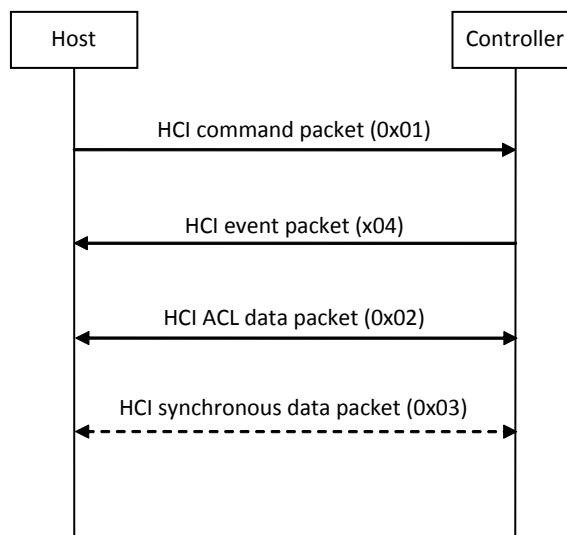


Figure 3.19 HCI UART Transport Layer.

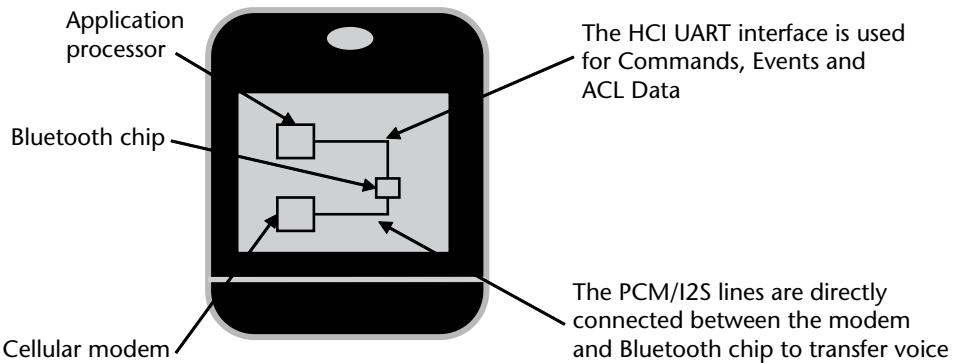


Figure 3.20 Typical HCI UART connections in a smartphone.

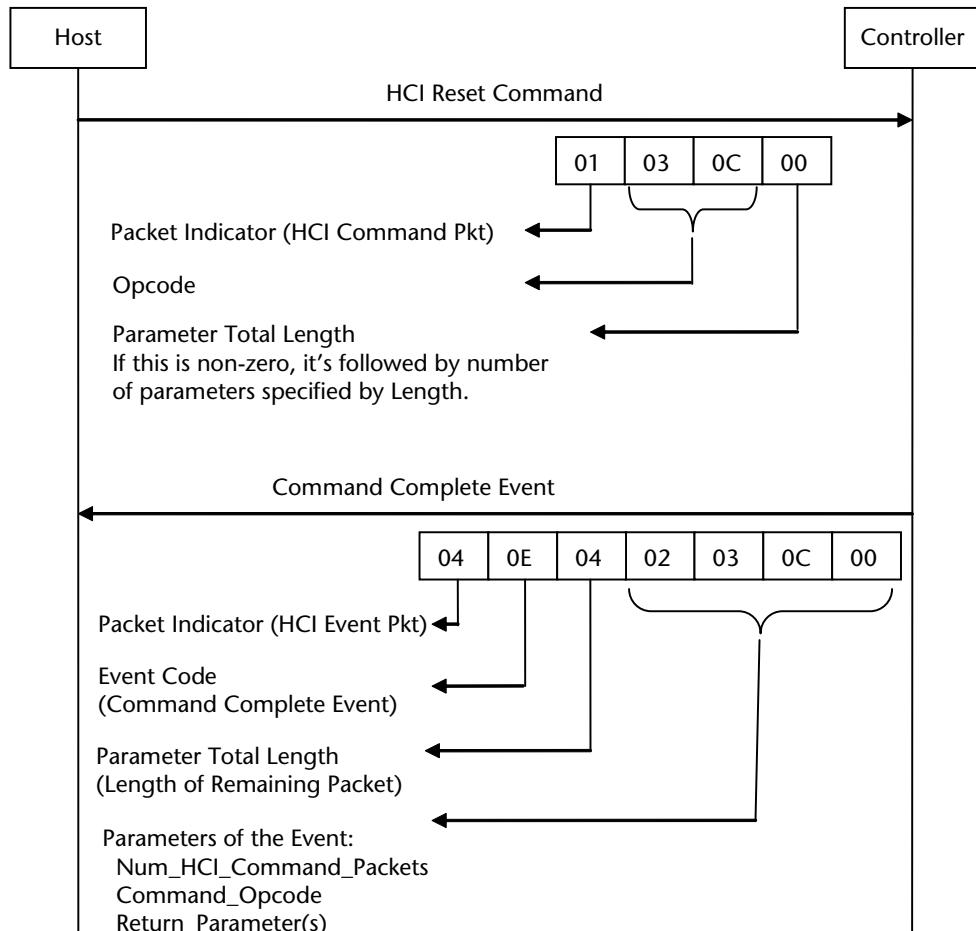


Figure 3.21 How to decode packets sent on HCI UART Transport Layer.

communications. This is done by the host by sending the HCI_Reset command to the controller.

In order to avoid synchronization losses, the Host and Controller use Hardware Flow Control. For details on this as well as the details on the settings of the UART you may refer to the Bluetooth Core specification (Ref [1]).

3.6 Security—Secure Simple Pairing (SSP)

Secure Simple Pairing was introduced in Bluetooth spec 2.1 + EDR. The main goal was to simplify the pairing procedure from the user perspective. It also introduced improved level of security as compared to previous versions.

SSP increased the security level as compared to previous version of the Bluetooth specification. Till Bluetooth spec 2.0 + EDR, only 4-digit numeric pin keys were supported. On top of that several devices used frequently used pin keys like 0000, 8888, or 1234. So hacking those codes was comparatively easier. SSP introduced 16 alphanumeric pin which makes it more difficult to hack the codes.

SSP has two security goals:

- Passive Eavesdropping Protection.
- Man-In-The-Middle (MITM) Attack Protection.

3.6.1 Passive Eavesdropping Protection

Passive Eavesdropping protection is provided by two mechanisms:

- *Strong Link Key:* The strength of a link key is dependent on the amount of randomness (entropy) in its generation. In previous versions of Bluetooth, the only source of this entropy was the 4-digit numeric PIN key. This was relatively easy to break in short time. In comparison SSP uses 16-bit alphanumeric PIN key which provides for much higher entropy.
- *Strong Encryption Algorithm:* SSP uses Elliptic Curve Diffie Hellman (ECDH) public key cryptography. This provides a high degree of strength against passive eavesdropping attacks. It's also computationally less complex than standard Diffie Hellman (DH76) cryptography and suitable for Bluetooth controllers which may have limited computational power.

3.6.2 Man-in-the-Middle (MITM) Attack Protection

An MITM attack occurs when a rogue device attacks by sitting in the middle of two devices that want to connect and relays messages between them. The two devices believe that they are directly talking to each other without knowing that all their messages are being intercepted and relayed by a third device which is setting between them. This is also known as active eavesdropping.

Let's say devices A and B want to make a connection and M is an attacking device (as shown in Figure 3.22).

M receives all information from A and relays it to B and vice versa. So, A and B have an illusion that they are directly connected. They are not aware of the existence of M between them. Since M is relaying the information between the two

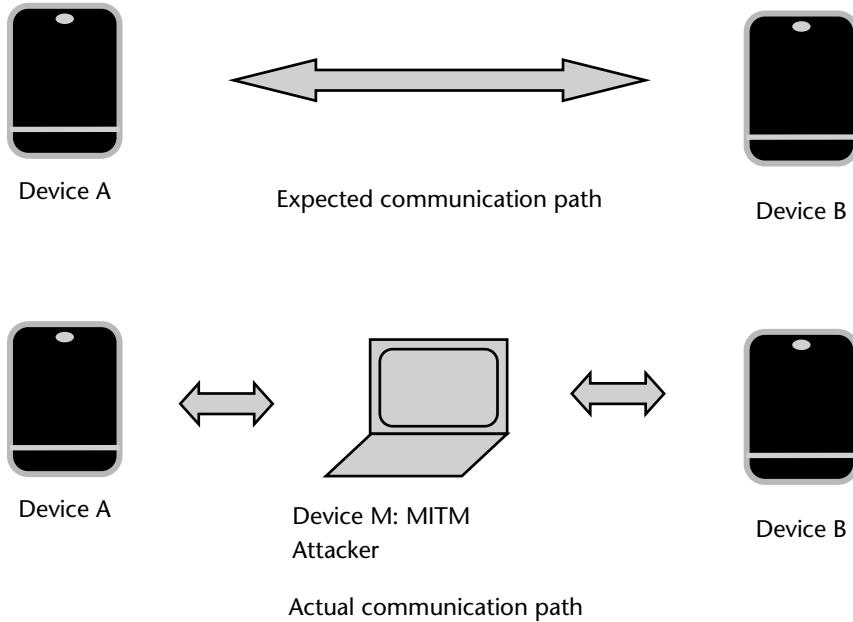


Figure 3.22 Example of MITM attack.

devices, it can interpret all this information and misuse it. Besides this, M can also attack by inducing rogue information between the two devices.

Since A and B can only communicate via M, this type of attack can get detected only if the connection to M is lost. In that case, A and B will not be able to communicate any further and user can detect an MITM attack.

To prevent MITM attacks, SSP provides two mechanisms: Numeric Comparison and Passkey entry. These are described in further detail below.

3.6.3 Association Models

SSP provides four association models based on the I/O capabilities of the two devices. These are as follows:

- Numeric Comparison;
- Just Works;
- Out of Band;
- Passkey Entry.

3.6.3.1 Numeric Comparison

This is used in scenarios where both devices are capable of displaying a six digit number and both are capable of having the user enter a binary “yes” or “no” response. This method displays a 6-bit numeric code on both the devices. The user is then asked whether the number is the same on both the devices. If the user enters yes on both devices, the pairing is successful.

This method has the following advantages:

1. It provides additional confirmation that the correct devices are being paired. This is especially true in cases where the device names are not unique and it's quite hard to remember and identify the device using BD_ADDR
2. It helps to provide protection against MITM attacks.
3. It is also applicable in scenarios where a device may not have a full-fledged keyboard (for entering the PIN) but has only a display. For this method a binary Yes/No input is sufficient.

3.6.3.2 Just Works

As the name implies, this method just works without any user intervention. It's designed for scenarios where the device does not have capability to enter 6 decimal digits nor it has capability to display 6 decimal digits.

It's quite commonly used in pairing mobile phones with mono headsets since mono headsets do not have display and pin entry capabilities. Internally this method still uses the Numeric Comparison though the numbers are not displayed to the user.

It provides protection against passive eavesdropping but does not provide MITM protection. So the security level is still higher than older 2.0 devices but not as good as Numeric Comparison.

3.6.3.3 Out-of-Band (OOB)

Out Of Band Pairing uses an external means for discovering the devices and exchanging pairing information. It's expected that the Out Of Band channel provides protection against MITM attacks to ensure that the security is not compromised.

Typically this could be NFC (Near Field Communication) where the user may touch the two devices. An option would be given to pair the two devices and if the user confirms, the pairing would be successful.

3.6.3.4 Passkey Entry

The passkey mechanism is used when one device has input capability but does not have display capabilities and the second device has display capabilities. One example of such scenario is pairing between keyboard and PC. The user is shown a six digit number on the device which has display capabilities and is then asked to enter this number from the device which has input capabilities. Pairing is successful if the value entered by the user is correct.

3.7 Practical Scenarios

This section describes how the HCI, Baseband Controller and Link Manager interact to implement the following practical scenarios:

- Inquiry;
- Connection Establishment;
- Disconnection.

3.7.1 Inquiry

Inquiry is the procedure to discover other Bluetooth devices in the vicinity. It is also known as scanning or discovering.

Prior to performing an inquiry, it is important that the device that is to be discovered be set in discoverable mode. This includes sending the `HCI_Write_Scan_Enable` command with *Inquiry Scan* parameter set to *Enabled*. Besides this, `HCI_Write_Inquiry_Scan_Activity` can also be used to configure additional parameters. Once this is done the device is said to be in inquiry scan mode (or discoverable mode).

The inquiry procedure is started by the host of the first device (which is supposed to discover other devices) by sending an `HCI_Inquiry` command to the controller. On receipt of this command, the controller initiates an inquiry. The devices in the vicinity respond with inquiry responses.

The controller collects the inquiry responses and provides the results of inquiry to the host in `Inquiry_Result` events. These events contain the `BD_ADDR`, `Class_Of_Device` and `Clock_Offset`. Multiple devices may be reported in a single `Inquiry_Result` event. Once the inquiry is complete, an `Inquiry_Complete` event is sent by the controller to the host (on the device that initiated the inquiry).

This is illustrated in Figure 3.23. (Some of the events are omitted to aid simplicity).

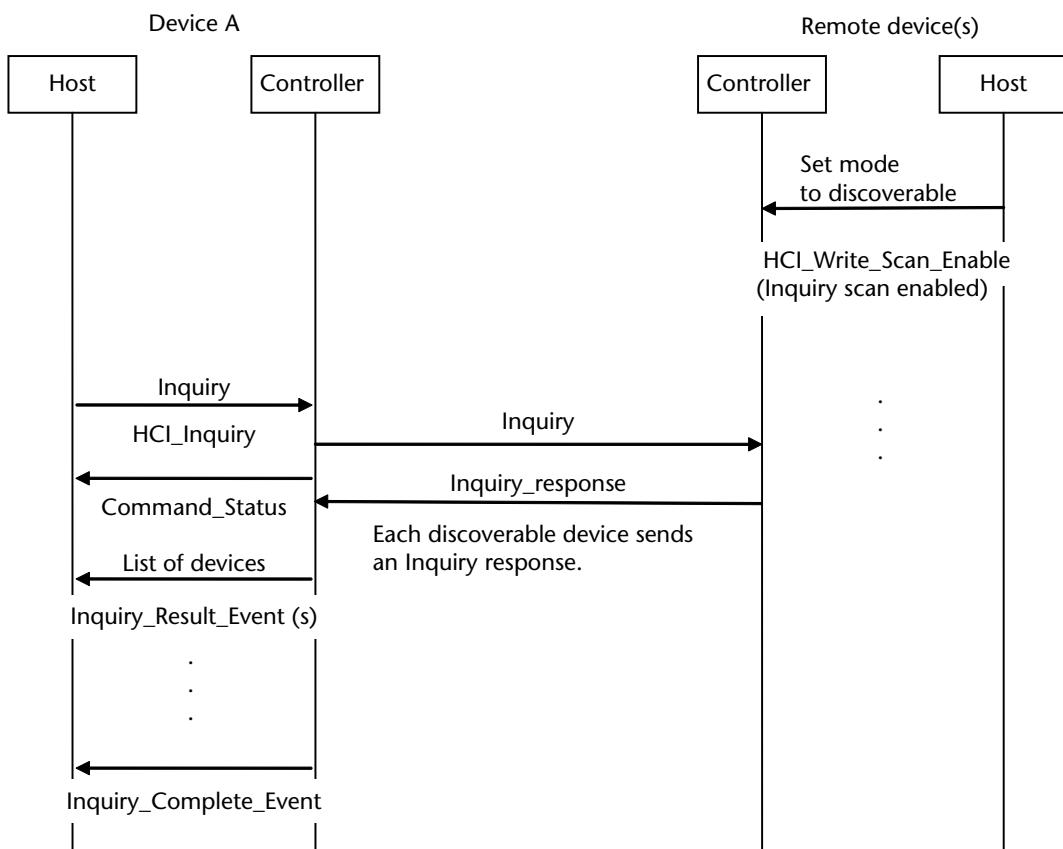


Figure 3.23 Inquiry procedure.

3.7.1.1 Periodic Inquiry

If the inquiry procedure is to be repeated periodically, then HCI_Periodic_Inquiry_Mode command can be used. This command takes the maximum and minimum period between consecutive inquiries as parameters. Once this command is given by the host, the controller performs an automatic inquiry periodically. This can be stopped by the host by sending the HCI_Exit_Periodic_Inquiry_Mode command.

A practical use of this is when a device wants to be regularly alerted of any devices which are coming in or going out of the vicinity. To achieve this periodic inquiry can be used. The time range between two consecutive inquiries is provided as a parameter. The new device list can be compared with the previous device list. If a device was not present in the previous list but is present in the new list then it has entered the Bluetooth vicinity recently. Any specific action like finding details about the device, connecting to it, sending a message or file to it can then be initiated.

3.7.1.2 Extended Inquiry Response (EIR)

The EIR enhancement was added in Bluetooth 2.1 + EDR version of the specification. If a device supports EIR it can provide some additional data while responding to the inquiry. This extended data may contain the name of the device, supported services, Received Signal Strength Indicator (RSSI) etc.

This is more efficient than a normal inquiry response since the inquiring device gets all the information in one go instead of first doing an inquiry, then a get name and finally a service search. This leads to an overall faster connection setup procedure.

3.7.2 Connection Establishment

The procedure to connect to a remote device is known as paging or connecting procedure. In the text below, the device that initiates the connection is referred to as the initiator and the device to which it connects is referred to as the target.

Prior to connecting, the target device (that is to be connected to) is set to Connectable mode. This is done by sending the HCI_Write_Scan_Enable command. This command takes Page_Scan as one of the parameters. The Page_Scan parameter is set to Enabled to make the device connectable. Besides this, HCI_Write_Page_Scan_Activity can be used to configure additional parameters. Once this is done the device is in page scan mode (or Connectable Mode).

Connection procedure is started by the host of the initiator by sending an HCI_Create_Connection command to the controller. The parameters of this command include BD_ADDR, Packet Types, Page_Scan_Repetition_Mode, Clock_Offset and Allow_Role_Switch.

On receipt of this command the controller initiates a connection request using the link manager command LMP_host_connection_req.

Figure 3.24 depicts a very simplified view of the connection establishment procedure. The optional parts like Feature Exchange, Authentication, Encryption and some of the HCI events are omitted to aid simplicity.

The initiator device becomes the Master after successful connection establishment.

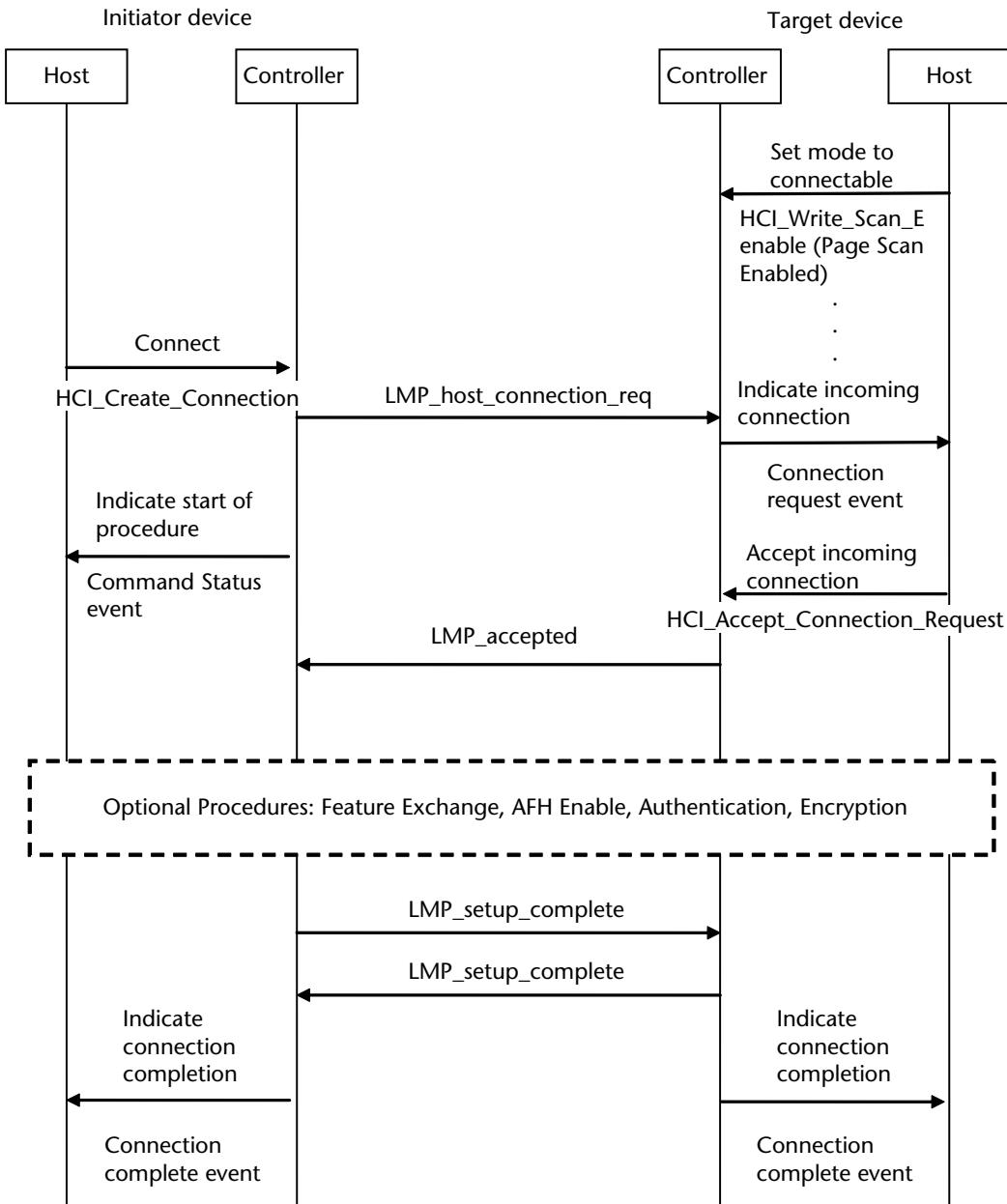


Figure 3.24 Simplified connection establishment procedure.

Figure 3.25 illustrates the disconnection procedure. Either of the devices may decide that the connection is no longer needed and initiate this procedure.

3.8 Summary

This chapter explained the lower layers of the Bluetooth Protocol stack. These include the stack layers up to the HCI interface: Bluetooth Radio, Baseband Controller, and Link Manager. These layers are typically implemented in a controller.

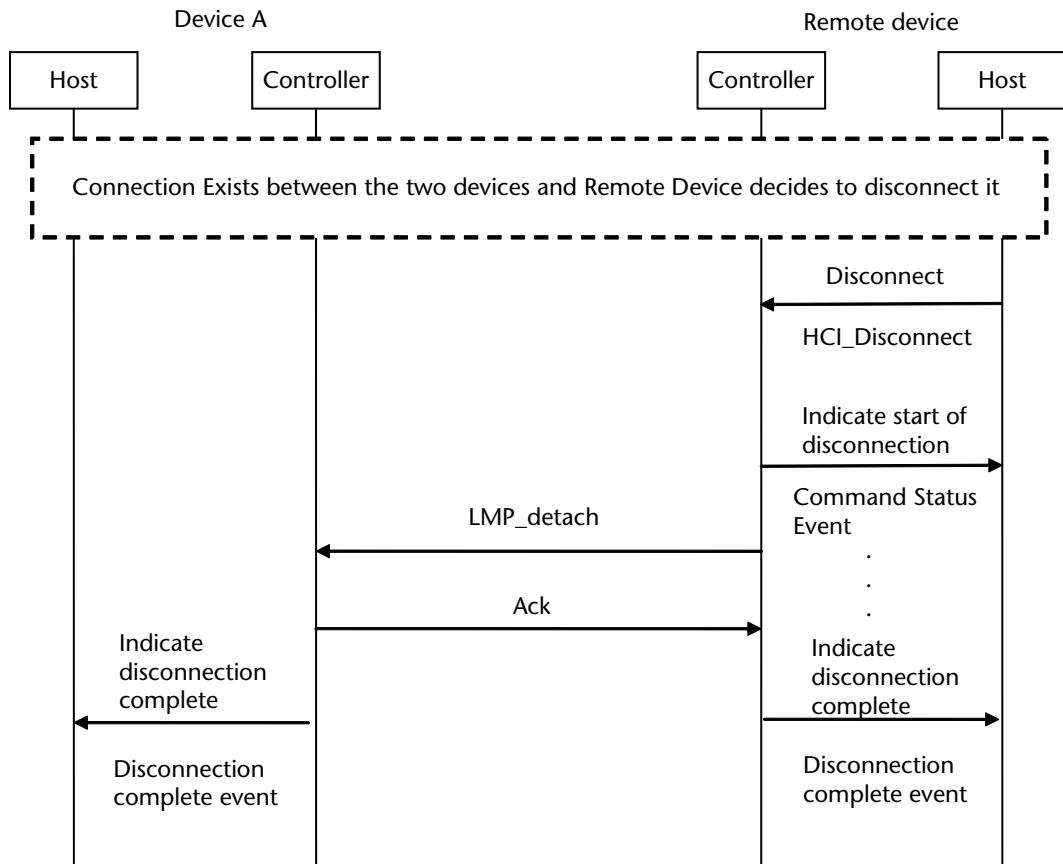


Figure 3.25 Disconnection procedure.

The Bluetooth Radio is responsible for transmitting and receiving the packets to and from the air interface. It uses frequency hopping across 79 channels in the ISM band to combat interference. The Baseband Controller is responsible for carrying out procedures like inquiry, connection, formation of piconet and scatternet, connection states, and low power modes. The Link Manager provides the functionality of link setup and control, security, Master-Slave role switch, etc.

The HCI interface provides a standard mechanism for interfacing Bluetooth upper layers with the controller.

The next chapter will be the last in the series of chapters explaining the Bluetooth architecture. It will cover the Bluetooth upper layers.

Reference

- [1] Bluetooth Core Specification 4.0 <http://www.bluetooth.org>.

Bluetooth Upper Layers and Profiles

4.1 Introduction

The previous chapter described the lower layers of the Bluetooth protocol stack. This chapter continues explanation of the Bluetooth protocol stack and covers the upper layers of the Bluetooth protocol stack and the profiles.

The detailed Bluetooth architecture was presented in Chapter 2. It is shown again in Figure 4.1 for ease of reference. The upper layers make use of the functionality provided by lower layers to provide more complex functionality like serial port emulation, transferring big chunks of data, streaming music, and synchronizing information. These help the applications to conveniently implement end user scenarios.

One of the design principles of the Bluetooth protocol stack was the reuse of existing protocols wherever possible instead of rewriting everything from scratch. On one hand this helped to easily and quickly build further on existing and proven technologies, on the other hand it also helped in reusing existing applications that were already implemented and available. Protocols such as Object Exchange (OBEX) and RFCOMM were adopted from other standard bodies and are referred to as *adopted protocols*. These are shown by shaded rectangles in Figure 4.1. OBEX was adopted from the IrOBEX protocol which was defined by the Infrared Data Association. So applications that were designed to run on Infrared transport can generally be reused to run on Bluetooth as well. Similarly RFCOMM was adopted from ETSI standard 07.10. It allows legacy serial port applications to be used on top of Bluetooth without much change.

Some protocols were defined from scratch. These were the protocols that provided the core Bluetooth functionality and are called *core protocols*. These are shown by plain rectangles in Figure 4.1. For example the Service Discovery Protocol (SDP) is one of the core protocols that allows for support of discovering the services of the devices in the vicinity. Since Bluetooth is an ad hoc peer to peer protocol, this layer was essential because there is no prebuilt infrastructure to provide such information and devices can come into vicinity at any time. So this protocol was defined to query the services from the device itself instead of the need of a central server to store information about all devices.

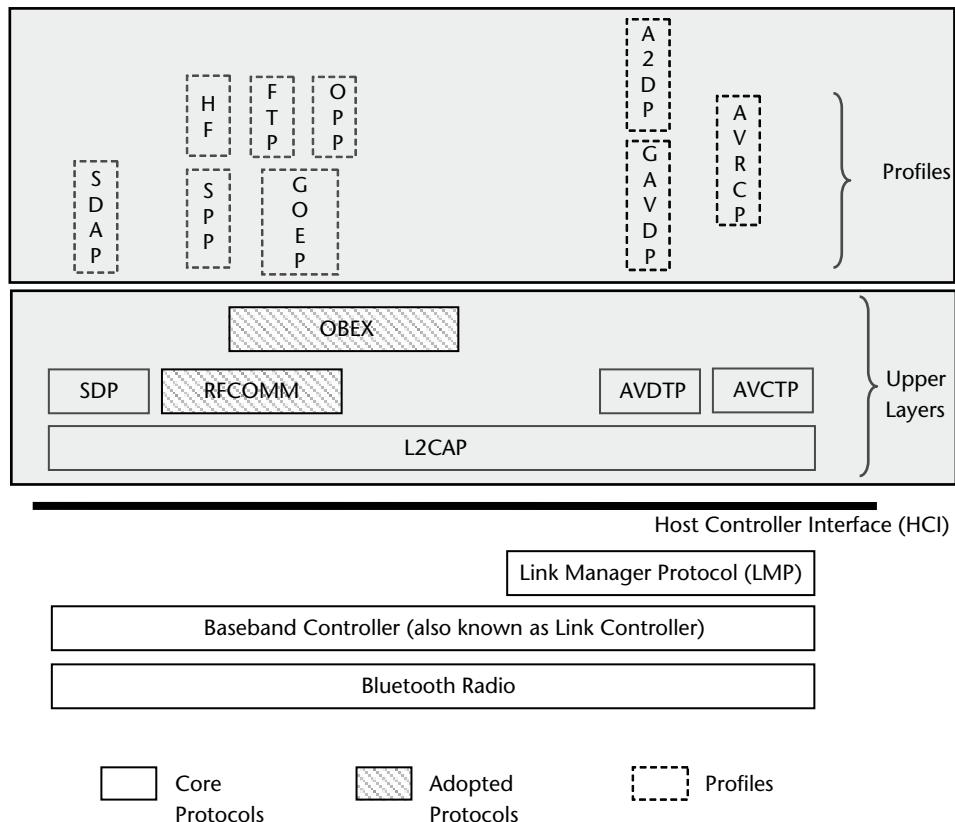


Figure 4.1 Detailed Bluetooth Architecture.

Bluetooth profiles provide a usage model of how the different layers of the protocol stack come together to implement a particular usage model. Profiles define the protocols and the features of each of the protocol that are needed to support a particular usage model. For example the File Transfer Profile (FTP) profile defines how and what features of the underlying protocols like OBEX, RFCOMM and L2CAP are needed in order to provide support for transferring files. Profiles are shown by dotted rectangles in Figure 4.1.

4.2 Logical Link Control and Adaptation Protocol (L2CAP)

The L2CAP protocol sits above the Baseband layer and provides data services to the upper layer protocols. It uses the ACL links to transfer packets and allows the protocols above it to send data packets up to 64 KB in length.

L2CAP is based on the concept of Channels. A channel represents a data flow path between L2CAP entities in remote devices. Channels may be connection-oriented or connectionless.

Connection-oriented L2CAP channels are used to transport point-to-point data between two devices. These channels provide a context within which specific properties may be applied to data transported on these channels. For example QoS

parameters may be applied to the connection-oriented channels. These channels are setup using the L2CAP_CONNECTION_REQUEST command before any data can be transferred. Once data transfer is completed, these channels are disconnected using the L2CAP_DISCONNECT_REQUEST command.

Connectionless L2CAP channels are generally used for broadcasting data though they may also be used for transporting unicast data. The Master uses these channels to broadcast the data to all Slaves in the piconet. These channels do not need separate procedures to setup and disconnect the channel. So latency incurred during the channel setup is removed.

Each endpoint of an L2CAP Channel is referred to as a Channel Identifier (CID). So CID is the local name representing a logical channel end point on the device. CIDs are assigned from 0x0001 to 0xFFFF. Out of these, CIDs from 0x0001 to 0x003F are reserved. These are known as fixed channels. The CID 0x0001 is reserved as the L2CAP signaling channel and 0x0005 is reserved as the L2CAP LE signaling channel. CID assignment is specific to the device and is done independent of the other devices. So it's possible, for example, to have one CID number assigned on one device and a different CID number assigned on the other device by the L2CAP entities executing on the respective devices.

The fixed channels (0x0001 for BR/EDR and additionally 0x0005 for LE) are available as soon as the ACL-U logical link is established with the remote device. The L2CAP Signaling Channel is used for negotiating configuration parameters and setting up the other channels.

The allocation of CID numbers is shown in Table 4.1.

Table 4.1 CID Name Space

CID	Description
0x0000	Null Identifier. Usage is not allowed.
0x0001	L2CAP Signaling channel. Used to send the signaling commands in the form of requests and responses. Some examples of signaling commands are: <ul style="list-style-type: none"> • Connection Request • Connection Response • Configuration Request • Configuration Response • Disconnection Request • Disconnection Response
0x0002	Connectionless channel. Used for the following: <ul style="list-style-type: none"> • Broadcast from Master to all Slaves in the piconet. There is no acknowledgement or retransmission of this data. • Unicast transmission from either a Master or Slave to a single remote device.
0x0003	AMP Manager Protocol. This is used for BT 3.0 + HS operations.
0x0004	Attribute Protocol. Attribute Protocol will be covered in detail in later chapters.
0x0005	LE L2CAP Signaling Channel.
0x0006	Security Manager Protocol (SMP): SMP will be covered in detail in later chapters.
0x0007 – 0x003E	Reserved for future use
0x0040 – 0xFFFF	Dynamically allocated by the L2CAP layer.

4.2.1 Modes of Operation

There are five modes of operation for L2CAP. The mode of operation is selected independently for each channel.

1. *Basic L2CAP Mode*: This is the default mode. The header contains only minimal information including the length of the packet and the channel ID.
2. *Flow Control Mode*: In flow control mode no retransmissions are done. The missing PDUs are detected and reported as lost.
3. *Retransmission Mode*: In retransmission mode a timer is used to ensure that all PDUs are delivered to the peer. The PDUs are retransmitted if they are not ack'd by the remote side.
4. *Enhanced Retransmission Mode*: The enhanced retransmission mode is similar to the retransmission mode and adds some enhancements to it. For example it adds a POLL bit to request a response from the remote L2CAP layer.
5. *Streaming Mode*: The streaming mode can be used for all streaming applications. In this mode, the PDUs from the transmitting side are numbered but are not acknowledged by the receiver. The numbering of PDUs ensures that they are processed in the correct sequence on the receiving side. A flush timeout is used so that if the PDUs are not sent within that timeout, they are flushed.

4.2.2 L2CAP PDUs

The L2CAP Protocol Data Unit (PDU) is the term used to refer to the packets that are sent and received the L2CAP layer. The PDUs contain control information or data.

L2CAP defines 5 types of PDUs:

1. *B-frame (Basic Frame)*: A B-frame is a PDU used in basic L2CAP mode for L2CAP data packets.
2. *I-frames (Information Frame)*: An I-frame is a PDU used in enhanced retransmission mode, streaming mode, retransmission mode and flow control mode. It contains additional information encapsulated in the L2CAP header.
3. *S-frame (Supervisory Frame)*: An S-frame is a PDU used in Enhanced retransmission mode, retransmission mode and flow control mode. It contains protocol information only and no data.
4. *C-frame (Control Frame)*: A C-frame is a PDU that contains L2CAP signaling messages that are exchanged between peer L2CAP entities. This frame is exchanged on the L2CAP signaling channel.
5. *G-frame (Group Frame)*: A G-frame is used on the connectionless L2CAP channel. It may be used to broadcast data to multiple Slaves or unicast data to a single remote device.

4.2.3 L2CAP Features

L2CAP performs the following major functions:

- Higher layer Protocol Multiplexing and Channel Multiplexing;
- Segmentation and Reassembly (SAR);
- Per Channel Flow Control;
- Error Control and Retransmissions;
- Streaming Channels;
- Quality of Service;
- Group Management.

L2CAP uses BR/EDR Controller, LE Controller or a Dual mode controller for transporting data packets. It can also use AMP controllers if they exist.

4.2.3.1 Higher layer Protocol Multiplexing and Channel Multiplexing

L2CAP permits several higher layer protocols to share the same ACL links. Each higher layer protocol is assigned a separate CID to transfer data. For example once an L2CAP channel is established with the remote device it may be shared by SDP, RFCOMM and other protocols. Each of these protocols will use a different CID to talk to the respect peer entity on the remote device. This is shown in Figure 4.2.

The Protocol/Service Multiplexer (PSM) field is used during L2CAP connection establishment to identify the higher level protocol that is making a connection on that particular channel. The PSM values are predefined for many of the protocols by the Bluetooth SIG and can be referred to on the Assigned Numbers page on the Bluetooth SIG website. Some of the PSM values for the protocols are show in Table 4.2.

L2CAP also allows the channel to be operated over different controllers though the channel can be active on only one controller at a time. This is useful in scenarios where the channel is initially established over BR/EDR controller and then it's moved to an AMP controller to achieve higher data throughput.

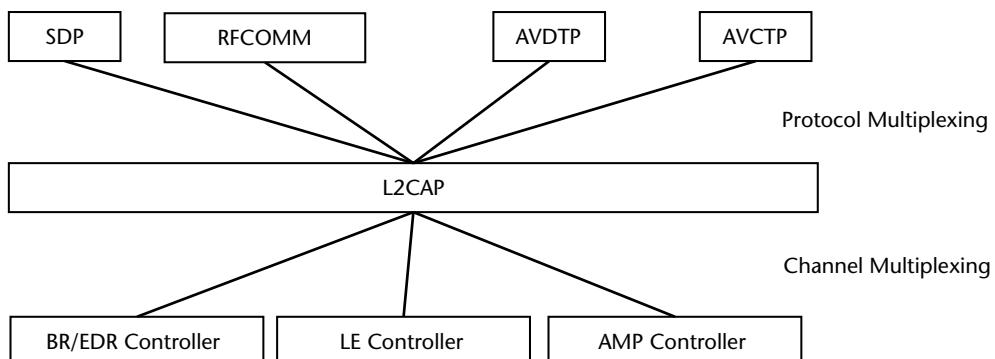


Figure 4.2 L2CAP: Protocol/Channel Multiplexing.

Table 4.2 Assigned Protocol/Service Multiplexer Values (PSM)

Protocol	PSM	Remarks
SDP	0x0001	Service Discovery Protocol
RFCOMM	0x0003	RFCOMM Protocol
TCS-BIN	0x0005	Telephony Control Specification, TCS Binary Protocol
TCS-BIN-CORDLESS	0x0007	Telephony Control Specification, TCS Binary Protocol
BNEP	0x000F	Bluetooth Network Encapsulation Protocol
HID_Control	0x0011	Human Interface Device Profile
HID_Interrupt	0x0013	Human Interface Device Profile
UPnP	0x0015	
AVCTP	0x0017	Audio/Video Control Transport Protocol
AVDTP	0x0019	Audio/Video Distribution Transport Protocol
UDI_C-Plane	0x001D	Unrestricted Digital Information Profile.

4.2.3.2 Segmentation and Reassembly (SAR)

L2CAP allows higher layer protocols to transmit and receive data packets up to 64 kilobytes. While transmitting, L2CAP breaks the packets into smaller packets depending on the packet size supported by the controller. On the receiving end, L2CAP receives all these segments and reassembles them to form the complete packet. This complete packet is then sent to the higher layers.

As an example, let's say RFCOMM sends a packet of 60 KB to L2CAP. L2CAP will break this packet into smaller chunks of 1021 bytes which can then be transferred over a BR/EDR controller (Note that this is the maximum packet size that can be transmitted over ACL using 3-DH5 packets. A BR/EDR controller may support this as the maximum size or a smaller size). This will result in 60 chunks of 1021 byte packets and one chunk of 180 bytes.

$$60 \text{ KB} = 61440 \text{ Bytes} = 60 * 1021 + 1 * 180$$

All these 61 chunks will be reassembled by the L2CAP on the receiving side and the original 60 KB RFCOMM packet will be recreated. This packet will then be given to the RFCOMM of the remote side.

(Note that the above example is a bit simplified. In practice, L2CAP will also prefix its own 4 byte header to the RFCOMM packet. So it will actually be segmenting and reassembling a packet of 61444 bytes. (61440+ 4 byte L2CAP header).)

Segmentation and Reassembly of packets provides the following advantages:

1. The higher layer protocol doesn't need to care about the size of packets that can be transmitted over the ACL link. The L2CAP layers of the peer devices negotiate a mutually suitable MTU size and use it for the data transfer. This makes the design of the higher layer protocol simpler.
2. Since the packet is split into smaller chunks, if there is an error in transmission of one of the chunks then only that chunk will be retransmitted. The whole packet doesn't need to be retransmitted.
3. L2CAP can interleave packets of different higher layer protocols. This ensures that if one of the higher layer protocols is sending a big packet, the other protocols don't get starved for bandwidth.

4. In general, the controllers have limited buffer space for keeping transmit and receive packets while the host may have much larger buffer space. So L2CAP allows the upper layer protocols to send bigger data packets even though the controller may support much smaller packet sizes.

The MTU (Maximum Transmission Unit) is specified by each device independently and is not negotiated between the two L2CAP entities. This means, for example, if a mobile phone is connected to the headset then the mobile phone may specify a higher MTU size while the headset may specify a smaller MTU size. The mobile phone will always send packets which are smaller than or equal to in length to the MTU size of the headset.

The MTU parameter is very significant in cases where speed of data transfer is important. Lower MTU values will result in the need of segmenting a big chunk into more number of smaller packets. This will decrease the overall speed of data transfer.

4.2.3.3 Per Channel Flow Control

Flow control mechanisms are already in place for the data that is transferred on the HCI interface but that is at an aggregate level to control the data that is flowing from the host to the controller and on the air from one controller to another. L2CAP extends this mechanism to provide individual flow control to each of the channels that are multiplexed on top of it. This means that the data for the RFCOMM layer may be stopped while the data for the SDP layer may still be allowed to flow.

4.2.3.4 Error Control and Retransmissions

The first level of error control is done by the baseband layer. If the received packet has an error, then it is not passed on to the host. L2CAP provides an additional level of error control that detects any erroneous packets that are not detected by the baseband. L2CAP retransmits packets if they are not received correctly on the remote side. Besides this, it's possible that some of the packets are dropped by the time they reach the host entity on the remote side. This could be, for example, if the packet had an error and was dropped by the baseband layer. The L2CAP layer retransmits these packets so that the layers above it receive all the packets in the correct sequence without any errors.

L2CAP uses a Transmit/Acknowledge mechanism. Each packet that is transmitted is acknowledged by the remote device. If the acknowledgement is not received, then L2CAP may decide to either retransmit that packet or to drop it. Whether packets on a certain channel are to be retransmitted or not is specified using the Flush Timeout option. There are three possible values:

1. No retransmissions at the baseband level.
2. Use a specified flush timeout and continue retransmitting till the time the timeout expires.
3. An infinite amount of retransmissions. In this case, the baseband continues retransmissions until the physical link is lost.

4.2.3.5 Streaming Channels

Streaming channels are useful in scenarios where data with a specific data rate (like audio) is being transferred. The audio applications can setup an L2CAP channel with the specific data rate. A flush timeout is used if L2CAP is not able to transfer packets within the correct time period to comply with that data rate. This ensures that packets don't get queued up indefinitely if, for example, the link quality deteriorates. Since audio packets are real time in nature, it's better to drop a delayed packet and transmit the next packet than to continue trying to retransmit the delayed packet.

4.2.3.6 Quality of Service

Isochronous data has time constraints associated with it. The information has a time bound relation with the previous and successive entities. Audio is a good example of isochronous data. For such a data, the lifetime is limited after which the data becomes invalid and there is no point in delivering that data anymore.

L2CAP can support both isochronous (Guaranteed bandwidth) and asynchronous (Best Effort) data flows over the same ACL logical link. This is done by marking the isochronous packets as automatically flushable and asynchronous packets as nonflushable in the Packet_Boundary_Flag in HCI ACL data packets. This flag was explained in detail in the HCI section in previous chapter. The automatically flushable packets will be automatically flushed by the controller if they are not transmitted within the time window of the flush timeout set for the ACL link.

4.3.3 L2CAP Signaling

The Signaling commands are used between L2CAP entities on peer devices for operations like setting up the connection, configuring the connection, and disconnection. The fixed channels (0x0001 and additionally 0x0005 for LE) are available as soon as the ACL-U logical link is established with the remote device. The L2CAP Signaling Channel is used for negotiating configuration parameters and setting up the other channels. The L2CAP commands are encapsulated within C-Frames (control frames).

The format of C-Frames is shown in Figure 4.3. The Channel ID is 0x0001 for BR/EDR signaling and 0x0005 for LE signaling. The Code field identifies the type of command. The Identifier field is used to match responses with the requests.

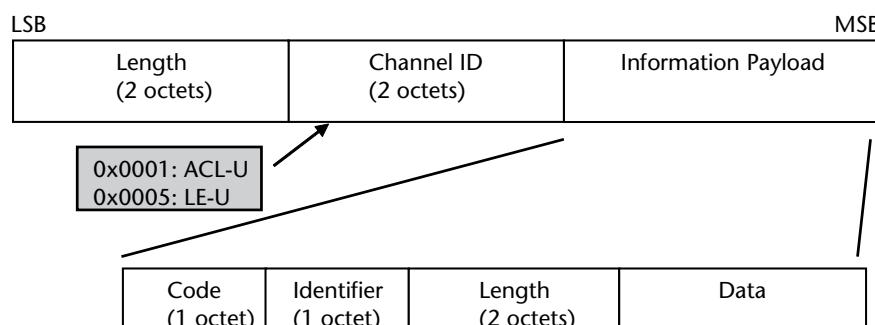


Figure 4.3 Format of L2CAP signaling PDUs (C-Frames).

The various L2CAP signaling packets are shown in Table 4.3.

Out of the commands mentioned in Table 4.3, Connection Parameter Update Request and Response are used only for LE. Command Reject can be used for both LE and BR/EDR. These will be explained in detail in Chapter 10.

An example of various L2CAP Signaling PDUs is shown in Figure 4.4. This figure shows an air sniffer capture of L2CAP signaling packets when RFCOMM uses the services of L2CAP.

Some of the points worth noting are:

1. The Signaling packets are being exchanged on Signaling channel (CID 0x0001). See Frames #22–#29.
2. The RFCOMM data packets are exchanged on CID 0x0040. This is the CID allocated to RFCOMM.
3. During the Connection Request, the master provided the following information:
 - a. Source Channel ID = 0x0040: The CID that the master will be using.

Table 4.3 L2CAP Signaling Packets

Code	Signaling Packet	Purpose
0x00	Reserved	
0x01	Command Reject	This is sent in response to a command packet that contains an unknown command or when sending the corresponding response is inappropriate.
0x02	Connection Request	This is used to create an L2CAP channel between two devices.
0x03	Connection Response	This is sent in response to a Connection Request.
0x04	Configure Request	This is used to negotiate configuration parameters of the connection. These include parameters like MTU, Flush Timeout, QoS, etc.
0x05	Configure Response	This is sent in response to Configure Request.
0x06	Disconnection Request	This is used to terminate an L2CAP channel.
0x07	Disconnection Response	This is sent in response to Disconnection Request.
0x08	Echo Request	This is used to request a response from the remote L2CAP entity. This request is generally used to check the status of the link.
0x09	Echo Response	This is sent in response to Echo Request.
0x0A	Information Request	This is used to request implementation specific information from the remote L2CAP entity.
0x0B	Information Response	This is sent in response to Information Request.
0x0C	Create Channel Request	This is used to create an L2CAP channel between two devices over the specific controller. This is used when BT 3.0 + HS is supported and an alternate controller is used.
0x0D	Create Channel Response	This is sent in response to Create Channel Request.
0x0E	Move Channel Request	These are used to move an existing L2CAP channel from one controller to another. These are used when BT 3.0 + HS is supported.
0x0F	Move Channel Response	
0x10	Move Channel Confirmation	
0x11	Move Channel Confirmation Response	
0x12	Connection Parameter Update Request	This command is sent from an LE Slave to an LE Master to update the connection parameters. This will be described in further detail in Chapter 10.
0x13	Connection Parameter Update Response	This is sent in response to Connection Parameter Update Request.

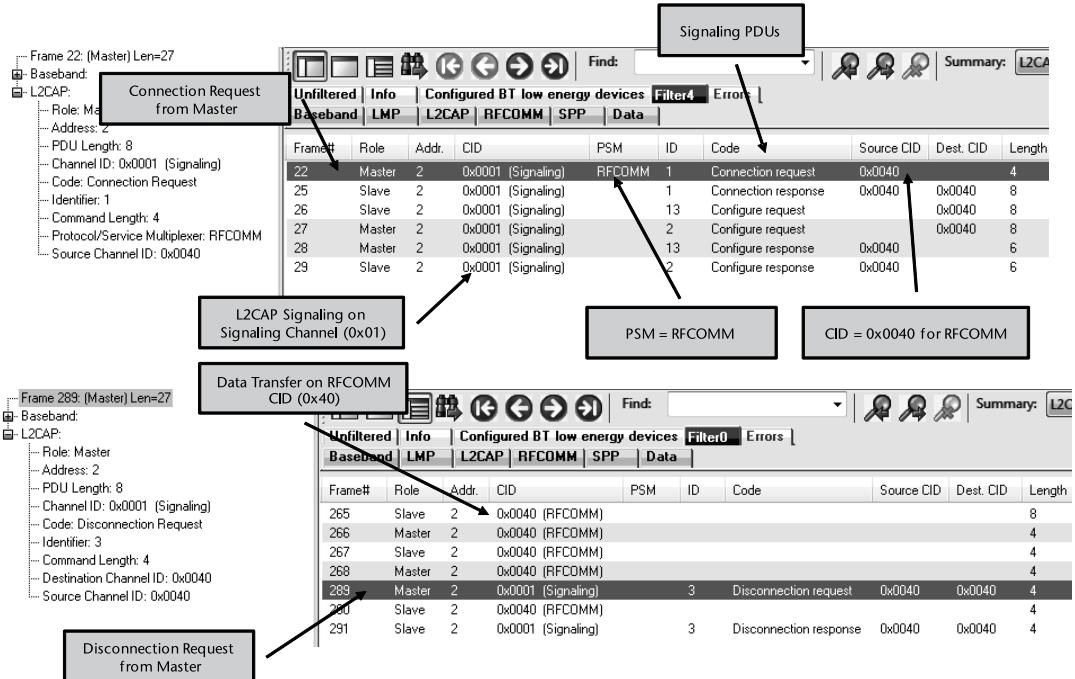


Figure 4.4 Air Sniffer Capture of L2CAP Signal Packets.

- PSM = RFCOMM: To indicate that L2CAP is creating a channel for RFCOMM to use.
- The configure request and configure response packets are exchanged between the Master and Slave in both directions to negotiate the connection parameters.
- Once the data transfer is done, the Master sends a disconnection request on the Signaling channel.

4.4 Service Discovery Protocol (SDP)

Bluetooth provides support for ad hoc connections. This means devices can dynamically discover each other and then decide to connect to each other. Before making the decision to connect to each other, one device may need to “discover” details about the other device. These details include which services are available and what are the characteristics of those services. Service discovery protocol provides a mechanism to discover the services provided by the remote devices and the characteristics of those services.

As an example, let’s say a laptop needs to play an audio file on Bluetooth wireless speakers. It will first do an inquiry to find out the devices in the Bluetooth vicinity. Once this is done, it will connect to these devices and search for the services provided by those devices. In order to play an audio file, it will search for a device that has the services registered for A2DP. Once it finds such a device, it may create an A2DP connection with that device to play the file.

(Note: In practice the laptop need not connect to all the devices to discover the A2DP service. It can already narrow down its search based on the Class of Device (CoD) information provided by that device during inquiry. It needs to only discover services on the devices of the Audio/Video Major class.)

SDP focuses primarily on providing a uniform method for discovering services available on the Bluetooth devices. It does not define the method to access those services once they are discovered. This is done by the other layers depending on the type of service. For example if a device provides a printing service, then the printing related profiles will make use of that service to print documents.

SDP follows a client server model. The services supported by a device are registered with an SDP server. The server maintains a list of these services in the form of service records. The SDP client queries for these services.

This is illustrated in Figure 4.5.

Only one SDP server is permitted per Bluetooth device. If the device does not need to provide any services to other devices, it can act as a client only device. In such cases it does not need to implement an SDP server.

4.4.1 Service Record, Service Attributes and Service Class

All the server applications register their services with the SDP server in the form of Service Records. A service is any entity that can provide information, perform an

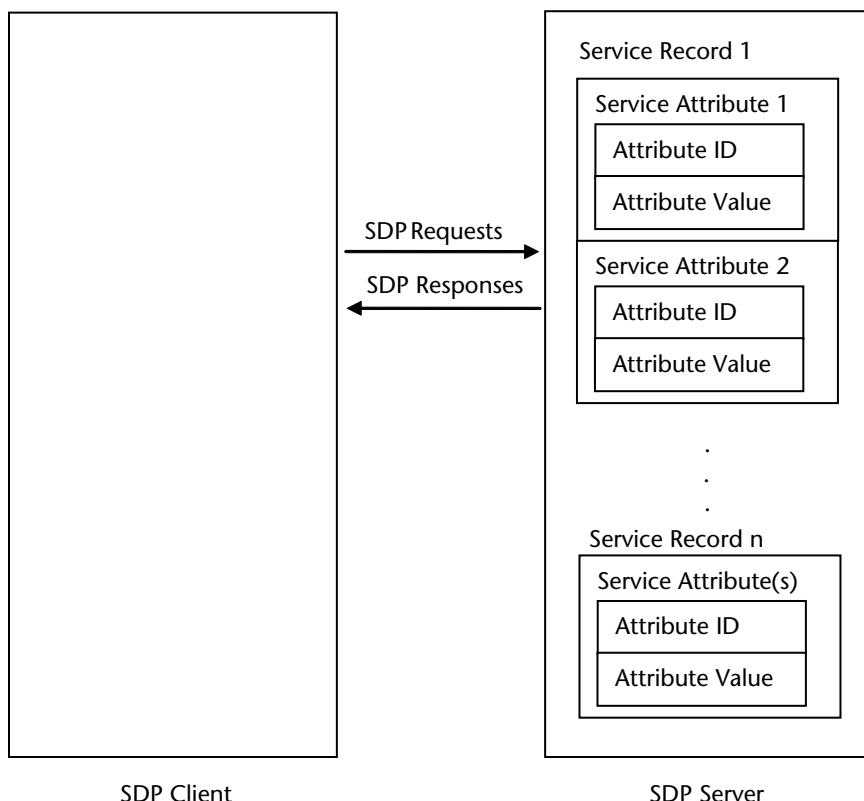


Figure 4.5 SDP Client and Server.

action, or control a resource on behalf of another entity. It may be implemented as software, hardware or a combination of both. For example a printer could provide the service of color printing; a smartphone could provide the service of a dial-up-networking gateway.

A service record contains all information that an SDP server wants to provide about the service to the SDP clients. This is in the form of a list of Service Attributes. Each Service Attribute describes a single characteristic of the service and is in the form of an Attribute ID and Attribute Value. An Attribute ID is a 16-bit unsigned integer that distinguishes each service attribute from other service attributes within a service record. An 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.

Each service is an instance of a service class. The service class definition provides the definitions of all service records that can be present in the service. Each service class is assigned a unique identifier called the service identifier. It is represented as a Universally Unique Identifier (UUID).

What is UUID?

A UUID is a universally unique identifier that is guaranteed to be unique across all space and time. UUIDs can be independently created in a distributed fashion. No central registry of assigning UUIDs is required.

A UUID is a 128-bit value. To reduce the burden of storing and transferring 128-bit values, a range of UUIDs has been pre-defined along with 16-bit or 32-bit aliases. A 16-bit or 32-bit UUID may be converted into 128-bit UUID by pre-defined formulas.

Table 4.4 provides a list of commonly used service attributes supported by SDP. Out of these, the first two attributes viz ServiceRecordHandle and ServiceClassIDList are mandatory to exist in every service record instance. The remaining service attributes are optional. A sample of some of these service attributes is shown in Figure 4.6.

4.4.2 Searching and Browsing Services

There are two ways for an SDP client to get the services from the SDP server:

- Search for Services.
- Browse for Services.

Using the Service Search transaction, the client can search for service records for some desired characteristics. These characteristics are in the form of Attribute Values. The search is in the form of a list of UUIDs to be searched. A service search

Table 4.4 Commonly Used Service Attributes

<i>Attribute Name</i>	<i>Description</i>
ServiceRecordHandle	It uniquely identifies each service record within an SDP server. It is used to reference the service by the clients.
ServiceClassIDList	A list of UUIDs representing the service classes that this service record conforms to. For example Headset Audio Gateway, Dial Up Networking.
ServiceRecordState	This is used for caching of service attributes. Its value is changed when any other attribute value is added, deleted, or changed in the service record. The clients can read this value to check if any values have changed in the service record since the last time they read it.
ServiceID	A UUID that uniquely identifies the service instance described by the service record.
ProtocolDescriptorList	A list of UUIDs for protocol layers that can be used to access the service. For example: L2CAP, RFCOMM.
BluetoothProfileDescriptorList	A list of elements to provide information about the Bluetooth profiles to which this service conforms to.
ServiceName	A string containing name of the service.
ServiceDescription	A string containing a brief description of the service
ProviderName	A string containing the name of the person or organization providing this service.

pattern is formed by the SDP client which is a list of all UUIDs to search. This service search pattern is sent to the SDP server in the service search request. This pattern is matched on the SDP server side if all the UUIDs in the list are contained in any specific service record. A handle to this service record is returned if a match is found.

Another technique to fetch the list of the service records on the server is by Browsing for Services. In this case the client discovers all the services of the server instead of any specific ones. This is useful when the client does not have previous knowledge of the type of services which the servers supports, so it's difficult to form a UUID pattern or if the client is interested in getting the complete list of services supported. An example of browsing the services of a smartphone using the BlueZ stack on Linux is shown in Figure 4.6.

4.4.3 SDP Transactions

SDP is a simple protocol with minimal requirements on the underlying transport. It uses a request/response model. Each transaction comprises one request PDU and one response PDU. The client sends one request and then waits for a response before sending the next request. So, only one request can be pending at any time. This makes the design of the client and server quite simple.

The different transactions that are supported by SDP are described in Table 4.5.

An example of the SDP transactions between a mobile phone and an A2DP headset is shown in Figure 4.7 and Figure 4.8.

Figure 4.7 shows the SDP_ServiceSearchAttributeRequest where the mobile phone queries the A2DP headset to check if the following services exist:

```
# sdptool browse 68:ED:43:25:0E:99
Browsing 68:ED:43:25:0E:99...
Service Name: Dialup Networking
Service RecHandle: 0x10000
Service Class ID List:
  "Dialup Networking" (0x1103)
  "Generic Networking" (0x1201)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 1
Profile Descriptor List:
  "Dialup Networking" (0x1103)
  Version: 0x0100

Service Name: Voice gateway
Service RecHandle: 0x10001
Service Class ID List:
  "Headset Audio Gateway" (0x1112)
  "Generic Audio" (0x1203)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 2
Profile Descriptor List:
  "Headset" (0x1108)
  Version: 0x0100

Service Name: Hands-free
Service RecHandle: 0x10002
Service Class ID List:
  "Handsfree Audio Gateway" (0x111f)
  "Generic Audio" (0x1203)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 3
Profile Descriptor List:
  "Handsfree" (0x111e)
  Version: 0x0105
```

BlueZ SDP command to browse SDP services on remote device

Service Record of First Service

- Service Name
- Service Record Handle
- Service Class ID List
- Protocol Descriptor List
- Profile Descriptor List

Service Record of Second Service

Figure 4.6 Example of browsing SDP services of a smartphone from BlueZ stack running on Linux.

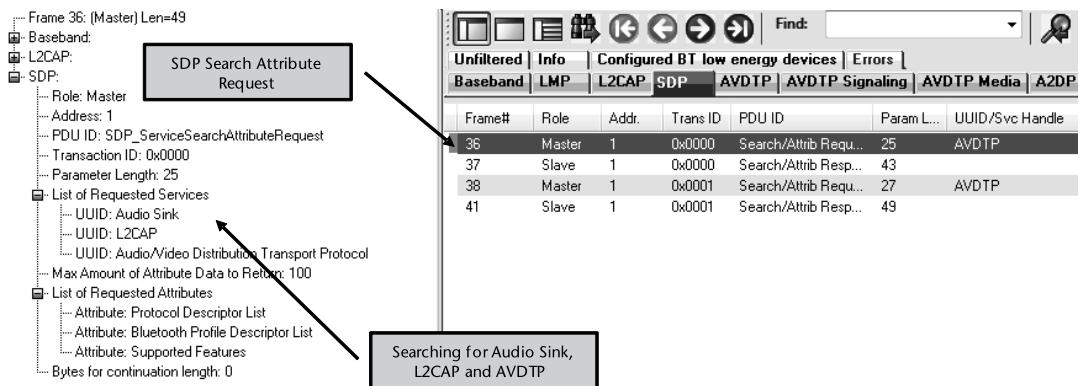
- Audio Sink;
- L2CAP;
- Audio/Video Distribution Transport Protocol (AVDTP).

Figure 4.8 shows the SDP_ServiceSearchAttributeResponse from the A2DP headset to the mobile phone. The A2DP headset responds to inform the support for the following:

- Audio/Video Distribution Transport Protocol (AVDTP) Version 1.0
- AVDTP is using PSM 0x0019 of L2CAP.
- Advanced Audio Distribution (A2DP) Version 1.0.
- Headphone is supported.
- Amplifier, Recorder and Speaker are NOT supported.

Table 4.5 SDP Transactions

PDU ID	Transaction	Description
0x00	Reserved	—
0x01	SDP_ErrorResponse	The SDP server sends this PDU if an error occurred and it cannot send the correct response PDU. This could be the case, for example, if the request had incorrect parameters.
0x02	SDP_ServiceSearchRequest	This PDU is sent by the SDP client to locate service records that match a service search pattern.
0x03	SDP_ServiceSearchResponse	Upon receipt of SDP_ServiceSearchRequest, the SDP server searches its service record data base and returns the handles of the service records that match the pattern using this PDU.
0x04	SDP_ServiceAttributeRequest	This PDU is sent by the SDP client to retrieve specified attribute values from a specific service record. (The service record handle would have been fetched already by the client using SDP_ServiceSearchRequest transaction).
0x05	SDP_ServiceAttribute_Response	The SDP server uses this PDU to provide the list of attributes (Attribute ID, Attribute Value) from the requested service record that was provided in the SDP_ServiceAttributeRequest.
0x06	SDP_ServiceSearchAttributeRequest	This PDU combines the capabilities of SDP_ServiceSearchRequest and SDP_ServiceAttributeRequest. The SDP client provides both the service search pattern and list of attributes to retrieve.
0x07	SDP_ServiceSearchAttributeResponse	The SDP server uses this PDU to provide a list of attributes (Attribute ID, Attribute Value) from the service records that match the requested service search pattern.

**Figure 4.7** Example of SDP_ServiceSearchAttributeRequest.

4.5 RFCOMM

The RFCOMM protocol is based on the ETSI (European Telecommunications Standards Institute) standard TS 07.10. It is referred to as an adopted protocol because it uses a subset of TS 07.10 protocol and makes some adaptations and extensions to that protocol.

The TS 07.10 is essentially a multiplexer protocol that allows a number of simultaneous sessions over a normal serial interface. Each session could be used for transferring various kinds of data, for example voice, SMS, data, GPRS etc. For details see the bibliography.

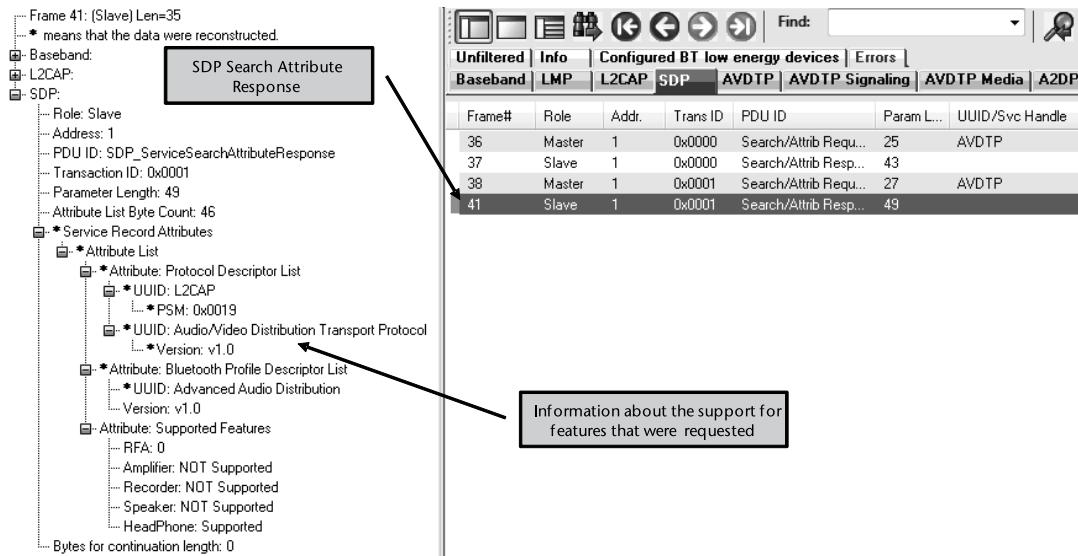


Figure 4.8 Example of SDP_ServiceSearchAttributeResponse.

RFCOMM provides the emulation of RS-232 serial ports on top of the L2CAP protocol. One of the first intended uses of the Bluetooth technology was as a cable replacement protocol. RFCOMM is the key component to enable this cable replacement. Broadly it may be compared to an RS-232 serial cable where the end points of the cable get replaced with RFCOMM endpoints and the cable is replaced with a Bluetooth connection. This may be used anywhere where serial cables are used to replace those cables with a wireless connection. Some examples are:

- Communication between PCs;
- Connection of mobile phone to headset;
- Connection of mobile phone to laptop.

RFCOMM supports up to 60 simultaneous connections between two Bluetooth devices. As an analogy this can be considered to be similar to two PCs connected to each other using up to 60 serial cables. The user can run separate applications on each of the serial ports.

There are broadly two types of communications devices:

- Type 1 devices are communication end points. As the name suggests, these devices are at the end of the communication path and are either the producer or consumer of data. For example a laptop that is used to browse the internet is a communication end point.
- Type 2 devices are part of the communication segment. These devices allow data to be relayed from one segment to another. For example a mobile phone may relay the data to the cellular network or a Bluetooth modem may relay the data to the telephony network.

RFCOMM supports both these types of devices.

RS-232 serial interface has following nine circuits which are used for data transfer and signaling.

- TD (Transmit Data) and RD (Receive Data) to carry the data.
- RTS (Request To Send) and CTS (Clear To Send) for Hardware Handshaking or Flow Control.
- DSR (Data Set Ready) and DTR (Data Terminal Ready) for Hardware Flow Control.
- CD (Data Carrier Detect) to indicate a connection to the telephone line.
- RI (Ring Indicator) to indicate an incoming ring signal on the telephone line.
- Signal Common to connect to common ground.

RFCOMM emulates these nine circuits. One of the advantages of this is that it provides backward compatibility with Terminal Emulation programs (like Hyper-terminal, TeraTerm etc). These terminal emulation programs can be run on virtual serial ports provided with an underlying RFCOMM connection and provide the same user experience as the wired serial ports connected through RS-232 cables.

As shown in Figure 4.9, up to 60 emulated ports can be active simultaneously though in practice a much lesser number of ports may be supported by the implementation. Each connection is identified by a DLCI (Data Link Connection Identifier).

DLCI 0 is used as a control channel. This is used to exchange Multiplexer Control commands before setting up the other DLCIs.

The DLCI value space is divided between the two communicating devices using the concept of RFCOMM server channels and direction bit. The server applications registering with RFCOMM are assigned a Server Channel Number in the range 1 to 30. The device that initiates the RFCOMM session is assigned the direction bit 1. (This is the lowest significant bit in DLCI).

So the server applications on the non-initiating side are accessible on DLCI 2, 4, 6, ..., 60 and the server application on the initiating side are accessible on DLCI 3, 5, 7, ..., 61.

One of the enhancements that RFCOMM made to TS 07.10 is the credit based flow control. It was introduced after Bluetooth spec 1.0b to provide a flow control mechanism between the two devices. At the time of DLCI establishment, the

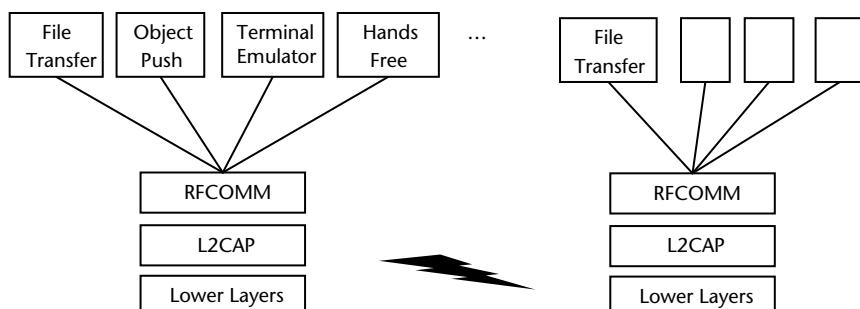


Figure 4.9 RFCOMM multiplexing.

receiving entity provides a number of credits to the transmitter. The transmitter can send as many frames as it has credits and decrement its credit count accordingly. Once the credit count reaches zero, it stops further transmission and waits for further credits from the receiver. Once the receiver is ready to receive more packets (For example after it has processed the previous packets) it provides further credits to the transmitter. This provides a simple and effective mechanism to emulate the flow control circuits of RS-232.

4.6 Object Exchange Protocol (OBEX)

Bluetooth technology has adopted the IrOBEX protocol from Infrared Data Association (IrDA). OBEX provides the same features for applications as IrDA protocol. So applications can work on both the Bluetooth stack and IrDA stack. That is why OBEX is also referred to as an adopted specification.

Version 1.1 of the OBEX specification provided for support of OBEX over RFCOMM. (It also defined optional OBEX over TCP/IP though it was not used by most of the protocol stacks). Version 2.0 of the specification provided for support of OBEX directly over L2CAP bypassing the RFCOMM layer. This helped in reducing the overheads of RFCOMM and providing higher throughputs, especially in the case of BT 3.0 + HS.

The OBEX protocol follows the client/server model. The purpose of this protocol is to exchange data objects. These data objects could be business cards, notes, images, files, calendars etc.

Some examples of usage of OBEX are provided below:

- **Synchronization:** OBEX could be used for the synchronization of data between two devices. For example to keep the contacts information and calendar in sync between the laptop and mobile phone.
- **File Transfer:** OBEX can be used for sending and receiving files, browsing folders, deleting files, etc.
- **Object Push:** OBEX can be used for sending (pushing) and fetching (pulling) objects like business cards, calendars, notes, etc. Standard formats for these objects are used to ensure interoperability. These formats are referred to as vCard, vCalendar, vMessage and vNotes (electronic business card, electronic calendar, electronic message, and electronic notes).

A device may implement the client role only, server role only or both. For example, a printer may support only the server role. Other devices may connect to the printer and push objects that are to be printed. On the other hand a mobile phone may support both a client and server role so that it can either push or pull objects from other devices or allow other devices to push and pull objects from it.

4.6.1 OBEX Operations

OBEX follows a client/server request-response mechanism as shown in Figure 4.10. The client is the initiator of the OBEX connection. Requests are issued by the client

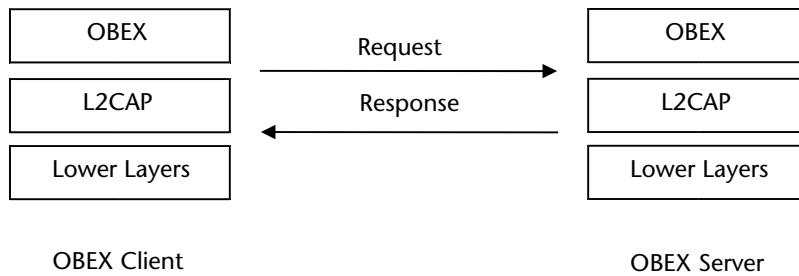


Figure 4.10 OBEX Client and Server (With OBEX over L2CAP).

and the server responds to these requests. The request/response pair is referred to as an operation.

The requests/responses are sequential in nature. After sending a request, the client waits for a response from the server before issuing another request.

The various operations supported by OBEX are shown in Table 4.6.

4.7 Audio/Video Control Transport Protocol (AVCTP)

AVCTP defines the transport mechanisms used to exchange messages for controlling Audio or Video devices. It uses point-to-point signaling over connection oriented L2CAP channels.

Two roles are defined:

- Controller (CT): This is the device that initiates an AVCTP transaction by sending a command message. The device that supports the controller functionality is also responsible for initiating the L2CAP channel connection on request of the application.
- Target (TG): This is the remote device that receives the command message and returns zero or more responses to the controller.

A complete AVCTP transaction consists of one message containing a command addressed to the target and zero or more responses returned by the target to the controller. A device may support both CT and TG roles at the same time.

Table 4.6 OBEX Operations

Operation	Meaning
Connect	This operation is used by the client to initiate a connection to the server and negotiate the parameters to be used for further operations (For example OBEX version number, maximum packet length)
Disconnect	This signals the end of OBEX connection.
Put	The Put operation is used by the client to push one object from the client to the server.
Get	The Get operation is used by the client to request to server to return an object to the client.
SetPath	The SetPath operation is used to set the “current directory” on the server side. All further operations are carried on from that directory after this command is sent.
Abort	The Abort request is used when the client decides to terminate an operation that was spread over multiple packets between the client and server.

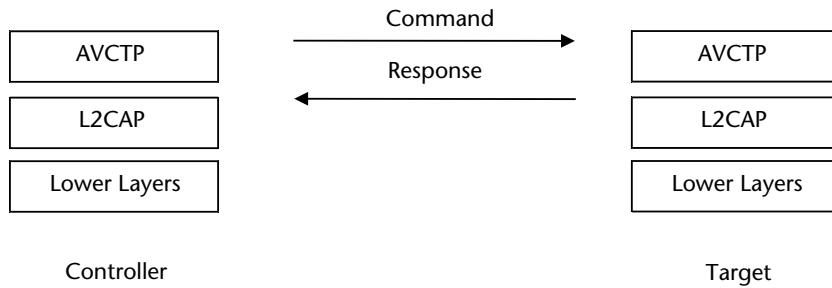


Figure 4.11 AVCTP Controller and Target.

AVCTP may support multiple profiles on top. It uses the concept of a Profile Identifier to allow applications to distinguish messages from different profiles.

4.8 Audio/Video Distribution Transport Protocol (AVDTP)

AVDTP defines the protocol for audio/video distribution connection establishment, negotiation and streaming of audio/video media over the Bluetooth interface. The transport mechanism and message formats are based on the RTP protocol which consists of two major protocols: RTP Data Transfer Protocol (RTP) and RTP Control Protocol (RTCP). AVDTP uses the L2CAP connection oriented channels for setting up the A/V streams and then streaming the data. A stream (or Bluetooth A/V stream) represents the logical end-to-end connection of streaming media between two A/V devices.

Two roles are defined:

- Source (SRC): This is the device where the streaming data originates.
- Sink (SNK): This is the device which receives the audio data.

As an example in the scenario of streaming data from a laptop to a Bluetooth stereo headset, the stream corresponds to the audio stream between the laptop and the Bluetooth headset. The laptop acts as a SRC device and the Bluetooth stereo headset acts as the SNK device.

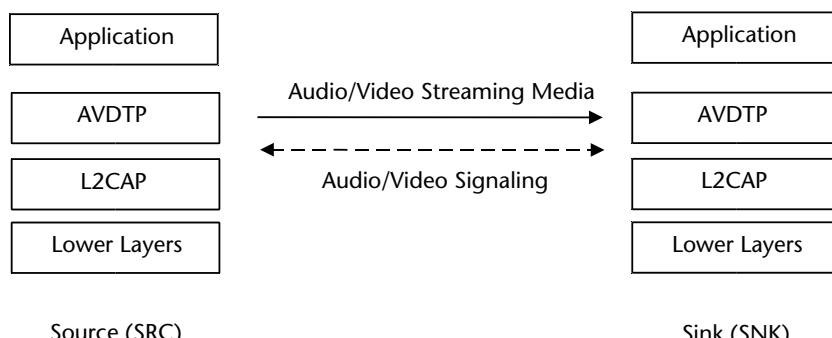


Figure 4.12 AVDTP Source and Sink.

A Stream End Point (SEP) is a concept to expose the available transport services and AV capabilities of the application in order to negotiate a stream. An application registers its SEPs in AVDTP to allow other devices to discover and connect to them.

AVDTP defines procedures for the following:

- Discover: To discover the Stream End Points supported in the device.
- Get Capabilities: To get the capabilities of the Stream End Point.
- Set Configuration: To configure the Stream End Point.
- Get Configuration: To get the configuration of the Stream End Point.
- Reconfigure: To reconfigure the Stream End Point.
- Open: Open a stream.
- Start: To start streaming.
- Close: To request closure of a Stream End Point.
- Suspend: To request that a Stream End Point be suspended.
- Security Control: To exchange content protection control data.
- Abort: To recover from error conditions.

A typical sequence of operations of AVDTP transactions is shown in Figure 4.13. In this scenario a mobile phone (acting as CT) creates a connection to an A2DP headset (acting as TG), streams a music file and then disconnects. The sequence of transactions that happen is as follows:

1. Frame #65: The mobile phone sends a command to discover the stream end points in the headset.
2. Frame #66: The headset responds with information about the stream end points. (In this example, the slave actually provides two stream end points.
 - a. One that supports MP3 codec.
 - b. Second that supports SBC codec.

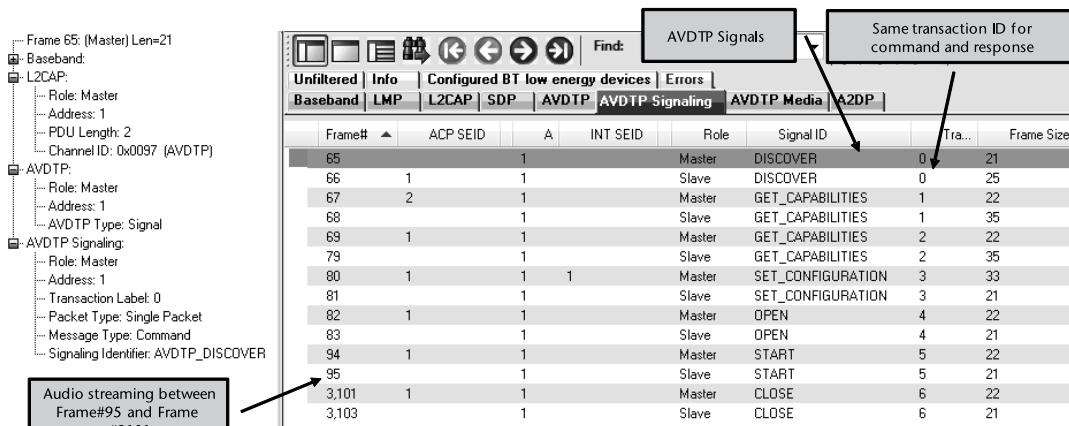


Figure 4.13 Example of AVDTP transactions between mobile phone and stereo headset.

3. Frame #67, #68: The mobile phone gets the capabilities of the first stream end point and the headset responds.
4. Frame #69 and #79: The mobile phone gets the capabilities of the second stream end point and the headset responds.
5. Frame #80: The mobile phone configures the stream end point on the headset with the parameters needed to stream the audio.
6. Frame #82: The mobile phone opens the stream.
7. Frame #94: The mobile phone starts streaming the audio data.
8. Frame #95 to Frame #3100: The audio data is streamed from the mobile phone to the headset.
9. Frame #3101: The mobile phone decides to close the stream.

4.9 Profiles

As explained at the beginning of this chapter, profiles can be considered to be vertical slices through the protocol stack. They provide information on how each of the protocol layers comes together to implement a specific usage model. They define the features and functions required from each layer of the protocol stack from Bluetooth Radio up to L2CAP, RFCOMM, OBEX, and any other protocols like AVCTP, AVDTP, etc. Both the vertical interactions between the layers as well as peer-to-peer interactions with the corresponding layers of the other device are defined.

Profiles help to guarantee that an implementation from one vendor will work properly with an implementation from another vendor. So they form the basis for interoperability and logo requirements. The profiles need to be tested and certified before a device can be sold in the market. A device can support one or more profiles at the same time.

Generic Access Profile (GAP) is mandatory to be implemented for all devices that support Bluetooth. Devices may implement more profiles depending on the requirements of the application. The dependencies amongst profiles are depicted in Figure 4.14. A profile is dependent on another profile if it uses parts of that profile. A dependent profile is shown in an inner box and the outer box indicates the profiles on which it is directly or indirectly dependent. GAP is shown in the outermost box since all other profiles are dependent on it.

For example Hands-free Profile is dependent on Serial Port Profile which is in turn dependent on Generic Access Profile. So the box for Hands-free Profile is shown within the box for Serial Port Profile. The Box for Serial Port Profile is in turn located inside the box for Generic Access Profile.

4.10 Generic Access Profile (GAP)

Generic Access Profile is a base profile which is mandatory for all devices to implement. It defines the basic requirements of a Bluetooth device. For BR/EDR it defines a Bluetooth device to include at least the following functionality:

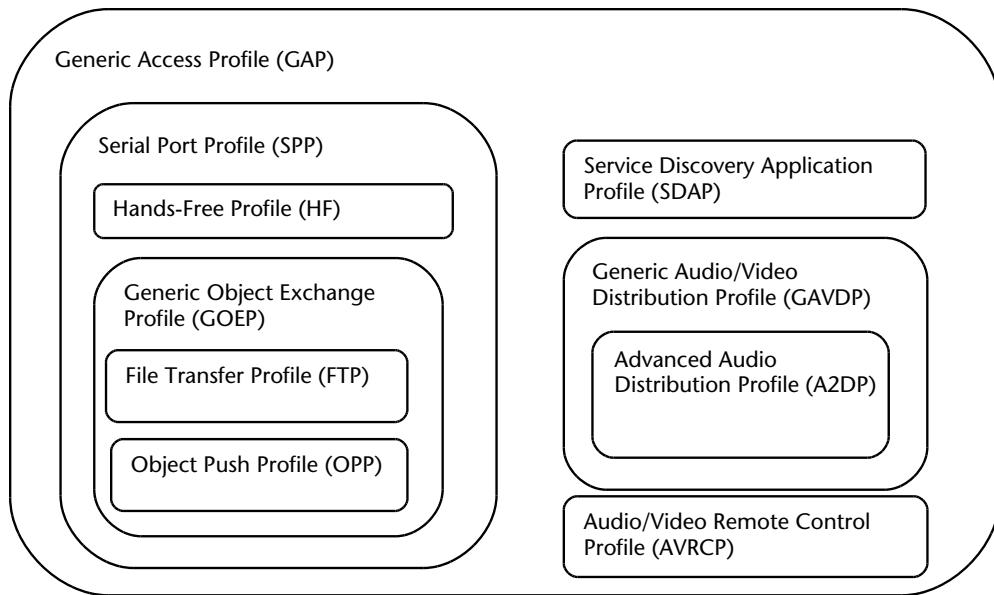


Figure 4.14 Profile dependencies.

- Bluetooth Radio;
- Baseband;
- Link Manager;
- L2CAP;
- SDP.

GAP defines how these layers come together to provide the Bluetooth functionality. It also defines procedures for the following:

- Device discovery;
- Connection establishment;
- Security;
- Authentication;
- Service discovery.

The purpose of GAP is:

- To introduce definitions, recommendations, and common requirements related to modes and access procedures that are used by transport and application profiles.
- To describe how the devices behave in various states like standby and connecting. Special focus is put on discovery, connection establishment and security procedures.
- To state requirements on user interface aspects, names of procedures, and parameters, etc. This ensures a uniform user experience across all devices.

GAP defines the procedures for both BR/EDR and LE device types. It defines three device types:

- BR/EDR: Devices that support Basic Rate and Enhanced Data Rate.
- LE Only: Devices that support Low Energy configuration.
- BR/EDR/LE: Dual mode devices that support both BR/EDR and LE.

The BR/EDR procedures will be covered in this section and LE procedures will be covered in Chapter 14.

4.10.1 Bluetooth Parameters Representation

GAP states requirements about the generic terms that should be used on the user interface (UI) level. These are useful not only when designing user interfaces but also in user manuals, documentation, advertisements, etc. This helps to ensure a uniform user experience irrespective of the vendor who makes the device or the application.

GAP defines the requirements for:

- Bluetooth Device Address (BD_ADDR)
 - On the user interface level, the address should be referred to as “Bluetooth Device Address” and represented as 12 hex characters possibly separated by “:” symbol. An example of representation of the BD Address is 00:AB:CD:EF:12:34.
- Bluetooth Device Name
 - This is the user friendly name that can be used to refer to a device. This is in the form of a character string which can be retrieved by remote devices using the remote name request. The maximum length of the character string is 248 bytes.
- Bluetooth Passkey (Bluetooth PIN)
 - The pairing process was explained in the previous chapter. The Bluetooth passkey may be used to authenticate two devices via the pairing procedure. The PIN may either be entered on the UI level (For example on the user interface on mobile phone or laptop) or stored in the device (For example the predefined PIN key stored in a headset).
- Class of Device (CoD)
 - Class of device provides information on the type of device and the type of services it supports. The Class of Device is retrieved during the inquiry procedure. GAP defines that the Class of Device parameters should be referred to as “Bluetooth Device Class” and “Bluetooth Service Type” on the UI level. These are obtained from various fields of the CoD.

4.10.2 Modes

GAP defines different modes in which the device can be in with respect to inquiry and connection. These modes are explained in this section.

4.10.2.1 Discoverability Modes

Inquiry is the procedure to discover devices in the Bluetooth vicinity. With respect to inquiry, a device can be in one of the following two discoverability modes:

- Non-Discoverable mode: In this mode a device does not respond to inquiry. So it cannot be discovered by other devices.
- Discoverable mode: In this mode the device is set to discoverable mode and it may respond to inquiry from remote devices. There are two discoverable modes:
 - Limited Discoverable mode: The limited discoverable mode is used by devices that are discoverable only for a limited amount of time, during temporary conditions, or for a specific event. The device responds to a device that makes a limited inquiry.
 - General Discoverable mode: This mode is used by devices that need to be discoverable continuously or for no specific condition. The device responds to any device that make a general inquiry.

4.10.2.2 Connectability Modes

Paging is the procedure used to connect to remote devices. With respect to paging, a device can be in one of the following two connectable modes:

- Non-connectable mode: In this mode, the device does not enter the PAGE_SCAN state. So it's not possible to connect to this device. (PAGE_SCAN state was explained in the previous chapter).
- Connectable mode: In this mode, the device periodically enters the PAGE_SCAN state to check for incoming connection requests. So other devices can connect to this device.

4.10.2.3 Bondable Modes (for Pairing)

Bonding is the process of pairing when the passkey is entered on the device for the purpose of creating a “bond” between two Bluetooth devices. It may or may not be followed later on by creation of a connection.

With respect to bonding, a device can be in one of the following two bondable modes:

- Non-bondable mode: In this mode, the device does not accept a pairing request from other devices. It may still accept an incoming connection from devices that do not require bonding.

- Bondable mode: In this mode, the device accepts a pairing request from other devices. Pairing can be either in the form of legacy pairing or secure simple pairing (SSP).

4.10.3 Idle Mode Procedures

The idle mode procedures include procedures for inquiry, name discovery, bonding etc. These are called idle mode procedures because generally these are initiated when the device is in the idle mode though these can also be done when the device is already connected (for example, to create a scatternet).

4.10.3.1 Inquiry

Inquiry is the procedure to discover information about remote devices. Devices can dynamically enter and move out of the Bluetooth vicinity. This procedure is useful to find out the list of devices that are currently in the vicinity along with some basic information about those devices. Using this information, a decision may be taken to further move on to stages like discovering name, creation of connection, etc.

Inquiry provides the following information about the remote device:

- BD_ADDR;
- Clock information;
- Class of Device;
- Information about Page scan mode;
- Extended Inquiry Response Information if it is supported by the device.

Bluetooth specification defines two types of inquiry:

- General Inquiry: This is used to discover devices which are set to discoverable mode and are set to do an inquiry scan with General Inquiry Access Code. (GIAC). This is used for devices that are made discoverable continuously or for no specific condition.
- Limited Inquiry: This is used to discover devices which are scanning with Limited Inquiry Access Code (LIAC). This is used for devices which are made discoverable only for a limited amount of time, during temporary conditions for a specific event.

Devices that are set to Limited discoverable mode are also discovered in General Inquiry. The term used on User Interface level is “Bluetooth Device Inquiry.”

4.10.3.2 Name Discovery

This procedure is used to get the Bluetooth name of the remote device. The term used on User Interface level is “Bluetooth Device Name Discovery.”

4.10.3.3 Device Discovery

This procedure is similar to the Inquiry procedure. It is used to find some additional information besides the one provided in inquiry.

Discovery provides the following information about the remote device:

- BD_ADDR;
- Clock information;
- Class of Device;
- Information about Page scan mode;
- Extended Inquiry Response Information;
- Bluetooth Device Name—This is the additional information that is not reported during inquiry.

The term used on User Interface level is “Bluetooth Device Discovery.”

4.10.3.4 Bonding

The bonding procedure is used to create a mutual trust relationship between two Bluetooth devices based on a common link key. The link key is created and exchanged during this procedure and is expected to be stored by both the Bluetooth devices. This link key is used for authentication when a connection is created.

The term used on the User Interface level is “Bluetooth Bonding.”

4.10.4 Establishment Procedures

These procedures refer to the link establishment, channel establishment and connection establishment. A Device discovery or inquiry procedure is done before establishment procedures to get information about the device to which the connection is to be established. These are shown in Figure 4.15.

4.10.4.1 Link Establishment

This procedure is used to create an ACL link between two devices. The term used on the User Interface level is “Bluetooth link establishment.”

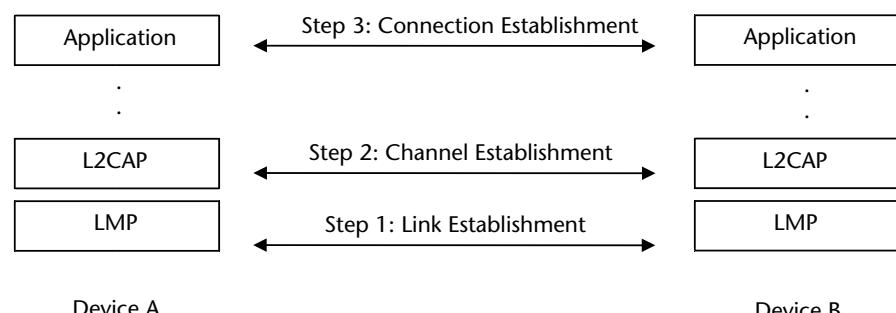


Figure 4.15 Establishment procedures.

4.10.4.2 Channel Establishment

This procedure is used to create an L2CAP channel between two devices. The term used on the User Interface level is “Bluetooth channel establishment.”

4.10.4.3 Connection Establishment

This procedure is used to establish a connection between applications on two Bluetooth devices. The term used on the User Interface level is “Bluetooth connection establishment.”

4.10.5 Authentication

Authentication was explained in the previous chapter. It is the process of verifying “who” is at the other end of the link. The authentication process starts when the two devices initiate a connection establishment. If the link is not already authenticated and the link key is not available, then the pairing procedure is started.

4.10.6 Security

There are two broad types of security modes: Legacy security mode and Simple Secure Pairing mode. Legacy mode is used by devices that do not support SSP.

Within legacy security mode, there are three types of security modes:

- Security mode 1 (non-secured): This mode means that no security is desired.
- Security mode 2 (service level enforced security): In this mode, the security is initiated after the connection establishment if the channel or service requires security. For example if security is required for a particular profile, then this security mode may be used.
- Security mode 3 (link level enforced security): In this mode, the security is initiated during the connection establishment.

Version 2.1 + EDR of the Bluetooth specification added Simple Secure Pairing and Security mode 4:

- Security mode 4 (service level enforced security): The security can be specified with the following attributes:
 - Authenticated link key required: This is the link key generated using numeric comparison, out-of-band, or passkey entry. It has protection against MITM attacks as explained in Chapter 3.
 - Unauthenticated link key required: This is the link key generated using just works method. It does not provide protection against MITM attacks.
 - Security optional: This is only used for SDP transactions.

It is possible for a device to support two security modes at the same time. This could be the case when it wants to connect to legacy devices using Security Mode 2 and devices that support Simple Secure Pairing with Security mode 4.

4.11 Serial Port Profile (SPP)

The serial port profile (SPP) defines the requirements for setting up emulated serial connections between Bluetooth devices. This provides similar user experience as compared to an RS232 cable connection between the two devices with the only difference that a physical wire is replaced by the Bluetooth connection between the two devices.

Bluetooth was originally designed as a cable replacement technology and this was amongst the first profiles that were used since it supports the cable replacement use case.

One of the strongest points about this profile is that it allows legacy applications (which were designed for RS232 serial ports) to use Bluetooth wireless connection instead of a wired connection. Generally another application (for example, a Bluetooth connection management application) is used to initially discover devices in the vicinity and create an SPP connection with the remote device. Once this is done, the legacy application can transparently use the Bluetooth connection as if it were a wired RS232 connection. As shown in Figure 4.14, SPP depends on GAP and re-uses the terms and procedures defined in the GAP profile. SPP is in turn used by other profiles like Hands-free and GOEP.

SPP defines two roles:

- Dev A: This is the device that initiates a Bluetooth connection.
- Dev B: This is the device that waits for a device to make a connection and then accepts the incoming connection.

Figure 4.16 shows a typical usage scenario of Serial Port Profile. It uses the RFCOMM, L2CAP and lower layers of the Bluetooth protocol stack. A port emulation layer is used to emulate the serial port. In general this layer is dependent on the operating system. For example on Linux, this layer may expose a virtual serial port device driver. A Legacy serial port application could be used on both devices to connect to the emulated serial port.

Besides this a separate Bluetooth application may be used for initial device discovery, services discovery, connection establishment, etc. Once a Bluetooth connection is established, the legacy application can start using the emulated serial port just like any other RS232 serial port.

4.12 Headset Profile, Hands-Free Profile

The Headset and Hands-free profiles define the set of functions to be used for a Bluetooth connection between a mobile phone and a handsfree device, for example a Bluetooth mono headset, Carkit installed in a car, etc. The Headset profile was

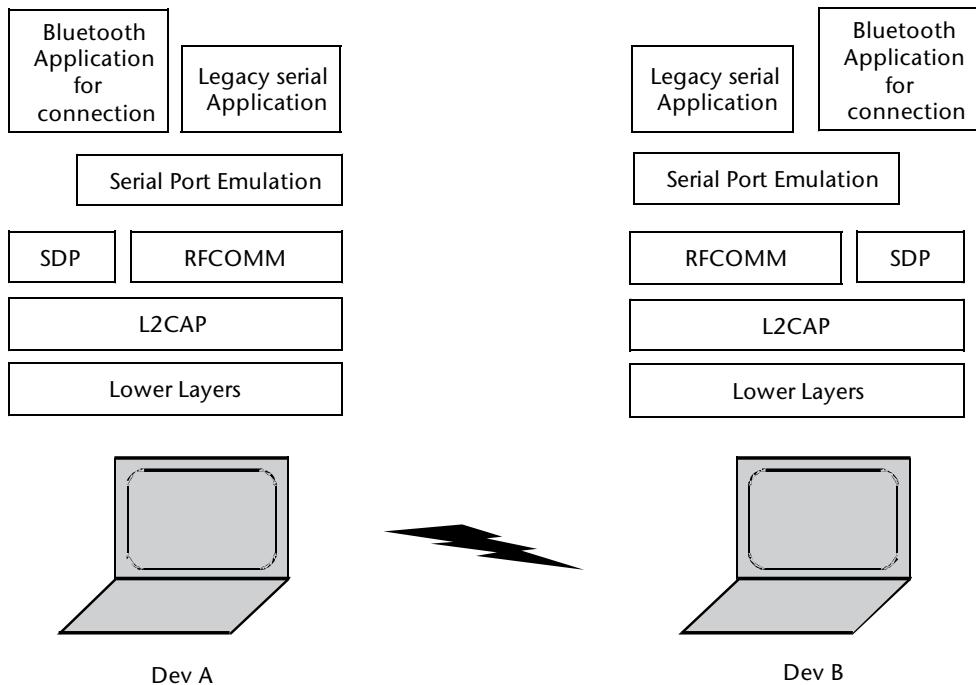


Figure 4.16 Typical usage scenario of serial port profile.

one of the first profiles defined by the Bluetooth specification. Later on this profile was superseded by the Hands-free profile. The Hands-free profile provides a super-set of the Headset profile functionality.

Some of the functionality defined by this profile includes:

- Connection related functionality.
 - Connection to a handsfree device so that the audio can be routed from the mobile phone to the handsfree device.
 - Accept an incoming call.
 - Reject an incoming call.
 - Terminate a call.
- Display of phone status like signal strength, roaming, battery level, etc, on the hands-free device.
- Transfer an audio connection from phone to hands-free or vice versa.
- Different options for placing a call from the hands-free device.
 - From a number supplied by the hands-free device.
 - By memory dialing.
 - Redial last number.
- Activation of voice recognition.
- Call waiting notification.
- Three-way calling.

- Caller line identification (CLI).
- Echo cancellation and Noise reduction.
- Remote audio volume control.

Hands-free profile defines the following two roles. These are shown in Figure 4.17.

- **Audio Gateway (AG):** This is the device that acts as a gateway for audio. Typically this is the mobile phone which acts as a gateway of the audio from the cellular network to the hands-free device.
- **Hands-Free unit (HF):** This is the device that acts as the audio input and output mechanism. This may also provide some means to control some of the functionality of the AG. Typically this is a Carkit or a handset.

The Hands-free profile defines two types of connections:

- **Service level connection:** This is the RFCOMM connection between the AG and HF which is used for transfer of control information.
- **Audio Connection:** This is the SCO or eSCO connection along with the complete audio path to route the audio (voice data) from the cellular network to the hands-free unit.

The Hands-free profile uses AT commands extensively on the Service level connection to perform various tasks. The AT commands used by this profile are a subset of the 3GPP 27.0.0.2 specification.

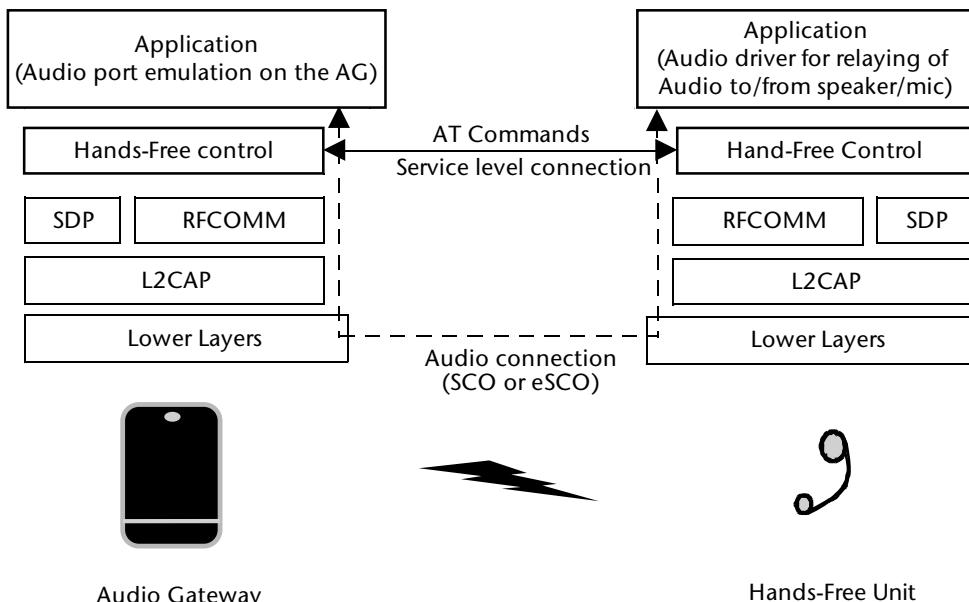


Figure 4.17 Hands-free profile.

4.13 Generic Object Exchange Profile (GOEP)

GOEP defines the requirements for devices like laptops, PDAs and smartphones to support capabilities to exchange objects. As shown in Figure 4.14, this profile is dependent on GAP and SPP. In turn it provides features to support profiles like FTP and OPP.

GOEP defines the following two roles. These are shown in Figure 4.18.

- Server: This is the device that acts as the object exchange server to and from which objects can be pushed and pulled.
- Client: This is the device that can push or pull objects to and from the server.

As shown in Figure 4.18, this profile makes use of the OBEX, SDP, RFCOMM, L2CAP and lower layers of the protocol stack. It provides the following major features:

- Establishment of an object exchange session: This feature is used in the beginning to establish a connection between the client and the server. The remaining features can only be used once this procedure is successfully completed.
- Pushing a data object: This feature is used to transfer an object from the client to the server.
- Pulling a data object: This feature is used to retrieve an object from the server to the client.

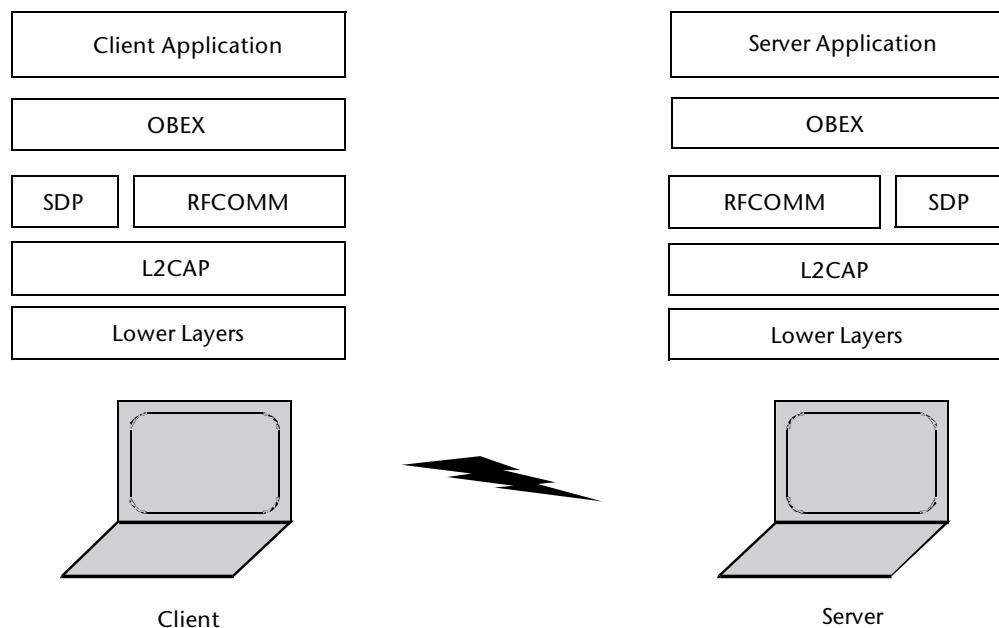


Figure 4.18 Generic Object Exchange Profile.

4.14 Object Push Profile (OPP)

This profile defines the requirements needed to support the object push usage model between two Bluetooth devices. It is dependent on GOEP, SPP and GAP.

OPP defines the following two roles. These are shown in Figure 4.19.

- Push Server: This is the device that acts as the object exchange server to and from which objects can be pushed and pulled.
- Push Client: This is the device that can push or pull objects to and from the push server.

As shown in Figure 4.19, this profile makes use of the OBEX, SDP, RFCOMM, L2CAP and lower layers of the protocol stack. It provides the following major features:

- Object Push: This feature is used to push an object to the inbox of another device. For example to push a business card or an appointment to a mobile phone.
- Business Card Pull: This feature is used to pull an object from the server. For example to pull a business card from a mobile phone.
- Business Card Exchange: This feature is used to exchange objects. For example to push a business card followed by a pull of the business card.

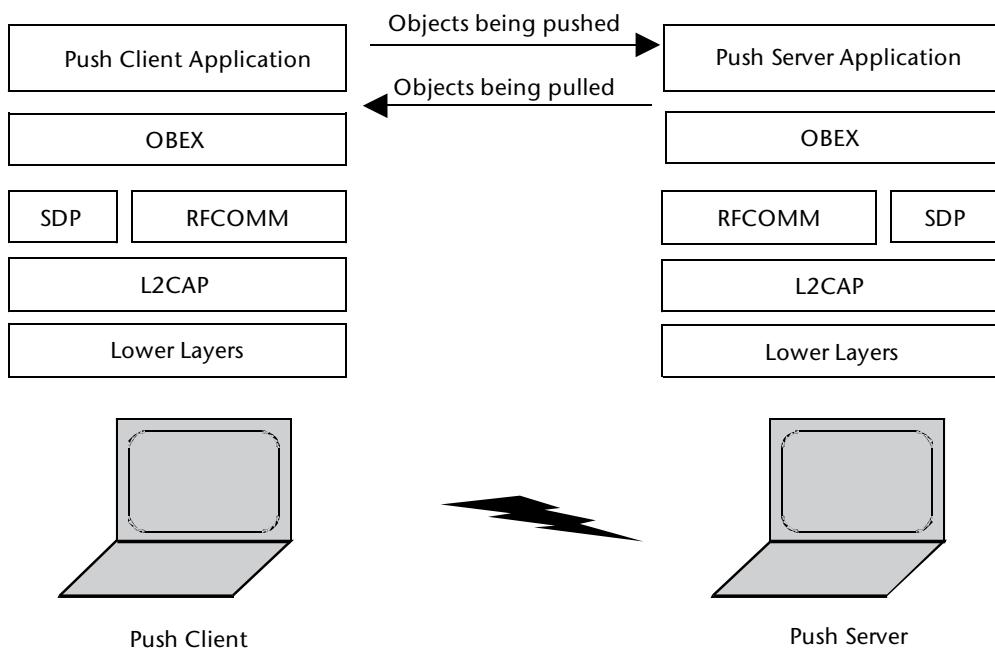


Figure 4.19 Object Push Profile.

The different objects that can be pushed by this profile are as follows:

- vCard: This is a format used for transferring contacts.
- vCalendar: This is a format used for transferring appointments.
- vMessage: This is the format used for messaging applications.
- vNote: This is the format used by notes applications.

References to the details of these formats are provided in the Bibliography section.

4.15 File Transfer Profile (FTP)

This profile defines the requirements needed to support the file transfer usage model between two Bluetooth devices. It is dependent on GOEP, SPP and GAP.

FTP defines the following two roles. These are shown in Figure 4.20.

- Server: This is the device that acts as the file transfer server to and from which files can be pushed and pulled. It also provides the folder browsing capabilities.
- Client: This is the device that can push or pull files to and from the server.

As shown in Figure 4.20, this profile makes use of the OBEX, SDP, RFCOMM, L2CAP, and lower layers of the protocol stack. It provides the following major features:

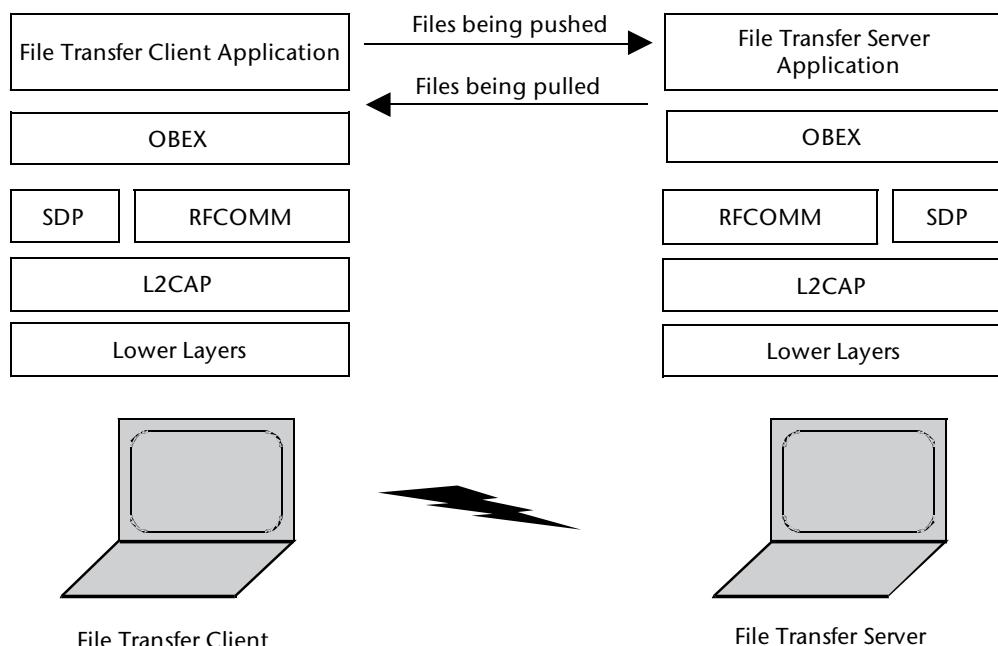


Figure 4.20 File Transfer Profile.

- Browse the file system: This feature allows support for browsing the file system of the server. This includes viewing the files and folders and navigating the folder hierarchy of the other device.
- File Transfer: This feature provides support for transferring files and folders from one device to another.
- Object manipulation: This feature allows manipulating the objects on the other device. This may include deleting files, creating folders, deleting folders, etc.

4.16 Generic Audio/Video Distribution Profile (GAVDP)

This profile defines the requirements for Bluetooth devices to support streaming channels for supporting audio/video distribution on ACL channels. It is dependent on GAP and uses AVDTP, L2CAP and lower layers.

GAVDP defines the following two roles. These are shown in Figure 4.21:

- Initiator (INT): This is the device that initiates the GAVDP signaling procedures.
- Acceptor (ACP): This is the device that responds to the incoming requests from the INT.

A typical use case of the profile is the streaming of audio between a laptop and headphones. In this case, the laptop may act as the INT. It would send request to establish a streaming channel, negotiate parameters, control the stream etc. The headphones would act as the ACP and respond to the requests made by the INT.

GAVDP provides support for the following two scenarios:

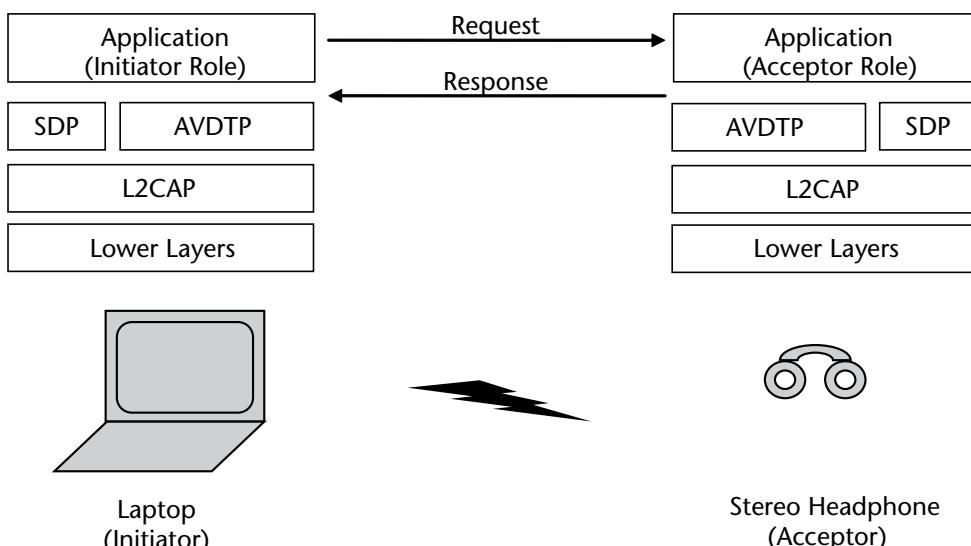


Figure 4.21 Generic Audio Video Distribution Profile.

- Setup the two devices for A/V data streaming and then create a connection between these two devices.
- Control the established streaming connection.

The detailed features and procedures supported by this profile are described in Table 4.7.

4.17 Advanced Audio Distribution Profile (A2DP)

This profile defines the requirements for Bluetooth devices to support high quality audio distribution. It uses the ACL channels for distribution of high quality audio. This is in contrast to the HF profile in which the SCO channels are used to transfer voice.

ACL channels are used because the bandwidth that is provided by SCO and eSCO channels is not sufficient to transfer the high quality audio data. In fact if raw audio data were to be streamed, then even the ACL channels don't have sufficient bandwidth. So, a codec is used to encode the data before sending and then decoding the data after it is received on the remote side.

Another difference from the Hands-free profile is that while Hands-free supports bi-directional transfer of voice data, A2DP supports audio data streaming in only one direction. This is in line with the use cases meant for these profiles. Hands-free is meant for transfer of voice data where users may be having a conversation on the mobile phone. A2DP is meant for transfer of audio data where a user may be listening to music on the wireless headset.

This profile is dependent on GAP and GAVDP.

A2DP defines the following two roles. These are shown in Figure 4.22:

- Source (SRC): This is the device that acts as the source of the digital audio stream to the SNK.

Table 4.7 GAVDP Features and Procedures

Feature	Procedure	Purpose
Connection	Connection Establishment	This procedure is used when a device needs to create a connection with another device. It includes AVDTP procedures for finding the stream end points and getting the capabilities.
	Start Streaming	This procedure is used when both the devices are ready to start streaming and is used to start or resume streaming.
	Connection Release	This procedure is used to release the stream.
Transfer Control	Suspend	This procedure is used to suspend the A/V stream.
	Change Parameters	This procedure is used to change the service parameters of the stream.
Signaling Control	Abort	This procedure may be used to recover from a loss of signaling message.
Security Control	Security Control	This procedure is used to exchange security control messages between the two devices.

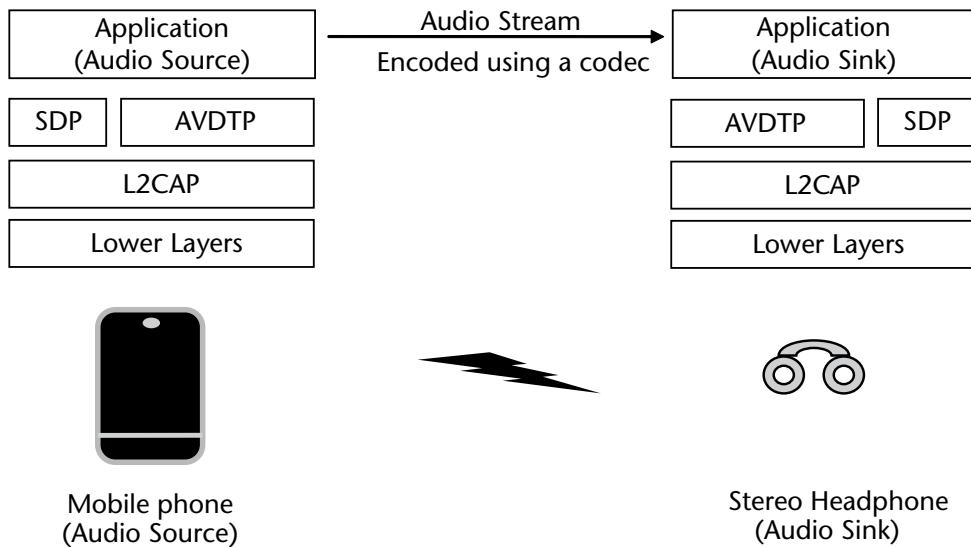


Figure 4.22 Advance Audio Distribution Profile (A2DP).

- **Sink (SNK):** This is the device that receives the audio stream from the SRC and processes it.

Typical use cases of this profile are:

- Play stereo music from a laptop to speakers. In this case the laptop acts as the SRC and the speakers act as the SNK.
- Play stereo music from mobile phone to the Bluetooth enabled music system in the car. In this case the mobile phone acts as the SRC and the Bluetooth enabled music system in the car acts as a SNK.

A2DP does not define point-to-multipoint distribution of audio (note that such cases are still supported by the Bluetooth technology. This can be done, for example, by creating two A2DP connections and routing the same audio on both the connections).

Since raw streaming of audio data requires lot of bandwidth, A2DP uses on-the-fly encoding and decoding of audio data. There are several codecs that could be used to encode and decode the data.

- Sub-Band Codec (SBC);
- MPEG-1,2 Audio;
- MPEG-2,4 AAC;
- ATRAC family;
- Non-A2DP Codecs: This allows the applications to use their own codecs.

Out of these codecs, supporting SBC is mandatory. All other codecs are optional. SBC is a low complexity codec. It needs less computational power compared to the other codecs and delivers good quality compression of audio samples in

real time. It's quite suitable for devices like headphones which may have limited resources like memory and computation power. It supports sampling frequencies from 16 KHz to 48 KHz. 48 KHz sampling frequency is sufficient for CD quality audio.

On the Audio source side, the audio samples coming from the application are encoded with the codec (e.g., SBC) before being given to AVDTP layer for transmission. These encoded samples are sent over the ACL link to the SNK. The AVDTP layer on the SNK side receives those samples. These are decoded and then given to the Audio Sink application. The audio sink application then plays those samples.

4.18 Audio/Video Remote Control Profile (AVRCP)

AVRCP defines the requirements for Bluetooth devices to support use cases related to control of A/V devices. This control can be considered similar to the control provided by a remote control of, let's say, a DVD player.

This profile is dependent on GAP. AVRCP defines the following two roles. These are shown in Figure 4.23:

- Controller (CT): This is the device that initiates a transaction by sending a command to the target.
- Target (TG): This is the device that receives the command, takes the requested action and sends back the response.

Typical use cases of this profile are:

- A headphone sending commands to the mobile phone to pause, play, fast forward, change tracks etc. In this case the headphone acts as the controller and the mobile phone acts as the target.
- A PC sending a command to a DVD player to pause video playback. In this case the PC is the controller and the DVD player is the target.

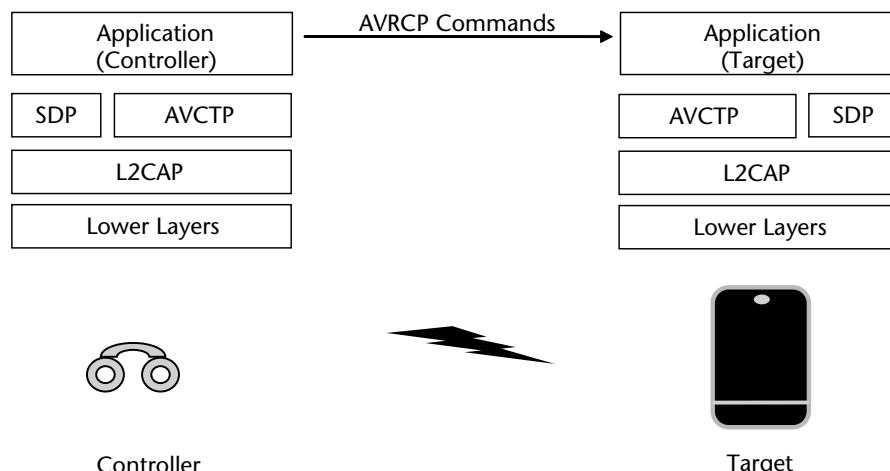


Figure 4.23 Audio/Video Remote Control Profile (AVRCP).

Typical operations that are carried out by devices that support this profile are:

- Retrieving information about the type of units and subunits supported by the device (e.g., Player/Recorder, Monitor/Amplifier, Tuner, etc.);
- Volume up;
- Volume down;
- Channel up;
- Channel down;
- Mute;
- Play;
- Stop;
- Pause;
- Rewind;
- Fast forward.

Not all devices support all operations. Rather the operations that are supported by the device depend on the type of the device and the features it supports.

This profile extensively uses the AV/C command set as defined in the 1394 trade association specification (See Bibliography). (Note: This is another good example where Bluetooth borrows from existing specifications instead of writing the specifications from scratch.)

4.19 Summary

This chapter explained the Bluetooth upper layers and profiles. Wherever possible, the Bluetooth protocols and profiles try to reuse implementations that are already available. These are referred to as adopted protocols. The protocols which are defined from scratch by the Bluetooth SIG are referred to as core protocols.

The profiles provide information on how each of the protocol layers comes together to implement a specific usage model. These define how end-to-end interactions take place between two Bluetooth devices and form the fundamental building block towards ensuring interoperability between devices from various vendors.

The Generic Access Profile (GAP) is a base profile which is mandatory for all devices to implement. A device may implement one or more of the other profiles depending on the end application that the device is intended to support.

Bibliography

Bluetooth Core Specification 4.0 <http://www.bluetooth.org>.

Bluetooth SIG, Specifications of the Bluetooth System, Profiles <http://www.bluetooth.org>.

Bluetooth Assigned Numbers, <https://www.bluetooth.org/assigned-numbers>.

GSM 07.10 version 6.3.0 Release 1997 aka ETSI TS 101 369.

Infrared Data Association, IrDA Object Exchange Protocol (IrOBEX) (<http://www.irda.org>).

Infrared Data Association, IrMC (Ir Mobile Communications) Specification.

IETF RFC3550 / RFC1889 (obsolete) RTP, A Transport Protocol for Real-Time Applications

3GPP 27.007 v6.8.0. <http://www.3gpp.org/ftp/Specs/html-info/27007.htm>.

The Internet Mail Consortium, vCard—The Electronic Business Card Exchange Format, Version 2.1, September 1996.

The Internet Mail Consortium, vCalendar—The Electronic Calendaring and Scheduling Exchange Format, Version 1.0, September 1996.

1394 Trade Association, AV/C Digital Interface Command Set—General Specification, Version 4.0, Document No. 1999026 and AV/C Digital Interface Command Set - General Specification, Version 4.1, Document No. 2001012 (<http://www.1394ta.org>).

1394 Trade Association, AV/C Panel Subunit, Version 1.1, Document No. 2001001 (<http://www.1394ta.org>).

Getting the Hands Wet

5.1 Introduction

After walking through the concepts of Bluetooth, it's a good time to start doing some practical experiments. This chapter will help you to bring up your own Bluetooth development environment in which you can start writing small scripts and programs. A bit of familiarity with the Linux systems, scripting with the shell and the C programming language is assumed.

This chapter will introduce some of the practical usage of the Bluetooth functionality. This will be extended in further chapters to Bluetooth Low Energy. So it's important that you understand the examples provided here and try out a few of them yourself.

This chapter provides examples based on the BlueZ stack. BlueZ is the “official Linux Bluetooth protocol stack.” Support for BlueZ can be found in many Linux distributions. In general it is compatible with any Linux system in the market. It provides support for the core Bluetooth layers and protocols. It is flexible, efficient, and uses a modular implementation. Further details about BlueZ can be found on the BlueZ website (<http://www.bluez.org>).

5.2 Ingredients

You will need the following:

1. A PC running any flavor of Linux. For the purpose of examples, Ubuntu 12.10 is used here, though any other Linux system will serve the purpose provided it's not too old.
2. A Bluetooth Dongle. If you search for Bluetooth dongle, you will find a lot of vendors offering USB based dongles. Any of those should be good. If your PC (or laptop) has an in-built Bluetooth device, then it should also be fine and you don't need an external dongle.
3. Some off-the-shelf devices that support Bluetooth. For example, mobile phone, mono headset, stereo headset, keyboard, mouse, printer. Depending on the devices that you have, you may be able to run different examples provided later in this chapter.

4. Some off-the-shelf devices that support Bluetooth Low Energy. This is the tricky part since only limited LE devices are available in the market as of writing this book. You will find development kits from some of the vendors which could be useful in developing LE applications. For the purpose of examples, a PTS dongle is used as a Bluetooth Low Energy device. This dongle is available for purchase from the Bluetooth SIG website.

5.3 Basic Bluetooth Operations

Before trying the examples provided in this section, you will need to connect the Bluetooth dongle to the PC and boot up Linux. Some of the examples provided here may also need root privileges.

5.3.1 Enabling and Disabling Bluetooth

The first command to try out is hciconfig. This command is used to configure the Bluetooth devices.

To check whether the Bluetooth dongle is properly connected and initialized, run the hciconfig command. It will give output similar to Figure 5.1. If the dongle is properly connected, then this command should show information like:

- BD_ADDR.
- Class of Device.
- Manufacturer name.
- Packet Types supported.
- Number of ACL buffers and buffer size: The screenshot below shows 10 buffers of 310 bytes each.
- Number of SCO buffers and buffer size: The screenshot below shows 8 buffers of 64 bytes each.

```
# hciconfig -a
hc0: Type: BR/EDR Bus: USB
      BD Address: 00:1B:DC:05:B5:B3  ACL MTU: 310:10  SCO MTU: 64:8
      UP RUNNING PSCAN
      RX bytes:1127 acl:0 sco:0 events:39 errors:0
      TX bytes:655 acl:0 sco:0 commands:38 errors:0
      Features: 0xff 0xff 0x8f 0x7e 0xd8 0x1f 0x5b 0x87
      Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
      Link policy: RSWITCH HOLD SNIFF PARK
      Link mode: SLAVE ACCEPT
      Name: 'ubuntu-1'
      Class: 0x6e0100
      Service Classes: Networking, Rendering, Capturing, Audio, Telephony
      Device Class: Computer, Uncategorized
      HCI Version: 4.0 (0x6) Revision: 0x1d86
      LMP Version: 4.0 (0x6) Subversion: 0x1d86
      Manufacturer: Cambridge Silicon Radio (10)
```

Figure 5.1 Hciconfig command.

To close the HCI interface, use the command:

```
hciconfig hci0 down
```

To open and initialize the HCI interface, use the command:

```
hciconfig hci0 up
```

Both these commands need root privileges. In these commands, hci0 indicates the hci interface on which the Bluetooth dongle is attached. It's possible that it's attached on hci1 or some other interface instead of hci0. The parameters of these commands will need to be changed accordingly.

5.3.2 Discovering Devices

There are two commands to discover the devices in the vicinity:

```
hcitool scan  
hcitool inq
```

hcitool inq performs a Bluetooth inquiry and reports information like BD_ADDR, Clock offset and Class of Device.

hcitool scan performs the Bluetooth inquiry as well as gets the Bluetooth Device names for all the devices that are found during inquiry.

The output of these two commands is shown in Figure 5.2.

5.3.3 Browsing Services

The following command can be used to browse the services of the remote devices:

```
sdptool browse [bdaddr]
```

The command queries the SDP server on the device specified by the BD_ADDR and shows the list of services supported.

A partial output of this command is shown in Figure 5.3.

```
#hcitool inq  
Inquiring ...  
68:ED:43:25:0E:99    clock offset: 0x298e        class: 0x7a020c  
00:17:83:DC:72:E9    clock offset: 0x7596        class: 0x1a0114  
  
# hcitool scan  
Scanning ...  
00:17:83:DC:72:E9    WM_nareshg  
68:ED:43:25:0E:99    BlackBerry 8520
```

Figure 5.2 Discovering devices.

```
# sdptool browse 68:ED:43:25:0E:99
Browsing 68:ED:43:25:0E:99 ...
Service Name: Dialup Networking
Service RecHandle: 0x10000
Service Class ID List:
  "Dialup Networking" (0x1103)
  "Generic Networking" (0x1201)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 1
Profile Descriptor List:
  "Dialup Networking" (0x1103)
  Version: 0x0100

Service Name: Voice gateway
Service RecHandle: 0x10001
Service Class ID List:
  "Headset Audio Gateway" (0x1112)
  "Generic Audio" (0x1203)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 2
Profile Descriptor List:
  "Headset" (0x1108)
  Version: 0x0100

Service Name: Hands-free
Service RecHandle: 0x10002
Service Class ID List:
  "Handsfree Audio Gateway" (0x111f)
  "Generic Audio" (0x1203)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 3
Profile Descriptor List:
  "Handsfree" (0x111e)
  Version: 0x0105
```

Service Record of First Service

- Service Name
- Service Record Handle
- Service Class ID List
- Protocol Descriptor List
- Profile Descriptor List

Service Record of Second Service

Figure 5.3 SDP browsing.

5.4 Real World Application—Café Bluebite

This section explains how to use the Bluetooth commands to implement a simple real world application. It is assumed that the reader is familiar with writing simple shell scripts using bash shell.

Let's say you have been requested by Café Bluebite (fictitious name) to create a Bluetooth based advertisement application for them. They are located in a shopping mall and they want to send the deal of the day to anyone who visits the mall. The deal is to be sent on the person's mobile phone via Bluetooth assuming that the person has his Bluetooth switched on when he enters the Mall.

5.4.1 Requirements Specification

The requirements specification provided by Café Bluebite is as follows:

- Advertise the deal of the day to any user who enters the shopping mall.

- Send this advertisement to the person's mobile phone using Bluetooth.

5.4.2 High Level Design

The high level design of the application is provided in the form of a flow chart in Figure 5.4

Some key points to note in the high level design are as follows:

- Step 1c
 - We switch off the discoverable and connectable mode of our own device. This is because we only want to send advertisements messages and we don't want to receive any messages from the remote devices.
- Step 7
 - We want to send messages to only Mobile phones and Tablets. There is no point in sending messages to headsets, keyboards, etc. because they don't have display capabilities.
- Step 11
 - This example uses an endless loop. In practice it will be needed to terminate this loop based on certain conditions.

5.4.3 Code

Before we start writing the code, let's look at the pre-requisites to run this program.

This program will use OBEX to send the advertisements to the remote device. It's possible that the obexftp software is not installed on your Linux system.

To check this you may run the command:

```
# obexftp  
The program 'obexftp' is currently not installed. You can install  
it by typing:  
apt-get install obexftp
```

This indicates the obexftp is not installed on your system. To install it, run the following command:

```
# apt-get install obexftp
```

You will need to be online to download the obexftp package. During installation it will ask for a few confirmations and then install obexftp on your system.

Now let's start writing the code for each of the steps mentioned above.

We will write the code as bash scripts for simplicity. You may either use a programming language or scripts in another shell depending on what you are conversant with.

Step 1: Initialize Bluetooth. The following steps assume that the Bluetooth controller is attached to hci0 interface. If it's attached to another interface, then the name of the correct hci device needs to be put here. This step also disables inquiry

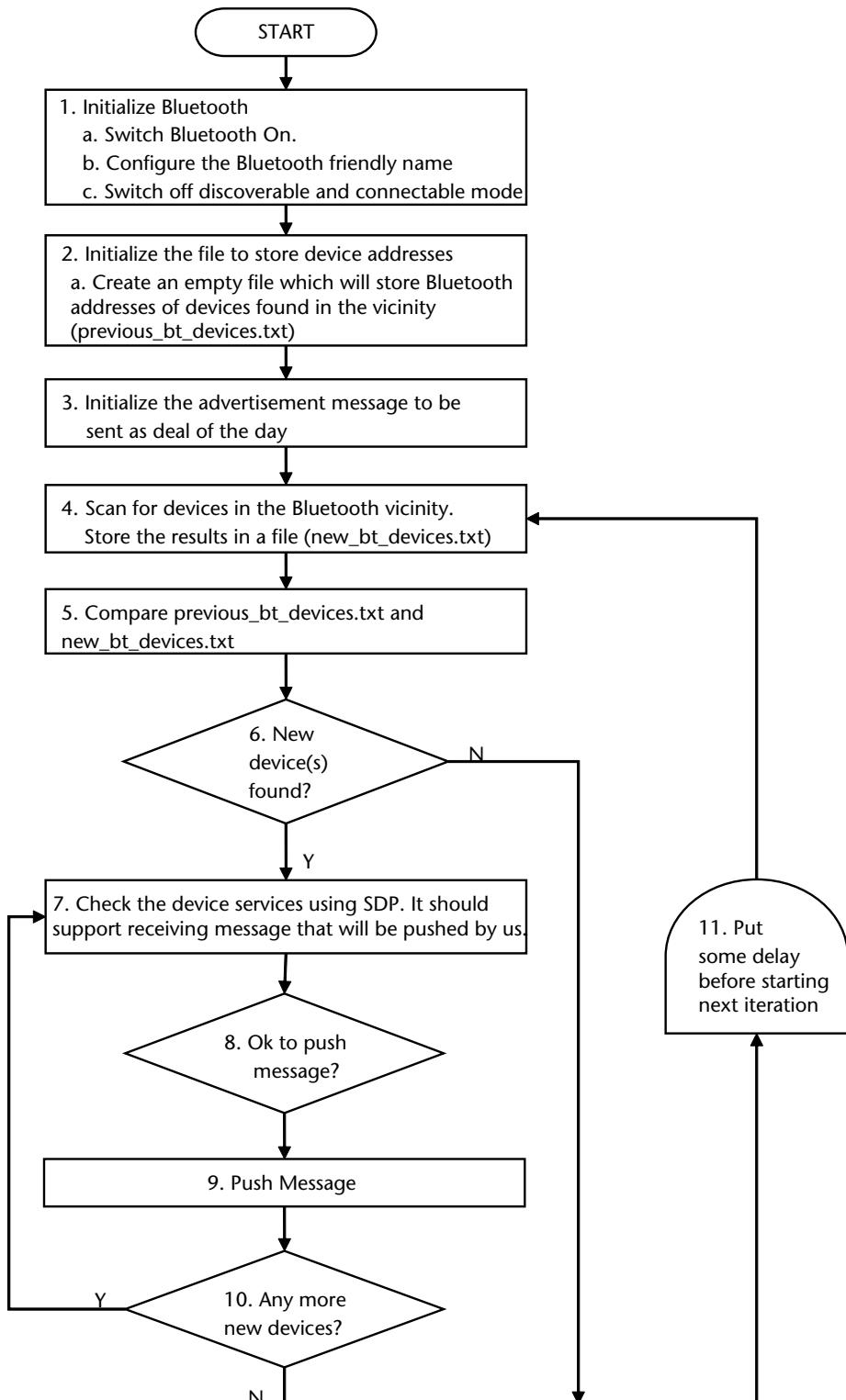


Figure 5.4 High level design—flow chart for Bluebite.

and page scans. After this other devices will not be able to discover our device or connect to our device.

```
echo "Initializing the Bluetooth Controller on hci0..."  
hciconfig hci0 up      # Initialize the Bluetooth controller.  
  
echo "Configuring the name to Cafe Bluebite..."  
hciconfig hci0 name "Cafe Bluebite" # Configure the BT name.  
  
echo "Switching off inquiry and page scans..."  
hciconfig hci0 noscan      # Disable page and inquiry scan.
```

Step 2: Create an empty file to store device names. This step is quite easy. We can just do a simple “> previous_bt_devices.txt” to create such a file.

```
# Create an empty file to store the previous list of devices  
> previous_bt_devices.txt
```

Step 3: Initialize the advertisement message to be sent as deal of the day.

```
# Initialize the advt message to be sent as deal of the day  
echo "Welcome to the shopping mall !!" > advt.txt  
echo "Visit Cafe Bluebite for exciting deals" >> advt.txt  
echo "Get 15% off if you show this message" >> advt.txt
```

Step 4: Scan for devices in the Bluetooth vicinity. Store the results in a file (new_bt_devices.txt). “hcitool inq” performs a Bluetooth inquiry for remote devices in the vicinity and reports the Bluetooth device address, clock offset and Class of Device for each device that is found.

We redirect the output of this command to the file temp_bt_devices.txt. The file temp_bt_devices.txt will contain the list of devices in the following format:

```
#hcitool inq  
Inquiring ...  
68:ED:43:25:0E:99  clock offset: 0x298e  class: 0x7a020c  
00:17:83:DC:72:E9  clock offset: 0x7596  class: 0x1a0114
```

We need only the BD_ADDRs out of this list. So the remaining information can be removed. We will use a combination of the following two commands to remove the remaining information:

tail -n +2: To remove the first line.

cut -c2-12: To keep only columns 2 to 18 which contain BD_ADDR.

```
# Perform an inquiry and store the results in a temporary file.  
hcitool inq > temp_bt_devices.txt  
# Remove extra info from the temporary file  
tail -n +2 temp_bt_devices.txt | cut -c2-18 > new_bt_devices.txt
```

If we use these two commands on the shell prompt, the file new_bt_devices.txt will contain the following.

```
68:ED:43:25:0E:99  
00:17:83:DC:72:E9
```

This file will be used in the following steps for comparing with the previous list of Bluetooth devices stored in previous_bt_devices.txt.

Step 5: Compare previous_bt_devices.txt and new_bt_devices.txt to find out the devices that have got added.

There are several methods to do this with varying level of complexity and simplicity. In the interest of simplicity we choose a very rudimentary method here.

We read each line of the file new_bt_devices.txt and check if that line was present in previous_bt_devices.txt. This comparison is done using the grep command. The grep command returns 0 if the line is found and 1 if it is not found.

One interesting thing in the code below is that we redirect the output of grep command to /dev/null. This is because the output of the grep command is the list of matching lines. Besides that it also sets the exit status (?) to 0 if a match was found and 1 if a match was not found. Since we are only interested in the exit status and not the list of matching lines, we let the list of matching lines go to /dev/null.

```
# Run a loop for all lines in new_bt_devices.txt
for line in $(cat new_bt_devices.txt)
do
    # Check if a match is found in previous_bt_devices.txt
    grep $line previous_bt_devices.txt > /dev/null
    # If a match is not found, then it's a new device
    if [ "$?" -eq "1" ]
    then
        echo "New Device found: $line"

    ##### Some more code will be put here in Step 7 #####
    fi
done
```

Step 7: Check the remote device services using SDP. For simplicity, let's create a function check_sdp that will do this.

check_sdp function will search the SDP records of the remote device to see if it supports the OBEX Object Push service. This service is referred to by SDP as OPUSH.

It first creates two temporary files temp_sdp.txt and temp_sdp1.txt. It then searches for the OPUSH service on the remote device using the sdptool command. If the OPUSH service is found, it finds the RFCOMM Channel number on which the service is present.

```
function check_sdp ()
{
    # Create a blank file to store the results of SDP search
    > temp_sdp.txt
    > temp_sdp1.txt

    # Do an SDP search for OPUSH service on the BD_ADDR
    sdptool search --bdaddr $1 OPUSH > temp_sdp.txt

    # The output is in temp_sdp.txt
    # if the device has OBEX Object Push service then we must
    # be able to find the string OBEX Object Push in the file
```

```

grep "OBEX Object Push" temp_sdp.txt > /dev/null
retval=$?

if [ $retval -eq 0 ]
then
    # Check for the Channel number
    grep "Channel: " temp_sdp.txt > temp_sdp1.txt
    retval=$?

    if [ $retval -eq 0 ]
    then

        # Channel number is in the format
        # Channel: 6
        # So extract the field after colon (:)
        channel_num=`cut -d: -f2 temp_sdp1.txt`
        echo OBEX Object Push service found on Channel Num-
ber: $channel_num
    fi
fi

# grep returns 0 if found. 1 otherwise.
# this is the same that our function has to return.
return $retval
}

```

Step 8: If the check_sdp function returned 0, it means that the device supports OPUSH and we can push the advertisement.

Step 9: Push the advertisement.

```

# Check if the device supports Object Push
echo "Checking Services..."
check_sdp $line

if [ "$?" -eq "0" ]
then
    echo "Device supports OPUSH. Pushing advertisement"
    push_advt $line
else
    echo "Device does not supports OPUSH."
fi

function push_advt ()
{
    echo executing obexftp --bluetooth $1 -B $channel_num -p
advt.txt
    obexftp --bluetooth $1 -B $channel_num -p advt.txt}

```

Step 10: Repeat this for any more devices found. This is already done by the for loop in Step 5.

Step 11: Put some delay before starting next iteration. Before doing this also copy the temp_bt_devices.txt file to previous_bt_devices.txt so that this file can be used for comparison to find whether any new devices came in the vicinity.

```
cp temp_bt_devices.txt previous_bt_devices.txt
sleep 5s
```

5.4.4 Complete Code

The previous section provided code in pieces. The complete code for Bluebite.sh is provided below.

```
#!/bin/bash

#####
# function check_sdp
# Input: BD_ADDR of the device for which SDP search is to be done
# Output: 0 if OPUSH record is found. 1 otherwise
# Synopsis: This function does an SDP Search for OBEX Object Push
#           service on the remote device
#####
function check_sdp ()
{
    # Create a blank file to store the results of SDP search
    > temp_sdp.txt
    > temp_sdp1.txt

    # Do an SDP search for OPUSH service on the BD_ADDR
    sdptool search --bdaddr $1 OPUSH > temp_sdp.txt

    # The output is in temp_sdp.txt
    # if the device has OBEX Object Push service then we must
    # be able to find the string OBEX Object Push in the file
    grep "OBEX Object Push" temp_sdp.txt > /dev/null
    retval=$?

    if [ $retval -eq 0 ]
    then
        # Check for the Channel number
        grep "Channel: " temp_sdp.txt > temp_sdp1.txt
        retval=$?

        if [ $retval -eq 0 ]
        then
            # Channel number is in the format
            # Channel: 6
            # So extract the field after colon (:)
            channel_num=`cut -d: -f2 temp_sdp1.txt`
            echo OBEX Object Push service found on Channel Number:
$channel_num
        fi
    fi

    # grep returns 0 if found. 1 otherwise.
    # this is the same that our function has to return.
    return $retval
```

```
#####
# function push_advt
# Input: BD_ADDR of the device to which advertisement is to be
#        pushed
# Output: None
# Synopsis: This function pushes the advertisement to the remote
#        device
#####
function push_advt (){

    echo executing obexftp --bluetooth $1 -B $channel_num -p
advt.txt
    obexftp --bluetooth $1 -B $channel_num -p advt.txt
}

echo "Initializing the Bluetooth Controller on hci0"
hciconfig hci0 up          # Initialize the Bluetooth controller.

echo "Configuring the name to Cafe Bluebite"
hciconfig hci0 name "Cafe Bluebite" # Configure the BT name.

echo "Disabling Simple Secure Pairing"
hciconfig hci0 sspmode 0

echo "Switching off inquiry and page scans."
hciconfig hci0 noscan      # Disable page and inquiry scan.

# Create an empty file to store the previous list of devices
> previous_bt_devices.txt

# Initialize the advt message to be sent as deal of the day
echo "Welcome to the shopping mall !!" > advt.txt
echo "Visit Cafe Bluebite for exciting deals" >> advt.txt
echo "Get 15% off if you show this message" >> advt.txt

echo "Starting infinite loop. Press Ctrl-C to exit program"

while [ 0 -eq 0 ]
do
    # Perform an inquiry and store the results.
    hcitool inq > temp_bt_devices.txt

    # Remove extra information from the temporary file and
    # store results in new_bt_devices.txt
    tail -n +2 temp_bt_devices.txt | cut -c2-18 > new_bt_devices.
txt

    num_devices=`cat new_bt_devices.txt | wc -l`
    echo Found $num_devices devices in this iteration

    # Run a loop for all lines in new_bt_devices.txt
    for line in $(cat new_bt_devices.txt)
    do
        # Check if a match is found in previous_bt_devices.txt
        grep $line previous_bt_devices.txt > /dev/null

        # If a match is not found, then it's a new device
        if [ "$?" -eq "1" ]
```

```

then
echo "New Device found: $line"

# Check if the device supports Object Push
echo "Checking Services..."
check_sdp $line

if [ "$?" -eq "0" ]
then
    echo "Device supports OPUSH. Pushing advertisement"
    push_advt $line
else
    echo "Device does not supports OPUSH."
fi
fi
done
# Sleep for 5 seconds
cp temp_bt_devices.txt previous_bt_devices.txt
sleep 5s
done # Infinite While loop

```

5.5 Disclaimer

The code provided here is only for educational purposes to illustrate the use of different Bluetooth related commands and features. It has been tested with only a few mobile phones and may not work with all phones or lead to some unknown problems. So you are advised to use it at your own risk. To make the code suitable for commercial use several enhancements, error checks, and exhaustive testing would be needed.

5.6 Summary

This chapter explained the basic requirements of bringing up a setup to try out some Bluetooth operations like enabling and disabling Bluetooth, discovering devices, etc. An example application was also developed to get accustomed to how the various Bluetooth operations can be invoked through simple shell commands.

With this chapter, the background of Bluetooth is completed. The next chapter onwards will focus on Bluetooth Low Energy.

Bibliography

Bluetooth Core Specification 4.0 <http://www.bluetooth.org>.

Bluetooth SIG, Specifications of the Bluetooth System, Profiles <http://www.bluetooth.org>.

BlueZ website (<http://www.bluez.org>).

Bluetooth Low Energy—Fundamentals

6.1 Introduction

As described in Chapter 1, Bluetooth Low Energy is the next major evolution of the Bluetooth technology. It specifies requirements for devices to have ultra low power consumption. This is a radical change from the direction in which the technology was evolving through previous versions. While the focus of previous versions was either feature enhancements or increase in the throughput, LE focused in an entirely new direction—how to cut down the power consumption drastically? LE technology is fully optimized from the ground up to ensure that the power consumption is kept to a minimum. This meant a complete redesign of several key components to ensure that all steps are taken to reduce the power requirements.

In general Bluetooth devices are battery powered. It's expected that the LE devices may have smaller batteries like the coin cell batteries (or even smaller ones). This technology focuses on reducing both the peak current and the average current. A reduced average current ensures that the battery drains down slowly. A reduced peak current means that the devices can continue operating even when the battery has started running down and the maximum current that the battery can provide has been reduced.

Some of the uses for LE were shown in Chapter 1. These included finding devices, alerting devices, proximity detection, sensors, healthcare, sports and fitness equipment, mobile payments, etc. It may be noted that none of those use cases focused on high throughput or transferring big chunks of data. Rather the use cases focused on transferring very short pieces of information that could be transferred only when needed, which may generally not be very frequent.

Typical LE use cases would include creating a connection, transferring a few bytes or kilobytes of data and then disconnecting. The connection time is so low that it's easier to reestablish the connection every time a data transfer is needed without any impact on the user experience. This is in contrast to BR/EDR use cases like a Hands-free connection that is maintained for a long time to ensure that there is least latency when there is an incoming call, or use cases like the exchange of big file in the case of FTP profile.

There are two broad classifications of Bluetooth systems.

- The first is the classic Bluetooth system that conforms to versions prior to 4.0 of the Bluetooth specification. It is also referred to as BR/EDR. BR stands for Basic Rate indicating that the device can support up to a maximum data rate of 721 kbps. EDR stands for Enhanced Data Rate indicating that the device can support up to a maximum data rate of 2.1 Mbps.
- The second is the LE system which conforms to 4.0 version of the Bluetooth specification and supports enhancements for ultra low power. These systems have lower complexity and lower cost compared to BR/EDR systems. The throughput is significantly lower. The maximum throughout is about 305 kbps though devices generally don't need or use such high rate data transfers.

A device can support only BR/EDR, only LE, or both BR/EDR and LE. A device which supports both BR/EDR and LE is also referred to as BR/EDR/LE or dual mode device. This will be explained in further detail later in this chapter.

The architecture of BR/EDR devices was covered in detail in previous chapters. This chapter and further chapters will focus on the architecture of LE devices.

6.2 Single Mode versus Dual Mode Devices

Depending on the functionality supported, the Bluetooth devices may be categorized into 3 types:

1. *BR/EDR Devices*: These are the classic Basic Rate/Enhanced Data rate devices which do not support the LE functionality.
2. *LE Only Devices or Single Mode LE Devices*: These devices support only LE functionality. Examples of these devices include watches, key fobs, heart rate monitors, thermometers, sports and fitness equipment, sensors, etc. These devices are expected to have ultra low power consumption and last for several months or years on coin cell batteries.
3. *BR/EDR/LE or Dual Mode Devices*: These devices support both BR/EDR and LE functionality. Typically these devices are smartphones, tablets, PCs, etc. These devices are expected to communicate with both the BR/EDR devices and single mode LE devices even at the same time. These devices don't have as stringent requirements on power consumption as the single mode LE devices since these have bigger batteries or are generally recharged frequently.

The use cases of LE only devices were explained in Chapter 1. Some of the possible use cases of Dual Mode devices are as follows:

1. A person is listening to music or taking a call on the Bluetooth headset and the child (or the pet) goes out of range. In this case the user would be alerted about the child (or the pet) going out of range by the proximity detection feature of LE so that the person can take immediate action.

2. A person walks into the home while on a call. The presence detection function of LE may automatically switch on the lights or switch on the air conditioning.
3. A person is jogging on a treadmill and wants to listen to music on the Bluetooth headset and at the same time monitor his/her heart rate, number of steps he/she has run, etc.
4. A person using a GPS enabled smartphone while cycling wants to see distance covered on the track and parameters like current heart rate, maximum heart rate, average heart rate, calories burnt, etc. In addition to displaying on the smartphone, the same data could be sent to the laptop or fitness center for further analysis using the smartphone as a gateway.

Table 6.1 provides information on compatibility between different categories of devices.

Some examples of communication between the various devices types are provided below:

1. BR/EDR to BR/EDR—Classic Bluetooth communications between:
 - a. Mobile phone to headset.
 - b. Laptop to laptop.
 - c. PC to printer.
2. Single Mode LE to Single Mode LE—Low energy ecosystem
 - a. Sports sensor displaying data on a watch.
3. Single Mode LE to Dual Mode
 - a. A laptop sending an alert to a key fob.
 - b. A heart rate monitor sending data to the hospital's computer using smartphone as the gateway.

6.3 Bluetooth Smart Marks

The Bluetooth smart marks were created to help consumers ensure compatibility among their Bluetooth devices. There are two trademarks from the Bluetooth SIG:

- Bluetooth Smart;
- Bluetooth Smart Ready.

Table 6.1 Compatibility Between Single Mode and Dual Mode Devices

	<i>BR/EDR</i>	<i>Single Mode LE</i>	<i>Dual Mode</i>
<i>BR/EDR</i>	Yes	No	Yes
<i>Single Mode LE</i>	No	Yes	Yes
<i>Dual Mode</i>	Yes	Yes	Yes

6.3.1 Bluetooth Smart (Sensor-Type Devices)

Bluetooth Smart devices are sensor-type devices that are used to collect a specific piece of information. After collecting this information, these devices send it to the Bluetooth Smart Ready devices. Bluetooth Smart devices include only a single mode LE radio and are expected to consume ultra low power.

Some examples of Bluetooth Smart devices are heart rate monitors, thermometers, sports equipment, etc. These devices collect a specific piece of information like heart rate or temperature and then relay it to the Bluetooth Smart Ready devices.

Bluetooth Smart trademark was developed to brand qualified devices as meeting the following three requirements:

1. Conform to Bluetooth 4.0 or higher with GATT based architecture.
2. Contain Single mode LE radio.
3. Use GATT-based architecture to enable a particular functionality (GATT-based architecture will be explained in detail in subsequent sections).

6.3.2 Bluetooth Smart Ready (Hubs)

Bluetooth Smart Ready devices are the devices that receive data sent from the classic Bluetooth and Bluetooth Smart devices and give it to applications that make use of that data. The applications could be running on these devices themselves or could be running anywhere else on the internet. These devices implement the dual mode radio and can connect to the BR/EDR devices as well as the Bluetooth Smart devices. Such devices have one single Bluetooth device address which is used for both the BR/EDR and LE radios. Some examples are phones, tablets, PCs etc.

Bluetooth Smart Ready mark was developed to brand qualified devices as meeting the following three requirements:

1. Conform to Bluetooth 4.0 or higher with GATT based architecture.
2. Contain Dual mode radio.
3. Provide a means by which the end user can choose to update the functionality for a Bluetooth Smart device on a Bluetooth Smart Ready device. For example if the user buys a new Bluetooth Smart device then new software can be installed on the smart phone to communicate to that device.

Figure 6.1 shows Bluetooth, Bluetooth Smart and Bluetooth Smart Ready devices. In this scenario, the mobile phone is the Smart Ready device and it is communicating to two devices at the same time:

1. Streaming audio data to a Bluetooth headset.
2. Collecting temperature information from a Bluetooth Smart thermometer and acting as a hub to relay that information to a server located in the hospital. The server can then take the appropriate action like informing the doctor or pharmacist.

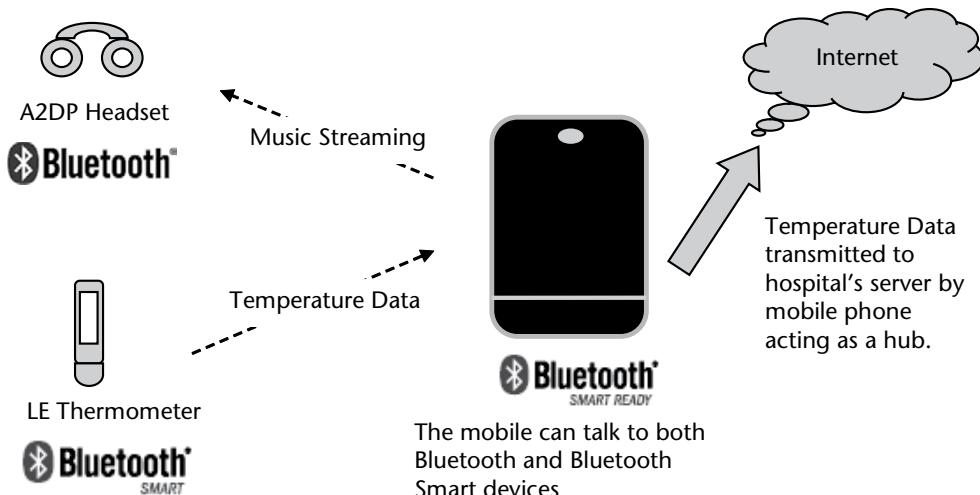


Figure 6.1 Bluetooth, Bluetooth Smart and Bluetooth Smart Ready devices.

6.4 LE Fundamentals

The devices which are based on BR/EDR require a recharge in a few days or few weeks and are generally using larger batteries than coin cell batteries. Take, for example, a Bluetooth keyboard, mouse, headset, etc. All have relatively large batteries and need a recharge every few days or weeks.

Achieving a power consumption of several months to several years with LE was not as easy as optimizing the power consumption of various layers in the Bluetooth architecture. That would not have led to drastic reduction in the power consumption. So LE has been designed almost from scratch to ensure that all possibilities to achieve ultra low power consumption have been incorporated. It is designed ground up for simplicity, low cost and ultra-low power consumption without compromising robustness, security, global usage, or ease of use. Most importantly the compatibility of dual mode devices with the existing BR/EDR devices has been preserved to ensure that nothing gets broken when manufacturers upgrade their existing devices to BT 4.0-based devices.

There are several enhancements done in BT 4.0 specification to achieve low power. Some of the fundamental concepts related to LE operation are introduced below. These will be explained at length in the following chapters.

6.4.1 Frequency Bands

Similar to the BR/EDR radio, the LE radio operates in the 2.4 GHz ISM band. This band is globally license free and is shared by several other wireless technologies. LE also uses a frequency hopping mechanism to combat interference. (Frequency hopping was explained in Chapter 3).

One important difference between BR/EDR and LE is that while BR/EDR uses 79 channels for frequency hopping, LE uses only 40 channels. Secondly there are

dedicated channels for advertising and sending data in the case of LE. This will be explained in detail later in Chapter 8.

6.4.2 Mostly Off Technology

LE can be termed as a “mostly off” technology. This means that the LE devices are expected to be sending data only occasionally and be in a switched-off state for the remaining time. For example, the heart rate monitor may collect all data and then send it across once per hour or once per day, the weighing machine may send the weight only once per day or once per week, or a temperature sensor may send the data only if the temperature crosses a certain threshold limit.

LE technology is designed in such a manner that the LE devices remain off most of the time and switch on only when they need to transmit some data. This ensures that the duty cycle (ratio of device off to device on) is almost close to zero and the device will only switch on when some specific conditions are triggered. Normally the device would just remain off.

6.4.3 Faster Connections

LE takes much less time to create connections as compared to BR/EDR. This is because LE uses only 3 dedicated advertising channels which can be used for creating connections. This is in contrast to BR/EDR where 32 channels are used for inquiry and paging (connection).

Since BR/EDR has more channels, the device takes more time in scanning across all channels before a connection can be created. A typical BR/EDR connection can take up to 20 milliseconds while in the case of LE the connection time is less than 3 milliseconds because the device has to scan on only 3 channels.

A faster connection means that whenever the device needs to send the data, it can quickly connect, transmit data, and then disconnect. The total time for this transaction may be in the range of only 3 to 4 milliseconds. This means that the LE radio needs to be switched on for a very short time. The shorter the time that the radio is switched on the lower the power consumption. The device may switch itself off until it needs to transmit data again.

This is in contrast to BR/EDR. For example, in the case of a headset that is connected to a mobile phone, the headset may remain connected to mobile phone for several days. During this time the headset would be in sniff mode. This would mean that it would be waking up periodically to see if the mobile phone has data to send. This would translate into continuous drain of battery since the headset would wake up periodically and check to see if the mobile phone has any data to send.

6.4.4 Reduced Functionality

LE incorporates some major reductions in the functionality so that it can target a specific market—the one which needs devices to be consuming ultra low power. So it cuts down heavily on the functionality to reduce the memory required to implement that functionality.

Some of the major reductions are:

1. Not mandatory to implement both transmitter and receiver—LE is designed in such a manner that a device can implement only transmitter, only receiver or both transmitter and receiver. For example an LE weighing machine may implement only the transmitter. As soon as it has measured the weight, it just transmits and does not need to receive anything. Such devices would reduce the silicon area to almost half as compared to devices which implement both transmitter and receiver.
2. No support for voice channels—LE is intended for devices that send short amount of data infrequently. It is not intended for devices like headsets which need to transfer continuous stream of voice data. So the SCO/eSCO functionality has been completely removed.
3. No support for Scatternet—LE provides support for only a piconet and the support for scatternet has been removed. This has simplified the state machine of the link layer since a device can only be in one piconet at a particular time. While BR/EDR supports only up to 7 devices in a piconet, an LE piconet can have any number of devices, limited only by the resources available on the piconet master device. So removal of scatternet functionality in LE did not put any restriction on the number of devices that could be connected together while it also simplified the link layer state machine.
4. No support for Master/Slave role switch—BR/EDR allowed the possibility to switch the roles of Master and Slave. In LE, once a connection is made, the role switch is not permissible. It is also clearly defined right in the beginning which device will end up being a Master and which device will end up being a Slave. This led to a lot of simplification of the link layer state machine.
5. No need for continuous polling of the link. In the case of BR/EDR, even if there is no data exchange, POLL/NULL packets are continuously exchanged to check if the remote device is still present, leading to consumption of power. In the case of LE, there is no need to continuously monitor the link. Devices can just shut down the link and recreate it whenever needed without any impact on the user experience. This is because the link setup time is far less in case of LE as compared to BR/EDR.
6. No support for sniff and park modes. The LE controllers are designed to be very simple and power efficient from ground up. The connection is created for a very short duration—only when the data is transferred, and then disconnected. So there is no need for separate power saving modes. It can be said that the default mode for LE is already power savings mode and so no additional power savings modes are defined.

6.4.5 Shorter Packets

LE uses packets of much shorter size as compared to BR/EDR which means that the time needed to transmit or receive them is lesser. So the radio will be switched on for a lesser time leading to savings in power consumption.

Besides this the buffer space needed to store the packets is much lesser. The maximum size of LE packet is 27 bytes which is much shorter than the maximum size of 1021 bytes supported by BR/EDR (As discussed in Chapter 3, the maximum size of 3DH5 packets is 1021 bytes).

Besides lesser times, shorter packets also require much lesser power to transmit. This is because of radio characteristics. If a long packet has to be transmitted, the radio needs to be in a high power state for a longer period of time, resulting in the heating up of silicon. This changes the material's physical characteristics and deviation in the transmission frequency which could result in a packet loss or even link loss. To counter this, the radio needs to be constantly recalibrated in order to ensure that the transmission frequency is correct. Recalibration logic makes the radio more complex and also requires power for the recalibration. In comparison shorter packets ensure that the silicon stays cool and the characteristics don't change. Hence no recalibration logic is needed.

6.4.6 Reduced Dynamic Memory Footprint

Another design principle used in LE is to optimize the usage of dynamic memory as much as possible. This is because firstly memory requires silicon area which contributes to the costs and secondly dynamic memory requires constant supply of current to retain the contents that are stored in it. So it adds to the current consumption of the device. (Dynamic memories typically need to be refreshed continuously so that they do not lose the contents. In contrast ROM and FLASH memories do not need to be refreshed).

To reduce the dynamic requirements on memory, LE incorporates the following:

1. Shorter packets—The amount of buffer memory space needed for storing packets is much smaller in case of LE due to shorter packet sizes. This was explained in the previous section.
2. Shorter headers—LE uses only a 32-bit access code to reduce the overall size of the packet. This again leads to less time to transmit the packet and lesser buffer requirement.
3. Simple Protocol—The LE protocol has been designed to be very simple so that there is least state information to be stored. For example the Link Manager has only 5 states and a very limited number of state transitions. Similarly the L2CAP layer has also been simplified a lot to support a very limited number of CIDs (Channel Identifiers) and signaling commands. These will be explained in detail in the subsequent chapters.
4. Uniform Packet Format—LE uses only one packet format for all types of packets. This makes the logic for creating packets on the transmitter side and parsing packets on the receiver side much simpler thereby reducing the code that has to be implemented.

6.4.7 Optimized Power Consumption of Peripherals

LE is designed in such a manner that the peripherals consume the least amount of power while the central devices could consume a bit more power to compensate. This is because generally the peripherals are resource constrained. They have smaller batteries, memory and limited processing power. In contrast, the central devices may have much higher processing power and much bigger batteries. These may even be powered from mains (For example a TV, PC, or a set-top box) or may be recharged frequently (For example a mobile phone or a tablet).

As an example, if a weighing machine gets connected to a mobile phone to send data, then it's more important to optimize the power consumption of the weighing machine as compared to optimizing the power consumption of the mobile phone. The mobile phone may be charged every day or every alternate day while the weighing machine's battery may be replaced only once in a few months or years.

The LE protocol is designed in such a manner that the peripheral needs to be powered on for as less time as possible while the central device may be continuously powered on to wait for any data coming from the peripheral. So in the case of weighing machine transmitting data to the mobile phone, the mobile phone may always be scanning for packets while the weighing machine may just transmit packets when it has some information to transmit. So the weighing machine will be powered on for far lesser time as compared to the mobile phone.

Another important point is that, in the wireless radio world, transmitting packets consumes much less power than receiving packets. So LE tries to reduce the time for which the peripherals are receiving packets. Instead of this the LE protocol is designed in such a manner that the peripherals mostly transmit packets instead of scanning and receive only for very short durations, if they really have to. So in the case of weighing machine example, the weighing machine would transmit (advertise) to consume less power while the mobile phone would receive (scan). The weighing machine, may in fact never receive anything and may not even have a receiver in it.

6.4.8 No Need for Continuous Polling

In the case of LE, for devices to remain connected, there is no need for continuous exchange of packets as long as the link layer is synchronized to the timing, frequency and access address parameters.

This is in contrast of BR/EDR where, to remain connected, continuous POLL/NULL packets are exchanged even if there is no data to send. BR/EDR does provide the possibility to put the connection into low power mode to reduce the exchange of packets but packets are still exchanged leading to power consumption.

6.4.9 Backward Compatibility with BR/EDR

There are billions of BR/EDR device already in use. Mobile phones, tablets and laptops have an attach rate of almost 100%. LE technology is designed in such a manner that dual mode devices which are based on the BT 4.0 specification are backward compatible with the existing devices. This means that the next generation

of mobile phones, laptops and other devices can be upgraded to BT 4.0 devices without breaking existing compatibility. Once these devices get upgraded to BT 4.0, they will also be able to talk to LE devices in addition to BR/EDR. So LE builds on to the ecosystem that has already been established by BR/EDR and paves the way for next millions and billions of devices.

A summary of the key LE features is shown in Table 6.2.

6.5 LE Architecture

LE has a layered architecture just like BR/EDR. The architecture of LE is shown in Figure 6.2. In many ways it's similar to the architecture of classic BR/EDR stack. It also uses the concept of protocols and profiles. (This was explained in Chapter 2).

The design of profiles is quite simplified in the case of LE. LE introduced the concept of GATT based profiles. Most of the common functionality that is needed by all profiles is moved into the ATT protocol and GATT profile. The profiles on top of GATT use the services that are provided by GATT and only implement the bare minimum things that are needed to support that specific use case.

In the next chapters, each of the protocol layers will be covered in details. The mechanisms that are employed by each of the protocol layers to reduce the power consumption will also be pointed out specifically so that the reader is able to appreciate the reasons why this technology is designed for ultra-low power operations.

Table 6.2 Summary of Key LE Features

<i>Connection Type</i>	<i>Frequency Hopping Spread Spectrum.</i>
Spectrum	2.4 GHz ISM Band. Regulatory range: 2400 – 2483.5 MHz.
Frequency Hopping	Across 40 RF channels. The channels are separated by 2 MHz.
Modulation	Gaussian Frequency Shift Keying (GFSK).
Maximum Data Rate	305 kbps
Maximum Data Packet size	27 bytes
Typical Range	30 m to 100 m.
Topology	Master Slave architecture. The number of slaves is limited only by the availability of resources on the master.
Connection Time	In the range of 2.5 milliseconds. LE supports a much lower connection time as compared to BR/EDR. So it's easier to just re-establish the connection and transfer data in case of LE instead of keeping the connection alive.
Data Security: Authentication Key	AES 128 bit key.
Data Security: Encryption Key	AES-128 (Stronger than BR/EDR)
Voice Channels	Not supported.
Applicability	Does not require line of sight. Intended to work anywhere in the world since it uses unlicensed ISM band.

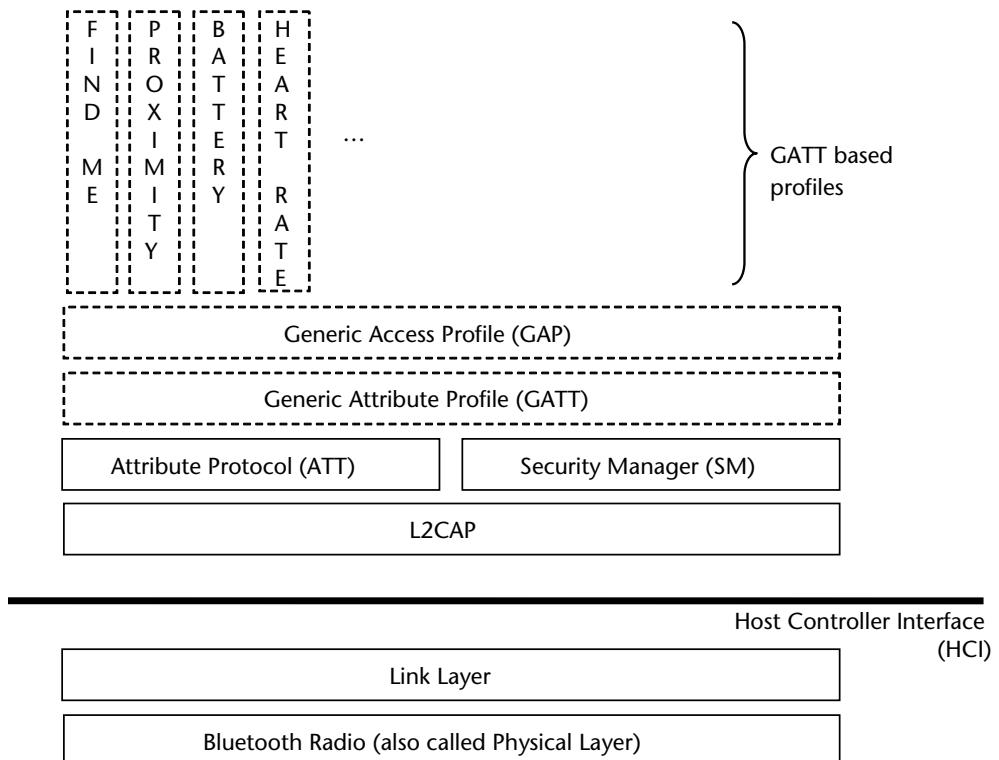


Figure 6.2 LE architecture.

6.6 Comparison between BR/EDR and LE

The architectural comparison of the BR/EDR and LE protocol stack is shown in Figure 6.3. The LE stack modifies some of the existing protocol layers and profiles like the L2CAP layer and the GAP profile. The LE radio has also been modified as explained above. LE also replaces some layers entirely to gain significant power savings. For example the link layer is defined from scratch.

In the case of dual mode devices, the implementation of some of the layers can be shared. For example a combined LE + BR/EDR radio can be used and the implementation of the L2CAP layer, HCI and GAP profile can be shared for the dual mode devices. This helps in maintaining backward compatibility as well as re-using the efforts that were spent already in developing code for these layers.

A broad level comparison between BR/EDR and LE is shown in Table 6.3. There are several other differences besides these and those differences will be highlighted in the next chapters at the appropriate places.

6.7 Summary

LE technology offers dual benefits. First, it builds on to the existing ecosystem of Bluetooth devices, and secondly, it offers a drastic reduction in the power

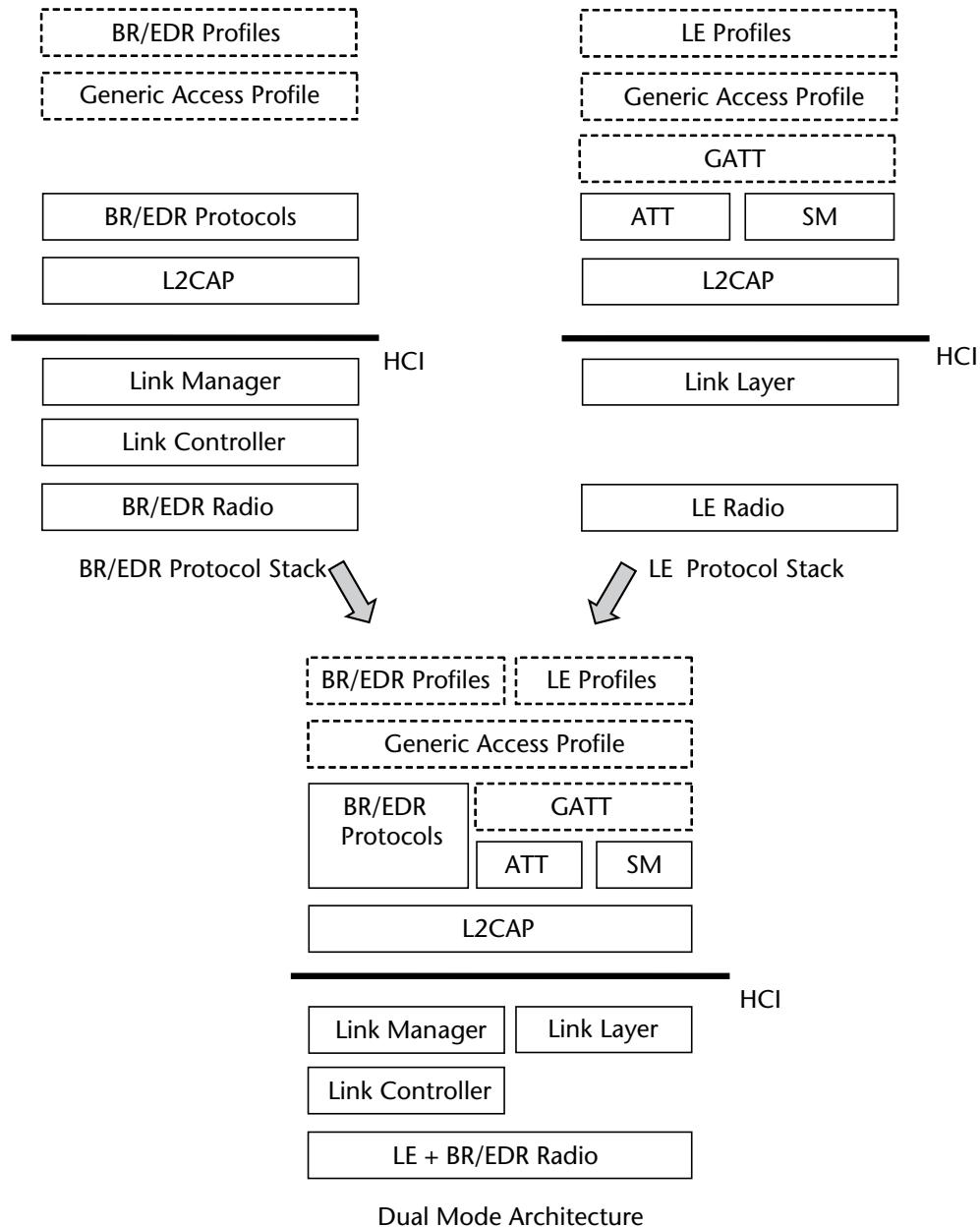


Figure 6.3 BR/EDR protocol stack, LE protocol stack and Dual Mode Architecture.

consumption for LE only devices. This has opened up several new use cases where this technology can be effectively deployed.

The architecture of LE was introduced in this chapter. Besides this some of the enhancements done by LE to reduce power consumption were briefly introduced. These will be explained in depth in next chapters.

Table 6.3 Comparison of BR/EDR and LE

Feature	BR/EDR	LE	Remarks
RF Channels and spacing	79 channels spaced by 1 MHz each	40 channels spaced by 2 MHz each	
Dedicated RF Channels	No dedicated RF channels. All are used for data.	3 channels dedicated for advertising. 37 channels dedicated for data.	
Range	Typically 10m to 30m. Can go to max of 100 m	Typically 30m to 50m. Can go to a max of 100m.	Typically LE based devices have a longer range
Connection time	In the range of 20 ms	Less than 3 ms	Much less time required to make a connection.
Maximum Packet Size	1021 bytes	27 bytes	LE uses much smaller packets.
Maximum Data Rate	BR: 721 kbps EDR: 2.1 Mbps	305 kb/s	LE supports much lower data rates. Even these data rates are seldom used.
Scatternet Support	Yes	No	LE simplifies implementation complexity by disallowing scatternets.
Master/Slave Role Switch	Supported	Not Supported	Simpler state machine.
Transmitter/Receiver	Both transmitter and receiver are mandatory	Device can have only transmitter, only receiver or both	LE saves a lot of silicon area if the device has to only transmit or only receive.
PDU Format	Several	One	LE has only one PDU format. This makes it simpler to create and parse packets
CRC Strength	16-bit	24-bit	Much more robust, especially in noise environments.
Voice Channels	Yes	No	No support for voice channels in LE.

Bibliography

Bluetooth Core Specification 4.0 <http://www.bluetooth.org>.

Bluetooth SIG, Specifications of the Bluetooth System, Profiles <http://www.bluetooth.org>.

SIG Low Energy Training, <http://www.bluetooth.org>.

Bluetooth Assigned Numbers, <https://www.bluetooth.org/assigned-numbers>.

Physical Layer

7.1 Introduction

This chapter explains the operation of the physical layer of LE. As shown in Figure 7.1, this is the bottom most layer in the protocol stack. It is responsible for sending and receiving data over the air.

7.2 Frequency Bands

Like BR/EDR, the LE radio also operates in the 2.4 GHz ISM band (ISM stands for Industrial, Scientific and Medical). This frequency band is globally unlicensed and is used by several other devices like remote control toys, cordless telephones, NFC, Wireless LAN, etc. Some microwave ovens also generate interference in this frequency band.

Since this band may be shared by multiple devices, there is a good possibility that there may be interference from the other devices. So, the frequency is continuously changed for subsequent transmissions so that a new frequency is chosen for exchanging the data. The pattern of changing the frequencies is predefined so that the devices that need to exchange data know which frequency to hop to for the next data exchange. This is known as frequency hopping. LE uses frequency hopping technique to combat interference and fading.

The frequency band is divided into 40 channels which are spaced 2 MHz apart. (In contrast BR/EDR uses 79 channels which are spaced 1 MHz apart). The channels are numbered from 0 to 39 starting at 2402 MHz. The frequencies of various channels are derived from the formula:

$$f(k) = 2,402 + k * 2 \text{ MHz}, k = 0, \dots, 39$$

The complete range used by LE is 2.400 to 2.4835 GHz. The frequency bands used by LE are shown in Figure 7.2.

Figure 7.3 shows the sniffer capture of transactions between a mobile phone and an LE device. It shows the transmissions on the 40 channels which include:

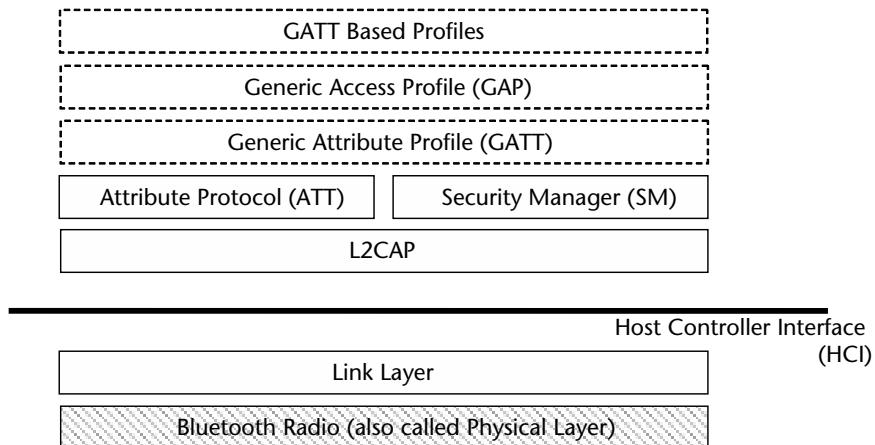


Figure 7.1 Bluetooth Radio in LE Protocol Stack.

RF Channels: 40 Channels with 2 MHz spacing between channels

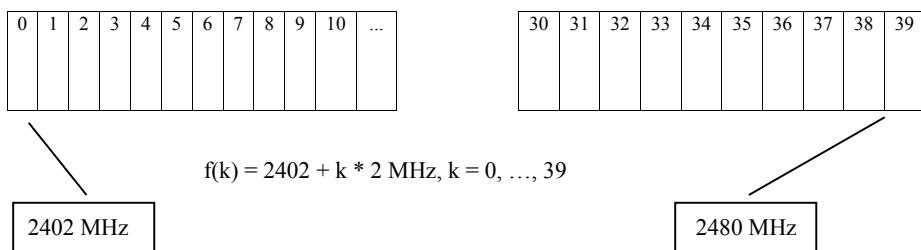


Figure 7.2 RF Channels used by LE.

1. Data transmissions on channel 0 to 36.
2. Advertisements on channels 37, 38, and 39.

As seen in the figure, the channels are spaced 2 MHz apart.

7.3 Transmitter Only, Receiver Only, or Both

An LE radio may have only a transmitter, only a receiver or both. This is in contrast to BR/EDR specification where it's mandatory for the BR/EDR radio to have both a transmitter and a receiver. This helps in reducing the cost and simplifying the design of devices which only need to transmit or receive data.

An example of this could be an LE based TV remote control. The remote control may need to only transmit the commands based on which buttons the user presses and not receive anything back from the TV. As per the LE specification, such a device is allowed to have only a transmitter.

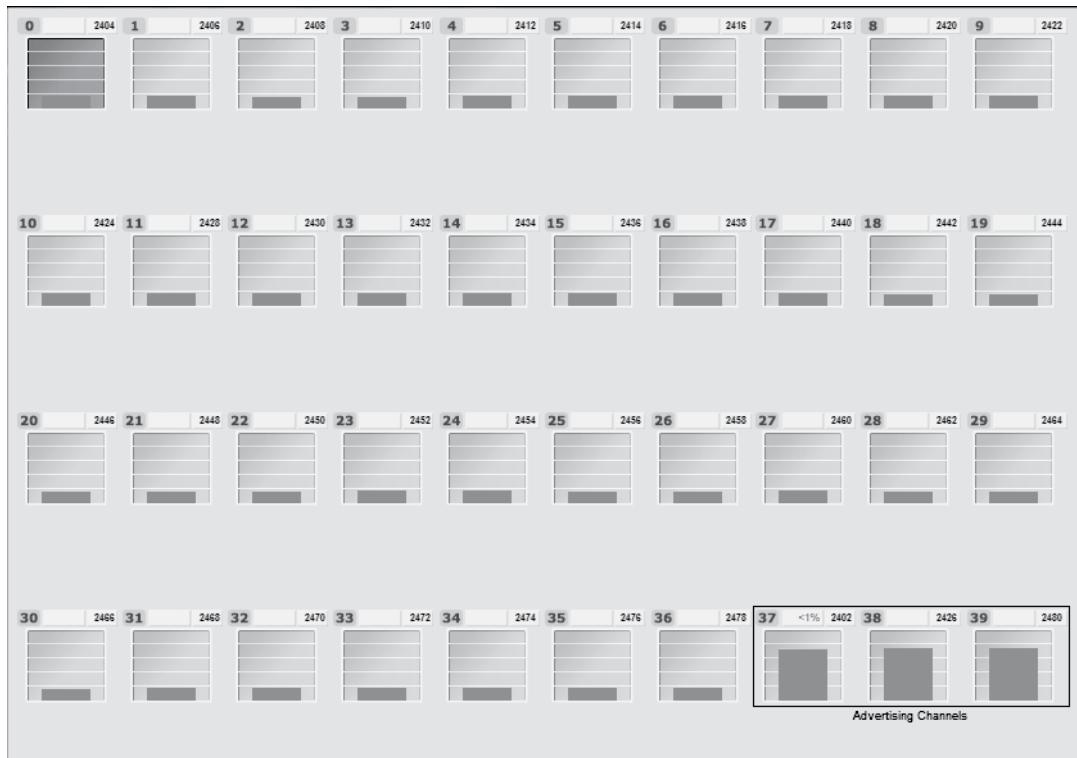


Figure 7.3 Sniffer capture of transmissions on LE channels.

7.4 Output Power

LE defines the transmitter output power to be in the range of 0.01 mW (-20 dBm) to 10 mW (+10 dBm). The device can change the output power dynamically to optimize the power consumption and reduce the interference on other equipment.

7.5 Range

Based on the output power defined in the previous section, LE devices support a range of about 30m to 100m.

7.6 Modulation Characteristics

LE uses Gaussian Frequency Shift Keying (GFSK) mechanism.

The bandwidth-bit period product BT=0.5 and the modulation index is between 0.45 to 0.55. A binary one is represented by a positive frequency deviation and a zero is represented by a negative frequency deviation.

The reference signal is defined in Table 7.1.

Some of these terms are beyond the scope of this book and are provided here just for reference. The details can be looked up in the Bluetooth specifications.

Table 7.1 Reference Signal for LE

<i>Modulation</i>	<i>GFSK (Gaussian Frequency Shift Keying)</i>
Modulation Index	0.5 +/- 1%
Bandwidth Bit period Product, BT	0.5 +/- 1%
Bit Rate	1 Mbps +/- 1 ppm
Modulation data for wanted signal	PRBS9
Modulation data for interference signal	PRBS15
Frequency accuracy better than	+/- 1 ppm

What is GFSK modulation?

Modulation is the process of mixing one signal with another signal. The signal that contains the information to be transmitted is called the *modulating signal*. It is mixed with a high-frequency signal called the *carrier signal*. Generally the frequency of the carrier signal is much higher than the modulating signals. For example the carrier signal would be in the range of GHz (2.4 GHz for Bluetooth) and the modulating signal would be in the range of MHz.

The three key properties of the carrier signal are: amplitude, phase, and frequency. Any of these can be modified during the process of mixing with the carrier signal. The resulting modulated signal is then transmitted to the remote side where it is demodulated to reconstruct the original signal.

For example, when a binary number (string of 0s and 1s) is to be transmitted over the air, it is mixed with a carrier signal, and then transmitted over the air. If the 0s and 1s were mixed to the amplitude then one level of amplitude would represent a 0 and another level of amplitude would represent a 1 after modulation. This modulated signal is now in a state that can be transmitted over the air. At the receiver side, the receiver would interpret the different amplitude levels as 0 and 1 and reconstruct the binary number.

Frequency Shift Keying (FSK) conveys information by varying the carrier signal frequency to represent a 0 or a 1. So a binary 1 can be represented by increasing the frequency of the carrier signal and 0 can be represented by decreasing the frequency of the carrier signal. One of the reasons for using frequency modulation to encode data compared to amplitude modulation is that generally the noise signals change the amplitude of a signal. So, modulation signals which ignore the amplitude of the signal are relatively more immune to noise.

Gaussian Frequency Shift Keying (GFSK) applies Gaussian filter to the modulating signal before it is mixed with the carrier signal. The Gaussian filter smoothens the shape of the frequency pulse so that high frequencies at the time of switching can be avoided. This helps to reduce the spectral width of the signal and is also called *pulse shaping*.

7.7 LE Timeline

Figure 7.4 shows a typical timeline of transactions happening on the air for LE. The air captures have been taken for one of the GATT profiles where initially the LE device is advertising. Then later on a dual mode device tries to discover the services and creates a connection to this device.

Some of the interesting points to note are:

1. The advertising packets are generally seen in sets of 3 packets. This is because the devices generally advertise consecutively on the three advertisement channels and then wait for some time to restart the advertisement.
2. Most of the space is empty space. This means that for most of the time, there are no transmissions happening. This confirms the statement in the previous chapter that LE is a “mostly off” technology.
3. The average throughput is very low. For this particular capture, this is in the range of 3 kbps. This is much lower than the average throughput seen in classic Bluetooth systems.
4. There could be some instances where lot of data is exchanged. During that time the throughput may go up, but even then it is peaking to a maximum of 17 kbps in this particular example. This is still quite low.

7.8 Summary

This chapter explained the physical layer which is the bottom layer of the LE protocol stack and is responsible for sending and receiving data over the air. Like the BR/

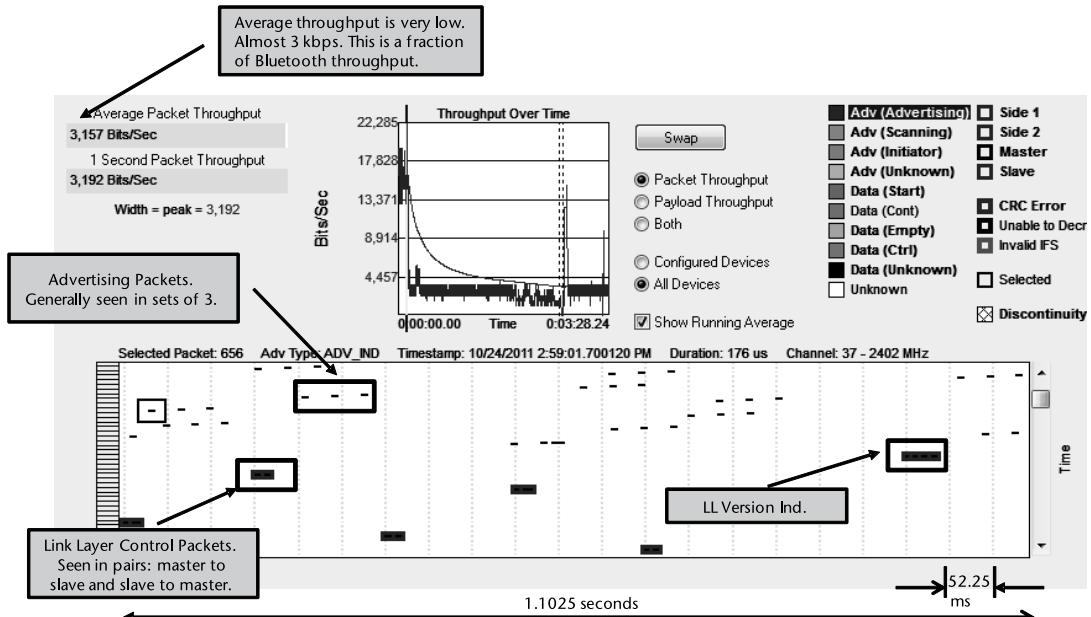


Figure 7.4 LE timeline.

EDR radio, it operates in the 2.4 GHz ISM band though it uses only 40 channels as compared to 79 channels in the case of Bluetooth.

The next chapter will focus on the link layer which is responsible for controlling, negotiating, and establishing the links. The link layer uses the services of the physical layer to send and receive packets. It provides the packets to the physical layer for transmission on the sender side and processes the received packets from the physical layer on the receiver side.

Bibliography

Bluetooth Core Specification 4.0 <http://www.bluetooth.org>.

Link Layer

8.1 Introduction

This chapter explains the operation of the link layer. This layer is responsible for controlling, negotiating and establishing the links, selecting frequencies to transmit data, supporting different topologies and supporting various ways for exchanging data. The position of Link Layer in the LE Protocol stack is shown in Figure 8.1. It sits on top of the Physical Layer and provides services to the L2CAP layer.

8.2 Overview of Link Layer States

The operation of the link layer can be described in terms of a very simple state machine with the following five states:

1. Standby State;
2. Advertising State;
3. Scanning State;
4. Initiating State;
5. Connection State.

In general LE devices support one single instance of the state machine, though multiple instances of the state machine are also permitted by the specification. If a device supports multiple instances of the state machine then only one state can be active at a time for each state machine. Besides this, at least one state machine that supports the advertising state or scanning state is mandatory.

The link layer state machine is shown in Figure 8.2. The five states are explained in brief below and will be explained in detail later in this chapter.

8.2.1 Standby State

This is the default state of the link layer. In this state, no packets are received or transmitted. A device can enter into this state from any of the other states.

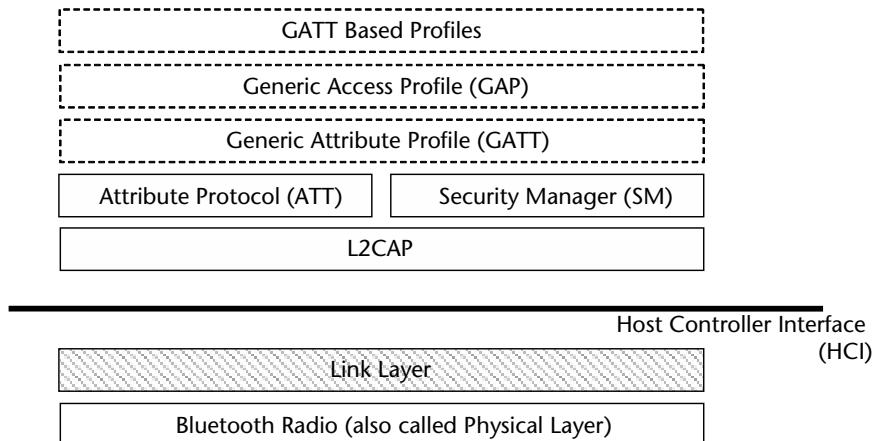


Figure 8.1 Link Layer in LE Protocol Stack.

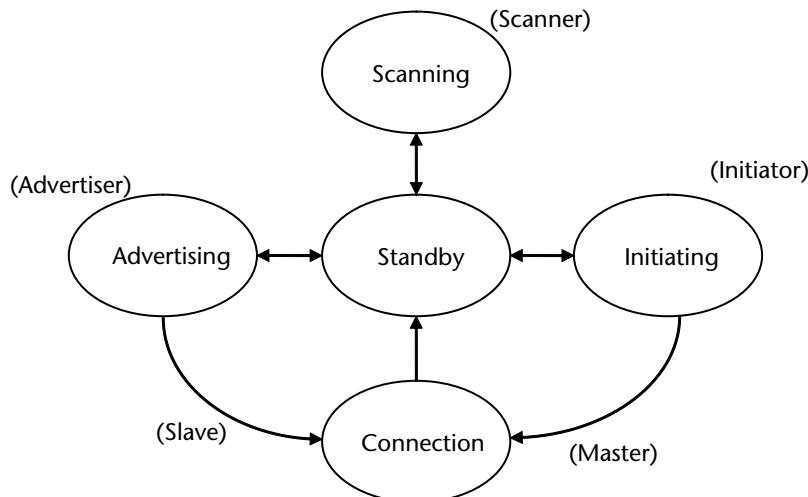


Figure 8.2 Link Layer State Machine—State Diagram and Roles.

8.2.2 Advertising State (Advertiser)

In the advertising state the link layer transmits advertising packets. It may also listen to the devices that respond to the advertising packets and then respond to those devices accordingly. This state can be entered from the standby state when the link layer decides to start advertising. A link layer in this state is known as the Advertiser.

An example of an Advertiser could be a thermometer which continuously advertises that “I’m a thermometer” to the devices around it. It may advertise additional information as well. For example it may also advertise that it has data to send. Any device in the vicinity that is in the scanning state may pick up that packet and query the thermometer for additional information or decide to connect to the thermometer.

8.2.3 Scanning State (Scanner)

In the scanning state the link layer listens to the packets from the Advertiser and may request the Advertiser to provide some additional information. This state can be entered from the standby state when the link layer decides to start scanning. A link layer in this state is known as the Scanner.

8.2.4 Initiating State (Initiator)

In the initiating state the link layer listens to the packets from the Advertiser and responds to those packets by initiating a connection. This state can be entered from the standby state when the Scanner decides to initiate a connection with the Advertiser. A link layer in this state is known as the Initiator.

8.2.5 Connection State (Master or Slave)

In the connection, the device is connected to another device. Two roles are defined—Master Role and Slave Role. This state can be entered either from the initiating state or from the advertising state.

- When entered from the initiating state, the device acts as a Master.
- When entered from the advertising state, the device acts as a Slave.

A link layer in Slave role can communicate with only one single device in Master role. This means the LE does not support scatternet scenarios similar to the ones that are supported in BR/EDR (This is another step towards simplification in LE).

8.3 Device Address

In contrast to BR/EDR where a device has only one Bluetooth device address (BD_ADDR), an LE device may either have a Public Address or a Random Address or both. It is mandatory to have at least one of these addresses so that the device can be identified.

8.3.1 Public Device Address

This address is similar to the BD_ADDR for BR/EDR devices. This was explained in Chapter 2. In fact in case of dual mode devices, both the LE controller and the BR/EDR controller should have the same address. The BD_ADDR of BR/EDR is referred to as the public device address by LE. Public Device Address is a globally unique 48-bit address. It is similar to an Ethernet MAC address and is, in fact, administered by the same organization, IEEE.

The public device address (BD_ADDR) consists of two fields:

1. 24-bit company id assigned by IEEE Registration authority. This is called the Organizationally Unique Identifier (OUI) [24 most significant bits] and is different for each company.

2. 24-bit unique number assigned by the company to each controller. [24 least significant bits]. This is different for each controller manufactured by the company.

8.3.2 Random Address

Random Address is a privacy feature of LE where the device can hide its real address and use a random address that can change over time. So the real address is not revealed at any time. This helps to ensure that a device cannot be tracked.

Consider an example where a person is wearing LE enabled shoes which keep on transmitting data about the number of steps that the person has walked. Somebody can listen to those packets and track the person. So wherever the person is moving, he or she can be followed by just listening to the packet transmitted from the shoes if the shoes are using a fixed address.

To prevent tracking, a random address can be used to transmit. A different random address can be used every time the shoes transmit data. This ensures that the person cannot be tracked. LE provides a mechanism to resolve that random address so that only the intended recipient of the data keeps getting the data and knowing that it is coming from the same device even though it may be from different addresses.

The random address is divided into two fields:

1. Hash field: This is contained in the 24 least significant bits.
2. Random field: This is contained in the 24 most significant bits.

If a device uses a random address, then it must be configured before advertising, scanning, or initiating.

The Generic Access Profile defines the random address to be of two types:

1. Static Address: A device may choose to initialize its static address to a new value after each power cycle but cannot change it while it is still powered. If the device changes its static address, the peer device will not be able to connect to it with the old address that they may have stored.
2. Private Address: The private address may further be of following two types:
 1. Non-resolvable private address: The peer device can never discover the real address.
 2. Resolvable private address: The peer device can derive the real address using the random address and the link key of the connection.

These addresses will be explained further in Chapter 14.

8.4 Physical Channel

As mentioned in the previous chapter, LE uses 40 RF channels in the 2.4 GHz ISM band. These channels are spaced 2 MHz apart.

The RF channels are divided into two physical channels:

- Advertising Physical Channel: This physical channel uses three RF channels, viz channel 0, 12 and 39 for the following activities:
 - Discovering devices.
 - Creating a connection.
 - Broadcasting and receiving data.
- Data Physical Channel: This physical channel uses the remaining 37 RF channels for communication between devices that are in the connection state.

The advertising channels are carefully chosen by the Bluetooth specification to be spread far apart to ensure that if there is interference from other devices in one of the frequency ranges, then at least other frequency ranges are available for advertising. For example, if there is interference in the frequency band 2400 MHz to 2420 MHz, then only channel 0 experiences interference and channels 12 and 39 can still be used for advertising.

Another reason behind the choice of these channels is to reduce interference with WiFi. Many of the devices (like mobile phones, tablets, and laptops) which support Bluetooth have a WiFi radio as well. So it's important that the interference with WiFi is reduced as much as possible.

When a WiFi device is switched on, it uses WiFi channels 1, 6, and 11 as default. The frequency range of the advertising channels is chosen carefully so as not to overlap with channels 1, 6 and 11 of WiFi.

The 40 RF channels are mapped to either a Data Channel Index or an Advertising Channel Index. This is shown in Figure 8.3. For example:

- RF Channel 1 is mapped to Data Channel 0.
- RF Channel 2 is mapped to Data Channel 1.
- RF Channel 13 is mapped to Data Channel 11.

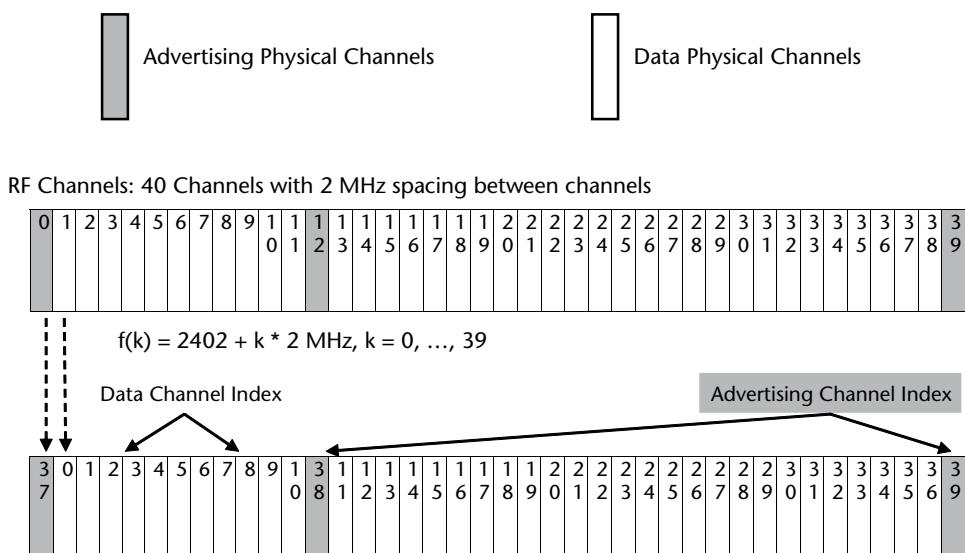


Figure 8.3 Advertising Physical Channels, Data Physical Channels and Mapping to RF Channels.

- RF Channel 0 is mapped to Advertising Channel 37.
- RF Channel 13 is mapped to Advertising Channel 38.
- RF Channel 39 is mapped to Advertising Channel 39.

The link layer uses only one physical channel at any given time.

8.5 Channel Map

Similar to BR/EDR, the Master's link layer classifies the data channels as used or unused. If the Master's link layer suspects interference on any of the channels, it can mark the channel as unused. This has two advantages:

- The channels which potentially have interference can be excluded from the frequency hopping pattern. This reduces the number of retransmissions that would have been done if the packets had been transmitted on those channels.
- The channels which have interference may, in fact, be used by other technologies which are also sharing the ISM band. So this reduces the impact of Bluetooth transmissions on those technologies.

The information about used and unused channels is provided as a bitmap by the Master to the Slave so that both the devices can use the same hopping frequencies. This channel bitmap is called a Channel Map.

The Master can keep on marking channels as unused if it suspects interference until it reaches a minimum of 2 used channels. This is the minimum number of channels that the Master has to use.

8.6 Adaptive Frequency Hopping

LE uses adaptive frequency hopping to hop frequencies across the 37 data channels. The algorithm used is very simple:

$$f_{n+1} = (f_n + \text{hopIncrement}) \bmod 37$$

- If f_n is a used channel, then it is used as it is.
- If f_n is an unused channel, then it is remapped to the set of good channels. (The link layer builds a remapping table to map all the unused channels to used channels.)

The Master sets the hop increment value at the time of creation of a connection. It sets it to a random value between 5 and 16. It may be noted that hop increment is a new concept that has been introduced in LE to simplify the calculation of the next hopping frequency. In BR/EDR, the next hop in the frequency hopping pattern was a function of the Master's parameters like clock and BD_ADDR. LE simplifies this by just using a random value between 5 and 16 as a hop increment to calculate the next hopping frequency. A simpler algorithm, of course, leads to

lesser number of gates when this is implemented in silicon (thereby reducing cost) and lesser amount of processing (thereby reducing power consumption). A random value is needed since if, by chance, there is a collision with another Master on one of the frequencies, the subsequent frequencies for the two Masters will be based on different random numbers. So, further collisions can be avoided.

8.7 Events

The physical channel is subdivided into time units which are known as events. There are two types of events:

- Advertising Events.
- Connection Events.

8.7.1 Advertising Events

Advertising events are used for transmissions on the advertising physical channels. At the start of each advertising event, the Advertiser sends an advertising packet. The Scanner receives this packet and depending on the type of the advertising packet, it may send a request back to the Advertiser. The Advertiser responds to that request within the same advertising event. After that the advertising event is closed. The Advertiser uses the next advertising channel for the next advertising packet. An example of three advertising events is shown in Figure 8.4.

Figure 8.5 shows a sniffer capture of six advertising events.

1. Frame #425: First advertising event (ADV_IND) on Channel 38.
2. Frame #426: Second advertising event (ADV_IND) on Channel 39.
3. Frame #427, #428, #429: Third advertising event on Channel 37. This consists of three packets.
 - a. ADV_IND packet from Advertiser to Scanner.
 - b. SCAN_REQ packet from Scanner to Advertiser.

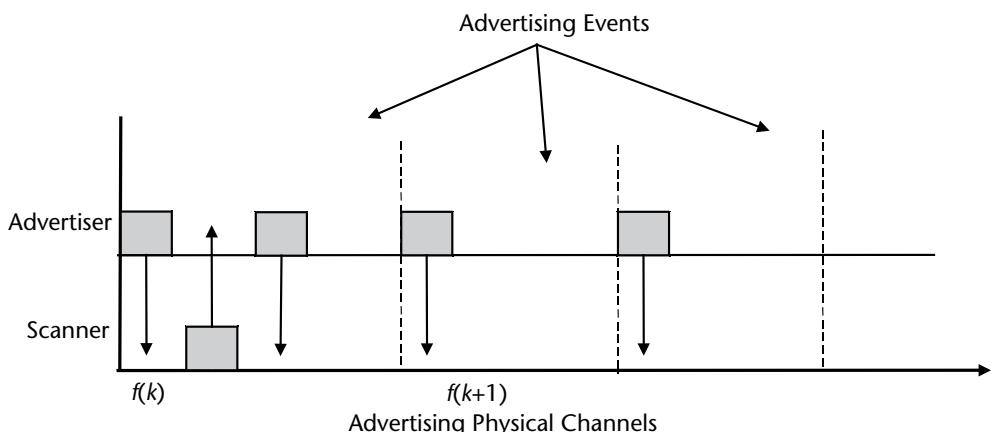


Figure 8.4 Advertising events.

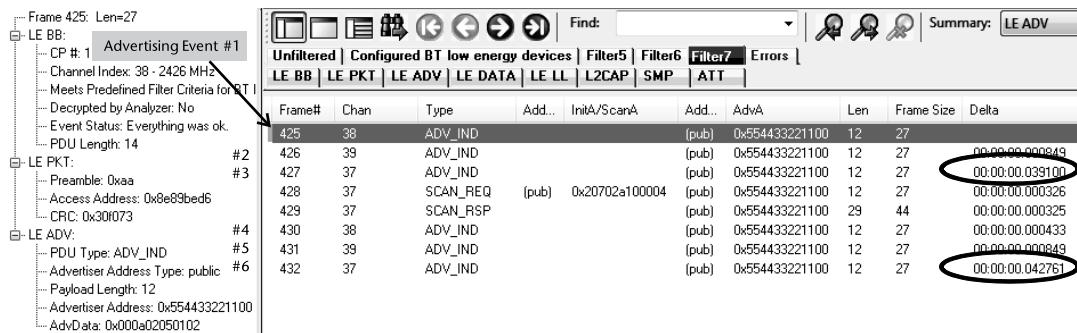


Figure 8.5 Example of advertising events.

- c. SCAN_RSP packet from Advertiser to Scanner in response to the SCAN_REQ packet.
- d. All these packets comprise one single advertising event and are sent on the same channel.
- 4. Frame #430: Fourth advertising event (ADV_IND) on Channel #38.
- 5. Frame #431: Fifth advertising event (ADV_IND) on Channel #39.
- 6. Frame #432: Fourth advertising event (ADV_IND) on Channel #37.

This indicates that the Advertiser is advertising on all the three advertising channels consecutively in rotation: Channel 37, Channel 38, Channel 39 and then again Channel 37, Channel 38, and so on.

There is one more thing worth noting from the Delta time stamps in the last column. The Advertiser advertises on Channels 37, 38, and 39 in quick succession. The approximate delta time between those is 300 to 400 microseconds. After that the Advertiser waits for about 39 or 42 milliseconds before starting the next cycle of advertisements. This means that there are quite big gaps between the advertisements so that the Advertiser can save battery power during that time.

8.7.2 Connection Events

Connection events are used to send data packets between the Master and Slave devices. The start of a connection event is called an Anchor Point. At the Anchor Point, the Master transmits a data channel PDU to the Slave. After that the Master and Slave send packets alternately during the connection event. The Slave always responds to a packet from the Master while the Master may or may not respond to a packet from the Slave. The connection event can be closed by either the Master or the Slave. All packets in a connection event are transmitted on the same frequency. Channel hopping occurs at the start of every connection event. An example of three connection events is shown in Figure 8.6.

The timing of connection events are determined by two parameters:

- Connection Event Interval (connInterval): The connInterval is the interval between two successive starting points of connection event. The start of a connection event is called an anchor point. So the connInterval is the time

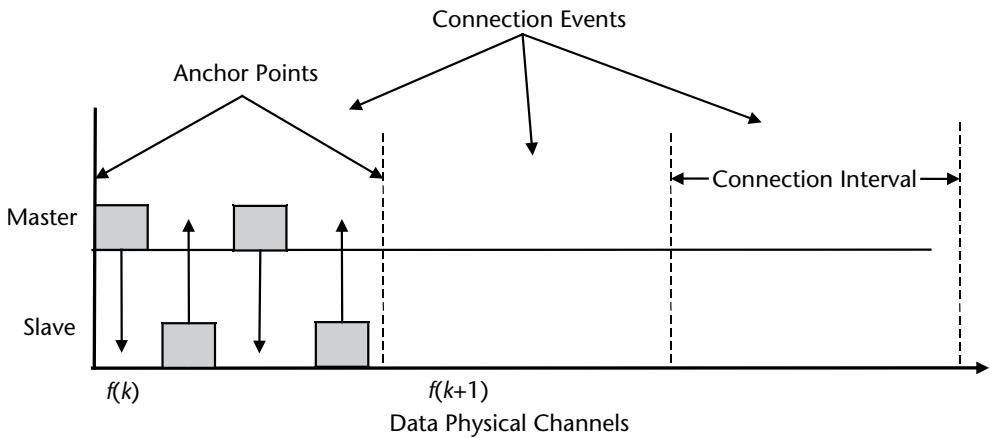


Figure 8.6 Connection events.

difference between two successive anchor points. It is a multiple of 1.25 ms and in the range of 7.5 ms to 4.0s.

- Slave Latency (connSlaveLatency): The connSlaveLatency indicates the number of consecutive connection events that the Slave can skip before listening to the Master. For example, if connSlaveLatency is set to ten then the Slave has to listen to every tenth connection event. If it is set to 0, then the Slave has to listen to every connection event.

Besides these, a Supervision Timeout (connSupervisionTimeout) parameter is used by both the Master and the Slave to detect whether a connection has been lost. If a packet has not been received for a duration of connSupervisionTimeout, then the connection is considered to be lost and no further packets are sent. The host is informed about the loss of connection.

The first connection event is scheduled after the CONNECT_REQ PDU. The master provides two parameters in the CONNECT_REQ PDU to indicate the transmit window:

- transmitWindowOffset: This indicates the time difference between CONNECT_REQ PDU and the transmit window.
- transmitWindowSize: This indicates the size of the transmit window.

8.8 Topology

The possible topologies for LE are shown in Figure 8.7. In scenario A, one Advertiser is sending advertising packets on the advertising physical channel. There are two scanners listening to those advertising packets. These scanners may request more information from the Advertiser or send a request to connect on the advertising physical channel.

Scenario B shows a piconet where one Master is connected to three Slaves. The data between the Master and the Slaves is exchanged on the data physical channels.

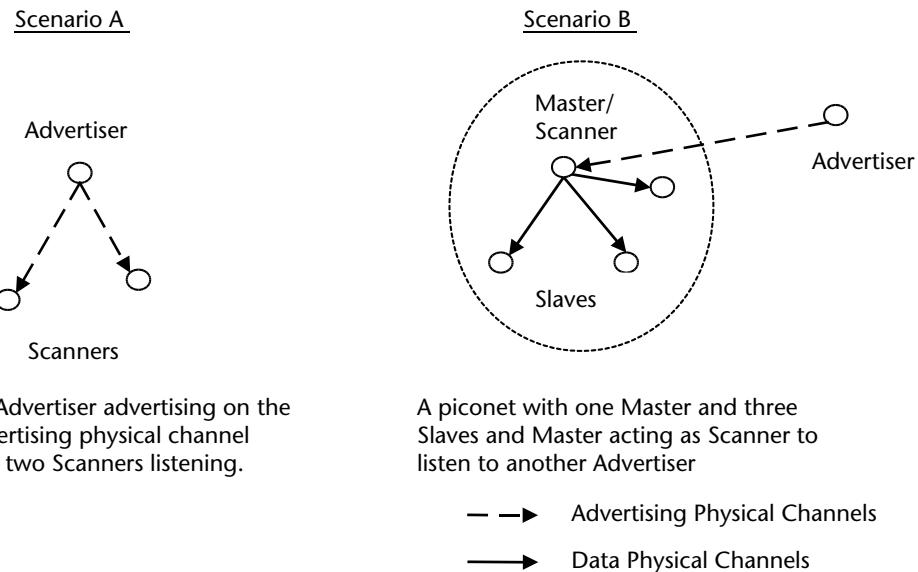


Figure 8.7 LE topology.

Besides this, the Master is also acting as a Scanner and listening to packets from an Advertiser on the advertising physical channel. In this scenario if the Master/Scanner decides to connect to the Advertiser, then it will send a connect request to the Advertiser on the advertising physical channel. If the connection is established successfully, the Advertiser will also join the same piconet and become a Slave of the same Master. Then the Master will have four Slaves.

While the BR/EDR specification put an upper limit of up to seven Slaves connected to a Master, the LE specification does not put any such upper limit. A Master can connect to as many Slaves as it wants. This is only restricted by the amount of resources available with the Master in terms of memory and processing power.

As with BR/EDR only one device can be a piconet Master and all other devices are piconet Slaves. All the communication is between the Master and Slave devices. The Slaves are not allowed to directly talk to each other.

An LE device can belong to only one piconet at a particular time. Scatternets are not supported. This restriction helps in simplifying the design of link layer for LE devices.

8.9 Packet Format

The link layer has only one packet format that is used for both advertising and data physical channels. This is in contrast to BR/EDR which has several packet formats (ID, NULL, POLL, FHS, DM). This is another step towards simplification by LE. The packet format is shown in Figure 8.8.

LSB	ACCESS ADDRESS (4 octets)	PDU (2 to 39 octets)	MSB
PREAMBLE (1 octet)			CRC (3 octets)

Figure 8.8 Link layer packet format.

8.9.1 Preamble

The preamble is used by the receiver to perform frequency synchronization, symbol timing estimation, and Automatic Gain Control (AGC) training. The Preamble is an alternate sequence of 1s and 0s. So it can be 10101010b or 01010101b. The receiver uses this sequence to synchronize its radio to the exact frequency and also adjust the gain so that the remaining packet is correctly received.

8.9.2 Access Address

The access address is used as a correlation code by devices tuned to the physical channel. Since LE uses a limited number of data channels, there is a possibility of unrelated LE devices using the same RF channel at the same time. The access address is used as a code to ensure that the transmission is indeed meant for the device that is receiving it. The access address is different for each link layer connection between any two devices.

A link layer is said to be connected to a channel if it is synchronized to the following parameters of the channel:

- Timing.
- Frequency.
- Access Address.

It is not mandatory for the link layer to be actively involved in data exchange to remain connected. This is another enhancement over BR/EDR. For the link layer to remain connected, it doesn't need to continuously send and receive data. In the case of BR/EDR, if the link layers are connected and they have no data to send, then they still exchange POLL/NULL packets continuously.

8.9.3 CRC

The CRC is a 24-bit checksum calculated over the PDU. One of the enhancements done in LE is that the CRC is 24-bit as compared to 16-bit in the case of BR/EDR. A 32-bit CRC helps to check for many more types of bit-errors compared to a 16-bit CRC. This leads to enhanced robustness especially in noise environments where the chance of multiple bit errors is higher.

8.9.4 PDU

The PDU is of two types:

- Advertising channel PDU: To transmit a packet on the advertising physical channel.
- Data channel PDU: To transmit a packet on the data physical channel.

8.9.4.1 Advertising Channel PDUs

The format of the advertising channel PDUs is shown in Figure 8.9.

The PDU Type field indicates the type of the advertising channel PDU. The TxAdd and RxAdd fields are defined for each PDU type separately and may not be valid for all PDU types. These are used to indicate whether the address contained in the payload is a public address ($\text{RxAdd}/\text{TxAdd} = 0$) or random address ($\text{RxAdd}/\text{TxAdd} = 1$).

The Length field indicates the length of the payload field in octets.

There are three types of advertising channel PDUs:

- Advertising PDUs: These PDUs are sent by the link layer in advertising state and received by the link layer in the scanning state or initiating state. The different type of advertising PDUs are shown in Table 8.1.
- Scanning PDUs: These PDUs are used by the link layer of the Scanner to request data from the Advertiser and by the Advertiser to respond to the request from the Scanner. The different types of scanning PDUs are shown in Table 8.2.
- Initiating PDUs: These PDUs are used by the link layer to initiate a connection to the Advertiser. There is only one type of PDU and that is shown in Table 8.3.

An example of the advertising and scanning PDUs was shown in Figure 8.5.

- The Advertiser sent ADV_IND PDUs in Frames #425, #426, #427, #430, #431 and #432.
- The Scanner sent SCAN_REQ PDU in Frame #428.
- The Advertiser responded with SCAN_RSP PDU in Frame #429.

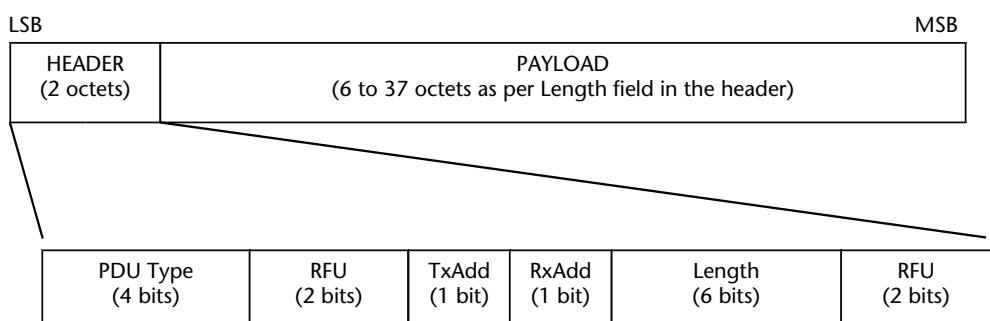


Figure 8.9 Advertising Channel PDU.

Table 8.1 Advertising PDUs

PDU Name	Advertising Event Type	Sender Link Layer State	Receiver Link Layer State
ADV_IND	Connectable Undirected	Advertising	Scanning/Initiating
ADV_DIRECT_IND	Connectable Directed	Advertising	Scanning/Initiating
ADV_NONCONN_IND	Non-Connectable Directed	Advertising	Scanning/Initiating
ADV_SCAN_IND	Scannable Undirected	Advertising	Scanning/Initiating

Table 8.2 Scanning PDUs

PDU Name	Scanning Event Type	Sender Link Layer State	Receiver Link Layer State
SCAN_REQ	Scanner requesting data from Advertiser	Scanning	Advertising
SCAN_RSP	Advertiser responding to the request from Scanner	Advertising	Scanning

Table 8.3 Initiating PDUs

PDU Name	Initiating Event Type	Sender Link Layer State	Receiver Link Layer State
CONNECT_REQ	Initiator requesting to connect to Advertiser	Initiating	Advertising

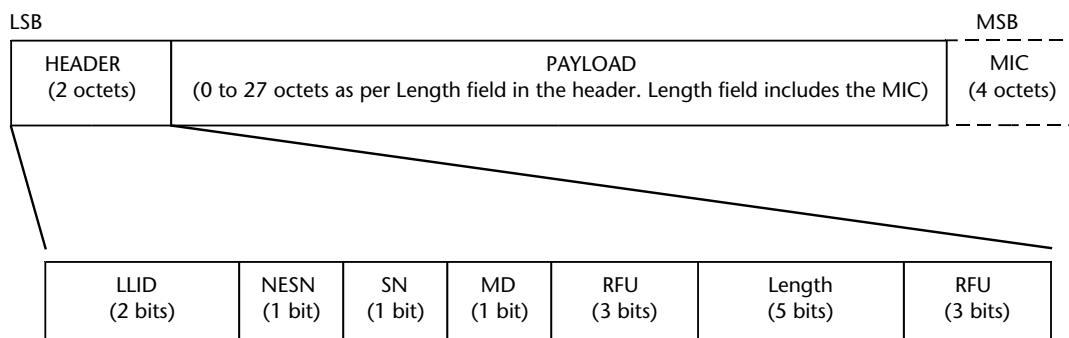
8.9.4.2 Data Channel PDUs

The format of the data channel PDUs is shown in Figure 8.10.

The Payload field is 0 to 27 octets in length. The Message Integrity Check (MIC) field is included only in the case of encrypted link layer connection when the payload field has a nonzero size. This is used to authenticate the data PDU.

The LLID indicates the type of the link layer PDU. There are two possible types:

- Link layer data PDU: This type of PDU is used to send L2CAP data.

**Figure 8.10** Data Channel PDU.

- Link layer control PDU: This type of PDU is used to control and negotiate the connection between the two link layers. There are various types of link layer PDUs defined for the link layer to exchange control information with the link layer of the other device. The various layer control procedures and corresponding PDUs will be explained later in this chapter.

MD indicates that the device has more data to send. It is used to decide whether the current connection event can be closed or not. Length indicates the size in octets of Payload and MIC fields. The SN field indicates the Sequence Number and the NESN field indicates the Next Expected Sequence Number. These two bits provide a very simple mechanism for acknowledgment and flow control of packets.

The SN bit identifies the current packet while the NESN bit identifies which packet from the peer device is expected next. If a packet is correctly received then the NESN bit is incremented. This serves as an acknowledgment to the sender that the packet has been received. If the packet had an error, then NESN bit is not incremented. This indicates to the sender that it has to resend the previous packet. This is shown in Figure 8.11.

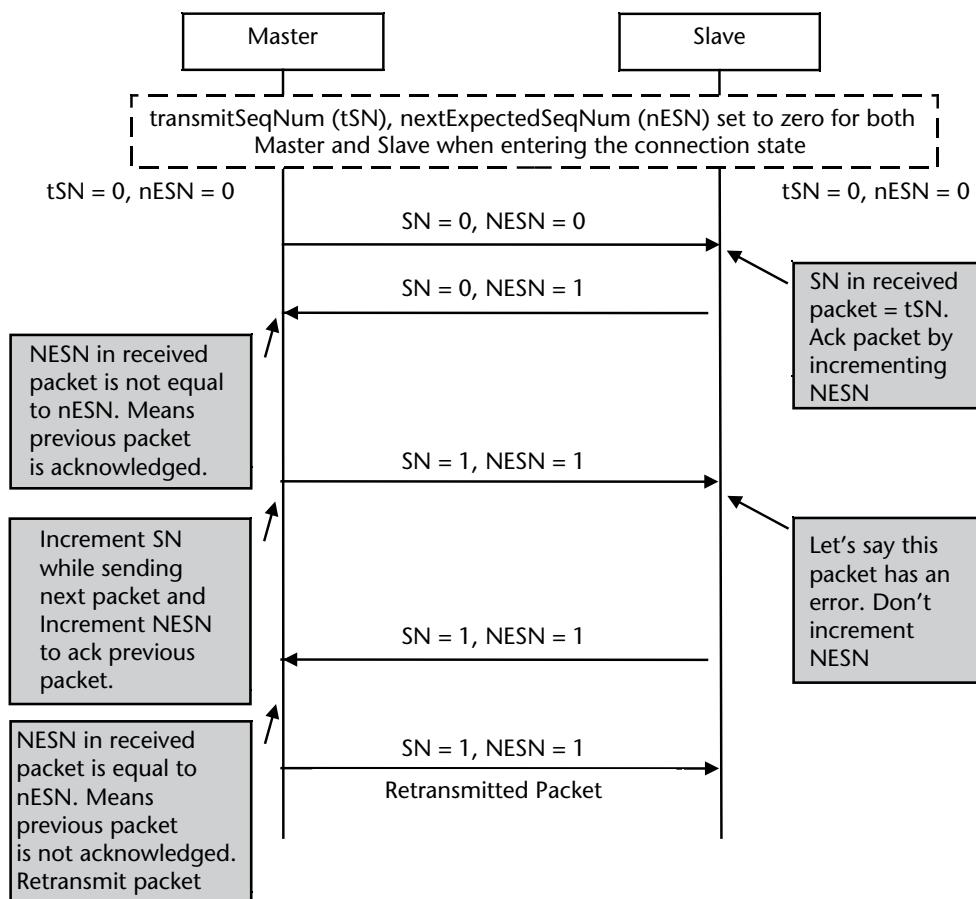


Figure 8.11 Acknowledgment and flow control using SN and NESN.

8.10 Bit Stream Processing

Figure 8.12 shows the sequence of steps that are carried out by the link layer before transmitting data and the link layer on the other side after receiving the data. The data to send is treated as a bit stream with the LSB first. At the time of transmission, the different steps that are performed are encryption of the data follower by CRC generation and then finally whitening. When the data is received on the receiving side, exactly the reverse of these steps are performed.

The Encryption stage on the transmitter side and the decryption stage on the receiver side are optional. Encryption is done only if the host requested an encrypted link.

Data whitening is the process used to avoid long sequences of zeros or ones while transmitting. Before transmitting the header and payload are scrambled with a data whitening word. This randomizes the data to reduce the possibilities of long sequences of zeros or ones. At the receiver end, the data is descrambled using the same data whitening word to get back the original data.

When a packet is received, the first step is to check for errors. This includes the following:

- Check the Access address to ensure that the packet is meant for the channel that the link layer is connected to.
- CRC checking.

One of the optimizations done by LE is that the encryption is done before CRC generation while transmitting data and decryption is done after CRC checking while receiving data. This is opposite to BR/EDR where CRC generation is done before encryption while transmitting and CRC checking is done after decryption while receiving. So in the case of LE, there are the following advantages:

1. CRC checking takes far lesser time as compared to decryption while receiving. So if CRC checking is done before decryption:
 - a. The received packet can be acknowledged as soon as the CRC check is complete. So the radio can be immediately switched off instead of waiting for the complete decryption process.
 - b. The complete decryption process can then be done offline when the radio is switched off. This helps in reducing the peak power consumption since only decryption is going on and the radio has been switched off.

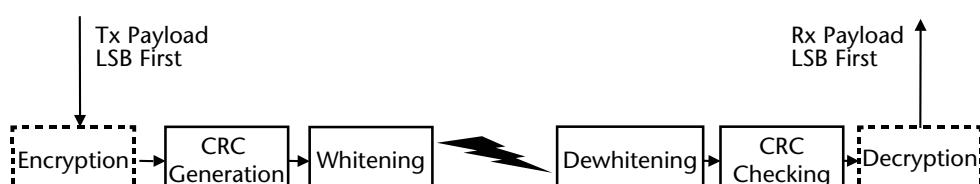


Figure 8.12 Link layer bit stream processing.

- c. If the packet got corrupted by the time it was received, then the CRC check will detect it early and the packet can be dropped immediately. For such packets power does not need to be consumed while doing the decryption.

8.11 Link Layer States

The five link layer states were briefly introduced at the beginning of this chapter. This section provides a detailed explanation of each of these states. Broadly the states can be categorized into Non-Connected States and Connection States as shown in Figure 8.13.

To reduce the complexity, LE does not allow scatternet scenarios. (Note that a scatternet is a combination of multiple piconets. It is formed in BR/EDR scenarios when one of the devices acts as Slave in two piconets or as Master in one piconet and Slave in another piconet.)

This imposes the following restrictions on LE devices:

1. The device cannot act as Master and Slave at the same time.
2. The device cannot also act in initiating state if it is already in Slave role. This is because it would lead to a Master connection in addition to the Slave role.
3. The device cannot have more than one Slave connections.
4. If the device is already operating in Connection or Initiating state, it cannot operate in Advertising state with advertising type that will lead to a Slave role connection.

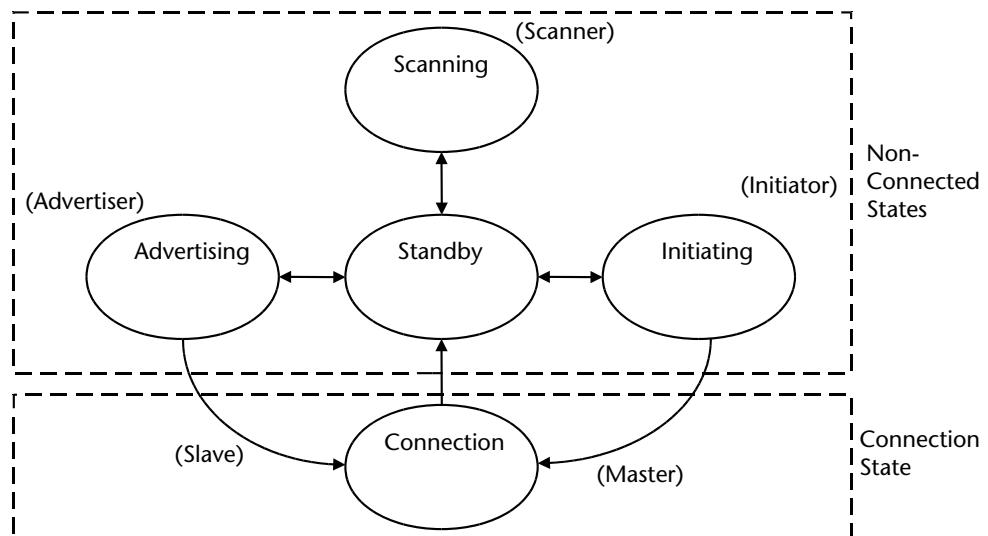


Figure 8.13 Link Layer States.

8.11.1 Nonconnected States

In the Nonconnected states the link layer can be in any one of the following four states:

1. Standby State.
2. Advertising State.
3. Scanning State.
4. Initiating State.

These states are explained in detail below.

8.11.1.1 Standby State

This is the default state of the link layer. No packets are sent or received in this state. From this state, a device can enter advertising state, scanning state or initiating state. It cannot go into connection state directly from the standby state.

A device can enter into this state from any other state. In fact, to aid simplicity of the state machine and reduce the number of possible combinations of transitions from one state to another, the link layer state machine has been designed to only have the minimum needed transitions and this state is used as an intermediate state. For example, to go from scanning state to initiating state the link layer first goes from scanning state to standby and then from standby to initiating state.

8.11.1.2 Advertising State

In this state, the link layer transmits advertising PDUs in advertising events.

During an advertising event, the link layer transmits one or more advertising PDUs on each of the used advertising channels (The host may request the link layer to use either all or a subset of the three advertising channels—37, 38 and 39). The device in this state is known as Advertiser.

The advertising events are of following four types. The advertising PDUs associated with each of these events were shown in Table 8.4.

1. Connectable undirected event.
2. Connectable directed event.

Table 8.4 Advertising Event Types, PDUs Used and Acceptable Responses

<i>Advertising Event Type</i>	<i>PDU Used</i>	<i>Acceptable Response PDU from Remote Device</i>	
		<i>Scanner (SCAN_REQ)</i>	<i>Initiator (CONNECT_REQ)</i>
Connectable Undirected Event	ADV_IND	Yes	Yes (From any Initiator)
Connectable Directed Event	ADV_DIRECT_IND	No	Yes (Only from the addressed Initiator)
Nonconnectable Undirected Event	ADV_NONCONN_IND	No	No
Scannable Undirected Event	ADV_SCAN_IND	Yes	No

3. Non connectable undirected event.
4. Scannable undirected event.

Connectable Undirected Event

The connectable undirected event is sent by an Advertiser when it wants another device to connect to it. It sends an advertising indication (ADV_IND) PDU on the advertising channel. The other device may also request for additional information before deciding to connect to the Advertiser.

The receiver of the connectable undirected event can be either in the scanning state or initiating state.

- If it's in the scanning state, it may request for more information using SCAN_REQ PDU.
- If it's in the initiating state, it may send a connect request using CONNECT_REQ PDU.

The two scenarios are shown in Figure 8.14. A device may use the first one or the second one or first one followed by the second one.

The payload in a connectable undirected event contains the following two fields:

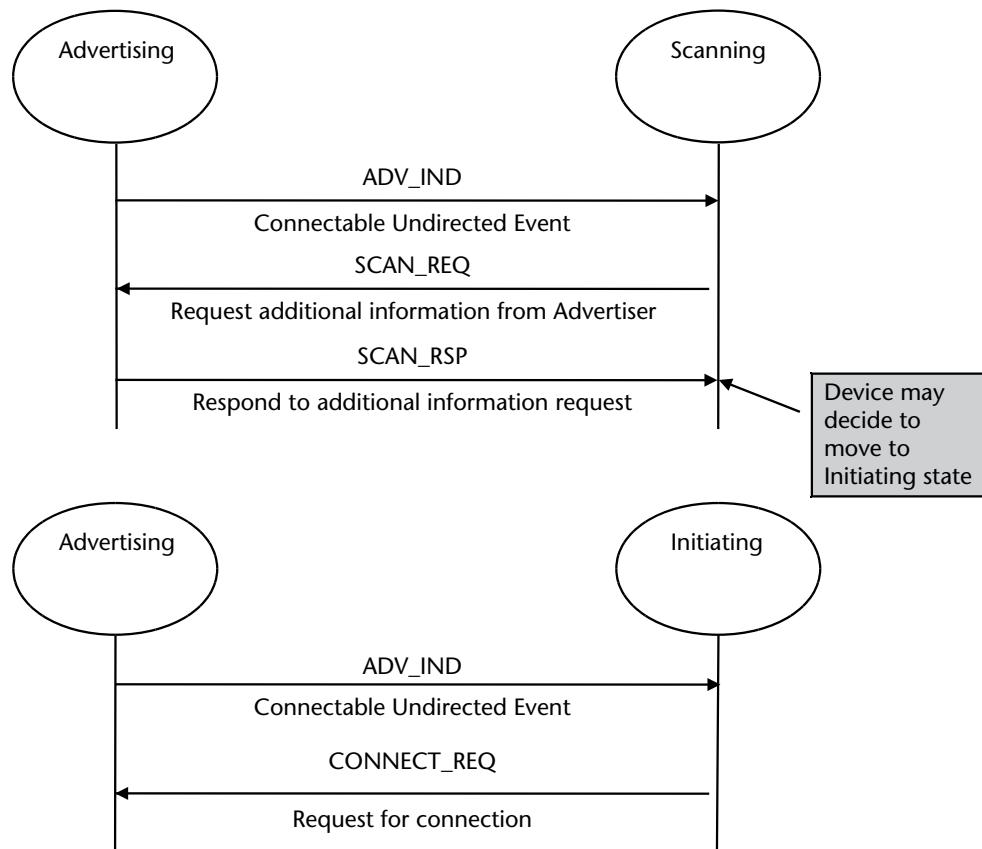


Figure 8.14 Connectable Undirected Advertising Event.

- AdvA (6 octets): Public address or random address of the Advertiser. The type is indicated by TxAdd field (See Figure 8.9).
- AdvData (0-31 octets): This contains the advertising data from the Advertiser's host.

One possible example of this could be a thermometer that is placed in a building.

- The thermometer could keep on advertising—‘I am a thermometer’;
- Any mobile phone in the vicinity could query—‘Do you display temperature in Fahrenheit?’;
- The thermometer could say—‘Yes’;
- The mobile phone could then connect to the thermometer and get the temperature.

An example of air logs for ADV_IND was shown in Figure 8.5. The Advertiser sends an advertising PDU in frame #425, #426, etc. In response to the ADV_IND packet sent in Frame #427, the Scanner requests for additional information using the SCAN_REQ PDU.

Connectable Directed Event

The connectable directed event type is used when an Advertiser wants a particular device to connect to it. It sends a directed advertising indication (ADV_DIRECT_IND) PDU on the advertising channel. The other device may request to connect to the Advertiser on receiving this PDU.

The ADV_DIRECT_IND PDU contains the device address of both the Initiator and the Advertiser. So only the Initiator for which the address was contained in the PDU is allowed to make a connection.

The payload in a connectable directed event contains the following two fields:

- AdvA (6 octets): Public address or random address of the Advertiser. The type is indicated by TxAdd field (See Figure 8.9).
- InitA (6 octets): Public address or random address of the Initiator. The type is indicated by RxAdd field (See Figure 8.9).

This is in contrast to connectable directed event where any device in the scanning or initiating state could request for additional information or connect to the advertiser. Here the request for additional information is not permitted and only a particular device can initiate the connection. The sequence diagram for this is shown in Figure 8.15.

One possible example of this could be a pedometer placed in the jogger's shoe. The pedometer may need to send information to the person's mobile phone.

- The pedometer could advertise: ‘I'm a pedometer. I want to send data to mobile phone A.’
- Mobile phone A could receive this request and connect to the pedometer and get the data and display to the person.

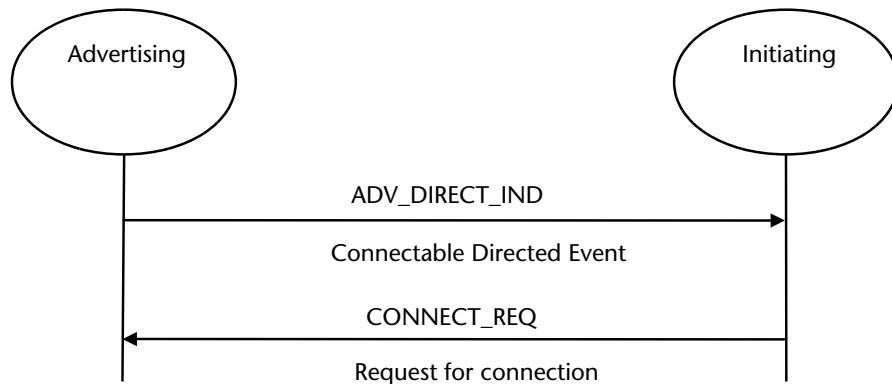


Figure 8.15 Connectable Directed Advertising Event.

Nonconnectable Undirected Event

The nonconnectable undirected event type is used when an Advertiser wants to provide some information to all devices but does not want the devices to connect to it or ask for more information. It sends a nonconnectable advertising indication (ADV_NONCONN_IND) PDU on the advertising channel. The other device may only listen to this information.

This is in contrast to connectable undirected and connectable directed events because the receiver (which has to be in the scanning state) can just receive this information. It can neither connect nor ask for more information.

The payload in a non-connectable undirected event contains the following two fields:

- AdvA (6 octets): Public address or random address of the Advertiser. The type is indicated by TxAdd field (See Figure 8.9).
- AdvData (0–31 octets): This contains the advertising data from the Advertiser's host.

The sequence diagram for this is shown in Figure 8.16.

One possible example of this could be a microwave oven.

- The microwave could advertise: 'I'm a microwave. The food is cooked. Please take it out.'

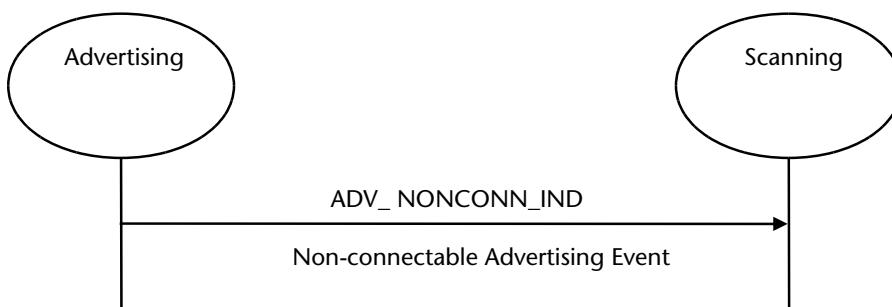


Figure 8.16 Nonconnectable Undirected Advertising Event.

- The people in the house could receive this information on their mobile phone, television or set top box and take appropriate action.

Another example could be an Advertiser at the airport:

- The Advertiser could advertise that flight ABC will take off from gate 123.
- Any person who is interested in the flight information could scan for that information.

Scannable Undirected Event

The scannable undirected event type is used when an Advertiser wants to allow a Scanner to request more information from it. It sends a scannable advertising indication (ADV_SCAN_IND) and the Scanner may request more information using the SCAN_REQ PDU.

This is slightly different from the Nonconnectable Undirected advertising event since in the nonconnectable advertising event, the Scanner cannot send any request back. Here the Scanner may send a scan request to get more information.

The payload in a scannable undirected event contains the following two fields:

- AdvA (6 octets): Public address or random address of the Advertiser. The type is indicated by TxAdd field (See Figure 8.9).
- AdvData (0–31 octets): This contains the advertising data from the Advertiser's host.

The sequence diagram for this is shown in Figure 8.17.

One possible example of this could be a remote control.

- The remote control could advertise: 'I'm a remote. A key has been pressed.'
- The Scanner could send a SCAN_REQ PDU to gather information about which key has been pressed.

A summary of the four advertising event types is provided in Table 8.4.

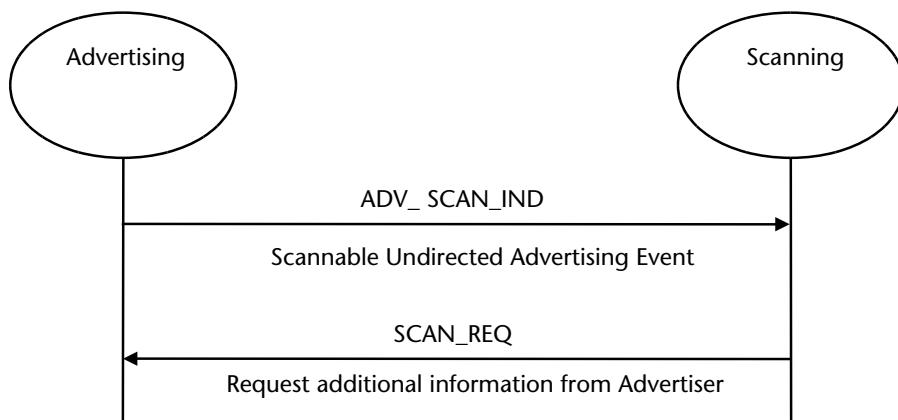


Figure 8.17 Scannable Undirected Advertising Event.

8.11.1.3 Scanning State

In this state, the link layer listens on the advertising channels (Advertising channels 37, 38, 39) for any PDUs from the Advertiser. The device in this state is known as Scanner.

The scanning events are of following two types:

- Passive Scanning.
- Active Scanning.

The advertising PDUs associated with each of these events were shown earlier in Table 8.2

Passive Scanning

In passive scanning, the link layer only receives the packets. It does not send back any packets. Once it receives the packets, it removes the duplicates and then sends the advertising reports to the host. The sequence diagram for this is shown in Figure 8.18.

Active Scanning

In active scanning, the link layer listens to the advertising PDUs and then depending on the advertising PDU type, it may request additional information from the Advertiser using the SCAN_REQ PDU.

The Scanner is permitted to send a SCAN_REQ only if the Advertiser used a connectable undirected event (ADV_IND PDU) or a scannable undirected event (ADV_NONCONN_IND PDU). These are the ones that are marked as Yes in the column for SCAN_REQ in Table 8.4.

The payload in a SCAN_REQ PDU contains the following two fields:

- ScanA (6 octets): Public address or random address of the Scanner. The type is indicated by TxAdd field (See Figure 8.9).
- AdvA (6 octets): Public address or random address of the Advertiser. The type is indicated by RxAdd field (See Figure 8.9).

The payload in a SCAN_RSP PDU contains the following two fields:

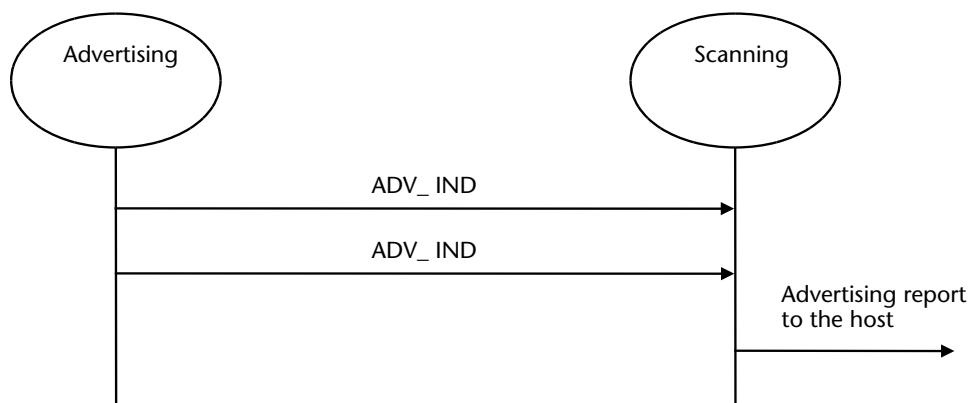


Figure 8.18 Passive Scanning.

- AdvA (6 octets): Public address or random address of the Advertiser. The type is indicated by TxAdd field (See Figure 8.9).
- AdvData (0–31 octets): This contains the advertising data from the Advertiser's host.

The sequence diagram for this is shown in Figure 8.19.

A practical example of active scanning was shown in Figure 8.5. At Frame #428, the Scanner sent a SCAN_REQ to get more information from the Advertiser.

8.11.1.4 Initiating State

In the initiating state, the link layer listens on the advertising channels (Advertising channels 37, 38, 39) for any PDUs from the Advertiser and if it's permitted, it sends a connection request to the Advertiser.

The Initiator is permitted to send a CONNECT_REQ only if the Advertiser used a connectable undirected event (ADV_IND PDU) or a connectable directed event (ADV_DIRECT_IND PDU). In the latter case, the address of the Initiator must match the address that was provided in the connectable directed event PDU. These are the scenarios that are marked as Yes in the column for CONNECT_REQ in Table 8.4. The sequence diagram for this is shown in Figure 8.20.

Figure 8.21 shows a sniffer capture of the connection initiation procedure. The following things may be observed:

- The Advertiser sent an advertising event (ADV_IND) in Frames #670, #671, and #672.
- The Initiator responded with a CONNECT_REQ in Frame #673 to create a connection.

8.11.2 Connection State

This state is entered when the Initiator sends a CONNECT_REQ PDU to the Advertiser. As shown in Figure 8.13, this state can be entered in two ways.

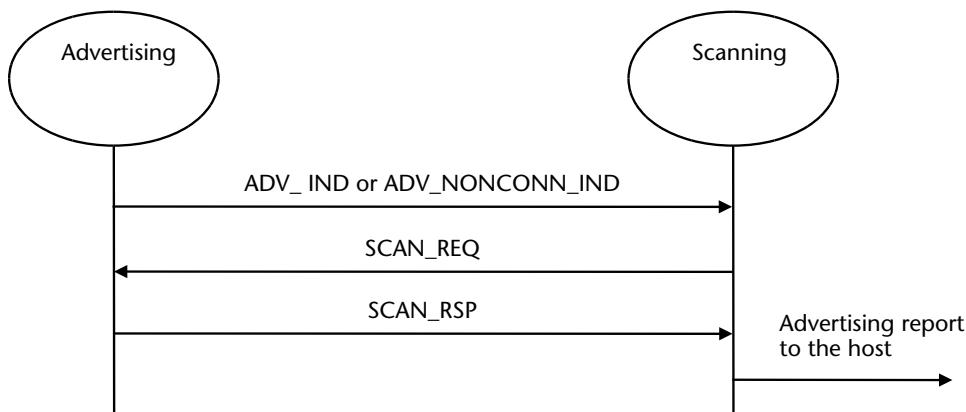


Figure 8.19 Active Scanning.

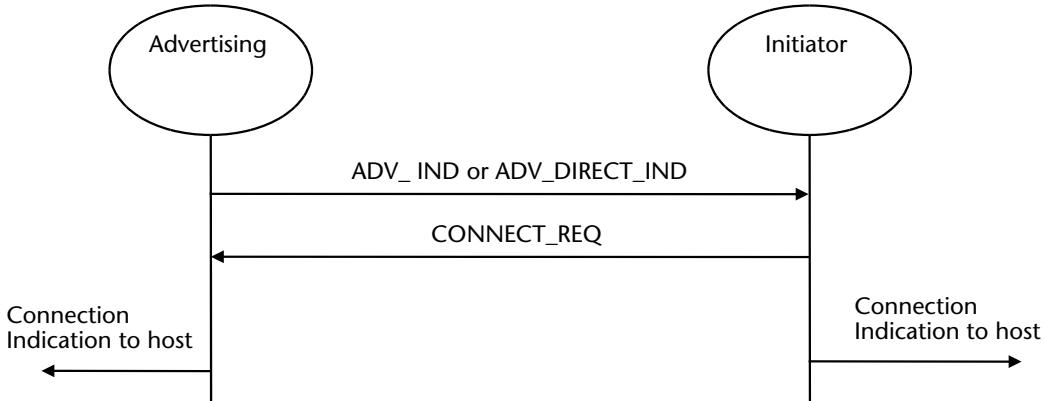


Figure 8.20 Initiating Connections.

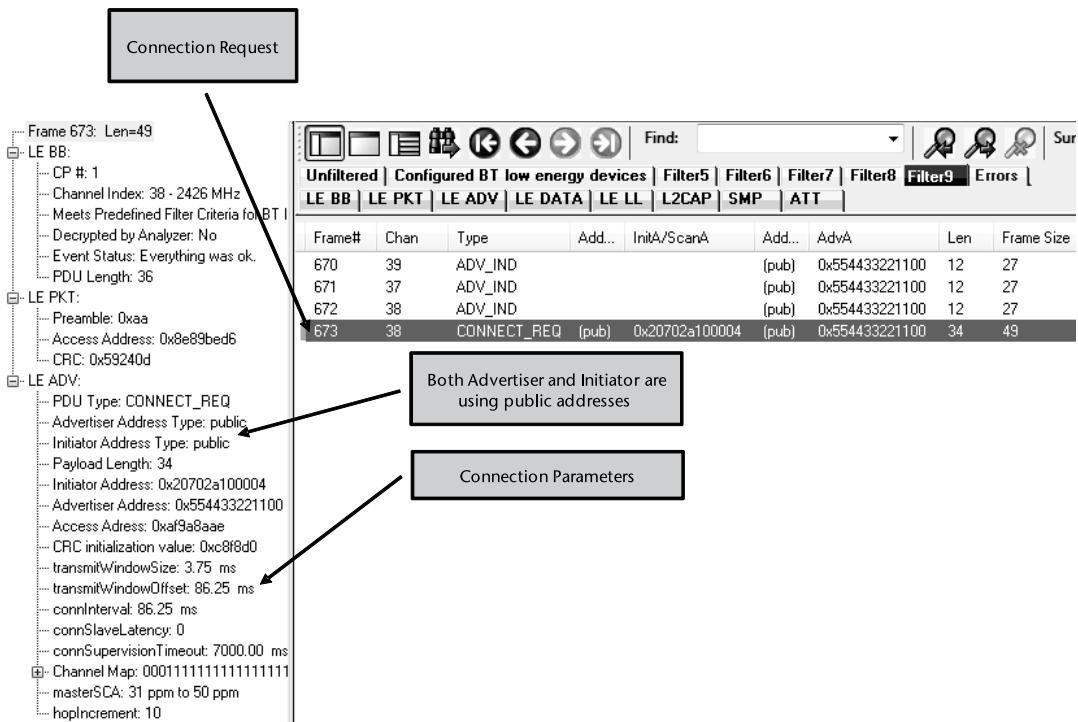


Figure 8.21 Example of Initiating a connection.

- From a link layer in the initiating state. The Initiator becomes the Master of the connection.
- From a link layer in the advertising state. The Scanner becomes the Slave of the connection.

After entering the connection state, the connection is considered to be created (but not established). After the connection is created, once a data channel packet has been received from the peer device, the connection is considered to be established.

The payload in a CONNECT_REQ PDU contains the following three fields:

- InitA (6 octets): Public address or random address of the Initiator. The type is indicated by TxAdd field (See Figure 8.9).
- AdvA (6 octets): Public address or random address of the Advertiser. The type is indicated by RxAdd field (See Figure 8.9).
- LLData (22 octets): This contains various parameters related to the connection.
 - AA (4 octets): The Access Address for the Link Layer's connection.
 - CRCInit (3 octets): Initialization value for the CRC calculation. (Random value.)
 - WinSize (1 octet): Indicates transmit window size.
 - WinOffset (2 octets): Indicates transmit window offset.
 - Interval (2 octets): Connection Interval (connInterval).
 - Latency (2 octets): Connection Latency (connSlaveLatency).
 - Timeout (2 octets): Connection Supervision Timeout (connSupervisionTimeout).
 - ChM (5 octets): Channel bitmap showing used and unused channels.
 - Hop (5 bits): Hop increment to be used for frequency hopping algorithm.
 - SCA (3 bits): Indicate worst case Master's sleep clock accuracy.

Some of these parameters were already explained earlier in this chapter in the section related to Connection Events. The sequence diagram for this is shown in Figure 8.22. Once the connection is established, the Master and Slave can exchange data channel PDUs in connection events.

Figure 8.21 showed a sniffer capture of the connection initiation procedure. The parameters of the CONNECT_REQ PDU are shown on the left hand side. These include the following:

- InitA (6 octets): Public address of the Initiator.
- AdvA (6 octets): Public address of the Advertiser.
- Access Address of the connection.
- LLData (22 octets) containing the following:
 - CRC Initialization value. Random value provided by Master to be used for CRC calculations.
 - transmitWindowSize = 3.75 ms.
 - transmitWindowsOffset = 86.25 ms.
 - connInterval = 86.25 ms.
 - connSlaveLatency = 0 (This means that the Slave has to listen for each connection event).
 - connSupervisionTimeout = 7000 ms. (This means that if no packet is received for 7 seconds then the connection is considered to be lost).
 - ChannelMap: Indicating which channels can be used for data transfer.

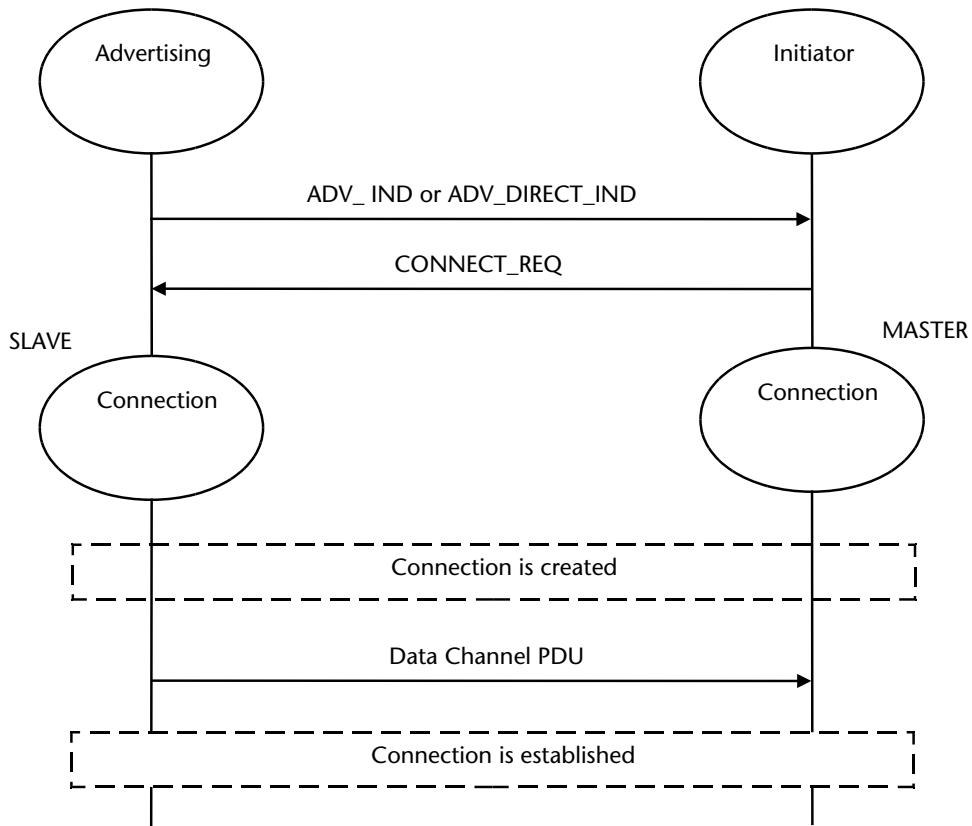


Figure 8.22 Connection state.

- HopIncrement: 10. This indicates the Hop Increment to be used in the algorithm to calculate the next hopping frequency.

8.12 Link Layer Control Procedures

The Link Layer Control Protocol (LLCP) is used to control and negotiate the connection between the two link layers. The summary of the link layer control procedures and the associated PDUs is shown in Table 8.5. The PDUs are encapsulated within the link layer control PDUs that are part of the data channel PDUs. The format of the data channel PDUs was explained earlier in Figure 8.10.

These procedures can only be invoked sequentially. This means that at any given time only one link layer procedure is initiated. The next link layer procedure can only be initiated after the previous one either completes or has a timeout. The only exception is the termination procedure which can be initiated at any time.

8.12.1 Connection Update Procedure

The connection update procedure is used to update the following link layer parameters of the connection:

Table 8.5 Link Layer Control Procedures and PDUs

<i>Link Layer Procedure</i>	<i>Link Layer Control PDU Name</i>	<i>Brief Purpose</i>
Connection Update Procedure	LL_CONNECTION_UPDATE_REQ	This procedure is used by the Master any time after entering the connection state to update the link layer parameters of a connection.
Channel Map Update Procedure	LL_CHANNEL_MAP_REQ	This procedure is used by the Master after entering the connection state to update the channel map to be used for frequency hopping.
Encryption Start Procedure	LL_ENC_REQ LL_ENC_RSP LL_START_ENC_REQ LL_START_ENC_RSP	This procedure is used by the Master to start encryption or to re-start encryption after a pause encryption procedure.
Encryption Pause Procedure	LL_PAUSE_ENC_REQ LL_PAUSE_ENC_RSP	This procedure is used if the Master wants to change the encryption key. A Pause procedure is followed by the Encryption Start procedure to change the link key.
Feature Exchange Procedure	LL FEATURE REQ LL FEATURE RSP	This procedure is used by the Master after a connection has been established to initiate exchange of feature set information.
Version Exchange Procedure	LL_VERSION_IND	This procedure is used by either the Master or the Slave after entering the connection state to exchange version information.
Termination Procedure	LL_TERMINATE_IND	This procedure is used in the connection state by either the Master or the Slave to terminate the connection.
Unused/ Unsupported PDU Rejection	LL_UNKNOWN_RSP LL_REJECT_IND	This PDU is sent as a response if an unused or unsupported PDU is received. This PDU is sent if a request from the other side is rejected. For example the Slave sends this response to Master if the Master tries to enable encryption and the Slave does not support it.

- Connection Interval.
- Connection Slave Latency.
- Connection Supervision Timeout.

This procedure can only be initiated by the Master after entering the connection state. The sequence diagram for this procedure is shown in Figure 8.23.

8.12.2 Channel Map Update Procedure

The channel map procedure is used to update the channel map of the connection. The channel map contains two parameters:

- Channel Map: The bitmap indicating which channels are enabled.
- Hop Increment: This indicates the number of channels to hop for each subsequent hop.

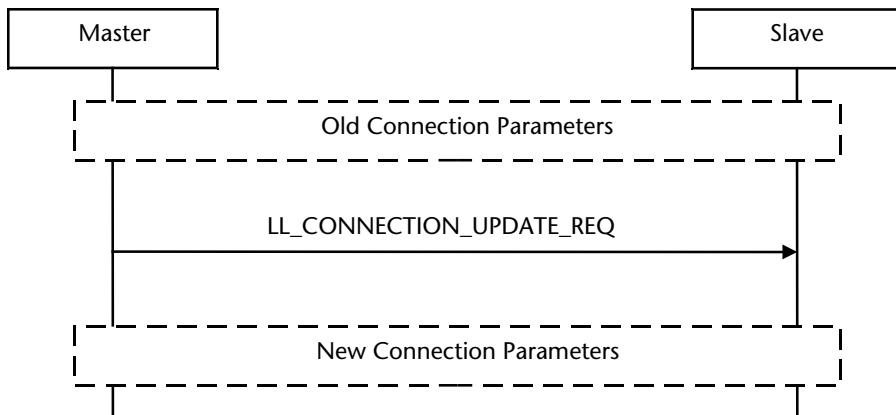


Figure 8.23 Connection Update Procedure.

This procedure allows the Master to disable frequency hopping on channels which potentially have interference thereby reducing the number of retransmissions that would have been required if the packets were transmitted on those channels. This is the key procedure that provides support for adaptive frequency hopping (AFH).

This procedure can only be initiated by the Master after entering the connection state. The sequence diagram for this procedure is shown in Figure 8.24.

8.12.3 Encryption Procedure

8.12.3.1 Encryption Start Procedure

The encryption start procedure is used by the link manager to enable encryption of packets. It is initiated by the host of the Master by sending a request to the link manager to start encryption. The host of the link manager also provides the Long_Term_Key (LTK). The Long_Term_Key is a 128-bit key used to generate the session key to be used for the encrypted connection. If the connection is not already encrypted, then the encryption start procedure is used.

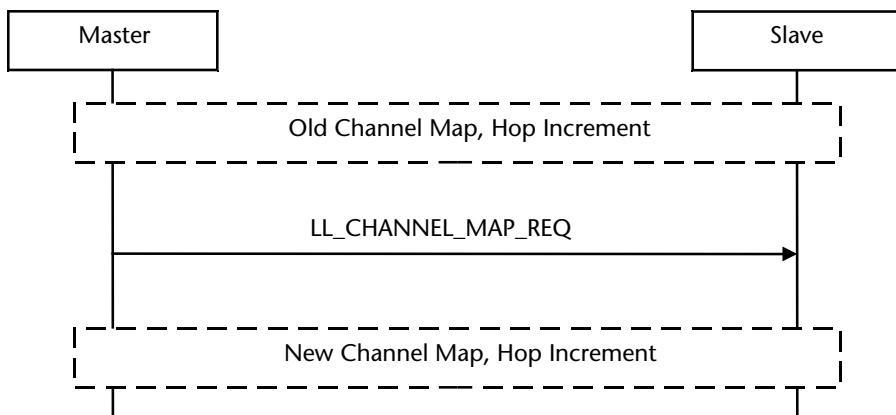


Figure 8.24 Channel Map Update Procedure.

If the connection is already encrypted, then the encryption pause procedure followed by encryption start procedure is used. The encryption pause procedure will be explained in the next section. The sequence diagram for this procedure is shown in Figure 8.25.

8.12.3.2 Encryption Pause Procedure and Encryption Restart Procedure

This procedure is used by the link manager if the link is already encrypted but a new encryption key is to be used without disconnecting the link. So, the link manager pauses encryption and then follows the same procedure as described in the previous section with the new encryption key.

Encryption restart procedure is very useful when the level of encryption has to be increased or decreased dynamically. So a link can be established with one level of security, and later, if the security needs to be increased it can be done using this procedure.

One example of this could be when a connection was established with a lower level of security but the security level needs to be increased (for example) to transmit some sensitive data. In that case, the encryption pause procedure is used followed by changing the link key and finally the encryption restart procedure.

The sequence diagram for this procedure along with the procedure to restart encryption with the new link key is shown in Figure 8.26.

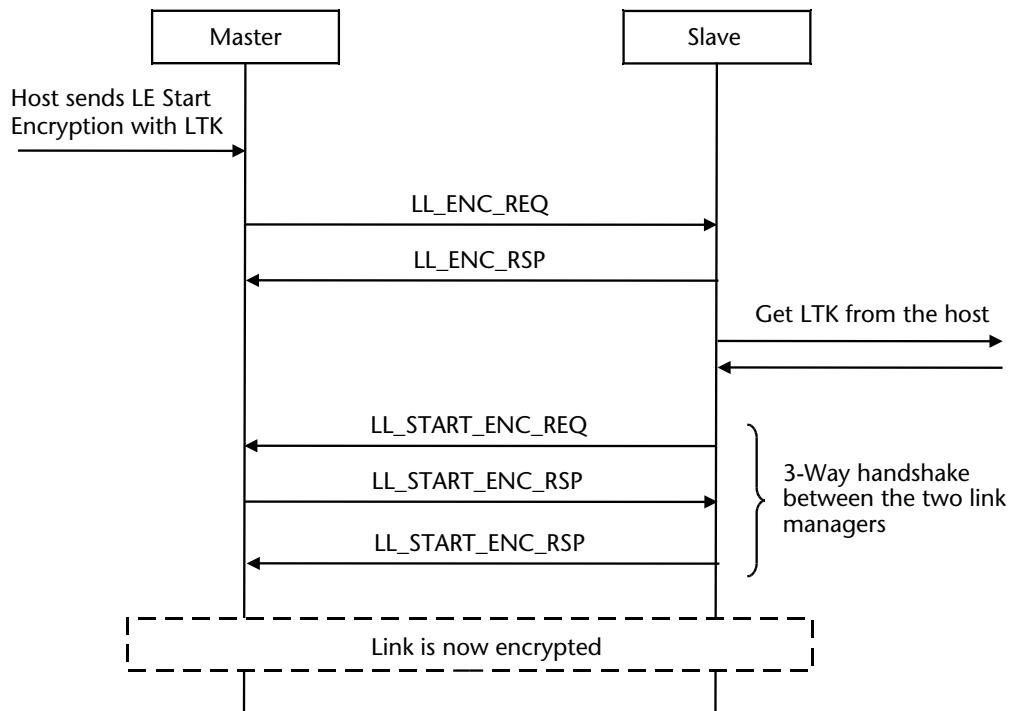


Figure 8.25 Encryption Start Procedure.

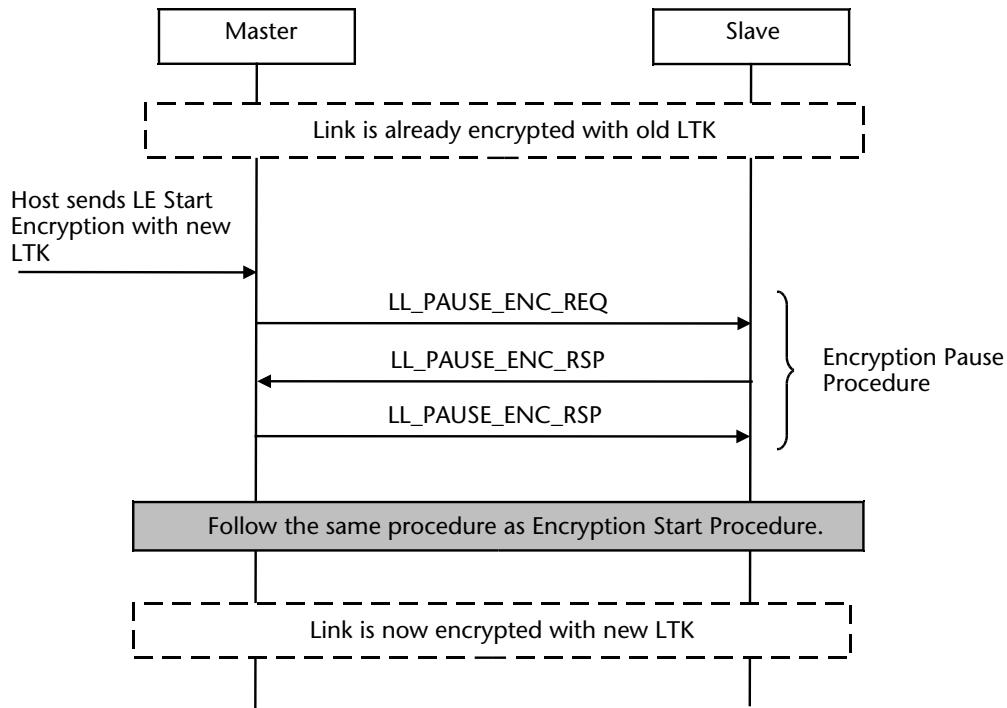


Figure 8.26 Encryption Pause Procedure and Encryption Restart Procedure.

8.12.4 Feature Exchange Procedure

The feature exchange procedure is used to exchange information about the current supported feature set.

Feature set is a bitmap that provides the feature capabilities of the Master or Slave. At present the feature set contains only information on whether encryption is supported or not. Other fields of this bitmap are reserved for future use. This procedure can only be initiated by the Master after entering the connection state. The sequence diagram for this procedure is shown in Figure 8.27.

8.12.5 Version Exchange Procedure

The version exchange procedure is used by either the Master or the Slave after entering the connection state to exchange version information. The version information consists of:

- Company ID;
- Link Layer Version;
- Sub Version Number.

The remote side responds with its own version information using the same PDU (LL_VERSION_IND) if it has not sent it in the past for the same connection. This procedure can be initiated by either the Master or the Slave after entering the connection state. The sequence diagram for this procedure is shown in Figure 8.28.

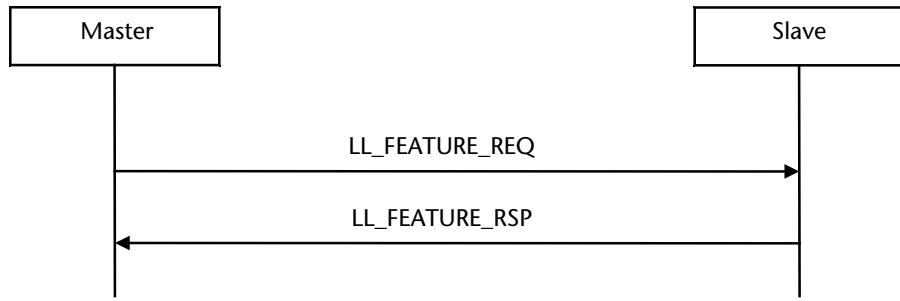


Figure 8.27 Feature Exchange Procedure.

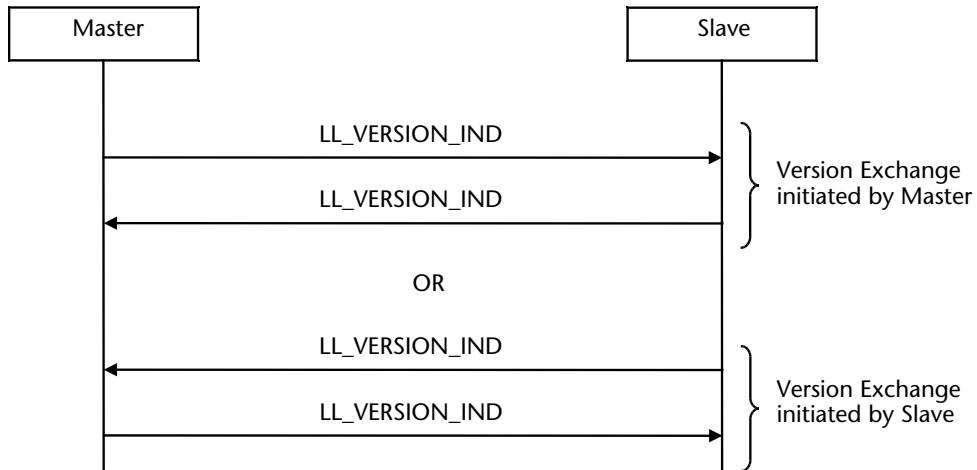


Figure 8.28 Version Exchange Procedure.

8.12.6 Termination Procedure

The termination procedure is used by either the Master or the Slave to terminate the current connection. This procedure can be used after entering the connection state. The sequence diagram for this procedure is shown in Figure 8.29.

8.12.7 Device Filtering and White List

One of the important power savings mechanisms introduced by LE is device filtering. With the use of device filtering, the link manager can be restricted to respond to only a certain set of devices. This is done through white lists that are maintained by the link layer. This is nothing but the set of devices that the link layer responds to (advertisers, scanners or initiators).

The transmissions from devices that are not in the white list are simply ignored. This reduces the number of transmissions made by the link manager thereby reducing the power consumption of the controller. In addition to this, it also reduces the communication the controller has with the host thereby reducing both the host's and controller's power consumption.

Besides power savings, the white list also eases the host processing in use cases where a host may wish to establish a connection with any one of the devices

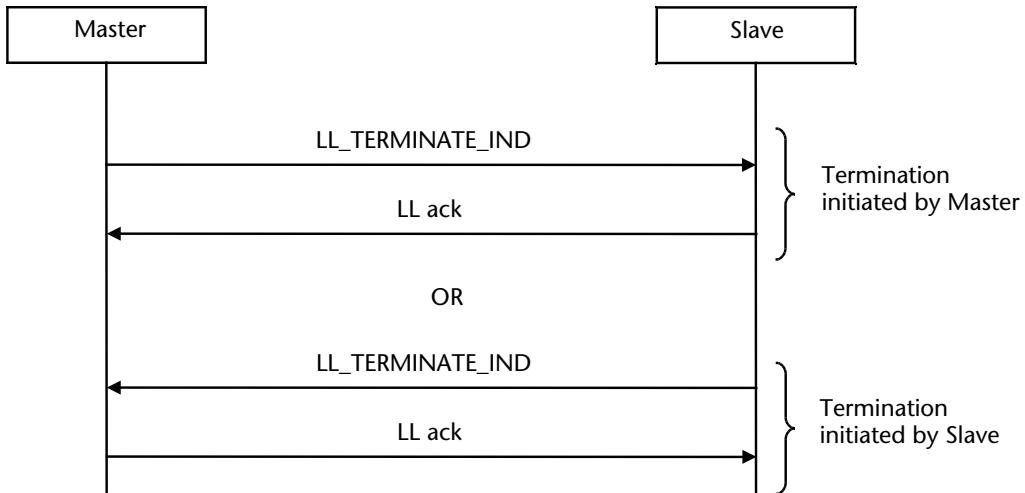


Figure 8.29 Termination Procedure.

amongst a list of devices (for example, if those devices provide similar data). At the time of LE connection, the host can specify a list of devices to connect to using white list. The controller connects to one of the available devices. The address to which the connection was established is returned in the `Peer_Address` parameter of `LE_Connection_Complete` Event. White List is very useful in such a scenario because the host does not have to repeatedly attempt making connections to each of the devices till the time it is able to successfully make a connection. This eases the host processing and lowers power consumption by reducing the communication between the host and the controller. The white list is initialized to empty by the controller at the time of reset. There are three filter policies that make use of the white lists. These are described below.

8.12.7.1 Advertising Filter Policy

This policy determines how the link layer of the Advertiser processes scan and connection requests.

There are four possible modes and one of these can be selected by the host by using the HCI command `HCI_LE_Set_Advertising_Parameters`.

- Process scan and connect requests only from devices in white list.
- Process scan and connect requests from all device (White list not in use). This is the default on reset.
- Process scan request from all devices but connect request from only devices in the white list.
- Process connect request from all devices but scan request from only devices in the white list

This policy is used only when the link layer is not using connectable directed advertising. In the case of connectable directed advertising, the Advertiser ac-

cepts the scan or connect request only from the device which it addresses in the advertising events. So this policy is not needed.

8.12.7.2 Scanner Filter Policy

This policy determines how the Scanner's link layer processes advertising packets. There are two possible modes and one of these can be selected by the host by using the HCI command HCI_LE_Set_Scan_Parameters.

- Process advertising packets only from devices in the white list.
- Process all advertising packets. (White list is not in use). This is the default on reset.

8.12.7.3 Initiator Filter Policy

This policy determines how the Initiator's link layer processes advertising packets. There are two possible modes and one of these is selected at the time of creation of a connection using the HCI_LE_Create_Connection command.

- Process connectable advertising packets from all devices in the white list.
- Process connectable advertising packets from a specific single device specified by the host. (White list is not in use). This is the default on reset.

8.13 Practical Examples

Figure 8.30 shows an air log capture of the transactions happening between the link layers of two devices to support the Proximity profile. These procedures are carried out after a connection has been established between the two devices.

The sequence of procedures that are carried out is as follows:

- Frames #676 and #677: The link layer of the Master and Slave exchange information about the versions that they support.
- Frames #702 to #708: These frames are used to start encryption on the link.
- Frames #2628, #2985, #3578: The link layer of the Master provides updated Channel Map to the Slave.
- Frame #4785: The Slave requests the link to be terminated.

8.14 Summary

As the name suggests, the link layer is responsible for the maintenance of the link. This includes establishing the link, selecting the frequencies, supporting different topologies and disconnecting the link. LE uses a very simple architecture for link manager with just five states. It imposes several restrictions in order to make the

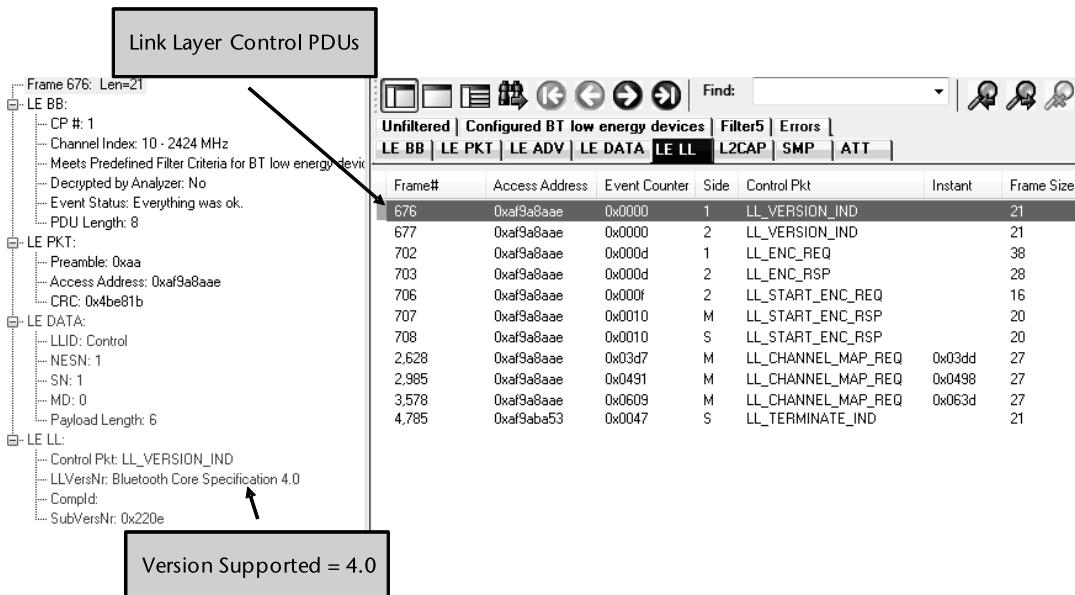


Figure 8.30 Practical example of Link Layer Transactions.

link manager state machine very simple, thereby reducing both the silicon cost of implementation as well as the power consumption.

LE uses dedicated channels for advertisement and data. The PDUs that can be exchanged between the link managers of the two devices were explained in detail in this chapter.

The next chapter will focus on HCI interface. The HCI interface is used by the upper layers to interface with the link manager.

Bibliography

Bluetooth Core Specification 4.0 <http://www.bluetooth.org>.

Bluetooth Low Energy Training and Marketing information from the Bluetooth SIG website.
<http://www.bluetooth.org>.

Host Controller Interface and Commands

9.1 Introduction

The host controller interface (HCI) provides a standard method of communication between the upper and lower layers of the protocol stack. In many implementations, the upper layers generally reside on a host and the lower layers reside on a separate Bluetooth controller chip. The HCI interface provides a communication mechanism between the host and the Bluetooth controller. The position of HCI interface in the LE protocol stack is shown in Figure 9.1.

The HCI was explained in detail in Chapter 3. LE reuses the specification of the HCI layer for BR/EDR and extends it with commands related to Low Energy. The LE controllers implement a reduced set of HCI commands and events that are only related to Low Energy while Dual mode controllers may implement both BR/EDR and LE commands.

Since the HCI layer is reused from the BR/EDR specification, it provides following major advantages:

1. All the code written for the HCI layer in BR/EDR can be reused with LE. This means the code to transmit commands, receive events, transmit, and receive data packets can be completely reused for LE. Only the support for LE specific commands and events needs to be added.
2. If the controller supports dual mode, then it is fully backward compatible with the BR/EDR controller. This means that a BR/EDR controller can be replaced with a dual mode controller without compromising on any existing BR/EDR functionality.
3. No software change is needed when replacing a BR/EDR controller with dual mode controller.

Before going further, it will be useful to read the sections related to Host Controller Interface in Chapter 3 because those are broadly applicable to LE as well. This chapter will provide details on the LE specific parts only.

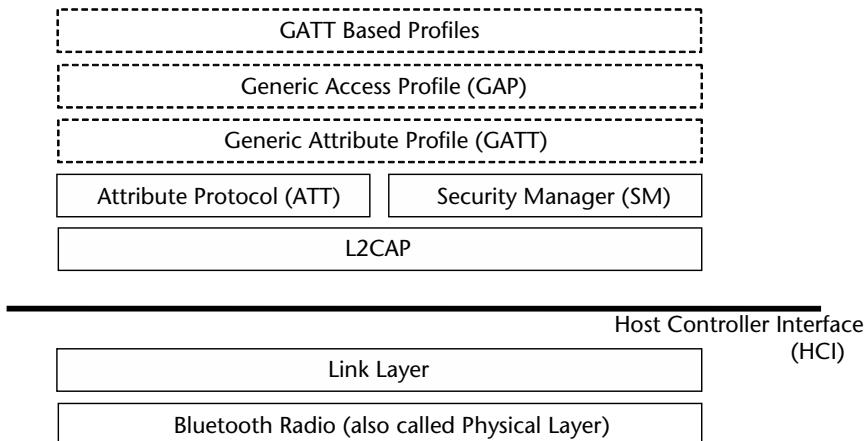


Figure 9.1 HCI in LE Protocol Stack.

9.1.1 HCI Packet Types

The different packet types that can be exchanged between the host and controller on the HCI interface were explained in Chapter 3. These are described here briefly for ease of reference. The format of HCI Command Packets and HCI ACL Data Packets is the same in the case of LE. The format of HCI Event Packets is slightly enhanced in the case of LE. All event packets are returned with the same event code (LE Meta Event) and a subevent code is used to identify the exact LE event. LE interface does not support Synchronous (SCO/eSCO) packets.

9.1.2 HCI Command Packets

The format of HCI Command packets was explained in Chapter 3. It is shown again in Figure 9.2. It consists of a 16-bit OpCode followed by an 8-bit Parameter Total Length field. The Parameter Total Length field specifies the total length of all parameters that are contained in the remaining packet measured in octets. This is followed by the command parameters. For the LE commands, the OGF (Opcode Group Field) is set to 0x08. The OGF occupies upper 6 bits of the opcode.

9.1.3 HCI Event Packet

The format of HCI Events Packets is shown in Figure 9.3. In the case of LE, all the events are encapsulated into one single event code called the LE Meta Event. The event code of LE Meta Events is 0x3E.

The Subevent code is used to identify the exact LE event that was generated by the controller. The remaining parameters depend on the type of the Subevent code.

9.1.4 HCI ACL Data Packet

The format of HCI ACL data packet is shown in Figure 9.4. All the data fields were explained in Chapter 3 and are valid for LE as well.

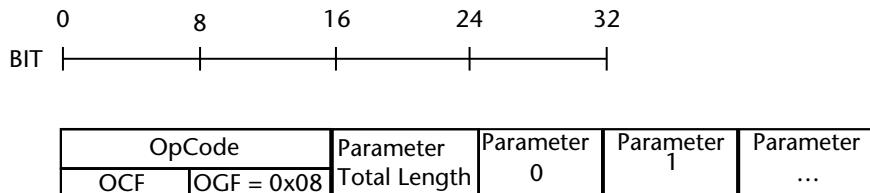


Figure 9.2 HCI Command Packet Format.

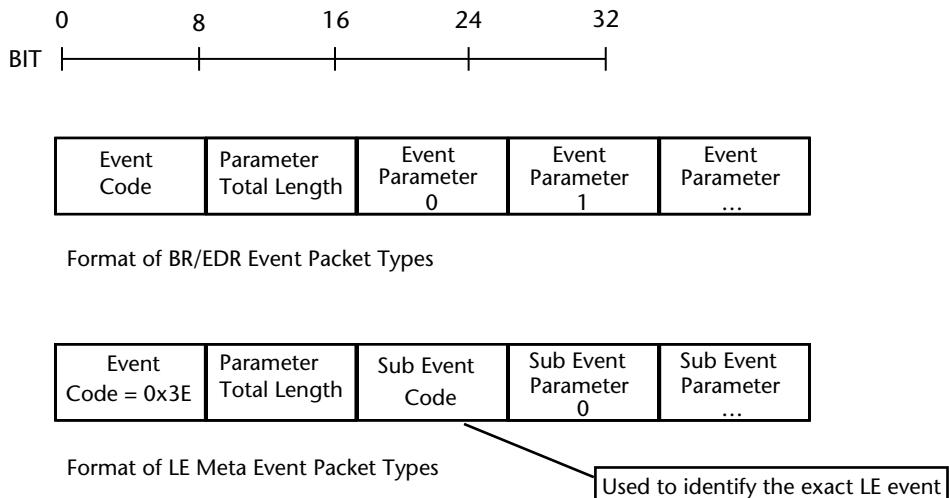


Figure 9.3 HCI Event Packet and LE Meta Event Packet Formats.

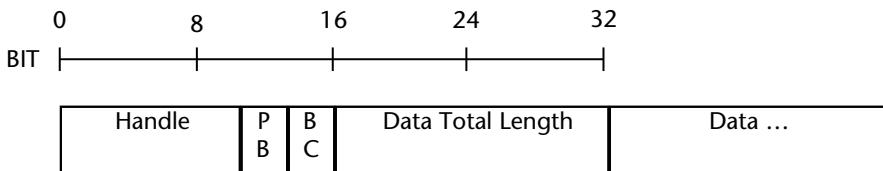


Figure 9.4 HCI ACL Data Packet.

9.2 HCI Commands and Events

The HCI commands are classified into several groups. The LE related commands in different groups are shown in Table 9.1. The commands that are applicable only to LE start with the HCI_LE prefix. The commands applicable to BR/EDR controllers as well as LE controllers start with HCI prefix. Similarly, the events specific to LE only start with an LE prefix while the events applicable to both BR/EDR and LE controllers do not have any prefix. LE specification tries to reuse the same HCI commands as BR/EDR wherever possible for LE functionality as well.

This section explains the various LE commands very briefly. The Bluetooth specification may be referred for a detailed explanation including the parameters of each command and event and the significance of each parameter.

Table 9.1 LE Related HCI Commands and Events

<i>Group</i>	<i>Commands</i>	<i>Events</i>
Device Setup	HCI_Reset	
Controller Flow Control	HCI_Read_Buffer_Size HCI_LE_Read_Buffer_Size	Number_Of_Completed_Packets_Event
Host Flow Control	HCI_Host_Buffer_Size HCI_Set_Event_Mask HCI_Set_Controller_To_Host_Flow_Control HCI_Host_Number_Of_Completed_Packets HCI_LE_Add_Device_To_White_List HCI_LE_Clear_White_List HCI_LE_Read_White_List_Size HCI_LE_Remove_Device_From_White_List HCI_LE_Set_Event_Mask	Data_Buffer_Overflow_Event
Controller Information	HCI_Read_Local_Version_Information HCI_Read_Local_Supported_Commands HCI_Read_Local_Supported_Features HCI_LE_Read_Local_Supported_Features HCI_LE_Read_Supported_States	
Remote Information	HCI_Read_Remote_Version_Information HCI_LE_Read_Remote_Used_Features	Read_Remote_Version_Information_Complete LE_Read_Remote_Used_Features_Complete
Controller Configuration	HCI_LE_Set_Advertise_Enable HCI_LE_Set_Advertising_Data HCI_LE_Set_Advertising_Parameters HCI_LE_Set_Random_Address HCI_LE_Set_Scan_Response_Data HCI_Read_LE_Host_Support HCI_Write_LE_Host_Support	
Device Discovery	HCI_LE_Set_Scan_Enable HCI_LE_Set_Scan_Parameters	HCI_LE_Advertising_Report_Event
Connection Setup	HCI_Disconnect HCI_LE_Create_Connection_Cancel HCI_LE_Create_Connection	Disconnect_Complete_Event LE_Connection_Complete_Event
Connection State	HCI_LE_Connection_Update	LE_Connection_Update_Complete_Event
Physical Links	HCI_LE_Set_Host_Channel_Classification	
Link Information	HCI_Read_Transmit_Power_Level HCI_Read_RSSI HCI_LE_Read_Advertising_Channel_Tx_Power HCI_LE_Read_Channel_Map	
Authentication and Encryption	HCI_LE_Encrypt HCI_LE_Long_Term_Key_Request_Reply HCI_LE_Long_Term_Key_Request_Negative_Reply HCI_LE_Rand HCI_LE_Start_Encryption	Encryption_Change_Event Encryption_Key_Refresh_Complete_Event HCI_LE_Long_Term_Key_Requested_Event
Testing	HCI_LE_Receiver_Test HCI_LE_Transmitter_Test HCI_LE_Test_End	

Some of the HCI commands and events are explained in further details below.

9.2.1 Device Setup

The device setup commands are used to initialize the controller and put it in a known state. Generally these are among the first commands sent to the controller.

There is only one command in this group for LE:

- **HCI_Reset:** This command is used to reset the controller. It also resets the link layer and puts it in the standby state with default values for all parameters for which default values are defined. Once the reset of the controller is complete, it sends back a **Command_Complete_Event** event back to the host.

9.2.2 Controller Flow Control

The controller flow control commands and events are used to control the data flow from the host to the controller. The buffers in the controller can either be separate for BR/EDR and LE or combined.

- If the buffers are separate, then the **HCI_LE_Read_Buffer_Size** command returns the number of LE buffers and **HCI_Read_Buffer_Size** returns the number of BR/EDR buffers.
- If the buffers are shared, then **HCI_LE_Read_Buffer_Size** returns 0 as the length of ACL data packets and **HCI_Read_Buffer_Size** is used to read the number of shared buffers.

Some of the commands and events in this group are:

- **HCI_Read_Buffer_Size:** This command is used at the time of initialization to find out the number of ACL data packet buffers in the controller, size of each buffer etc. The host can send as many data packets to the controller as the number of buffers reported by the controller. Then it has to wait for a **Number.Of.Completed.Packets.Event** to find out how many packets were processed. This gives an indication of how many buffers in the controller got freed up. After that it can send those many more ACL data packets.
- **HCI_LE_Read_Buffer_Size:** This command is used to find out the total number of LE ACL data buffers in the controller and the maximum size of each buffer.
- **Number.Of.Completed.Packets.Event:** This event is sent by the controller to indicate the number of packets that it has processed (either transmitted or flushed). This indicates to the host that those buffers have been emptied and the host can send more data to the controller. This provides a flow control mechanism between the host and the controller.

The Host to Controller Data Flow Control was explained in Chapter 3 for BR/EDR. The Data Flow Control for LE is also similar. This is shown in Figure 9.5.

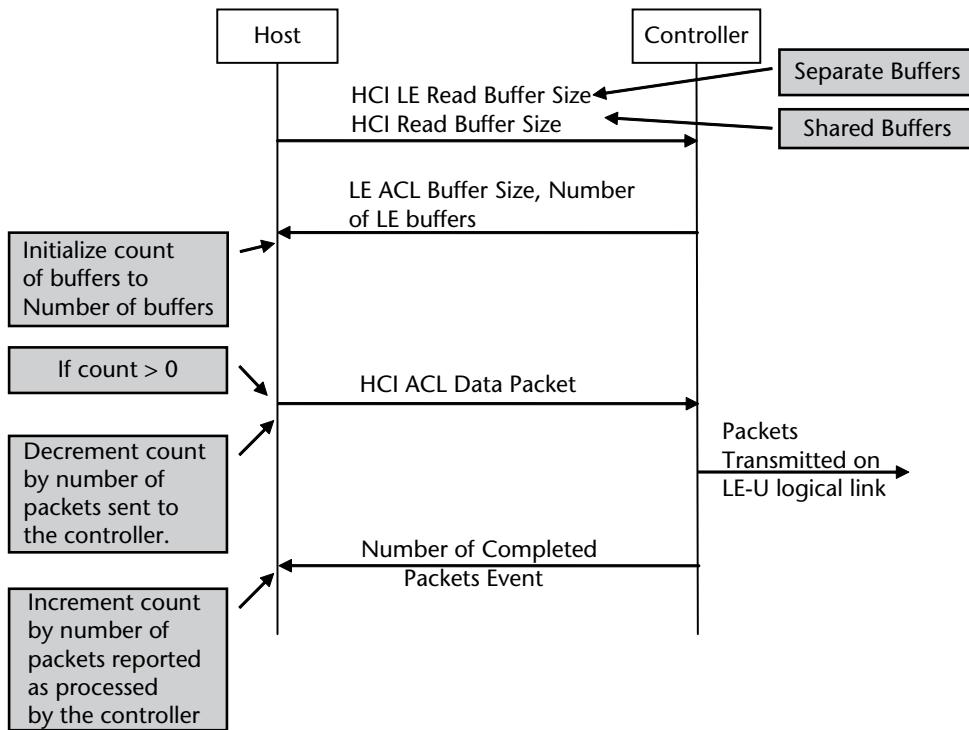


Figure 9.5 Host to controller data flow control.

During initialization the host sends the HCI_LE_Read_Buffer_Size command to the controller to get information about the LE buffers in the controller. The controller returns the following parameters:

1. Size of each LE ACL buffer (HC_Le_ACL_Data_Packet_Length).
2. Number of LE ACL buffers (HC_Total_Num_Le_ACL_Data_Packets).

If the controller returns the first parameter as 0, then it means that the controller is using shared ACL buffers for BR/EDR and LE. Subsequently, the HCI_Read_Buffer_Size command can be used to find the length and number of these shared buffers. Once the host has information on the number of ACL buffers, it knows that at any given time, it can have a maximum of that many packets outstanding (to be processed) on the controller side. For example if the Number of ACL buffers was four, then the host can have a maximum of four packets outstanding on the controller side.

The host maintains a count of the number of ACL buffers available in the controller. It initializes this value to the number of ACL buffers it received in the response to Read Buffer Size command. After that, every time it sends a packet to the controller, it decreases this count by one. Once the controller has completed processing one or more packets, it sends that count in the Number of Completed Packets event. The host knows that some additional buffers have been freed up on the controller side and it increases its count by the number of packets reported in the Number of Completed Packets event.

9.2.3 Host Flow Control

The host flow control commands and events allow the flow control to be used towards the host. In most implementations host flow control is not used. This may still be used if the host has a slow processor or limited memory space.

The following commands are included in this category:

- **Host_Buffer_Size_Command:** This command is used by the host to provide the controller information about the data buffers present in the host.
- **Set_Controller_To_Host_Flow_Control_Command:** This command is used to turn the flow control on and off for the data flowing from the controller to the host.
- **Host_Number_Of_Completed_Packets_Command:** This command is used by the host to indicate to the controller the number of packets that it has processed. The controller can use this information to send additional packets to the host.
- **Data_Buffer_Overflow_Event:** This event is used by the controller to indicate that its data buffers have overflowed because the host has sent more packets than the number of free buffers in the controller.
- **Set_Event_Mask:** This command is used to configure which events are generated by the controller for the host.

The following commands for the white list are also included in this category:

- **HCI_LE_Read_White_List_Size:** This command is used to read the total number of white list entries that can be stored in the controller.
- **HCI_LE_Clear_White_List:** This command is used to clear the whole white list.
- **HCI_LE_Add_Device_To_White_List:** – This command is used to add a device to the white list.
- **HCI_LE_Remove_Device_From_White_List:** This command is used to remove a device from the white list.

The white lists were explained in Chapter 8 and some of the use case scenarios of white lists are shown at the end of this chapter.

9.2.4 Controller Information

The controller information commands are used by the host to find out the information about the controller.

Some of the commands in this category are:

- **HCI_Read_Local_Version_Information:** This command reads the values of version information from the controller. It can be used to detect whether the controller supports LE functionality or not.

- **HCI_Read_Local_Supported_Command:** This command reads a bit mask indicating which of the HCI commands are supported by the controller.
- **HCI_Read_Support_Features_Command:** This command reads a bit mask indicating which of the features are supported by the controller.
- **HCI_LE_Read_Support_Features_Command:** This command reads a bit mask indicating which of the LE features are supported by the controller. At present it contains only one bit indicating whether the controller supports encryption or not. It may be extended in further versions of the specification.
- **HCI_LE_Read_Supported_States:** This command is used to retrieve the set of states and state combinations that the controller supports. For example this command is used to know whether directed advertising state is supported.

9.2.5 Remote Information

The remote information commands and events allow the device to find out information about the remote device's configuration.

Some of the commands and events in this category are:

- **HCI_LE_Read_Remote_Used_Features:** This command requests a list of used features from the remote device. At present the used features bit mask only provides information whether encryption is supported on the remote side or not.
- **LE_Read_Remote_Used_Features_Complete:** This event is generated once the reply from the remote side is received. It reports back to the host the bitmap of features that the remote side supports.

9.2.6 Controller Configuration

The controller configuration commands are used by the host to configure the controller.

Some of the commands in this category are:

- **HCI_LE_Set_Advertise_Enable:** This command is used to request the controller to start or stop advertising.
- **HCI_LE_Set_Advertising_Data:** This command is used to set the data used in advertising packets. A maximum of 31 bytes of advertising data can be included.
- **HCI_LE_Set_Advertising_Parameters:** This command is used to set various parameters related to advertising like the type of advertising, which channels to advertise on, minimum and maximum advertising interval, etc.
- **HCI_LE_Set_Random_Address:** This command is used to set the random address.
- **HCI_LE_Set_Scan_Response_Data:** This command is used to provide data in scanning packets.

- **HCI_Read_LE_Host_Support:** This bit is used to read the current setting of the following two bits in the controller:
 - LE_Supported_Host.
 - Simultaneous LE and BR/EDR to same device capable host.
- **HCI_Write_LE_Host_Support:** This bit is used to indicate that the host supports LE and whether it supports simultaneous LE and BR/EDR links to the same device. The local host sets these bits in the controller to indicate to remote devices about its LE capabilities:
 - LE_Supported_Host.
 - Simultaneous LE and BR/EDR to same device capable host.

9.2.7 Device Discovery

The device discovery commands and events allow the device to discover other devices in the vicinity.

Some of the commands and events in this group are:

- **HCI_LE_Set_Scan_Enable:** This command is used to enable and disable scanning. Scanning is used to discover devices in the vicinity.
- **HCI_LE_Set_Scan_Parameters:** This command is used to set the parameters like scan type, scan interval, etc.
- **HCI_LE_Advertising_Report_Event:** This event is used to indicate to the host that an advertising report has been received. Multiple reports can be sent in one single event.

9.2.8 Connection Setup

The connection setup commands and events allow a device to make a connection to another device.

Some of the commands and events in this group are:

- **HCI_LE_Create_Connection:** This command is used to create a connection to an Advertiser which is connectable.
- **LE_Connection_Complete_Event:** This event is received on both the Master and Slave side once a connection has been created.
- **HCI_Disconnect:** This command is used to terminate an existing connection.
- **Disconnection_Complete_Event:** This event is used to indicate that a connection has been terminated.

9.2.9 Connection State

The connection state commands and events allow the host to configure the link.

Some of the commands and events in this group are:

- **HCI_LE_Connection_Update:** This command is used to change the parameters related to a connection, like connection interval, supervision timeout, etc.
- **LE_Connection_Update_Complete_Event:** This event is used to indicate to the host that the parameters related to a connection have been changed.

9.2.10 Physical Links

The physical link commands and events allow the configuration of a physical link.
There is only one command related to LE in this group:

- **HCI_LE_Set_Host_Channel_Classification:** This command is used by the host to provide a channel map to the controller. The channel map indicates to the controller which of the 37 data channels it can use.

9.2.11 Link Information

The link information commands and events allow the host to read information about a link.

Some of the commands in this group are:

- **HCI_Read_Transmit_Power_Level:** This command is used by the host to get the value of the transmit power level used to transmit for a given connection handle.
- **HCI_Read_RSSI:** This command reads the Received Signal Strength Indicator (RSSI) value from a controller.
- **HCI_LE_Read_Advertising_Channel_Tx_Power:** This command is used to read the transmit power level that is used for advertising channel packets.
- **HCI_LE_Read_Channel_Map:** This command is used to read the current channel map used for the specified connection handle.

9.2.12 Authentication and Encryption

The authentication and encryption commands and events allow authentication of a remote device and then encryption of a link.

Some of the commands and events in this group are:

- **HCI_LE_Encrypt:** This command is used by the host to request the controller to encrypt some data. The host provides the key and data to be encrypted. The controller encrypts the data and returns the encrypted data in a command complete event.
- **HCI_LE_Long_Term_Key_Requested_Event:** This event is used by the controller to request the Long Term Key from the host.
- **HCI_LE_Long_Term_Key_Request_Reply:** This command is used by the host to provide the Long Term Link Key.

- **HCI_LE_Long_Term_Key_Request_Negative_Reply:** This command is used by the host to refuse to provide the Long Term Link Key.
- **HCI_LE_Rand:** This command is used by the host to request to the controller to generate an 8 octet random data and send to the host.
- **HCI_LE_Start_Encryption:** This command is used to request the controller to encrypt a connection.
- **Encryption_Change_Event:** This event is used to indicate that the change of encryption mode has been completed.
- **Encryption_Key_Refresh_Complete_Event:** This event is used to indicate to the host that the encryption key was refreshed on the given connection handle.

9.2.13 Testing

The testing commands allow the controller to be put in a special test mode so that testing can be performed. These consist of commands to test the receiver and the transmitter.

Some of the commands in this group are:

- **HCI_LE_Receiver_Test:** This command is used to test receiver functionality of the device. The remote tester generates reference test packets which are received by the receiver.
- **HCI_LE_Transmitter_Test:** This command is used to request the controller to start transmitting test reference packets. The remote tester can read and verify these packets.
- **HCI_LE_Test_End:** This command is used to stop any test which is in progress.

9.2.14 Usage of White Lists

The commands to reset and set the devices in the white list were described earlier. Once a white list has been set, the following commands are used to enable the use of white list:

- The **HCI_LE_Set_Advertising_Parameters** command takes the parameter **Advertising_Filter_Policy** which is used to select whether to allow scan and connect requests from all devices or just the devices present in White List.
- The **HCI_LE_Set_Scan_Parameters** command takes the parameter **Scanning_Filter_Policy** which is used to select whether to accept advertising packets from all devices or just devices present in the white list.
- The **HCI_LE_Create_Connection** command takes the parameter **Initiator_Filter_Policy** which is used to determine which Advertiser to connect to.

9.3 Practical Sequence Diagrams

This section provides sequence diagrams for some of the commonly used procedures. The interaction of the link manager with the host, along with the corresponding HCI Commands and Events, is also shown to provide the complete view on how each of the procedures are used in practical scenarios. In the interest of simplicity, the command status and command complete events are not shown. These will also be generated when these commands are sent out in practice.

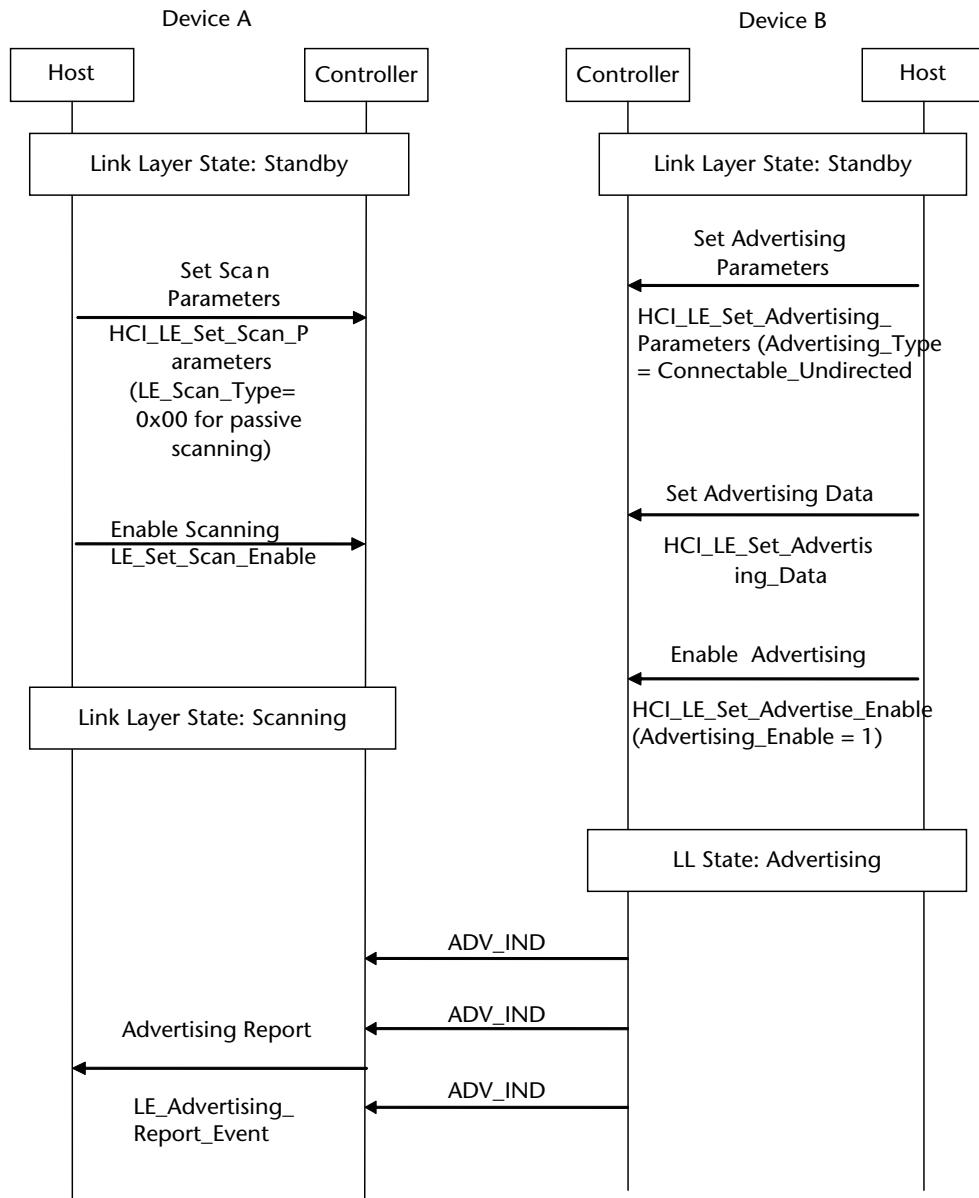


Figure 9.6 Typical sequence for passive scanning.

9.3.1 Passive Scanning

A typical sequence for passive scanning is shown in Figure 9.6.

9.3.2 Typical Sequence for Active Scanning

A typical sequence for active scanning is shown in Figure 9.7.

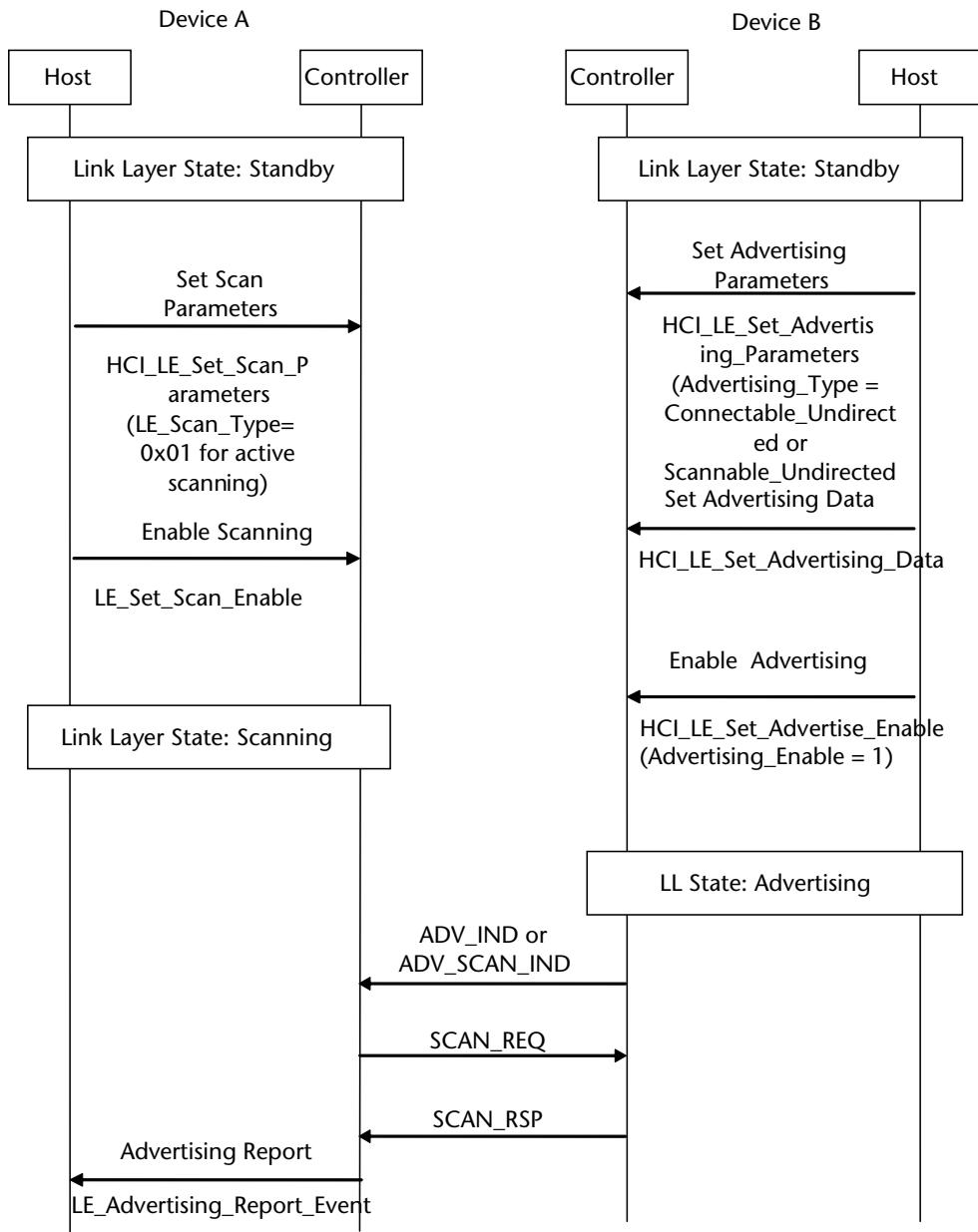


Figure 9.7 Typical sequence for active scanning.

9.3.3 Connection Establishment

A typical sequence for connection establishment is shown in Figure 9.8.

9.3.4 Setting up White list

Figure 9.9 shows a typical sequence for setting up a white list and then various scenarios in which a white list may be used.

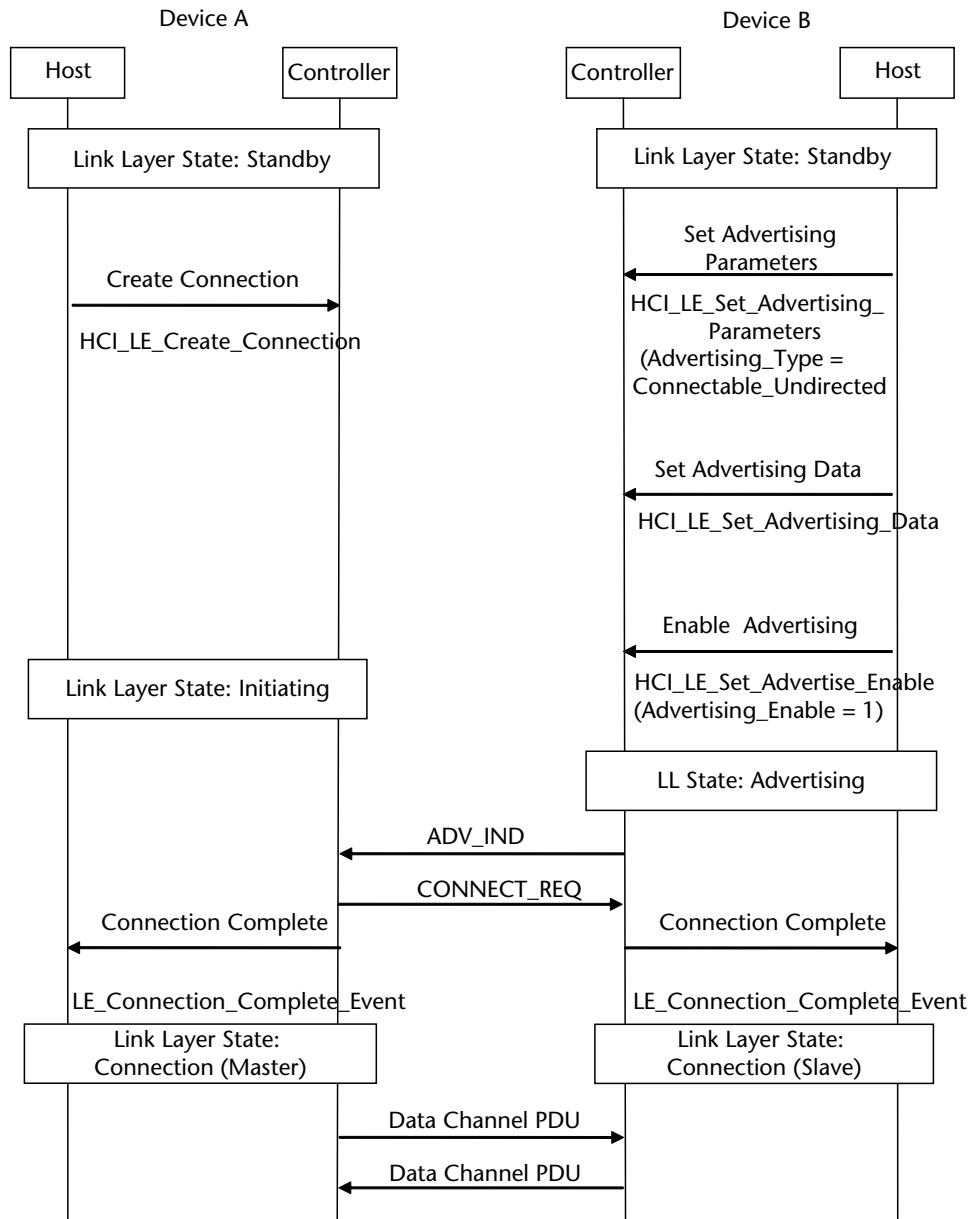


Figure 9.8 Typical sequence for connection establishment.

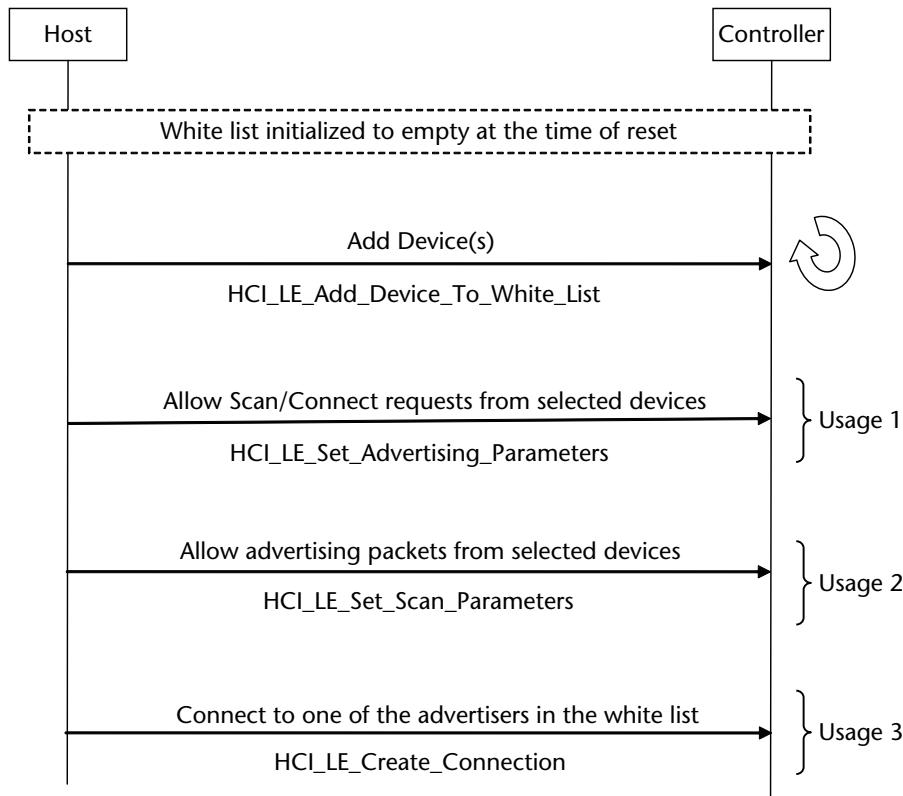


Figure 9.9 Usage of White Lists.

9.4 Summary

The host controller interface provides a communication interface between the upper layers and lower layers. LE specification has reused the HCI layer and enhanced it to add support for LE related commands and events. This chapter explained many of the LE related commands and events. It also provided sequence diagrams for some of the typical use cases of LE.

This chapter completes the information about the LE lower layers. Subsequent chapters will focus on the LE upper layers and profiles.

Bibliography

Bluetooth Core Specification 4.0 <http://www.bluetooth.org>.

Logical Link Control and Adaptation Protocol (L2CAP)

10.1 Introduction

The Logical Link Control and Adaptation Protocol (L2CAP) layer acts as an interface between the higher layer protocols and the lower layers. The position of L2CAP in the LE protocol stack is shown in Figure 10.1.

L2CAP for BR/EDR was explained in detail in Chapter 4. LE reuses the L2CAP functionality of BR/EDR and simplifies the functionality in a major way to make it suitable for LE devices. A significant part of the L2CAP functionality is not needed for LE devices. That functionality has been removed in order to keep the implementation small and simple.

The Bluetooth specification defines the HCI interface as an optional interface. In case of systems which include the HCI interface, the L2CAP layer uses the HCI layer to send data by encapsulating it into HCI ACL Data packets. In the systems in which the HCI layer is not present, the L2CAP layer invokes the functionality of Link Layer directly (maybe through some API mechanism) to send the packets.

Before going further, it will be useful to read the sections related to L2CAP in Chapter 4 because those are broadly applicable to LE as well. This chapter will provide details on the LE specific modifications to L2CAP.

10.2 PDU and SDU

An SDU (Service Data Unit) is a packet that contains data originating from the upper layers (For example Attribute protocol). The L2CAP entities transfer this SDU transparently from the upper layer of one side to the upper layer of the other side.

A PDU (Protocol Data Unit) is a packet containing L2CAP protocol information and may contain data from upper layers as well (SDU). So an SDU will be encapsulated into an L2CAP PDU before transmitting to the remote side. The remote side L2CAP will extract the header information and provide the SDU to the upper layers on the remote side. All SDUs are encapsulated into one or more L2CAP PDUs. The PDU and SDU are illustrated in Figure 10.2.

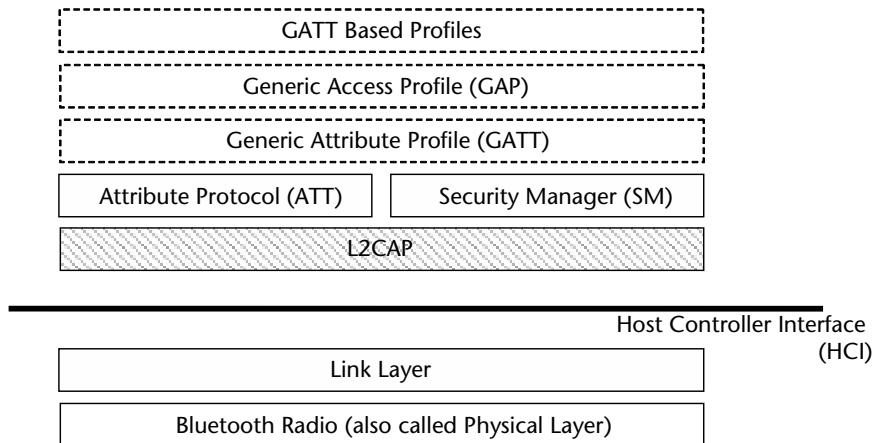


Figure 10.1 L2CAP in LE Protocol Stack.

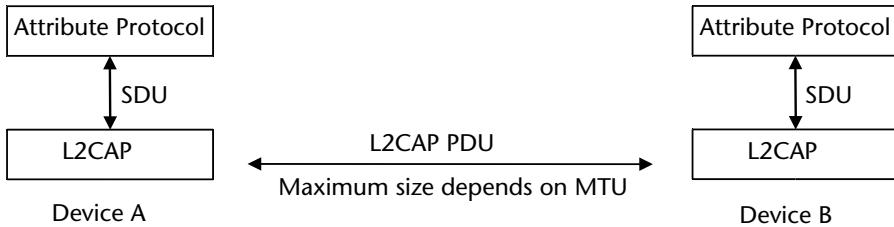


Figure 10.2 PDU and SDU.

10.3 Basic Assumptions

The L2CAP protocol is designed with the following basic assumptions about the controller:

1. The packets are delivered in the correct sequence by the controllers on both sides. This means that the packet which was transmitted first by the host on the transmitter side will be received first by the host on the receiver side.
2. Only one LE-U logical link exists between the two devices.
3. The controllers provide a degree of reliability by including error detection, and retransmission mechanisms.
4. The controllers provide flow control mechanisms for data going over the air as well as data going over the HCI transport layer. This ensures that data does not get overwritten at any stage.

10.4 Maximum Transmission Unit (MTU)

The MTU is used to inform the other side the maximum size of SDU it is capable of accepting. The minimum MTU for LE is 23 octets. This means that an LE device will certainly receive a packet of 23 octets, but, it may receive a bigger packet as well. It is possible that the two devices which are connected may have different

MTU sizes. The sender always keeps the MTU size of the receiver in mind when sending a packet and does not exceed that MTU size. In case the sender transmits a packet that exceeds the MTU size of the receiver, the receiver sends back a Command_Reject to indicate that it cannot accept this packet.

The minimum supported MTU for LE is much smaller than that of BR/EDR. In the case of BR/EDR, the minimum supported MTU is 48 octets if extended flow specification is not supported and 672 if extended flow specification is supported. The smaller MTU size of 23 octets in the case of LE helps to keep the packet sizes of LE smaller, thus, leading to savings of buffer space needed for transmit/receive and the power consumed in exchanging the packets.

10.5 L2CAP Features

Similar to BR/EDR, L2CAP provides the abstraction of channels to layers on top of it. It provides the following features:

1. Fixed Channel Identifiers.
2. Fragmentation and Defragmentation of data.
3. Multiplexing and Demultiplexing of various channels over a shared logical link (LE-U).

10.5.1 Fixed Channel Identifiers

As explained in Chapter 4, L2CAP is based on the concept of channels. A channel identifier (CID) is a local name representing a logical channel end point on the device. All channels going over an LE physical link are mapped to a single LE-U logical link.

In the case of BR/EDR, CID 0x0001 is fixed for signaling and CID 0x0002 is fixed for connectionless data. The remaining CIDs are dynamically allocated. So, for example, if SDP layer wants to create a connection, L2CAP dynamically assigns a CID to it.

In the case of LE, the CID assignment is simplified by using only fixed CIDs. The fixed CIDs for LE are shown in Table 10.1. CID 0x0005 is fixed as the L2CAP signaling channel. Only two protocols can use the services of L2CAP layer in the case of LE, viz Attribute Protocol and Security Manager Protocol. Both of these are assigned fixed CIDs. This is very much simplified as compared to BR/EDR where the CIDs for various protocols are allocated dynamically.

All these channels are available as soon as the LE-U logical link is setup. The higher layers can start sending data to L2CAP for these channels that is transmitted to the remote device. So ATT protocol can send data over CID 0x0004 as soon as

Table 10.1 CID Name Space for LE

CID	Description
0x0004	Attribute Protocol. Attribute Protocol will be covered in detail in later chapters.
0x0005	LE L2CAP Signaling Channel.
0x0006	Security Manager Protocol (SMP). SMP will be covered in detail in later chapters.

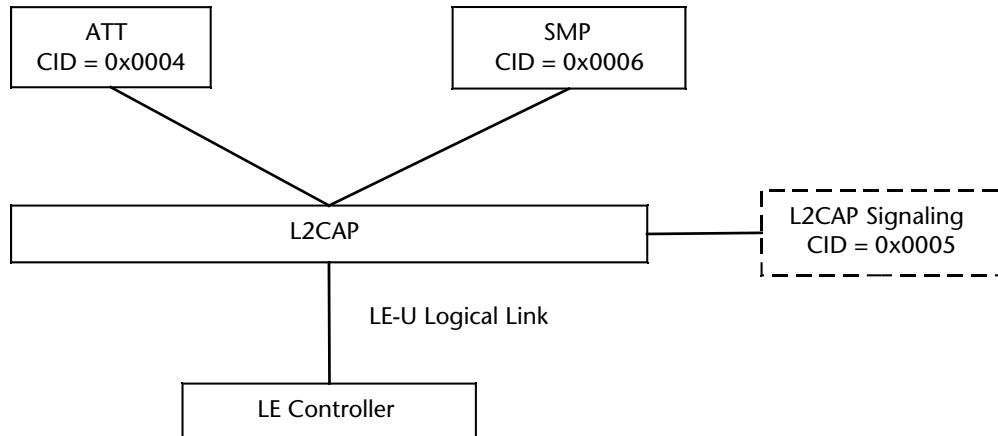


Figure 10.3 Channel Multiplexing.

the link layer creates a connection to the remote device. This is much simpler than, for example, SDP sending data over L2CAP in the case of BR/EDR. In the case of BR/EDR the Connect and Configure signals need to be exchanged between the two devices to establish an L2CAP connection before any data can be sent. This is one of the significant enhancements made in LE toward faster connection setup time. The number of PDUs exchanged between the two devices to setup the connection before transferring higher layer data is reduced to zero in the case of L2CAP.

10.5.2 Fragmentation and Defragmentation of Data

L2CAP allows higher layer protocols to send bigger chunks of data to it even though the LE PDUs at the link layer are restricted to much smaller sizes. In such cases, L2CAP fragments the data according to the size of the ACL buffers that are present for LE in the controller before sending it to the controller for transmission. At the time of reception, L2CAP re-assembles the data to reconstruct the original packet that was sent by the higher layers. It then sends the reconstructed packet to the higher layers.

10.5.3 Channel Multiplexing

As shown in Table 10.1, three L2CAP channels are defined in the case of LE:

1. ATT with CID = 0x0004.
2. SMP with CID = 0x0006.
3. L2CAP's own signaling channel with CID = 0x0005.

L2CAP performs the multiplexing and demultiplexing of these three channels on top of the shared LE-U logical link. This means that it accepts data from these three channels and multiplexes it on the shared link at the time of transmitting. At the time of receiving, it demultiplexes the data and provides the data to the appropriate higher layer entity.

10.6 Data Packets

As mentioned in Chapter 4, L2CAP supports five modes of operation for BR/EDR.

- Basic L2CAP Mode (used in LE as well);
- Flow Control Mode;
- Retransmission Mode;
- Enhanced Retransmission Mode;
- Streaming Mode.

Out of these five modes of operation, only the Basic Mode is used for LE. The PDU that is exchanged in the Basic L2CAP Mode is also referred to as a B-Frame. The format of B-Frame is shown in Figure 10.4.

The Length field indicates the size of the information payload. It can go up to 65535 bytes and is used during recombination when the different fragments are reassembled at the receiver side to reconstruct the whole packet. The Channel ID is 0x0004 for ATT and 0x0006 for SMP. The Information Payload contains the higher-layer data. So in the case of LE it contains the data that is sent by ATT or SMP.

As compared to BR/EDR, L2CAP supports only the Basic L2CAP mode. This simplifies the design of the L2CAP layer to a large extent while still not compromising on the functionality required by the layers that sit on top of L2CAP(ATT and SMP). This is another major step towards decreasing the cost and power consumption of LE devices.

10.7 L2CAP Parameters

Table 10.2 shows the L2CAP Parameters that are used by both ATT and SMP.

A Flush Timeout of 0xFFFF means that the baseband continues to do retransmissions until the link is dropped if the packet is not acknowledged.

LE requires only Best Effort service. This means that there is no guarantee that the data will be received by the remote side. (In the case of BR/EDR, L2CAP also



Figure 10.4 LE L2CAP Data Packet Format (B-Frame).

Table 10.2 L2CAP Parameters for ATT and SMP

Parameter	Value
MTU	23
Flush Timeout	0xFFFF (Infinite)
Quality of Service	Best Effort
Mode	Basic Mode

supports Guaranteed QoS option which guarantees a specific amount of bandwidth for the particular L2CAP channel. This option is not supported in the case of LE.)

10.8 L2CAP Signaling

As shown in Table 10.1 LE uses channel 0x0005 as the Signaling channel. The signaling channel is available as soon as the lower layer logical transport is set up and L2CAP traffic is enabled. The commands on the signaling channel are in the form of requests and responses.

LE simplifies the procedure for sending commands on the signaling channel. While on the BR/EDR signaling channel (0x0001) multiple commands can be sent within a single PDU, in the case of LE signaling channel (0x0005) only one command can be sent per PDU. This makes the logic for decoding the packets simpler on the receiving L2CAP entity.

The PDUs that contain L2CAP signaling messages are known as C-Frames (Control Frames). These are used only on the L2CAP Signaling channel. The format of C-frames is shown in Figure 10.5.

The code identifies the type of command that is being sent on the Signaling channel. The commands that are allowed on LE Signaling channel are shown in Table 10.3. This list is much smaller than the number of commands that are supported on the BR/EDR signaling channel. This allows the LE software to be simpler in terms of the number of commands that it needs to process.

The identifier field is used to match the responses with the requests. This field is set by the requesting device and the responding device uses the same value while responding. When the requesting device receives the response back, it can identify

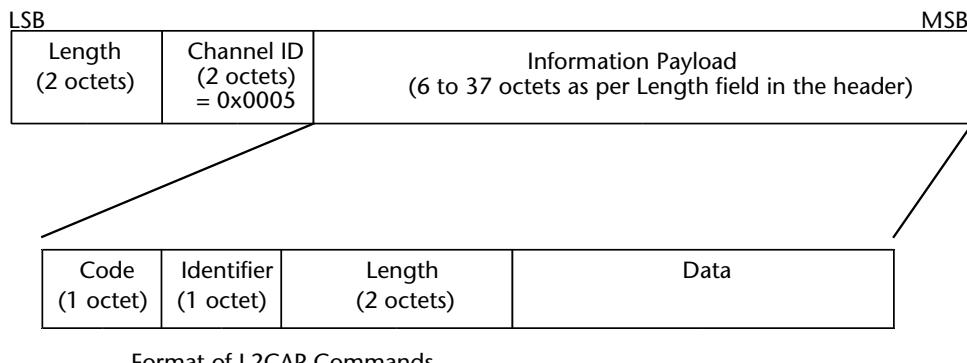


Figure 10.5 L2CAP PDU on signaling channel (C-Frame).

Table 10.3 L2CAP Signaling Commands

Code	Command	Direction
0x01	Command Reject	Both Directions
0x12	Connection Parameter Update Request	Slave to Master
0x13	Connection Parameter Update Response	Master to Slave

which request that response is for. The length field indicates the size of the Data field. The Data field is variable in length and the size depends on the command that is being sent.

10.8.1 Command Reject

The command reject PDU is sent as a response if the command code was not identified or the length was incorrect. It contains a reason field to indicate why the packet was rejected. The reason can be any of the following:

- Command not understood;
- Signaling MTU exceeded;
- Invalid CID in request.

10.8.2 Connection Parameter Update Request

This packet is sent by the LE Slave to the LE Master to request a set of new connection parameters. The connection parameters include:

- Interval Min: The minimum value for the connection event interval.
- Interval Max: The maximum value for the connection event interval.
- Slave Latency: This defines the Slave latency of the connection in number of connection events. For example if Slave Latency is 4, then it will listen to a packet from the Master on every 4th anchor point. If it is 0, then the Slave will listen to a packet from the Master on every anchor point.
- Timeout Multiplier: The connection supervision timeout can be calculated from this field as follows:
 - Connection Supervision Timeout = Timeout Multiplier * 10 ms.

If the Master decides to accept this request, then it sends the new set of connection parameters to the Slave using the link layer connection update procedure. The connection update procedure at the link layer level was explained in Chapter 8. If the Master decides not to accept the parameters, then it rejects the request using the Command Reject.

It may be noted that this command can only be sent from the LE Slave to the LE Master and not the other way round. If the LE Master needs to change the connection parameters, then it can directly use the link layer procedure.

10.8.3 Connection Parameter Update Response

This packet is sent by the LE Master to the LE Slave in response to the connection parameter update request. If the Master accepts the parameters sent by the Slave, it also sends the connection parameter update to the controller using the HCI_LE_Connection_Update command so that the controller can start the link manager procedure to update the connection. A typical sequence for updating the connection parameters is shown in Figure 10.6.

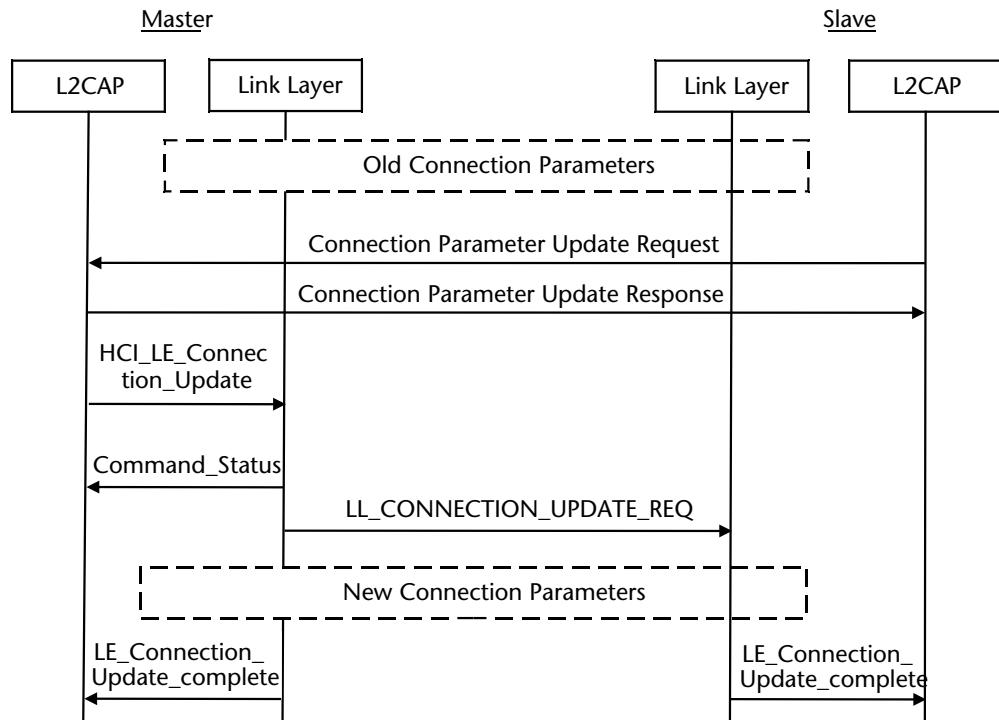


Figure 10.6 Typical Connection Parameter Update Sequence.

10.9 Practical Examples

Figure 10.7 shows a practical example of the L2CAP PDUs being exchanged between two devices. The following points may be observed:

- The CID used for ATT is 0x0004 and CID used for SMP is 0x0006.
- There are no CONNECT and CONFIGURE requests as in the case of BR/EDR. In the case of L2CAP, once the connection is established at the link layer level, data PDUs can be exchanged.

The screenshot shows a software interface for monitoring L2CAP PDUs. The left pane displays a tree view of PDU types: Frame 685, LE BB, LE PKT, LE DATA, L2CAP, and ATT. A callout points from "L2CAP PDUs" to the L2CAP entry in the tree. The right pane is a table of L2CAP PDUs with columns: Frame#, Role, Addr., CID, PSM, ID, Code, Source CID, Dest. CID, and Length. The table shows the following entries:

Frame#	Role	Addr.	CID	PSM	ID	Code	Source CID	Dest. CID	Length
685			0x0004 [Attribute Protocol]						14
686			0x0004 [Attribute Protocol]						7
689			0x0004 [Attribute Protocol]						5
690			0x0006 [LE Security Manager Protocol]						7
693			0x0006 [LE Security Manager Protocol]						7
694			0x0006 [LE Security Manager Protocol]						17
697			0x0006 [LE Security Manager Protocol]						17

Annotations highlight specific entries: "CID = 0x0004 for ATT" points to the first two rows, and "CID = 0x0006 for SMP" points to the last four rows.

Figure 10.7 Example of L2CAP exchanges.

10.10 Summary

The L2CAP layer provides data services to SMP and ATT protocols. Since there are only two layers on top of it, L2CAP layer use fixed CIDs for signaling leading to a simpler implementation.

In the case of LE, the L2CAP layer is simplified in a major way. It includes only three signaling commands: Command Reject, Connection Parameter Update Request, and Response. Even these commands don't need to be sent in the beginning to start data exchange between two layers. As soon as the link layer connection is established, the upper layers can provide data to L2CAP to send to the remote device.

The next two chapters will focus on Security Manager and Attribute Protocol. These are the two entities which use the services provided by L2CAP.

Bibliography

Bluetooth Core Specification 4.0 <http://www.bluetooth.org>.

Security Manager (SM)

11.1 Introduction

The security manager defines the procedures for pairing, authentication, and encryption between LE devices. This is needed once a link layer connection has been established and if security is requested on that particular connection. As shown in Figure 11.1, the security manager is located above L2CAP in the LE architecture. It uses the services of L2CAP to carry out its different procedures.

The security manager protocol is used to generate and store various keys (like encryption and identity). It uses a key distribution approach where each device generates and controls the keys it distributes. The security manager is also responsible for generating random addresses and resolving random addresses to known device identities. The security manager allows for keys from 56-bits to 128-bit length in 8-bit steps. The key length is defined by the profile or application that requests security. If devices have lower processing power or need less security, then they need not generate all bits of the 128-bit keys. They can generate a lesser number of bits (subject to a minimum of 56-bits) and set the remaining bits to 0.

11.2 Security in Host Instead of Controller

One major difference between the security architecture of BR/EDR and LE is that the security in BR/EDR is handled in the link manager, while in the case of LE, the security related procedures are moved to the host. So all procedures related to key generation and distribution of keys are performed by the host. This helps to keep the cost of LE-only controllers low and also provides more flexibility to the host. If the key generation algorithms need to be upgraded (for example, to increase the level of security offered by the device) then only the software in the host needs to be changed without any modification in the controller. The encryption of data just before transmitting the packets over the air is still done by the controller in both LE and BR/EDR.

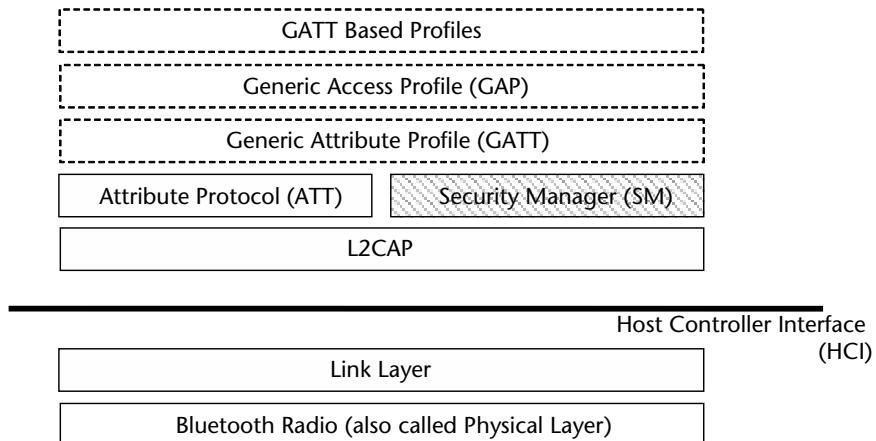


Figure 11.1 Security Manager in LE protocol stack.

11.3 Asymmetrical Architecture

The security manager has an asymmetrical architecture. This means that the architecture is designed so that the memory and processing requirements for the responding device are much lower than the memory and resource requirements of the initiating device.

This is another enhancement done by LE to optimize the power consumption of the peripheral devices. In general the initiating device could be a dual mode device like a mobile phone which has much higher memory, processing power, and battery power available. In comparison a peripheral device like a key fob could have limited memory, processing power, and battery power. So the architecture is designed in such a manner that more processing power and memory is used on the mobile phone resulting in lesser memory and processing power used at the key fob end.

11.4 Security Breaches

Before going into the details of security manager, it is important to understand the possible security breaches in a wireless system. These will be explained in this section. Later sections will explain how the security manager helps to prevent these security breaches.

11.4.1 Passive Eavesdropping

Passive Eavesdropping attacks occurs when an attacker starts listening to the data being exchanged between two devices. If the attacker is present at the time of initial pairing, it can listen to the keys that are being exchanged during pairing. After that it can use the same keys to decrypt all the information being sent between the two devices.

Passive eavesdropping is difficult to detect. This is because, the passive eavesdropper can be anywhere in the Bluetooth range (anywhere between 10 meters to

100 meters). So the eavesdropper need not be visible to the user. Eavesdropping can be done, for example, from another room. This is shown in Figure 11.2.

11.4.2 Man-in-the-Middle (MITM) (Active Eavesdropping)

MITM was explained in Chapter 3. The explanation is repeated here for convenience. An MITM attack occurs when a rogue device attacks by sitting in the middle of two devices that want to connect and relays messages between them. The two devices believe that they are directly talking to each other without knowing that all their messages are being intercepted and relayed by a third device which is between them. This is also known as active eavesdropping.

Let's say devices A and B want to make a connection and M is an attacking device (as shown in Figure 11.3). M receives all information from A and relays it to B and vice versa. So, A and B have an illusion that they are directly connected. They are not aware of the existence of M between them. Since M is relaying the

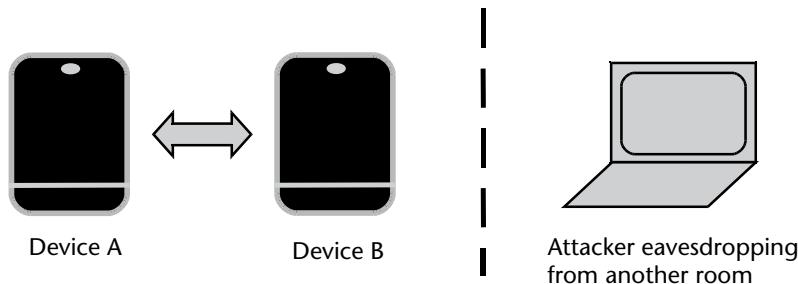


Figure 11.2 Example of Passive Eavesdropping.

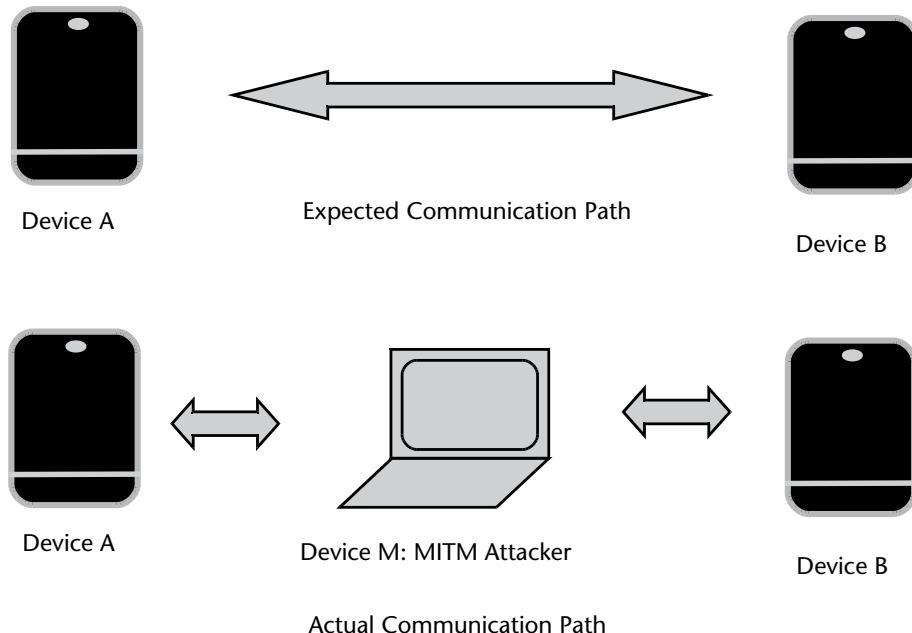


Figure 11.3 Example of MITM attack.

information between the two devices, it can interpret all this information and misuse it. Besides this, M can also attack by inducing rogue information between the two devices.

11.4.3 Tracking

Since many of the LE devices are intended to be carried by the user, an attacker can potentially try to track the transmissions coming from these devices to track the person. For example, if a person is wearing an LE watch or carrying a key fob which are advertising, then an attacker can read those advertising packets and follow the person by just following the advertising packets.

LE provides a privacy feature in which the LE devices can use a random address and change this random address frequently. So only devices that are previously authenticated by the person can resolve that address. This means that only the authenticated devices can map the random address to find out the exact device from which the packets are being transmitted. This prevents tracking from any unauthorized person.

11.5 Pairing Methods

The pairing method used to pair the devices depends on the Input/Output capabilities of the two devices. For example, if one of the devices is capable of entering a 6-digit number and another is capable of displaying a 6-digit number, then the Passkey entry method can be used.

Security Manager provides three types of pairing methods:

1. Just Works.
2. Passkey Entry.
3. Out of Band.

11.5.1 Just Works

In the case of Just Works pairing no passkeys are exchanged at the UI level. This is used when at least one of the devices does not have a mechanism to either display a 6-digit number or enter 6-digit numbers. An example of this could be a mono headset which does not have display capability or the ability to enter a passkey.

11.5.2 Passkey Entry

This method is designed for use when one device has input capability and the other device has display capability. In this method, a 6-digit numeric passkey is displayed on one of the devices and the user is asked to enter that passkey on the second device. For example, if a keyboard is being paired to a PC, then the PC may display a 6-digit numeric passkey which can be entered from the keyboard. Another example could be a remote being paired to a TV. The TV can display the passkey which is entered from the remote control.

11.5.3 Out of Band

This is designed for scenarios where an Out-of-Band mechanism can be used to transfer security information. For example, if NFC is used, then the user may touch the two devices to exchange the security information. These three methods will be explained in further detail later in this chapter.

11.6 Security Properties

Security Manager provides three types of security:

1. Authenticated MITM protection.
2. Unauthenticated No MITM protection.
3. No security.

None of the pairing methods provide protection against passive eavesdropper if the eavesdropper starts listening during the pairing process. This is because, at present, predictable or easily established values are used as temporary keys in the beginning of the pairing. Future versions of the specifications are expected to support more secure methods to support passive eavesdropping protection. If the pairing is done when the eavesdropper is not listening, then the communication channel is secure from any passive eavesdroppers and further communication can go on without any risk.

11.6.1 Authenticated MITM Protection

Authenticated MITM protection can be achieved by using the passkey entry pairing method or by using Out Of Band (OOB) pairing.

- In the case of the passkey entry pairing method the passkey is displayed on one device and entered from another device. So there is no way that an MITM can relay this information between these two devices.
- In the case of Out Of Band, some other technique apart from Bluetooth is used for exchanging the pairing information. For example, NFC could be used where the two devices touch each other to exchange the pairing information. Here again, if the OOB method being used provides MITM protection, only then the MITM attacks can be prevented.

11.6.2 Unauthenticated no MITM Protection

Unauthenticated no MITM protection does not provide protection from MITM attacks. This is achieved by using the Just Works pairing method. In this method, no passkeys are exchanged at the UI level. It is still more secure than No security since the keys are exchanged between the two devices though they are not exchanged at the UI level. In this case the keys are automatically generated by the two devices.

11.6.3 No Security

In the no security mode, there is no support for authentication or encryption. In this case the information transferred may not be sensitive. For example, an LE thermometer sending temperature data may use No Security Mode. Similarly No Security Mode may be used to fetch information like the device manufacturer's name, device model number, etc.

11.7 Cryptographic Functions

Security manager provides the following cryptographic functions which serve as enablers for various security operations described later.

1. Security function e .
2. Random address function ah .
3. Confirm value generation function $c1$.
4. Key generation $s1$.

11.7.1 Security Function e

The security function e is used to generate 128-bit encrypted data from 128-bit plain text data. It uses the AES-128 bit cypher.

```
encryptedData = e(key, plaintextData)
```

This function can be implemented in the host or the controller. If this functionality is implemented in the controller, then the host can use it via the HCI command HCI_LE_Encrypt. This function is used by the next three cryptographic functions.

11.7.2 Random Address Function ah

The random address function ah is used in generating a hash value used in the resolvable private address.

11.7.3 Confirm Value Generation Function c1

The confirm value generation function $c1$ is used to generate the confirm values used during the confirm procedure of the Security Manager Protocol.

11.7.4 Key Generation Function s1

The key generation function $s1$ is used to generate the Short Term Key (STK) during the pairing process.

11.8 Pairing

The pairing process is used to establish the keys used to encrypt the link. Once the link is encrypted, the various keys to resolve the random address, verify signed data and encrypt future links are exchanged between the Master and the Slave.

Pairing is a three phase process. The first two phases are mandatory while the third phase is optional. The three phases are:

1. Pairing Feature Exchange.
2. Short Term Key (STK) Generation.
3. Transport Specific Key Distribution [Optional].

Phase 1 and Phase 2 may be performed on a link (encrypted or not encrypted) while Phase 3 is only performed on a link which is encrypted using the STK generated in Phase 2. These three phases are shown in Figure 11.4 and will be explained in detail in the next sections.

11.8.1 Phase 1: Pairing Feature Exchange

In this phase, the devices exchange their capability information, key sizes, the keys that they can distribute, and the keys that they expect the remote side to distribute. This phase is also used in identifying which method of pairing they can use in the second phase. Pairing procedure is always initiated by the device in the Master role. The following is the set of features which are exchanged between the Master and the Slave during this phase.

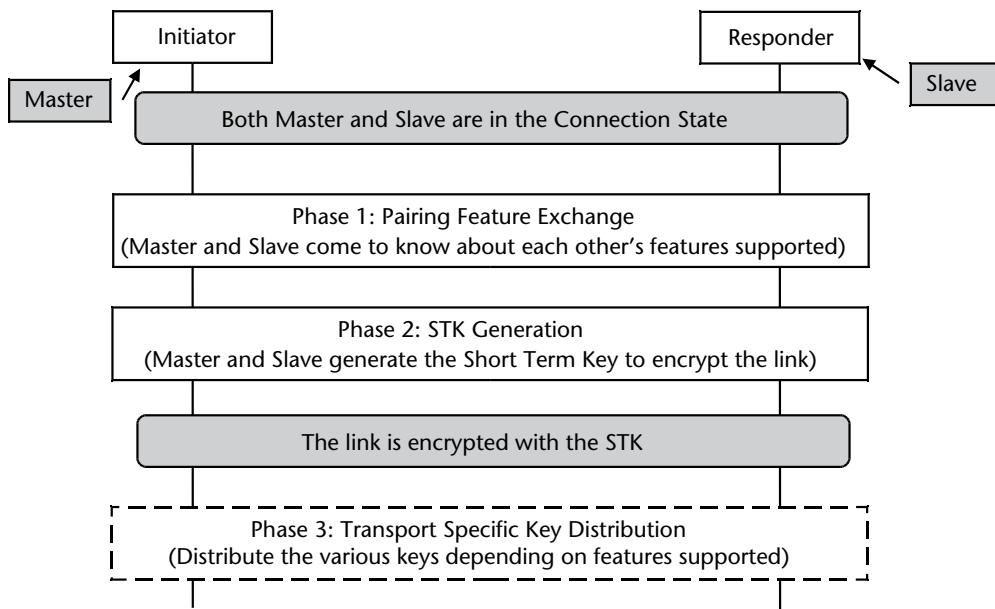


Figure 11.4 Three Phase Pairing Process.

11.8.1.1 IO Capability

The IO Capability provides information about the input and output capabilities of the device. It is used during the pairing feature exchange procedure (step 1 of the pairing process) to inform the IO capabilities of the device. Based on the IO capabilities of the two devices, the appropriate pairing procedure is selected in Phase 2 of the pairing process.

There are three possibilities with regards to the input capabilities of the device:

1. *No Input*: The device does not have any capability to take user input. An example of this could be a weighing machine which may not have buttons on it to take user inputs.
2. *Yes/No*: The device has some mechanism which can be used to indicate a ‘yes’ or ‘no’ input from the user. An example of this could be a key fob which has a couple of buttons which the user can use to indicate a ‘yes’ or a ‘no’.
3. *Keyboard*: The device has a numeric keyboard which can be used to input the numbers ‘0’ to ‘9’ and two buttons to indicate ‘yes’ and ‘no’. An example of this could be a remote control which has a numeric keypad which can be used to input the numbers ‘0’ to ‘9’.

There are two possibilities with regards to output capabilities of the device:

1. *No Output*: The device does not have any display capability to display a 6-digit decimal number. An example of this could be a remote control which does not have display capability.
2. *Numeric Output*: The device has the ability to display or communicate a 6-digit decimal number to the user. An example of this could be a watch which can display the 6-digit decimal number to the user.

Based on the input and output capabilities of the device, the IO capability field is mapped as shown in Table 11.1. As an example, a device which has No Input capability and Numeric Output is termed as DisplayOnly since it can only display but it cannot take user input.

The IO capability of the two devices is used to decide the appropriate pairing method to use in Step 2 out of the following three possibilities:

1. Just Works.
2. Passkey Entry.
3. Out of Band (OOB).

Table 11.1 IO Capability

<i>Input Capability</i>	<i>Output Capability</i>	
	<i>No Output</i>	<i>Numeric Output</i>
No Input	NoInputNoOutput	DisplayOnly
Yes / No	NoInputNoOutput	DisplayYesNo
Keyboard	KeyboardOnly	KeyboardDisplay

11.8.1.2 OOB Authentication Data

In order to have additional security, an LE device may support the use of an external mechanism to generate data to authenticate the device. For example, an LE device may use the NFC protocol to generate authentication data. This provides additional security because the NFC protocol may involve touching the two devices and data exchange on a different protocol than LE. So the possibility of an intruder listening to that data is significantly reduced.

11.8.1.3 Encryption Key Size

LE supports an encryption key between 7 octets (56-bits) and 16 octets (128-bits). The higher the number of bits used, the higher the level of encryption. So the devices can select the encryption key size based on the resources available (memory, processing power) and the level of security needed. This value is exchanged between the two devices during Phase 1, and the lesser of the two values is used as the encryption key size for the next phases.

11.8.1.4 Repeated Attempts

This feature is similar to the support provided in BR/EDR for repeated attempts. This feature is invoked when a pairing procedure fails. If the pairing procedure fails, the device has to wait a certain interval before again trying to pair to a remote device or allowing a new pairing procedure from a remote device. This waiting interval increases exponentially with each failed attempt. The waiting interval keeps increasing with each failed attempt until the time it reaches an implementation defined maximum value. It helps to prevent an intruder from repeatedly trying the pairing procedure with several different keys.

For example, if an intruder device A tries to connect to device B. If the pairing procedure fails, then B sets the waiting interval to i . During this waiting interval i it does not respond to any Pairing Request command or Security Request command from A. If A tries to pair after this waiting interval, and if the pairing procedure fails again, then B increases the waiting interval to $2 * i$. If the pairing procedure fails a third time, the waiting interval increases to $4 * i$. This makes it increasingly time consuming (and therefore difficult) for the intruder device A to try several different keys in order to pair to B.

11.8.2 Phase 2: Short Term Key (STK) Generation

After the Pairing Feature Exchange, both devices are aware of the capabilities of each other. Based on these capabilities, an appropriate paring method is selected. This can be one of the following three methods.

1. Just Works.
2. Passkey Entry.
3. Out of Band (OOB).

The selection of the method is done as follows:

1. If both devices support the OOB pairing method, then the OOB pairing method is used.
2. If both devices have not set the MITM option, then the Just Works association model is used.
3. Otherwise, the IO capabilities of both the devices are used to determine the appropriate method.

In this phase a Temporary Key (TK) is generated by each device based on the pairing method that is selected. This TK is used to generate the STK and encrypt the link. The formulae for generating TK and STK are explained in the Bluetooth specification.

11.8.2.1 Just Works

This is the simplest pairing mechanism. It does not provide protection against eavesdropping and MITM attacks during the pairing process. Once the pairing process is over, this procedure provides security by using encryption.

11.8.2.2 Passkey Entry

This method is used if one of the devices supports display capability and the other device supports keyboard capability. In this method, a 6-digit numeric passkey is displayed on one of the devices and the user is asked to enter that passkey on the second device. For example, if a TV is being paired with a remote, then the passkey will be displayed on the TV and the user will be asked to enter that passkey on the remote. This method provides protection against MITM attacks but limited protection against eavesdropping.

11.8.2.3 Out of Band (OOB)

This method is used if both the devices support the OOB Authentication Data feature. In this method the level of security depends on the security offered by the OOB mechanism used.

11.8.3 Phase 3: Transport Specific Key Distribution

This phase is optional and is performed only on a link which is encrypted using STK. In this phase, the Master and Slave distribute the keys to each other.

The various keys that are distributed in this phase are:

- Long Term Key (LTK).
- Encrypted Diversifier (EDIV) and Random Number.
- Identity Resolution Key (IRK).
- Public Device Address or Static Random Address.
- Connection Signature Resolving Key (CSRK).

These keys are briefly described below.

11.8.3.1 Long Term Key (LTK)

The Long Term Key (LTK) is a 128-bit key that is used to generate the key for an encrypted connection. The LTK is provided by the host to the controller on both the Master and the Slave side. The Master and Slave controllers use a combination of LTK, EDIV and Rand to encrypt the link.

11.8.3.2 Encrypted Diversifier (EDIV) and Random Number (Rand)

Encrypted Diversifier (EDIV) is a 16-bit stored value to identify the LTK. A new EDIV is generated every time a unique LTK is distributed. Random Number (Rand) is a 64-bit value used to identify the LTK. A new Rand is generated every time a unique LTK is distributed. A combination of LTK, EDIV and Rand is used by the Master and the Slave to encrypt the link.

11.8.3.3 Identity Resolution Key (IRK)

Identity Resolution Key (IRK) is a 128-bit key used to generate and resolve random addresses. Random address is a privacy feature that is introduced in LE. This allows a device to frequently use a different random address so that it's difficult to track that device. The random address can be resolvable or non-resolvable: This will be explained in detail in Chapter 14. The resolvable address is generated in such a manner that it can be resolved by the peer device if the peer device has the IRK along with the random address. In that case, the peer device will be able to identify the device transmitting data.

The IRK is used along with a random number to generate the random address. This random address is used for all further interactions with other devices. If a device wants a remote device to identify it, then it provides the IRK to that device. The remote device can then identify which device the packet is coming from by using a combination of the IRK and the random address. All other devices which don't have the IRK cannot identify which device is sending the packet and therefore cannot know if the random address belongs to the same device.

A Master that receives an IRK from a Slave can resolve the Slave's random device address. Similarly a Slave that receives an IRK from the Master can resolve the Master's random device address.

11.8.3.4 Connection Signature Resolving Key (CSRK)

LE provides the feature to sign the data. To sign the data, the sending device appends a 12-octet signature after the data PDU. The receiving device verifies the signature to check if the data is coming from a trusted source.

Connection Signature Resolving Key (CSRK) is a 128-bit key used to sign data and verify signatures on the receiving side. A different CSRK is used for each peer device to which signed data is to be sent.

11.9 Security Manager Protocol

The Security Manager Protocol (SMP) is used for pairing and key distribution. The L2CAP CID 0x0006 is used for all SMP commands. The format of the SMP commands is shown in Figure 11.5.

The Code field identifies the type of the command. The length and format of the Data field depends on the type of the command.

A timer of 30 seconds is used for all SMP procedures. If the timer expires, then the SMP procedure is considered to have failed and the higher layers are notified. If the higher layers need to restart the procedure, then a new physical channel needs to be established.

The various command codes supported by SMP are shown in Table 11.2.

11.9.1 Commands Used During Phase 1 (Pairing Feature Exchange)

In this phase, the Initiator (Master) and the Responder (Slave) exchange the information about paring features that they support so that the appropriate set of features can be used in Phase 2. The commands that are used during Phase 1 are shown in Figure 11.6. At the end of this phase, a pairing method is selected based on the Pairing Request and Pairing Response.

11.9.1.1 Security Request

This command is used by the Slave if it wants to request the Master to initiate security. In response to this command, the Master may send the Pairing Request command.

11.9.1.2 Pairing Request

The Pairing Request command is used to start the first phase of the pairing process which is Pairing Feature Exchange. The Initiator provides the set of features it supports in this command to the remote device. This includes the following:

- IO Capability: This indicates the IO capabilities of the device like Display-Only, KeyboardOnly, KeyboardDisplay etc. The different IO Capabilities were shown in Table 11.1.
- OOB Data Flag: This indicates whether Out Of Band data is available or not.
- AuthReq Flag: This indicates the requested authentication requirements. For example, whether bonding is requested or not or whether MITM protection is requested.

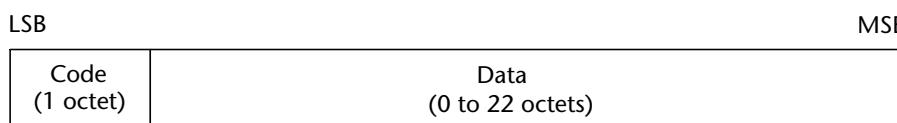
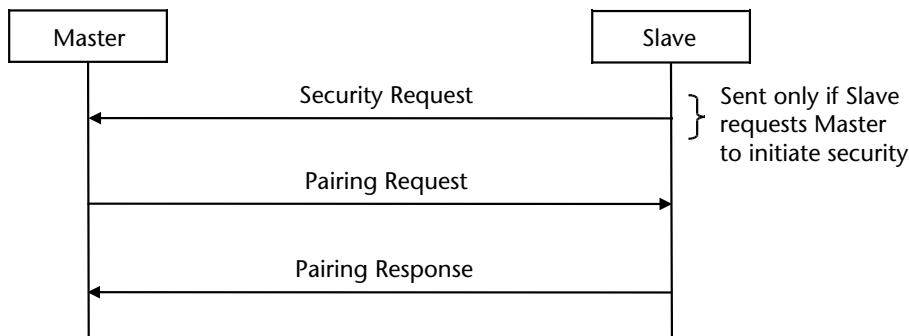


Figure 11.5 Format of SMP commands.

Table 11.2 SMP Command Codes

<i>Code</i>	<i>Description</i>	<i>Phase</i>
0x00	Reserved	—
0x01	Pairing Request	Phase 1
0x02	Pairing Response	Phase 1
0x03	Pairing Confirm	Phase 2
0x04	Pairing Random	Phase 2
0x05	Pairing Failed	Phase 2
0x06	Encryption Information	Phase 3
0x07	Master Identification	Phase 3
0x08	Identity Information	Phase 3
0x09	Identity Address Information	Phase 3
0x0A	Signing Information	Phase 3
0x0B	Security Request	Phase 1 (Used if Slave requests initiation of security procedures)
0x0C – 0xFF	Reserved	—

**Figure 11.6** Sequence of commands used in Phase 1.

- Maximum Encryption Key Size: This indicates the maximum encryption key size that is supported by the device.
- Initiator Key Distribution: This field indicates which keys the Initiator is requesting to distribute in Phase 3 of the pairing process.
- Responder Key Distribution: This field indicates which keys the Initiator is requesting the Responder to distribute during Phase 3 of the pairing process.

The format for the Initiator Key Distribution and Responder Key Distribution field is shown in Figure 11.7. A Practical example of Pairing Request is shown in Figure 11.10.

11.9.1.3 Pairing Response

The Pairing Response command is used by the responding device to respond to the pairing request. This command completes Phase 1 of the pairing process.

The different fields of this command are similar to the ones used in the Pairing Request command. In this command, the Responder confirms the exact Initiator and Responder keys to be used in Phase 3. If the Master had set any of the fields

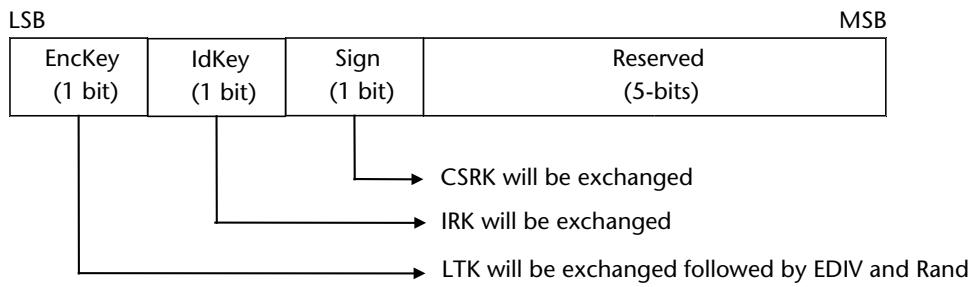


Figure 11.7 Format of Initiator Key Distribution and Responder Key Distribution.

to zero in the Pairing Request command, then the Slave does not set them to one. Then, depending on its own capabilities, the Slave sets the remaining bits in the Initiator Key Distribution and Responder Key Distribution fields.

Once the Master receives the Pairing Response command, both devices know which keys will be exchanged in Phase 3. A Practical example of Pairing Response is shown in Figure 11.11.

11.9.2 Commands Used During Phase 2 (STK Generation)

This phase is started after successful completion of Phase 1. After the Pairing Feature Exchange has completed, a paring method is selected for Phase 2. As mentioned earlier, there are three pairing methods that can be used:

1. Just Works.
2. Passkey Entry.
3. Out of Band Pairing.

In this phase a Short Term Key (STK) is generated and the link is encrypted using that STK. The commands that are used during Phase 2 are shown in Figure 11.8.

11.9.2.1 Pairing Confirm

The Pairing Confirm command is used by both devices to send the confirm value to the peer device. This is a 128-bit value which is generated by the Master and the Slave based on the pairing method that is selected. The Master generates a 128-bit Mconfirm and sends it to the Slave and the Slave generates a 128-bit Sconfirm and sends it to the Master.

11.9.2.2 Pairing Random

The Pairing Random command is used by both devices to send the random number used in generating the confirm value that was sent in the previous Pairing Confirm command.

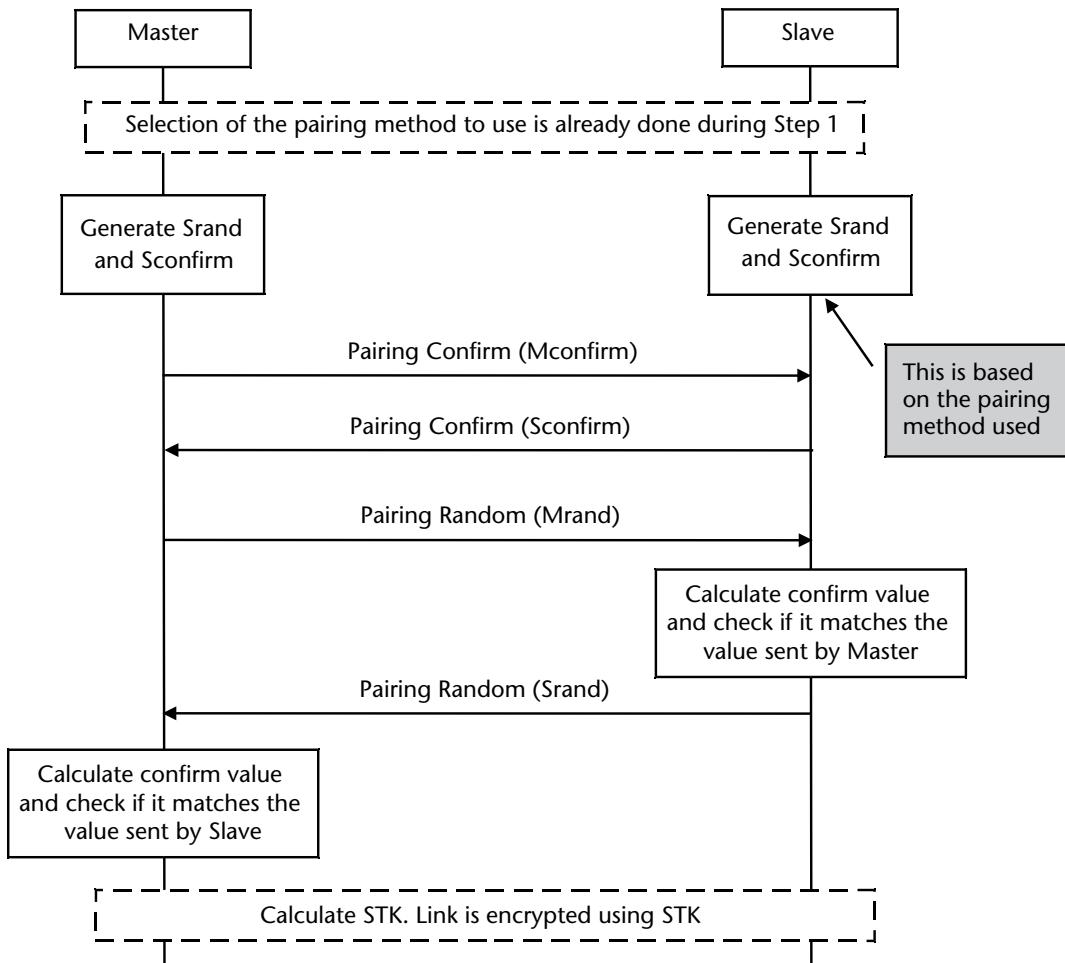


Figure 11.8 Sequence of commands used in Phase 2.

The Master sends the Mrand value to the Slave. The Slave calculates the confirm value based on this Mrand and checks if the value matches the value sent by the Master in the previous step (Mconfirm). If the value matches, then the Slave sends its own random number (Srand) to the Master. The Master does a similar calculation to check if the calculated value matches the confirm value that was sent by the Slave. After the Master verifies the confirm value, it calculates the STK and encrypts the link with the STK.

11.9.2.3 Pairing Failed

The pairing failed command is used if there is a failure during pairing. For example, if the confirm value received from the remote side does not match the confirm value calculated, then a Pairing Failed command is sent. A practical message sequence chart of these procedures is shown in Figure 11.12.

11.9.3 Commands Used During Phase 3 (Transport Specific Key Distribution)

This phase is optional and started after successful completion of Phase 2. By the end of Phase 2, the link is encrypted and during this phase the transport specific keys can be distributed from the Master to the Slave and vice versa. The various keys that can be distributed are shown in Figure 11.9.

The keys are distributed first by the Slave and then by the Master in the sequence that is shown in Figure 11.9. Some or all of these keys may be distributed depending on what was agreed during Phase 1. The keys to be distributed during this phase were indicated in Phase 1 in the Key Distribution Field of the Pairing Request and Pairing Response commands. For example, the LTK is distributed from the Slave to the Master only if the Slave had earlier set the EncKey bit in the Key Distribution Field of the Paring Response command that it sent to the Master in Phase 1. (See Figure 11.7).

11.9.3.1 Encryption Information

The Encryption Information command is used to distribute the LTK to be used for encrypting the connection. It contains the following parameter:

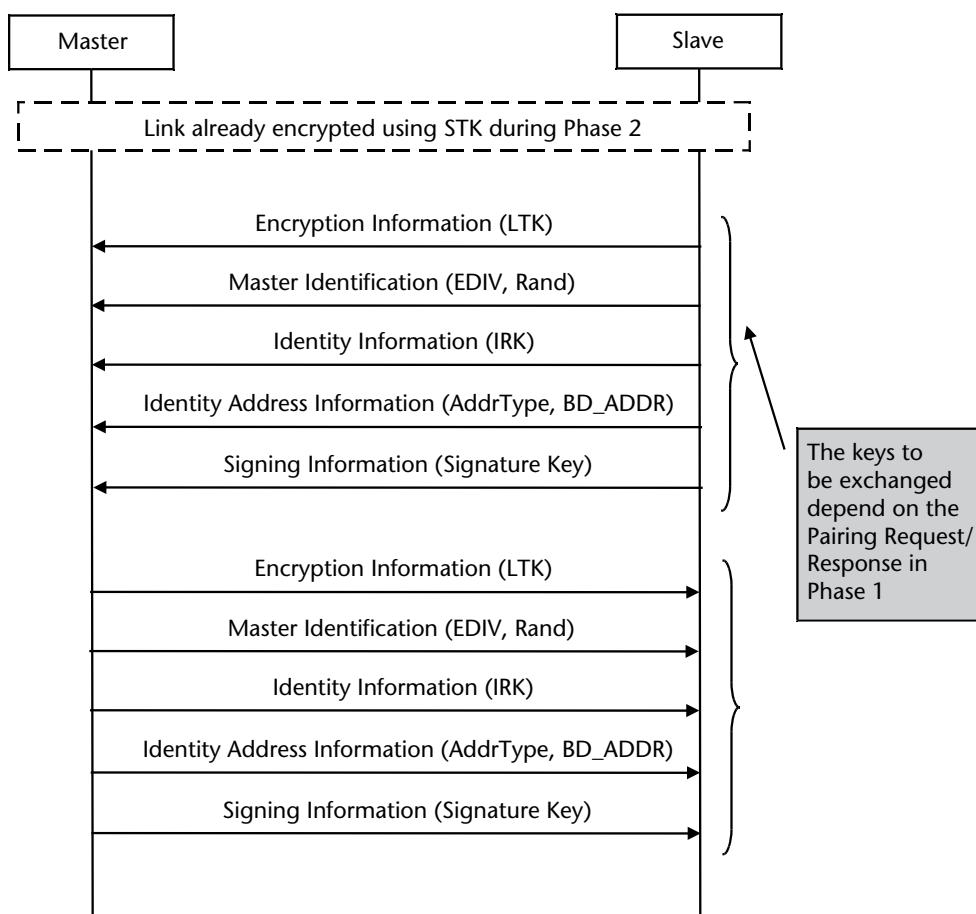


Figure 11.9 Distribution of transport specific keys in Phase 3.

- LTK: 128-bit.

This command is only sent when the link has been encrypted with STK.

11.9.3.2 Master Identification

The Master Identification command is used to distribute the EDIV and Rand to be used for encrypting the connection. It contains the following parameters:

- EDIV: 16-bit.
- Rand: 64-bit.

This command is only sent when the link has been encrypted with STK.

11.9.3.3 Identity Information

The Identity Information command is used to distribute the IRK to be used for resolving the device. It contains the following parameter:

- IRK: 128-bit.

This command is only sent when the link has been encrypted with STK.

11.9.3.4 Identity Address Information

The Identity Information command is used to distribute the public device address or the static random address. It contains the following parameters:

- AddrType: 8-bit: To identify whether the address is a public device address or a static random address.
- BD_ADDR: 48-bit: The public device address or the static random address.

This command is only sent when the link has been encrypted with STK.

11.9.3.5 Signing Information

The Identity Information command is used to distribute the CSRK to be used for signing the data. It contains the following parameter:

- CSRK: 128-bit.

11.10 Practical Examples

Sample air sniffer captures of SMP transactions between an Initiator and Responder are shown in Figure 11.10 and Figure 11.11. The security procedures start at Frame

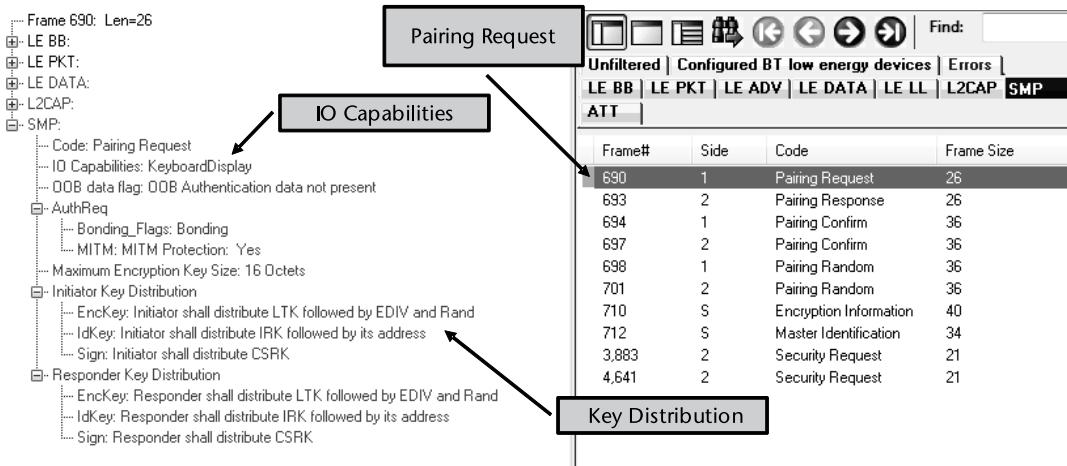


Figure 11.10 Example of Pairing Request.

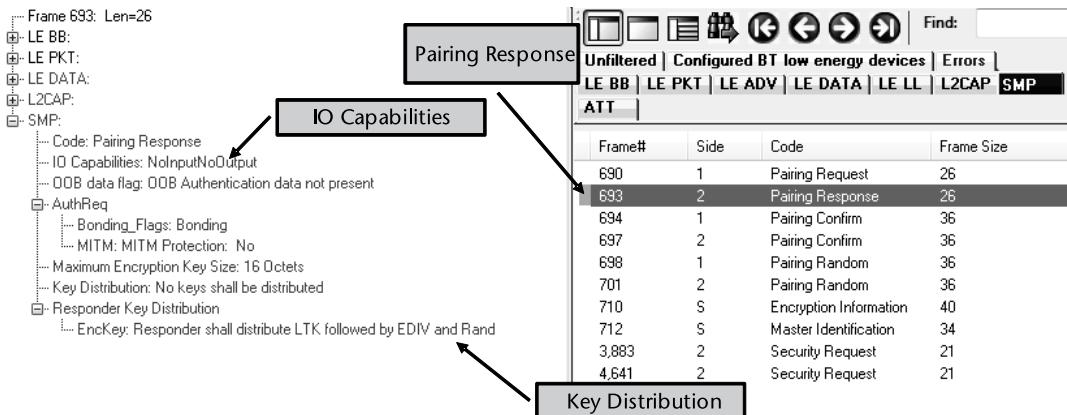


Figure 11.11 Example of Pairing Response.

#690 with a Pairing Request. There are two devices that are identified as Side 1 and Side 2. Side 1 is the Initiator and Side 2 is the Responder.

The parameters of the Pairing Request procedure are shown on the left side of Figure 11.10. The main points to note are:

1. IO Capabilities are set to KeyboardDisplay. This means that the device has a Keyboard and a Display.
2. MITM Protection has been set to Yes.
3. The Initiator Key Distribution specifies the keys that the Initiator will distribute. In this example the Initiator specifies the following:
 - a. *Encryption Key*: Initiator shall distribute LTK followed by EDIV and Rand.
 - b. *Id Key*: Initiator shall distribute IRK followed by its address.
 - c. *Sign*: Initiator shall distribute CSRK.

4. The Responder Key Distributor specifies the keys that the Initiator requests the Responder to distribute. In this example, the Initiator has specified the following keys for the Responder to distribute.
 - a. *Encryption Key*: Responder shall distribute LTK followed by EDIV and Rand.
 - b. *Id Key*: Responder shall distribute IRK followed by its address.
 - c. *Sign*: Responder shall distribute CSRK.

The Paring Response procedure is shown in Figure 11.11. The parameters of the Pairing Response are shown on the left side of the figure. The main points to note are:

1. IO Capabilities are set to NoInputOutput. This means that the device neither has a Keyboard nor a Display.
2. MITM Protection has been set to No.
3. The Responder Key Distribution specifies the keys that the Responder will distribute. In this example the Responder specifies only the following:
 - a. *Encryption Key*: Initiator shall distribute LTK followed by EDIV and Rand.

Since one device supports KeyboardDisplay while the other supports NoInput-NoOutput, the Just Works pairing method would be used.

The detailed message sequence chart corresponding to the air logs in Figure 11.10 and Figure 11.11 is shown in Figure 11.12. The SMP_M indicates the Master and SMP_S indicates the Slave.

The various phases of the pairing procedure are depicted as follows:

1. *Phase 1*: Phase 1 consists of the Master sending the Pairing Request and the Slave responding with a Pairing Response.
 - a. Pairing Request from the Master specifies the IO Capabilities as KeyboardDisplay and the MaxKeySize to be 16 octets.
 - b. Pairing Response from the Slave specifies the IO Capabilities of the Slave as NoInputNoOutput and the MaxKeySize as 16 octets.
2. *Phase 2*: Phase 2 consists of Pairing Random and Pairing Confirm exchanged between the Master and the Slave.
 - a. The Master sends a Pairing Confirm which includes a 128-bit Mconfirm.
 - b. The Slave responds with a Pairing Confirm which includes a 128-bit Sconfirm.
 - c. After this the Master sends the Pairing Random with the Mrand value. The Slave calculates the confirm value based on this Mrand and checks if this matches the value sent by the Master in the Pairing Confirm Request.
 - d. The Slave checks that the value matches and then sends the Pairing Confirm with its own Mrand value. The Master calculates the confirm

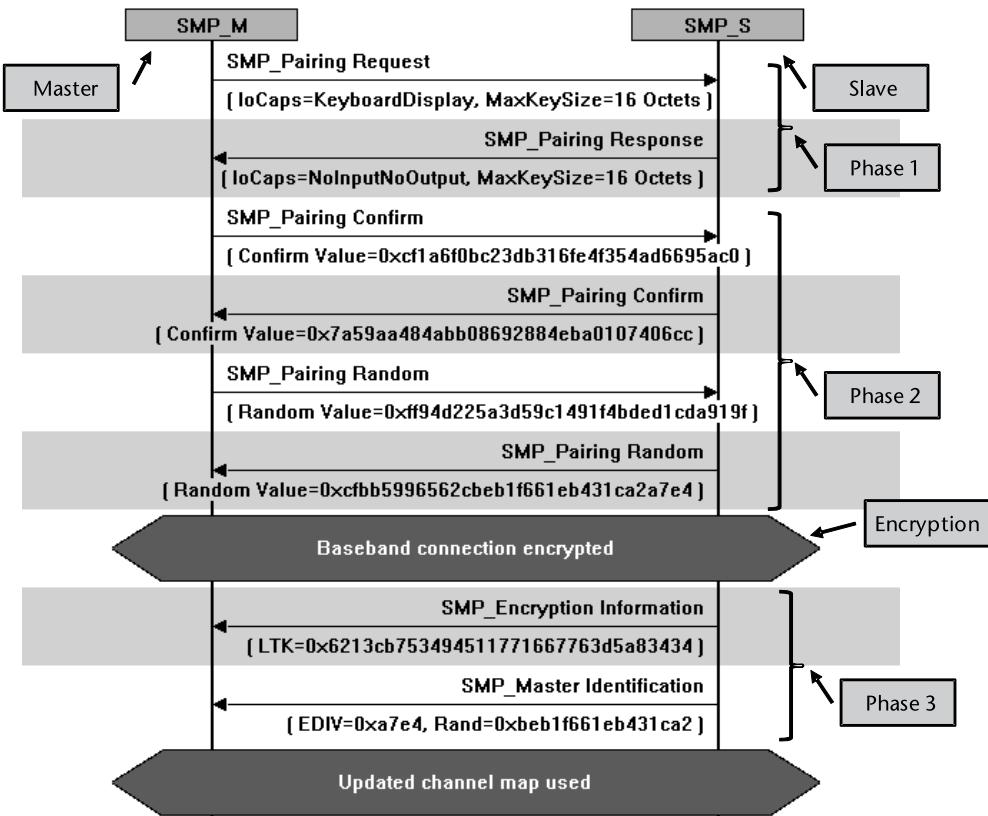


Figure 11.12 Example of SMP Message Sequence Chart.

value based on this Strand and checks if this matches the value sent by the Slave in the Pairing Confirm Request.

3. *Phase 3:* After successful completion of Phase 2, the link is encrypted and the key distribution can start. The set of keys that will be distributed was specified in Phase 1.
 - a. Encryption Information is used to distribute the 128-bit LTK.
 - b. Master Identification is used to distribute 16-bit EDIV and 64-bit Rand.

11.11 Summary

The security manager is responsible for carrying out security related procedures like pairing, authentication, and encryption. The security manager architecture in LE introduces several enhancements as compared to BR/EDR. This includes moving the security manager block into the host in the controller to keep the controller requirements to a minimum as well as using asymmetrical architecture to perform less processing in one device at the cost of performing more processing on the peer device. This is done because generally in LE communication one device (like a mobile phone) has abundant resources while the other device (like a sensor) has very

restricted resources and the device with limited resources tries to conserve battery power by doing as little processing as possible.

The pairing process is one of the important processes defined by security manager. It uses a three phase process. The first phase broadly includes exchange of capabilities, the second phase includes exchanging of short term keys, and the third phase involves the exchange of actual keys that will be used in subsequent procedures.

Bibliography

Bluetooth Core Specification 4.0 <http://www.bluetooth.org>.

NIST Publication FIPS-197 (<http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>).

Attribute Protocol (ATT)

12.1 Introduction

The attribute protocol provides a mechanism for discovering attributes of a remote device, reading the attributes and writing the attributes. As shown in Figure 12.1 ATT sits above L2CAP layer and uses L2CAP as the transport mechanism for transferring data. In turn, ATT provides services to the Generic Attribute Profile.

Attribute protocol follow a client server model. The server exposes a set of attributes to the client. The client can discover, read and write those attributes. The server can also notify or indicate the client about any of the attributes.

A device can implement only a client role, only a server role, or both client and server roles. At any given time, only one server can be active on a device.

12.2 Attribute

An attribute is something that represents data. It could be thought of as just any data at any given time when the device is in any given state. It could be the position, size, mass, temperature, speed, or any other data that the device wants to share with other devices. ATT protocol is designed to push or pull that data to or from a remote device. Besides that ATT protocol also supports setting notifications and indications so that the remote devices can be alerted when that data changes.

Some examples of attributes are:

- The temperature provided by a thermometer.
- The unit in which the temperature is provided.
- The name of a device.
- The manufacturer name and model number of a device.

Besides containing the value of the data, an attribute has three properties associated with it:

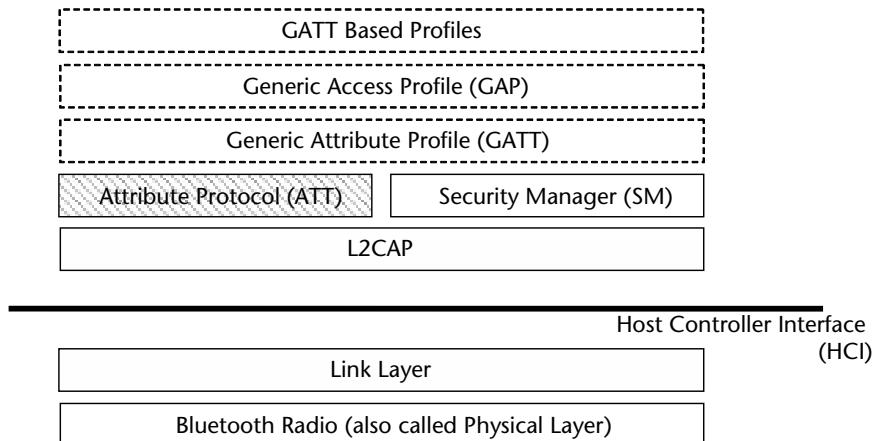


Figure 12.1 ATT Protocol in LE Protocol Stack.

1. Attribute Type.
2. Attribute Handle.
3. Access Permissions.

The structure of an attribute is shown in Figure 12.2.

The clients can discover the handles of the various attributes on the server and can read or write these attributes provided they have sufficient permissions to do so. The server can also inform the client of the value of the attribute at any particular time. The various operations that can be carried out between the client and the server will be explained in detail in further sections.

12.2.1 Attribute Type

The attribute type specifies what that particular attribute represents. This allows the client to understand the meaning of that particular attribute. Some examples of attribute type are:

- <<Primary Service>>
- <<Health Thermometer Service>>
- <<Manufacturer Name>>
- <<Serial Number>>

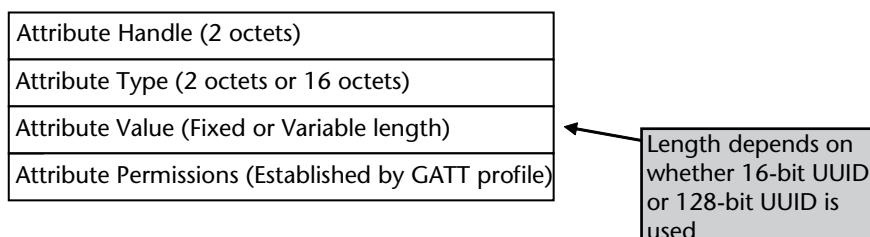


Figure 12.2 Attribute Structure.

The attribute type is identified by a Universally Unique Identifier (UUID). A UUID is a 128-bit value which is considered to be unique over space and time. The implementation could either use the set of predefined UUIDs defined by the Bluetooth SIG or define its own UUIDs.

In general, a shorter form of UUIDs is used. This shorter form is 16-bit. The 16-bit UUIDs are assigned by the Bluetooth SIG and share the same namespace as the 16-bit SDP UUIDs. These are published on the Bluetooth Assigned Numbers page on the Bluetooth SIG website.

The 128-bit value can be derived from the 16-bit value by combining it with the Bluetooth_Base_UUID as follows:

$$128\text{-bit UUID} = 16\text{-bit UUID} * 2^{96} + \text{Bluetooth_Base_UUID}$$

Note: This can also be derived by replacing the xxxx in the following with the hexadecimal value of the 16-bit UUID 0000xxxx-0000-1000-8000-00805F9B34FB.

12.2.2 Attribute Handle

All attributes on the server are assigned a unique, non-zero attribute handle. This handle is used by the client in all operations with the server to identify the attribute.

It is allowed to dynamically add or remove attributes on the server as long as the new attributes are not assigned a handle which has already been used by any other attribute in the past (even if that attribute has been deleted). This ensures that clients always get a unique attribute handle.

Once an attribute has been assigned an attribute handle, it should not change over time. This ensures that the clients can keep accessing that attribute with the same handle.

The attributes on the server are ordered by the attribute handle.

An attribute handle of 0x0000 is reserved and an attribute handle of 0xFFFF is known as the maximum attribute handle.

12.2.3 Attribute Permissions

Each attribute has a set of permissions associated with it which determines the level of access that is permitted for that particular attribute. The attribute permissions are used by a server to determine whether a client is allowed to read or write an attribute value and whether Authentication or Authorization is required to access that particular attribute.

Attribute permissions are a combination of following three permissions:

1. Access Permissions: This could be set to one of the following:
 - Readable.
 - Writable.
 - Readable and Writable.
2. Authentication Permissions: These are used by the server to determine if an authenticated physical link is required when a client attempts to access that attribute or when the server has to send a notification or indication to client. This could be set to one of the following:
 - Authentication Required.
 - No Authentication required.

3. Authorization Permissions: These are used by the server to determine if a client needs to be authorized before accessing an attribute value. This could be set to one of the following:
 - Authorization Required.
 - No Authorization Required.

If the client does not have sufficient permissions an error code is returned. Some of the error codes are shown below:

- Read Not Permitted: The attribute cannot be read.
- Write Not Permitted: The attribute cannot be written.
- Insufficient Authentication: The attribute requires authentication before it can be read or written.
- Insufficient Authorization: The attribute requires authorization before it can be read or written.
- Insufficient Encryption Key Size: The Encryption Key Size used for encrypting this link is insufficient.
- Insufficient Encryption: The attribute requires encryption before it can be read or written.

12.2.4 Attribute Value

An attribute value is an octet array that contains the actual value of the attribute. The length of the attribute can be either fixed or variable:

- Fixed length—The length can be one octet, two octet or four octet.
- Variable Length—The attribute can be a variable length string.

To simplify things, ATT does not allow multiple attribute values to be transmitted in a single PDU. A PDU contains only one attribute value and if the attribute value is too long to transmit in a single PDU, it can be split across multiple PDUs. There are some exceptions to this though—when a client requests for multiple attributes to be read and the attributes have a fixed length, then the response can contain multiple attributes.

12.2.5 Control Point Attributes

The attributes that cannot be read, but can only be written, notified or indicated are called control point attributes. These are generally used to control the behavior of a device and therefore called control point attributes.

One example of Control Point Attribute is the Alert Level characteristic defined by the Immediate Alert Service. The remote devices can write the alert level into this attribute (It can be one of “No Alert”, “Mid Alert”, or “High Alert”). Once the alert level is written the device may take some specific action like flashing an LED or sounding an alarm. This characteristic can only be written but cannot be read.

12.2.6 Grouping of Attribute Handles

ATT allows a set of attributes to be grouped together. In this case, a specific attribute is placed at the beginning of the other attributes that form the group. The clients can request the first and the last handle associated with the group of attributes.

The groups are defined by the higher layer profile, GATT. GATT defines grouping of attributes for three attribute types:

- <<Primary Service>>
- <<Secondary Service>>
- <<Characteristic>>

Once the attributes are grouped together, operations like Read By Group Type can be done on those attributes.

These will be explained in detail in the next chapter.

12.2.7 Atomic Operations

At any point of time, multiple clients may be connected to a server. The server treats each operation from the client as an atomic operation which is not impacted by any other client initiating an operation at the same time.

For example if a client is writing a large amount of data by splitting it into chunks, then the server treats this as an atomic operation. The commands to do this, viz Prepare Write Request and Execute Write Request, will be explained in further detail later in this chapter. This operation is not impacted if another client starts doing another write operation in parallel.

12.3 Attribute Protocol

The attribute protocol defines how a device will discover, read and write the attributes of another device. The following two roles are defined.

1. Attribute Server—The attribute server exposes a set of attributes and their associated values to a peer device.
2. Attribute Client—The attribute client can discover, read and write the attributes on a server. In addition to this it can be indicated and notified by the server.

The attribute protocol supports data to be transferred both from the client to server and vice versa. The client can read or write the attributes of the server. In addition to this the server can also notify the client about any attribute.

The PDUs supported by ATT can be of following six method types:

1. Request—This PDU is sent by the client to the server and the server responds with a response.
2. Response—This PDU is sent by the server in response to a request from the client.

3. Command—This PDU is sent by the client to the server. No response is received for this PDU.
4. Notification—This PDU is sent by the server to the client. The client does not send anything back.
5. Indication—This PDU is sent by the server to the client and the client responds with a confirmation.
6. Confirmation—This PDU is sent by the client to confirm the receipt of an indication.

These are shown in Figure 12.3.

12.3.1 PDU Format

The format of Attribute protocol PDUs is shown in Figure 12.4.

The Opcode contains the following:

- Method—Identifies the method for which this PDU is sent.
- Command Flag—Indicates whether this is a command. There is no acknowledgement from the server for a command.
- Authentication Signature Flag—if this is 1, then an Authentication Signature is appended to the PDU. (Authentication Signature will be explained shortly.)

The Attribute Parameters contain the data as per the specified Method.

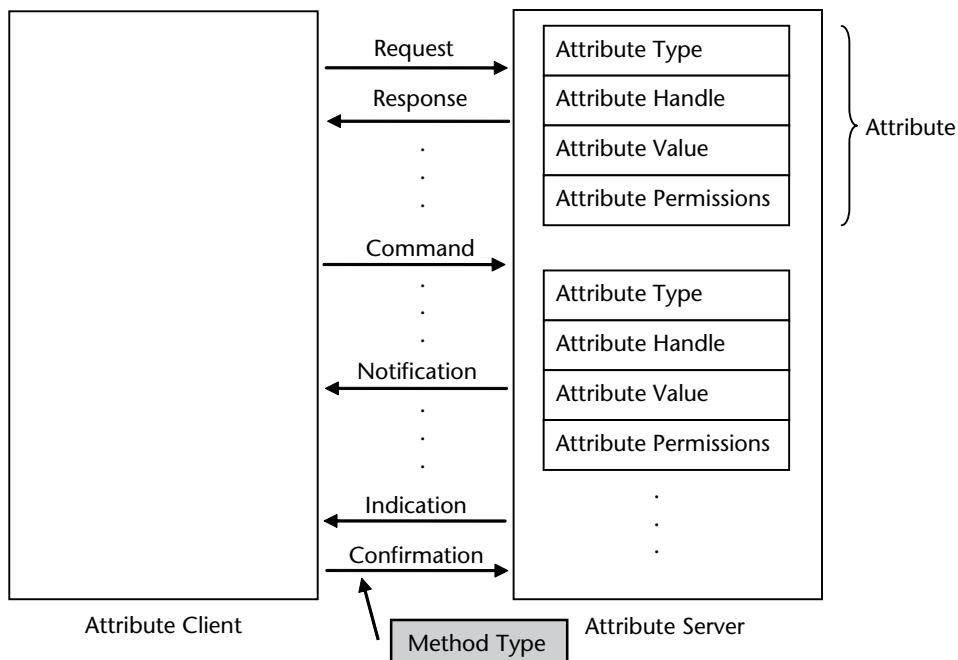


Figure 12.3 Attribute Client and Server.

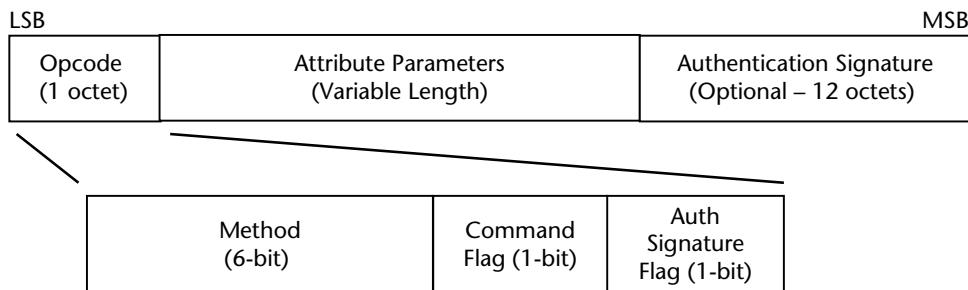


Figure 12.4 Format of Attribute Protocol PDU.

The Authentication Signature is an optional field and is used for signing the data that is present in this PDU. If this field is present, then it provides a signature for the Opcode and Attribute Parameters fields. This field may be used if the link is not encrypted but security is required.

12.3.2 Sequential Transactions

The Attribute protocol is a sequential request-response, indication-confirmation protocol.

- For commands and responses—The client sends one request to the server and then waits for the response before sending the next request.
- For indications and confirmations—The server sends one indication to the client and then waits for the confirmation from the client before sending the next indication.

This introduces a flow control mechanism for request-response and indication-response transactions. The flow control mechanism is valid only for request-response pairs and indication-confirmation pairs. So it is possible to have sequences like request-indication-response-confirmation or request-notification-response.

Flow control of each client and server is independent.

A request-response pair or an indication-confirmation pair is called a transaction. ATT defines a transaction timeout of 30 seconds. This means that if a request is sent from the client to the server and the response is not received within 30 seconds, then the transaction is considered to have timed out and the client can inform the higher layers which initiated the transaction about the failure.

For notifications and commands, there is no response from the other side. These can be sent at any time and there is no flow control. The notifications and commands that cannot be processed are just discarded. So these PDUs are considered to be unreliable.

One example of notifications is the intermediate temperature measurements sent by a server acting as a thermometer. If the client receives those notifications, then it displays the intermediate temperature value, otherwise those values just get discarded.

12.4 Methods

As mentioned previously, the PDUs supported by ATT can be of following six method types:

1. Request: Invokes a Response;
2. Response;
3. Command;
4. Notification;
5. Indication: Invokes a Confirmation;
6. Confirmation.

12.4.1 Request and Response Type Methods

The different PDUs for Request and Response type method are shown in Table 12.1. These PDUs are sent from the client to the server. The response that can be sent by the server for each request is also shown. Besides that response, an Error Response can also be sent from the server to the client.

12.4.1.1 Exchange MTU Request

The Exchange MTU Request is sent by the client to the server to:

- Inform the server of the client's maximum receive MTU size.
- Request the server to respond with the server's maximum MTU size.

This request has the following parameter:

- Client Rx MTU: Client Receive MTU Size.

This request is sent only by the client to the server. It is sent only once during the connection.

Table 12.1 Methods of Request and Response Type

<i>Request</i>	<i>Response</i>
Exchange MTU Request	Exchange MTU Response
Find Information Request	Find Information Response
Find By Type Value Request	Find By Type Value Response
Read By Type Request	Read By Type Response
Read Request	Read Response
Read Blob Request	Read Blob Response
Read Multiple Request	Read Multiple Response
Read By Group Type Request	Read By Group Type Response
Write Request	Write Response
Prepare Write Request	Prepare Write Response
Execute Write Request	Execute Write Response

The default value of ATT_MTU is 23 octets for LE. The exchange MTU request is used by the client if it wants to use an ATT_MTU bigger than this value. A bigger value will finally be used if both client and server support that bigger value.

The message sequence chart of the Exchange MTU Request is shown in Figure 12.5.

12.4.1.2 Exchange MTU Response

The Exchange MTU Response is sent by the server to the client in response to the Exchange MTU Request. It contains the server's receive MTU size

This response has the following parameter:

- Server Rx MTU: Server Receive MTU Size.

The message sequence chart of the Exchange MTU Response is shown in Figure 12.5.

After Exchange MTU request and response is complete, both the client and server set their MTU_SIZE (ATT_MTU) to the minimum of server and client receive MTU sizes. Both of them use the same size for all further requests and responses.

A practical example of Exchange MTU Request and Response is shown in Figure 12.19 towards the end of this chapter.

12.4.1.3 Find Information Request

The Find Information Request is sent by the client to the server to get a list of attribute handles and their types. The structure of an attribute was shown in Figure 12.2. This request is used to get the first two fields (Attribute Handle, Attribute Type) of that structure for a range of attributes.

The range of attributes is specified by two parameters:

- Starting Handle: Starting handle of the range of handle numbers to search.
- Ending Handle: Ending handle of the range of handle numbers to search.

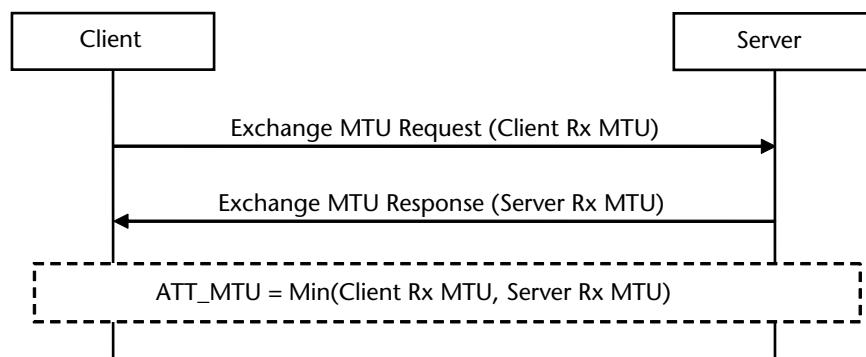


Figure 12.5 Exchange MTU Request and Response.

The message sequence chart of the Find Information Request is shown in Figure 12.6.

12.4.1.4 Find Information Response

The Find Information Response is sent by the server to the client in response to the Find Information Request. It contains the handle-UUID pairs (Attribute Handle, Attribute Type) for the range of Attribute Handles that was provided by the client.

The parameters of this response are:

- Format—Specifies the format of information data—Whether the handles contain 16-bit UUID or 128-bit UUID.
- Attribute Data List—handle-value pairs containing Attribute Handle and Attribute Value.

The UUID can be either 16-bit or 128-bit. The type that is returned is contained in the Format field.

Since the range of attributes can be long, this response can be split across multiple response packets but the handle-UUID pairs are not split across packets. The handle-UUID pairs are sent in ascending order of the handle value.

The message sequence chart of the Find Information Response is shown in Figure 12.6.

12.4.1.5 Find By Type Value Request

The Find By Type Value Request is sent by the client to the server to get a list of attributes which contain a particular Attribute Type and Attribute Value.

The parameters of this request are:

- Starting Handle: Starting handle of the range of handle numbers to search.
- Ending Handle: Ending handle of the range of handle numbers to search.
- Attribute Type: UUID that specifies the Attribute Type to find.
- Attribute Value: Attribute Value to find.

The UUID can be specified as only 16-bit.

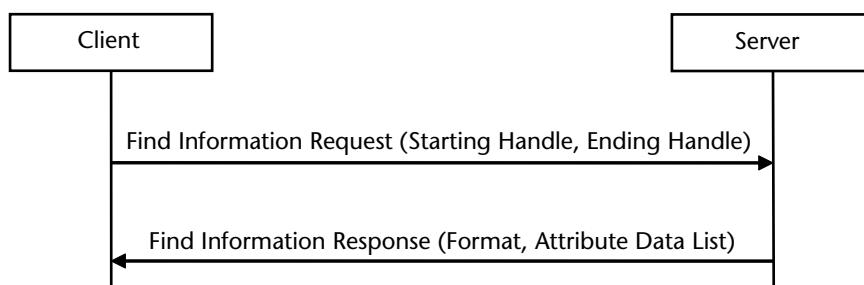


Figure 12.6 Find Information Request and Response.

The message sequence chart of the Find By Type Value Request is shown in Figure 12.7.

12.4.1.6 Find By Type Value Response

The Find By Type Value Response is sent by the server to the client in response to the Find By Type Value Request. It contains the list of handles that correspond to attributes which exactly match the Attribute Type and Attribute Value provided by the client in the Find By Type Value Request.

The parameters of this response are:

- Handle Information List: List of handles that match the Attribute Type and Attribute Value.

The message sequence chart of the Find By Type Value Response is shown in Figure 12.7.

12.4.1.7 Read By Type Request

The Read By Type Request is sent by the client to the server to get a list of attributes of a particular Attribute Type.

The parameters of this request are:

- Starting Handle: Starting handle of the range of handle numbers to search.
- Ending Handle: Ending handle of the range of handle numbers to search.
- Attribute Type: UUID that specifies the Attribute Type to find.

The UUID can be specified as either 16-bit or 128-bit.

The message sequence chart of the Read By Type Request is shown in Figure 12.8.

12.4.1.8 Read By Type Response

The Read By Type Response is sent by the server to the client in response to the Read By Type Request. It contains the list of handle-value pairs (Attribute Handle,

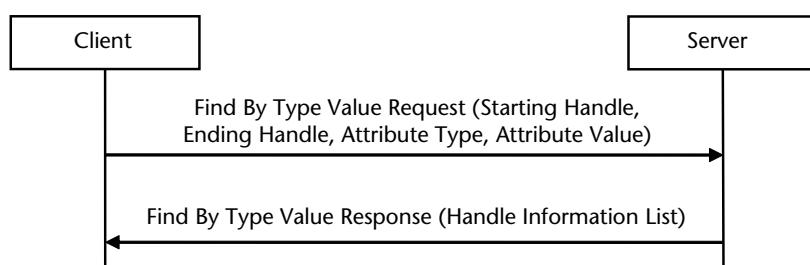


Figure 12.7 Find By Type Value Request and Response.

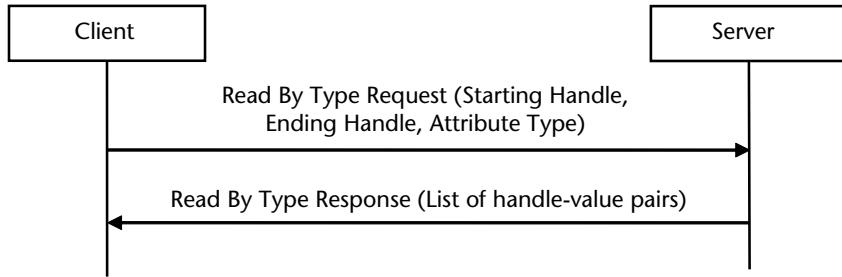


Figure 12.8 Read By Type Request and Response.

Attribute Value) that correspond to attributes which exactly match the Attribute Type provided by the client in the Read By Type Request.

The parameters of this response are:

- Length: The size of each attribute handle-value pair.
- Attribute Data List: handle-value pairs containing Attribute Handle and Attribute Value.

Since the range of attributes can be long, this response can be split across multiple response packets but the handle-value pairs are not split across packets. The handle-value pairs are sent in ascending order of the handle value.

The message sequence chart of the Read By Type Response is shown in Figure 12.8.

12.4.1.9 Read Request

The Read Request is sent by the client to the server to read the Attribute Value of an attribute.

This request has the following parameter:

- Attribute Handle: The handle of the attribute that is to be read.

The message sequence chart of the Read Request is shown in Figure 12.9.

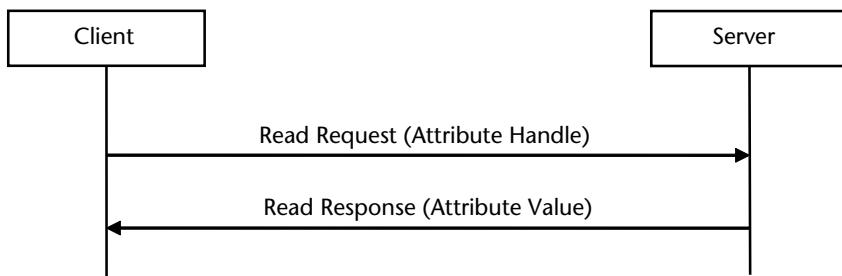


Figure 12.9 Read Request and Response.

12.4.1.10 Read Response

The Read Response is sent by the server to the client in response to the Read Request. It contains the Attribute Value corresponding to the attribute specified by the Attribute Handle provided by the client in the Read Request.

The parameters of this response are:

- Attribute Value: Attribute Value corresponding to the Attribute Handle.

If the Attribute Value is too big to fit in the ATT_MTU -1 size, then only the ATT_MTU-1 octets are returned and the remaining octets can be read by the Read Blob request.

The message sequence chart of the Read Response is shown in Figure 12.9.

12.4.1.11 Read Blob Request

The Read Blob Request is sent by the client to the server to read the Attribute Value of an attribute starting from a particular offset. This request is useful in scenarios where the Attribute Value is long and cannot fit ATT_MTU-1 octets. In that case the Read Response reads only the first ATT_MTU-1 octets and the remaining octets can be read using the Read Blob Request

The parameters of this request are:

- Attribute Handle: The handle of the attribute that is to be read.
- Value Offset: The offset from where the part of Attribute Value should be read. The Value Offset parameter is based from zero. This means that the first octet has an offset of zero.

The message sequence chart of the Read Blob Request is shown in Figure 12.10.

12.4.1.12 Read Blob Response

The Read Blob Response is sent by the server to the client in response to the Read Blob Request. It contains a part of the Attribute Value corresponding to the attribute specified by the Attribute Handle provided by the client in the Read Blob request. The starting octet of the part of the Attribute Value corresponds to the Value Offset that was provided in the Read Blob Request.

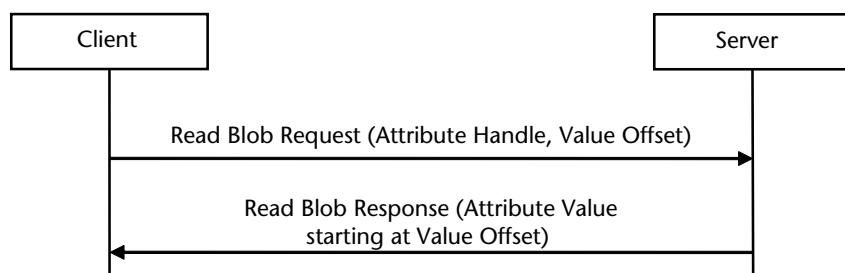


Figure 12.10 Read Blob Request and Response.

The parameter of this response is:

- Part Attribute Value: octets of the Attribute Value starting at Value Offset.

If the part of the Attribute Value that is to be returned is too big to fit in the ATT_MTU -1 size, then only the ATT_MTU-1 octets are returned and the remaining octets can be read by another Read Blob Request by adjusting the offset.

The message sequence chart of the Read Blob Response is shown in Figure 12.10.

12.4.1.13 Read Multiple Request

The Read Multiple Request is sent by the client to the server to read the Attribute Values of two or more attributes. This request is useful in scenarios where multiple attributes are to be read using one single request.

This request has the following parameter:

- Set of Handles: A set of two or more attribute handles.

The message sequence chart of the Read Multiple Request is shown in Figure 12.11.

12.4.1.14 Read Multiple Response

The Read Multiple Response is sent by the server to the client in response to the Read Multiple Request. It contains a set of Attribute Values corresponding to the attributes specified by the Attribute Handles provided by the client in the Read Multiple Request. The set of values are just concatenated in the same order in which the Attribute Handles were provided in the Read Multiple Request.

The parameter of this response is:

- Set of Values: A set of two or more values corresponding to the Set of Handles.

The message sequence chart of the Read Multiple Response is shown in Figure 12.11.

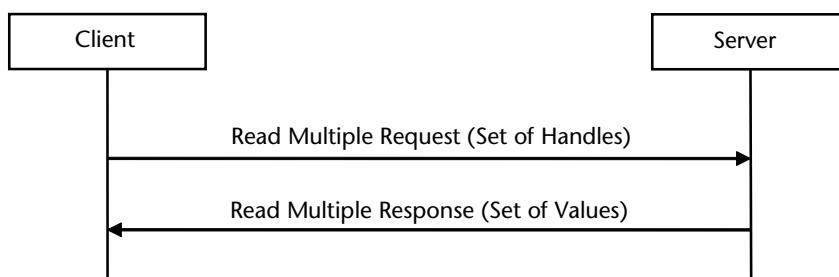


Figure 12.11 Read Multiple Request and Response.

12.4.1.15 Read By Group Type Request

The Read By Group Type Request is sent by the client to the server to read the Attribute Values of a particular group of services. This request is similar to the Read By Type Request. The only difference is that it specifies an Attribute Group Type instead of an Attribute Type.

The GATT Profile provides support for grouping attributes into various types. This request is used to read the attributes of the <>Primary Service<> and <>Secondary Service<> groups. This will be explained in detail in the next chapter.

The parameters of this request are:

- Starting Handle: Starting handle of the range of handle numbers to search.
- Ending Handle: Ending handle of the range of handle numbers to search.
- Attribute Group Type: UUID that specifies the Attribute Group Type to read.

The message sequence chart of the Read By Group Type Request is shown in Figure 12.12.

A practical example of Read By Group Type Request that is used to retrieve the Primary Services from a server is shown in Figure 12.20 towards the end of this chapter.

12.4.1.16 Read By Group Type Response

The Read By Group Type Response is sent by the server to the client in response to the Read By Group Type Request. It contains the handles and values of the attributes with the specified Attribute Group Type.

The parameters of this response are:

- Length: Size of each attribute data.
- Attribute Data List: This contains the Attribute Handle, the End Group Handle and the Attribute Value.

The End Group Handle is the handle of the last attribute within a group.

The message sequence chart of the Read By Group Type Response is shown in Figure 12.12.

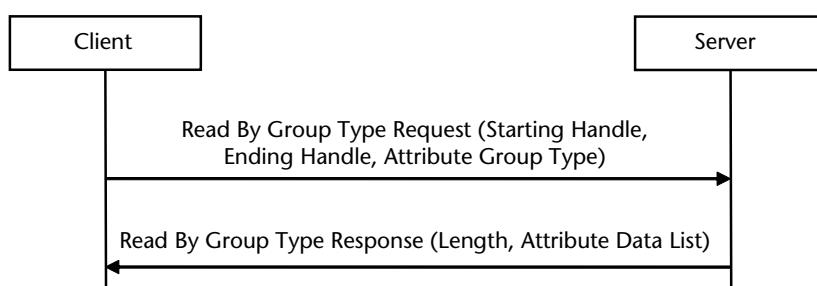


Figure 12.12 Read By Group Type Request and Response.

A practical example of Read By Group Type Response that is used to retrieve the Primary Services from a server is shown in Figure 12.21 towards the end of this chapter.

12.4.1.17 Write Request

The Write Request is sent by the client to the server to request the server to write the value of an attribute and acknowledge that it has written it by a Write Response. If there is any error while writing (For example insufficient permissions), then an Error response is sent by the server to the client.

The parameters of this request are:

- Attribute Handle: The handle of the attribute.
- Attribute Value: The value to be written

The message sequence chart of the Write Request is shown in Figure 12.13.

12.4.1.18 Write Response

The Write Response is sent by the server to the client in response to the Write Request. It confirms that the Attribute Value has been successfully written.

This response does not have any parameters.

The message sequence chart of the Write Response is shown in Figure 12.13.

12.4.1.19 Prepare Write Request

The Prepare Write Request is sent by the client to the server to prepare to write the value of an attribute. This is useful in scenarios where the Attribute Value is long and cannot fit one PDU. In this case the client sends multiple Prepare Write Request PDUs to the server and the server queues those requests. Once all Prepare Write Request PDUs have been sent, the client sends an Execute Write Request PDU. After the Execute Write Request PDU is received, the server writes all the queued octets of the Attribute Value that were received in the Prepare Write Request.

The new Attribute Values take effect only after the Execute Write Request has been executed on the server.

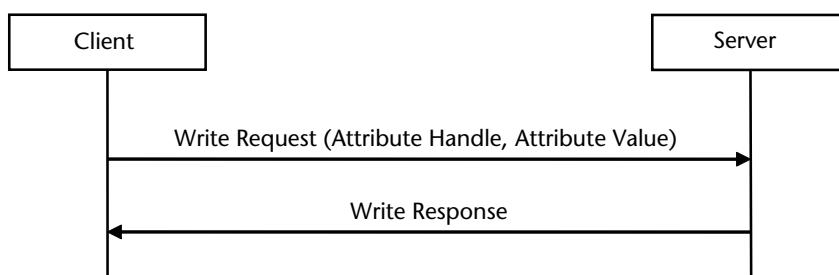


Figure 12.13 Write Request and Response.

If a server is serving multiple clients, then it queues the Prepare Write Requests from each of the clients separately. The execution of queue from one client does not affect the queue from another client.

The parameters of this request are:

- Attribute Handle: The handle of the attribute.
- Value Offset: The offset of the first octet to be written.
- Part Attribute Value: The octets of the Attribute Value to be written starting at Value Offset.

The message sequence chart of the Prepare Write Request is shown in Figure 12.14.

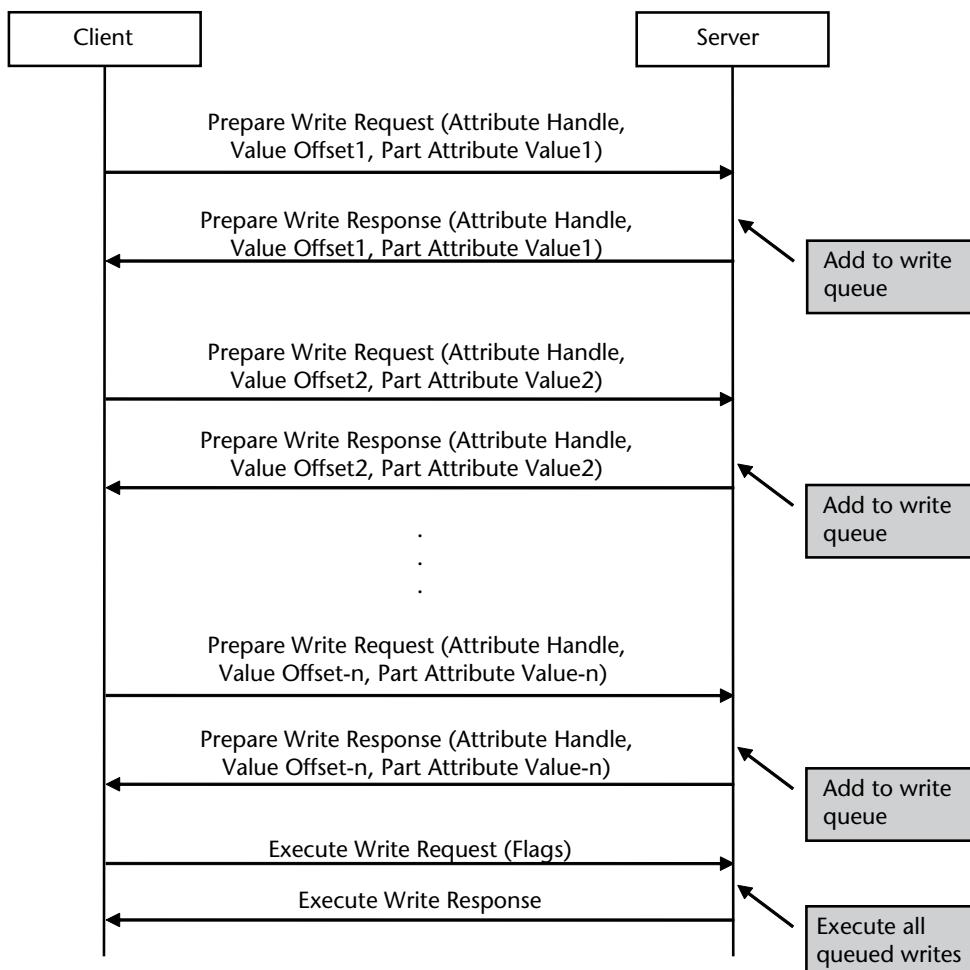


Figure 12.14 Prepare Write Request and Response, Execute Write Request and Response.

12.4.1.20 Prepare Write Response

The Prepare Write Response is sent by the server to the client in response to the Prepare Write Request. It is used to acknowledge that the value has been received by the server and placed in the write queue.

The parameters of this request are the same as those of the Prepare Write Request. These are:

- Attribute Handle: Same as the one provided in Prepare Write request
- Value Offset: Same as the one provided in the Prepare Write request.
- Part Attribute Value: Same as the one provided in the Prepare Write request.

The message sequence chart of the Prepare Write Response is shown in Figure 12.14.

12.4.1.21 Execute Write Request

The Execute Write Request is sent by the client to the server to either execute the write or cancel the write of all prepared values that were inserted in the write queue of the server using Prepare Write Requests.

If a server is serving multiple clients, then it queues the Prepare Write Requests from each of the clients separately. The execution of queue from one client does not affect the queue from another client.

This request has the following parameter:

- Flags – 0x00: Cancel all prepared writes. 0x01: Write all pending prepared values.

If the Flags parameter contains 0x01, then all pending writes that were queued earlier are written in the same order in which they were queued.

The message sequence chart of the Execute Write Request is shown in Figure 12.14.

12.4.1.22 Execute Write Response

The Execute Write Response is sent by the server to the client in response to the Execute Write Request. It is used to confirm that all the values that were earlier queued have been successfully written.

This response does not have any parameters.

The message sequence chart of the Execute Write Response is shown in Figure 12.14.

12.4.2 Command Type Methods

The different PDUs for Command type are shown in Table 12.2. These PDUs are sent from the client to the server. There is no response or Error Response from the server in response to these PDUs.

Table 12.2 Methods of Command Type

Write Command
Signed Write Command

12.4.2.1 Write Command

The Write Command is sent by the client to the server to request the server to write the value of an attribute. Generally this command is used for control-point attributes. The server does not acknowledge that it has written the value. Even if there is an error, the server does not send an Error Response.

The parameters of this command are:

- Attribute Handle: The handle of the attribute.
- Attribute Value: The value to be written.

The message sequence chart of the Write Command is shown in Figure 12.15.

12.4.2.2 Signed Write Command

The Signed Write Command is sent by the client to the server to request the server to write the value of an attribute after including an authentication signature with the data PDU. Generally this command is used for control-point attributes. The server does not acknowledge that it has written the value. Even if there is an error (like authentication signature verification fails), the server does not send an Error Response.

The parameters of this command are:

- Attribute Handle: The handle of the attribute.
- Attribute Value: The value to be written.
- Authentication Signature: 12-octet authentication signature.

The Authentication Signature provides the feature of data signing. The server verifies the authentication signature to check the authenticity of the client before actually writing that value.

The message sequence chart of the Signed Write Command is shown in Figure 12.16.



Figure 12.15 Write Command.

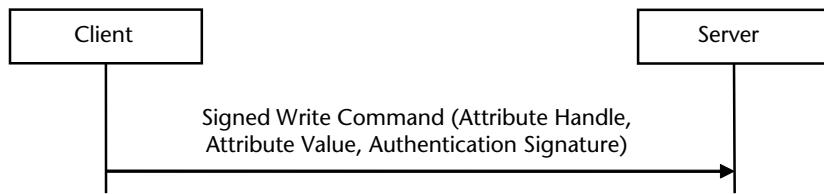


Figure 12.16 Signed Write Command.

12.4.3 Notification Type Methods

The different PDUs for Notification type are shown in Table 12.3. These PDUs are sent from the server to the client. There is no response from the client for these PDUs.

12.4.3.1 Handle Value Notification

The Handle Value Notification is sent by the server to the client to provide information about an attribute at any time.

The parameters of this notification are:

- Attribute Handle: The handle of the attribute.
- Attribute Value: The current value of the attribute.

The message sequence chart of the Handle Value Notification is shown in Figure 12.17.

12.4.4 Indication and Confirmation Type Methods

The different PDUs for Indication and Confirmation type are shown in Table 12.4. The Indication PDUs are sent from the server to the client. The client responds with a Confirmation PDU.

Table 12.3 Methods of Notification Type

Handle Value Notification

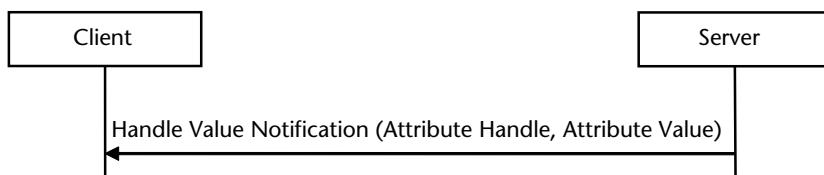


Figure 12.17 Handle Value Notification.

Table 12.4 Methods of Indication and Confirmation Type

<i>Indication</i>	<i>Confirmation</i>
Handle Value Indication	Handle Value Confirmation

12.4.4.1 Handle Value Indication

The Handle Value Indication is sent by the server to the client to provide information about an attribute at any time.

The only difference from Handle Value Notification is that in this case the client responds with a Handle Value Confirmation. In case the value of the attribute is long, the client can use a Read Blob Request to get the entire value of the attribute after receiving the notification.

The parameters of this indication are:

- Attribute Handle: The handle of the attribute.
- Attribute Value: The current value of the attribute.

The message sequence chart of the Handle Value Notification is shown in Figure 12.18.

12.4.4.2 Handle Value Confirmation

The Handle Value Confirmation is sent by the client to the server in response of a Handle Value Indication to confirm that it has received the Handle Value Indication.

This confirmation does not have any parameters.

The message sequence chart of the Handle Value Confirmation is shown in Figure 12.18.

12.5 Practical Examples

12.5.1 Exchange MTU

The air log captures of Exchange MTU Request and Response are shown in Figure 12.19. It shows that the client sends an Rx MTU size of 525 octets while the server responds with the Server Rx MTU of 23 octets. After these two PDUs are exchanged, both client and server will set the MTU to the minimum of the two values.

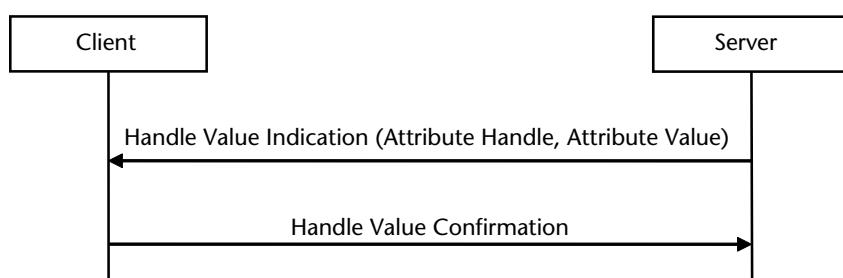


Figure 12.18 Handle Value Indication and Confirmation.

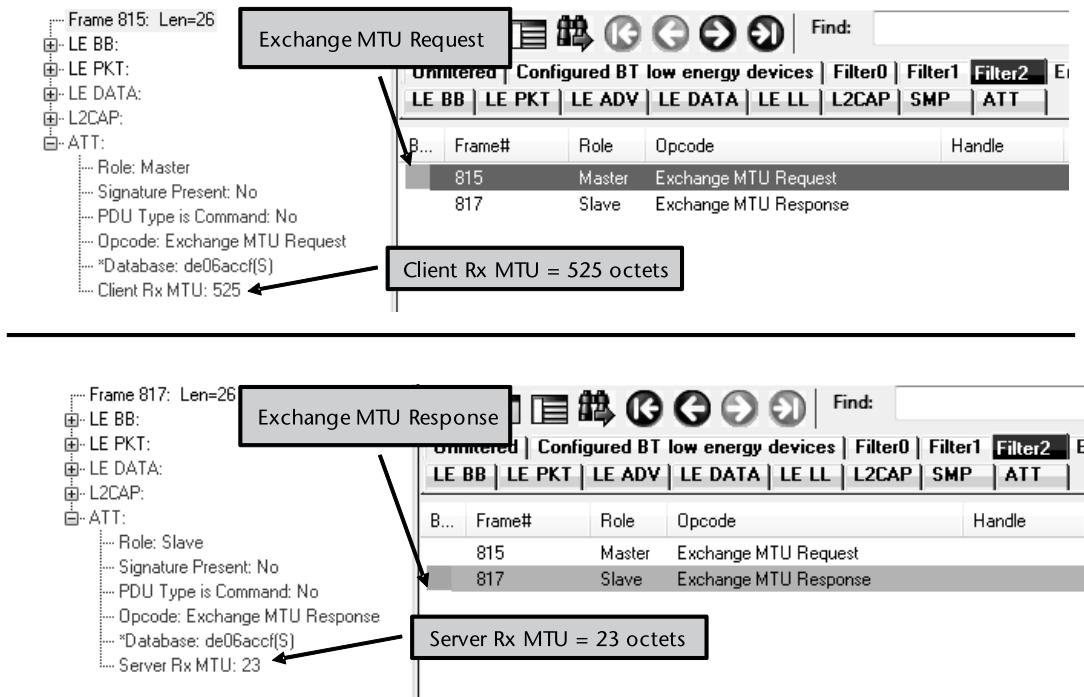


Figure 12.19 Example of Exchange MTU Request and Exchange MTU Response.

This means that both the client and the server will set the MTU size to 23 octets and will not be transmitting any packets with data more than 23 octets.

12.5.2 Reading Primary Services of a Device

The list of primary devices supported by a remote device can be read by using the Read By Group Type Request with the UUID of <<Primary Service>>.

This is done by the client first sending the Read By Group Type Request to the server with the following parameters.

- Starting Attribute Handle = 1
- Ending Attribute Handle = 65535
- Attribute Group Type = Primary Service

This is shown in Figure 12.20.

The server responds to this request with a Read By Group Type Response which contains the Attribute Data List. In this example, the server returns the length as 6 octets for each handle-value pair and three service declarations in the response:

- Length: The size of each attribute handle-value pair = 6 octets.
- Attribute Data List: handle-value pairs containing Attribute Handle and Attribute Value.

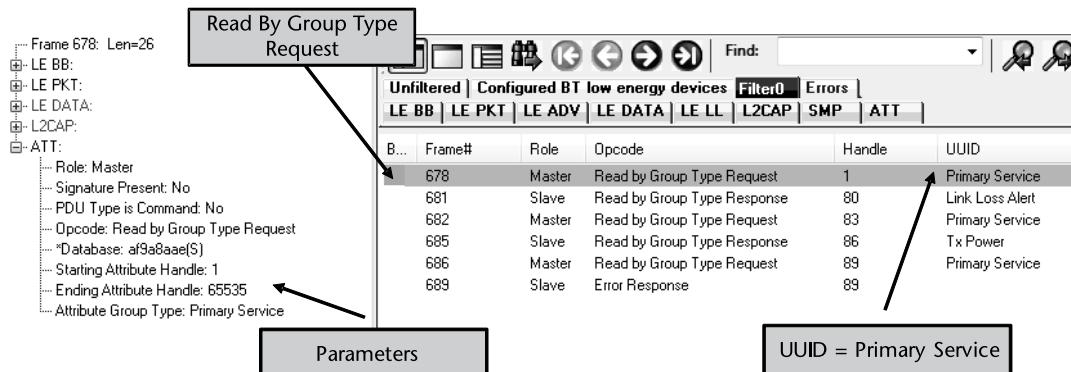


Figure 12.20 First Read By Group Type Request for getting Primary Services.

1. Generic Access Profile with Starting Attribute Handle 0x01 and Ending Attribute Handle 0x07.
2. Generic Attribute Profile with Starting Attribute Handle 0x16 and Ending Attribute Handle 0x19.
3. Link Loss Alert with Starting Attribute Handle 0x80 and Ending Attribute Handle 0x82.

This is shown in Figure 12.21. It may be noted that the last handle that is returned is 0x82. This indicates to the client that if it wants to retrieve further services, then it should start with the next higher attribute handle, i.e., 0x83.

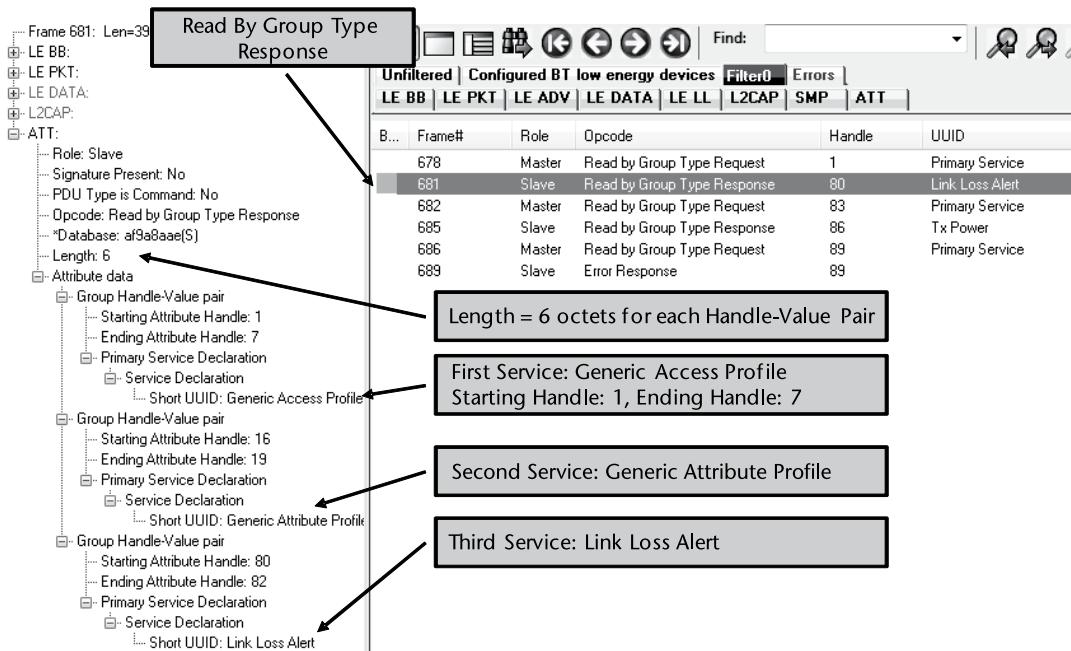


Figure 12.21 First Read By Group Type Response providing the primary services.

The client then continues to read more primary services. This time it sends the Starting Attribute Handle as 0x83 and all other parameters same as the previous request. This is shown in Figure 12.22.

The client responds with the set of services contained between the Attribute Handles 0x83 and 0x65535. This includes:

- Length: The size of each attribute handle-value pair = 6 octets.
- Attribute Data List: handle-value pairs containing Attribute Handle and Attribute Value.
 1. Immediate Alert Service with Starting Attribute Handle 0x83 and Ending Attribute Handle 0x85.
 2. Tx Power Service with Starting Attribute Handle 0x86 and Ending Attribute Handle 0x88.

This is shown in Figure 12.23.

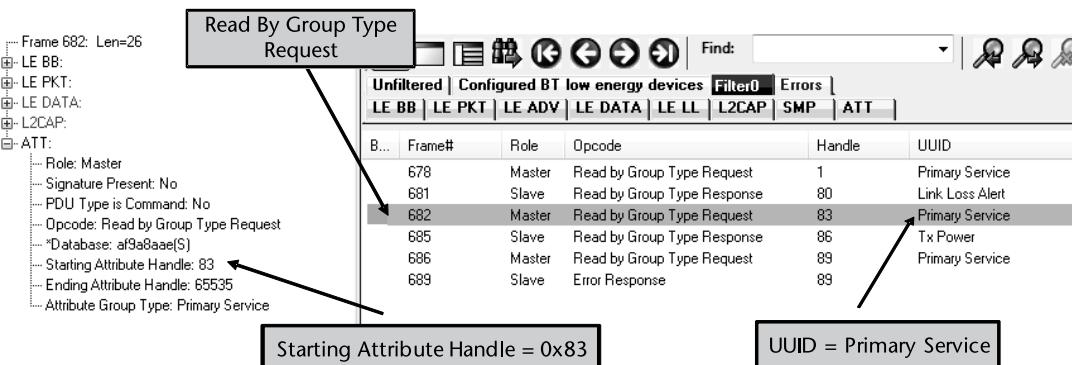


Figure 12.22 Second Read By Group Type Request.

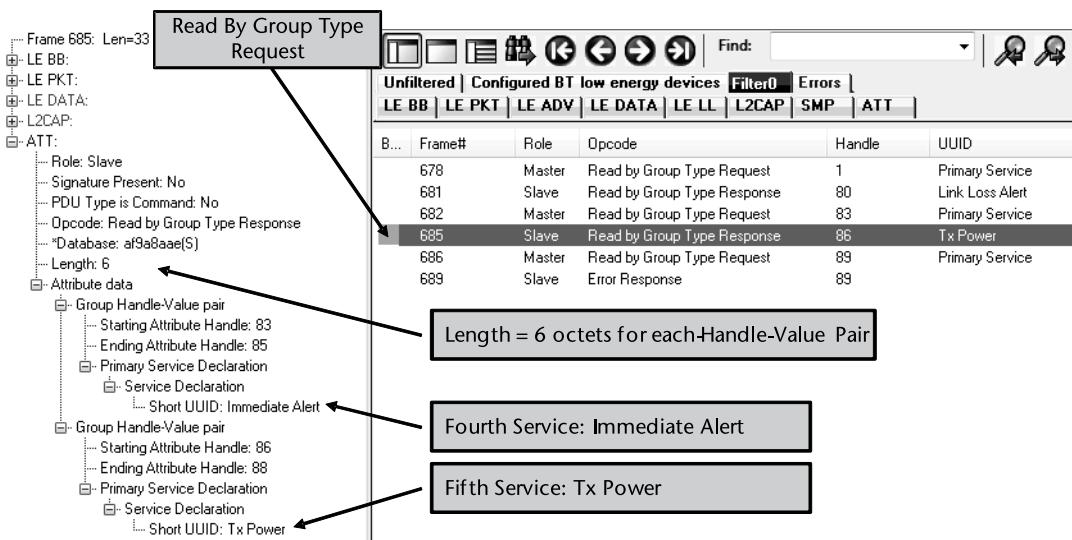


Figure 12.23 Second Read By Attribute Type Response.

This time the client receives the last attribute handle as 0x88. So it sends a third Read By Attribute Type Request, this time with the Starting Handle 0x89 to retrieve the next set of primary services.

This time the server does not have any more Primary services to report between the specified Attribute Handles. So it responds with an Error Response. This is shown in Figure 12.24.

This indicates to the client that it has received all the attributes that it had requested.

12.6 Summary

The Attribute Protocol provides a very simple mechanism to exchange information between devices in the form of attributes. An attribute could be any data that a device decides to share with other devices. Besides the data, the attribute also contains additional information about the data like type of data, access permissions, etc.

ATT follows a client-server model. To keep the protocol simple, only sequential transactions are supported. This means that the next transaction is not initiated until a previous transaction is completed. In general the amount of data transferred in LE is quite small, typically in the range of 10–20 octets so that it can be fit in a single LE ACL data packet. Still ATT provides features to read and write longer

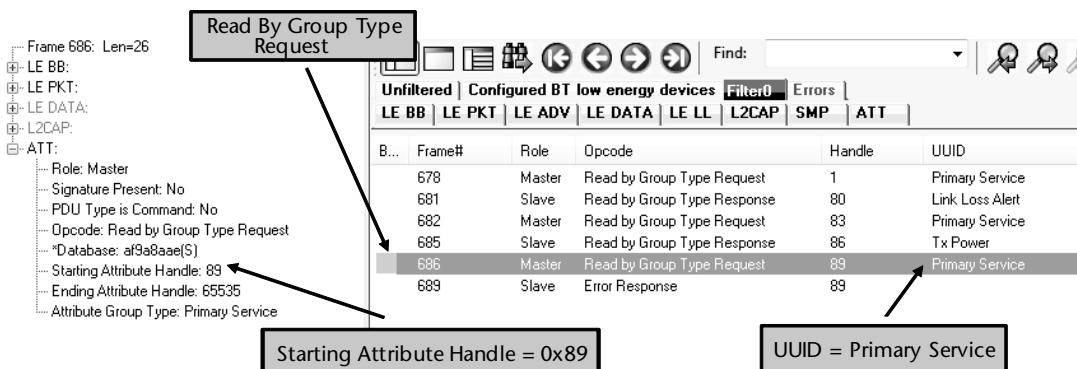


Figure 12.24 Third Read By Attribute Type Request.

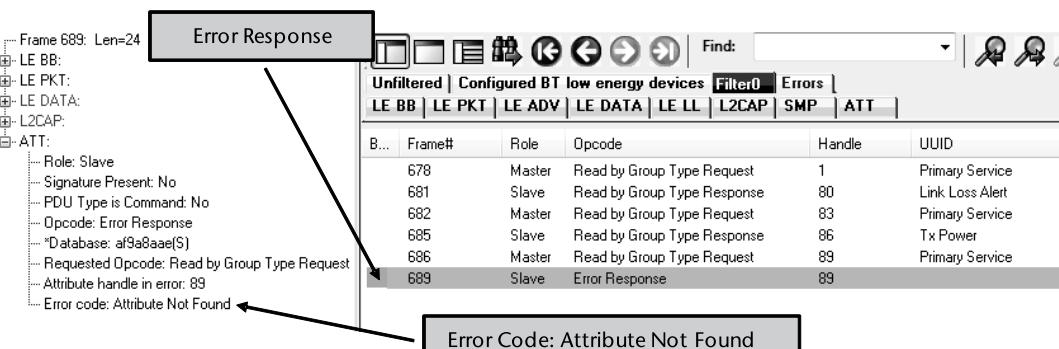


Figure 12.25 Error Response from the server indicating no further attributes.

amount of data by breaking that into smaller prepare transactions followed by an execute transaction.

The services of ATT are used by the Generic Attribute Profile which defines a hierarchy of services and characteristics using these attributes. This will be explained in detail in the next chapter.

Bibliography

Bluetooth Core Specification 4.0 <http://www.bluetooth.org>.

Assigned Numbers Specification: <https://www.bluetooth.org/Technical/AssignedNumbers/home.htm>.

Generic Attribute Profile (GATT)

13.1 Introduction

In the previous chapter, the ATT protocol described the structure of an attribute and the mechanisms of accessing those attributes. The Generic Attribute Profile (GATT) defines a framework of services and characteristics using these attributes as building blocks. It also defines how a device will discover, read, write, notify, and indicate the characteristics. GATT also defines the mechanisms for configuring the broadcast of attributes. The position of GATT in LE protocol stack is shown in Figure 13.1.

The ATT protocol defines a flat set of attributes and mechanisms to access those attributes. ATT protocol does not define any hierarchy of the attributes. GATT allows the server to define a hierarchy so that the attributes are grouped into primary and secondary services and these services can include characteristics. Each of these characteristics has its own set of permissions which can be defined by GATT or a higher layer entity.

The GATT and ATT can be used on both BR/EDR and LE transport. It is mandatory to implement GATT and ATT for LE since these provide the basic capability to discover services of a remote device (similar to SDP in the case of BR/EDR). It is optional to implement GATT and ATT on the BR/EDR transport.

The GATT profile defines the client and server roles similar to the SDP profile. A device may act as a server, a client, or both. In general the LE-only devices (like sensors) are quite simple and only implement the server roles with services and characteristics needed to fulfill the application requirements.

The GATT profile is defined in such a manner that most of the complexity is encapsulated within this profile. The profiles that are on top of GATT are very simple.

The symbols <> are used to indicate a Bluetooth SIG defined UUID. For example <Characteristic> is used to indicate a Bluetooth SIG defined UUID for Characteristic. It is a 128-bit unique value. In general, a shorter 16-bit equivalent of this value is used.

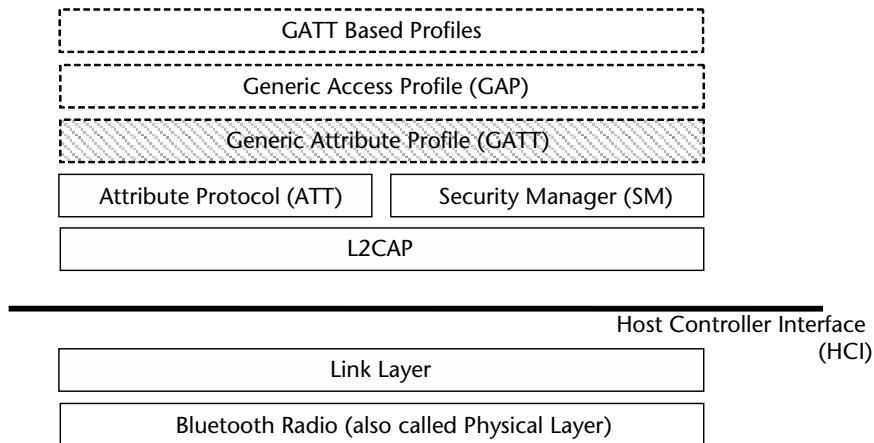


Figure 13.1 GATT in LE Protocol Stack.

13.1.1 Profile Dependencies

Similar to BR/EDR profiles, the dependencies of GATT based profiles are shown in Figure 13.2. A device can support one or more profiles at the same time.

Generic Access Profile (GAP) and Generic Attribute Profile (GATT) are mandatory for all devices that support LE. Devices may implement more profiles depending on the requirements of the application. The dependencies amongst profiles are depicted in Figure 13.2. One profile is dependent on another profile if it uses parts of that profile. A dependent profile is shown in an inner box and the outer box indicates profiles on which it is directly or indirectly dependent. GAP and GATT are shown in the outermost box since all other profiles are dependent on it.

GATT based profile architecture is much simpler than the architecture of BR/EDR profiles. It has only two layers. The outermost layer is GAP and GATT. All other profiles are located inside it at the same level. There is no further layering between the profiles (this is another step towards simplicity in LE).

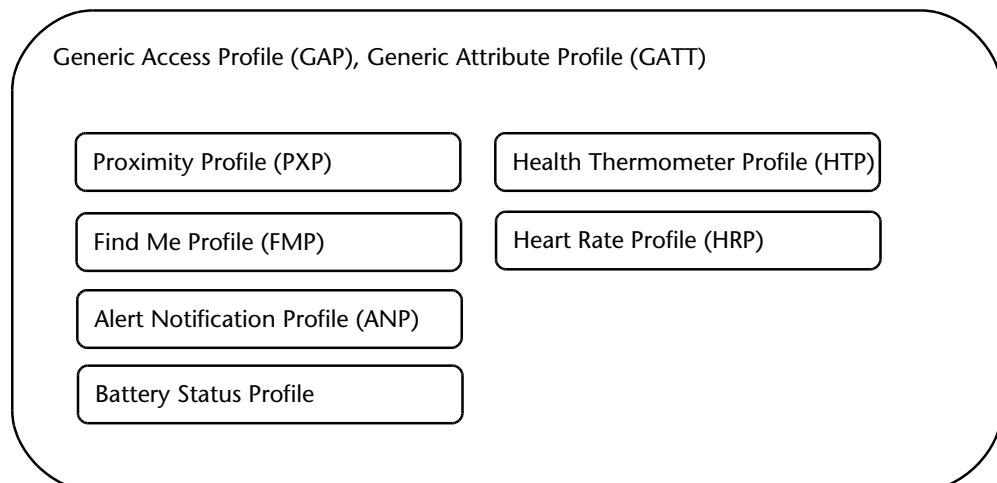


Figure 13.2 GATT-based profile dependencies.

13.1.2 GATT-Based Profile Architecture

The Bluetooth Core 4.0 specification introduced a very simple GATT based profile architecture. It is mandatory for LE devices and optional for BR/EDR devices.

In this architecture, the bulk of the profile complexity is embedded in GATT. The profiles on top are very simple and only needed to implement the functionality particular to devices of that particular category. For example, the health thermometer profile only has to implement the functionality specific to thermometer devices, like characteristics for providing the temperature. The remaining functionality like discovering the services, reading or writing attributes, or setting an indication are provided by GATT.

Another simplification introduced by LE is that profiles are defined in such a manner that the number of mandatory characteristics a device needs to support is kept to a minimum with the remaining characteristics defined as optional. So, depending on the complexity that the device can handle, it can either implement only minimum mandatory characteristics or many more optional characteristics.

For example, the device information service defines several characteristics including:

1. Manufacturer Name String.
2. Model Number String.
3. Serial Number String.
4. Hardware Revision String.
5. Firmware Revision String.
6. Software Revision String.

It is mandatory to implement any one of the characteristics and the remaining characteristics may be optional.

GATT defines the concept of a service and a characteristic. A profile can provide one or more services. Each service comprises one or more characteristics. This is shown in Figure 13.3.

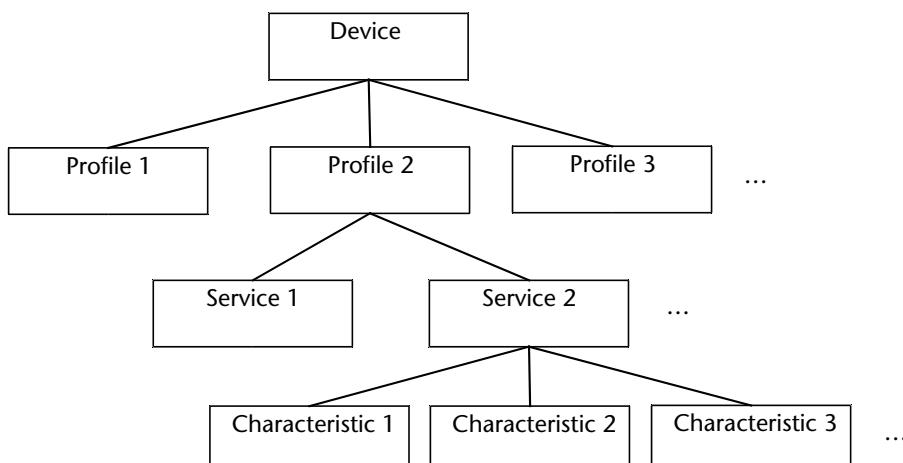


Figure 13.3 Relationship between profiles, services, and characteristics.

Grouping of attributes—An Analogy.

As an analogy, let's take the example of an organization which contains, let's say 500 people. It would be very difficult to manage the organization if all the 500 people are organized in a flat structure. So they would be grouped into various departments. There could be a marketing department, an admin department, an engineering department, and so on. The engineering department may further be organized into subdepartments like software engineering, hardware engineering, architecture, development, and testing. Organizing the people into various departments and subdepartments makes the functioning of the organization smoother, better organized, and effective.

ATT defines a flat structure of attributes and relevant operations for those attributes. This could be considered to be the 500 people in the organization.

GATT organizes those into profiles, services, and characteristics. A profile could be considered similar to a department like HR or admin. Each department could be functioning independently. Similarly profiles are independent of each other. Each profile can provide one or more services. This could be considered to be services provided by each department. HR could provide services of payroll, training, and so on.

Finally, each service could either contain subservices or contain one or more characteristics. In our analogy, subservices could be the subdepartments like software engineering, or hardware engineering. The characteristics would be the people who are providing these services of payroll, training, engineering etc.

In summary, GATT groups similar attributes into structures which are easy to manage instead of one large collection of attributes. So it groups all attributes related to temperature together in one service, time together in another service, and heart rate in a third service. A profile may contain one or more such services.

The GATT-based profile architecture is illustrated in Figure 13.4 using four profiles as examples.

The Proximity Profile (PXP) has two roles—Monitor role and Reporter role. In the reporter role, it provides three services:

1. Link Loss Service—Mandatory.
2. Immediate Alert Service—Optional.
3. Tx Power Service—Optional.

The Link Loss Service contains the Alert Level Characteristic. The Tx Power Service contains the Tx Power Level Characteristic.

The Find Me Profile (FMP) has two roles—Locator and Target. In the locator role, it does not provide any services. In the target role, it provides only one service:

1. Immediate Alert Service—Mandatory.

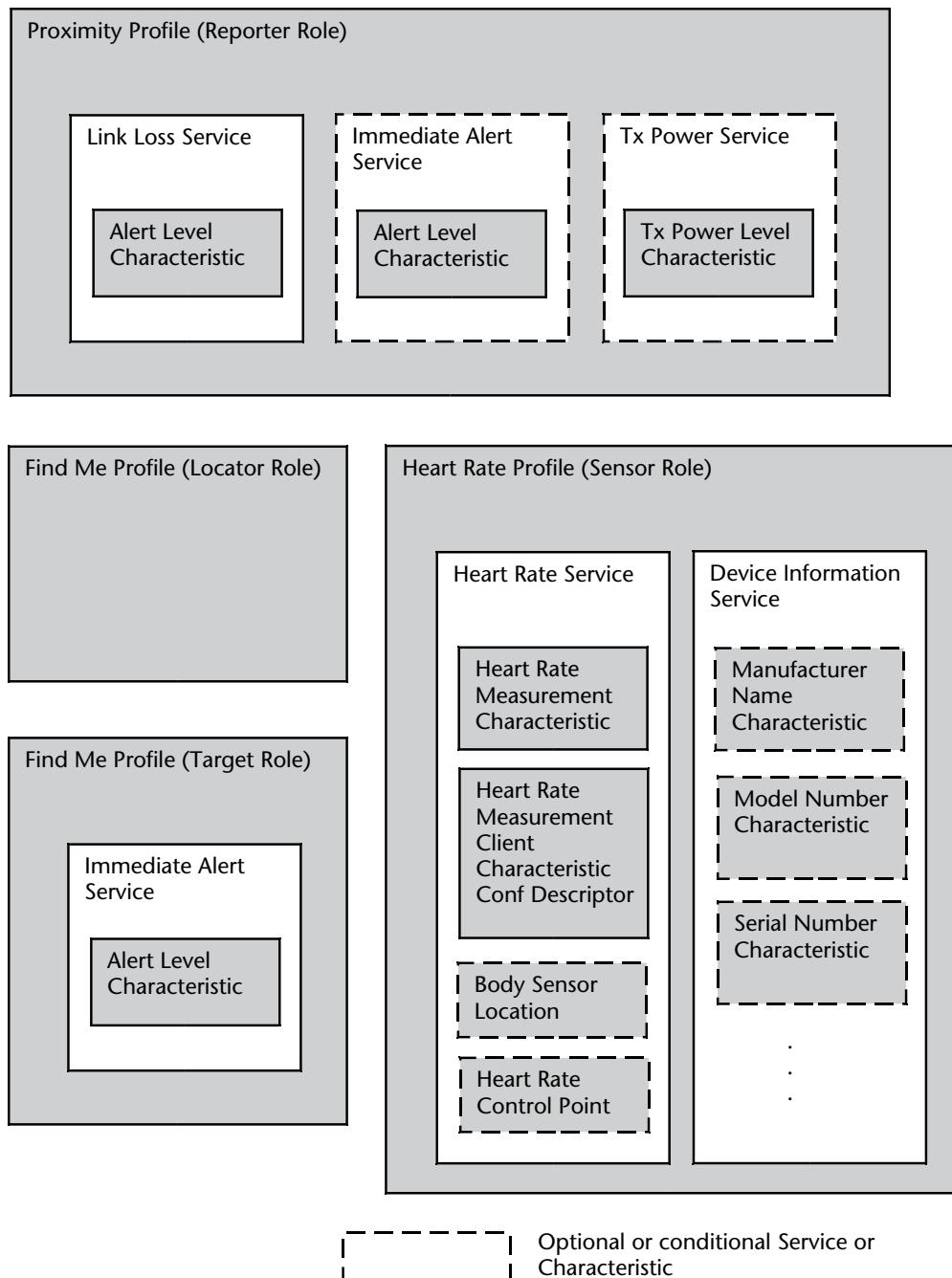


Figure 13.4 Example of profiles, services, and characteristics.

While the Immediate Alert Service is optional in PXP, it is mandatory in FMP. If a device supports both profiles, then this service can be shared with both of the profiles.

The Heart Rate Monitor profile (HRP) has two roles—Collector and Sensor. In the sensor role, it provides two services:

1. Heart Rate Service—Mandatory.
2. Device Information Service—Mandatory.

The Heart Rate Service (HRS) provides four characteristics:

1. Heart Rate Measurement— Mandatory.
2. Heart Rate Measurement Client Characteristic Configuration Descriptor—Mandatory.
3. Body Sensor Location—Conditional.
4. Heart Rate Control Point—Conditional.

13.1.2.1 Profile

A profile is composed of one or more services that are needed to fulfill a function. Some of the GATT-based profiles will be explained in detail in Chapter 15.

13.1.2.2 Service

A service can be considered to be a data structure used to describe a particular function or a feature. A service is a collection of characteristics. Besides characteristics, a service may reference other services using include definitions.

The specification defines the services separately from the profiles. So it is possible for two or more profiles to use some common services if those are needed to fulfill the particular use those profiles are supporting.

For example, the Device Information Service (DIS) shown in Figure 13.4 is a generic service that provides information about the device like the manufacturer name, serial number, model number, etc. This service can be included in the profiles which need to provide this information to the remote devices. As another example, the Immediate Alert Service (IAS) is included in both Find Me Profile and Proximity Profile. In the Find Me Profile, this service is mandatory while in the Proximity Profile this service is optional.

Every device must expose at least two services:

1. Generic Access Service.
2. Generic Attribute Service.

A service is defined by its service definition. The service definition may contain:

- References to other services using include definitions.
- Mandatory characteristics.
- Optional characteristics.

The service definition will be explained in detail later.

A service is defined in a specification which is different from the profile specifications. So there are service specifications for Immediate Alert Service, Device Information Service, Link Loss Service, etc.

There are two types of services:

1. Primary Service.
2. Secondary Service.

Primary Service

A primary service exposes the primary usable functionality of the device. The Primary services of a device are discovered using the Primary Service Discovery procedure that will be explained later in this chapter. For example, in the practical example shown in the previous chapter, the primary services discovered from the remote device included:

- Link Loss Alert.
- Immediate Alert Service.
- Tx Power Service.

Secondary Service

A secondary service is intended to be referenced from a primary service, another secondary service, or other higher layer specification. It provides auxiliary functionality for the device. It is only relevant in the context of the service or higher layer specification that references it.

Referenced Service

GATT provides a mechanism for a service to reference other services. When a service references another service, the entire referenced service becomes a part of that service. This includes any characteristics or included services as well. GATT permits the references to be nested without any restrictions on the depth of the references.

The service definition contains an include definition at the beginning of the service definition to include another service.

13.1.2.3 Characteristic

A characteristic is a value used in a service along with information about that value. A characteristic is defined by a characteristic definition. A characteristic definition contains a characteristic declaration, characteristic properties and a characteristic value. Besides this, it may optionally contain one or more characteristic descriptors that describe the value or permit configuring that value.

Some examples of characteristics are:

- Device Name Characteristic to provide the friendly name of the device to remote devices.
- Manufacturer Name, Model Number, Serial Number Characteristics to provide information about the device.
- Temperature Measurement Characteristic to provide a temperature measurement.
- Temperature Type Characteristic to describe the location on the human body where the temperature is measured.

Some examples of characteristic descriptors are:

- Characteristic Descriptor for the Temperature Measurement Characteristic to configure that characteristic to generate an indication when the temperature measurement is ready.
- Characteristic Descriptor for the Blood Pressure Measurement Characteristic to configure that characteristic to generate an indication whenever the blood pressure measurement is ready.

Figure 13.5 shows a simplified view of the Heart Rate Profile to illustrate the relationship between profile, service and characteristic.

13.2 Roles

Similar to ATT, GATT defines the following two roles:

1. *Client*: The client initiates transactions to the server and can receive responses from the server. This includes commands and requests sent to the server and responses, indications and notifications received from the server.
2. *Server*: The server receives the commands and requests from the client and sends responses, indications and notifications to the Client.

A device can act in both the roles at the same time. The various transactions between a client and the server are shown in Figure 13.6.

An example of a GATT client and server is shown in Figure 13.7. The mobile phone acts as a GATT client and the heart rate monitor acts as a GATT server. The mobile phone sends requests to the heart rate monitor to get information from it like the Model Number. The heart rate monitor sends the Model number in the response. Once the Heart Rate Monitor has collected a measurement, it may send a notification to the mobile phone of the Heart Rate Measurement Value.

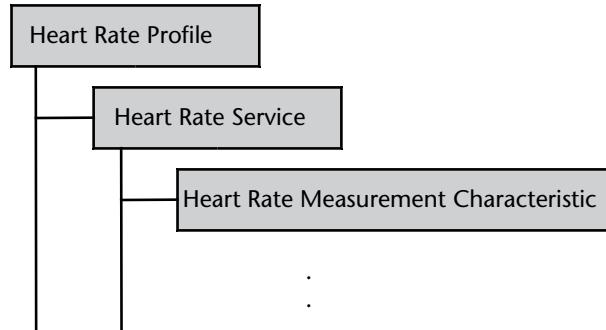


Figure 13.5 Profile, service, and characteristic for heart rate profile.

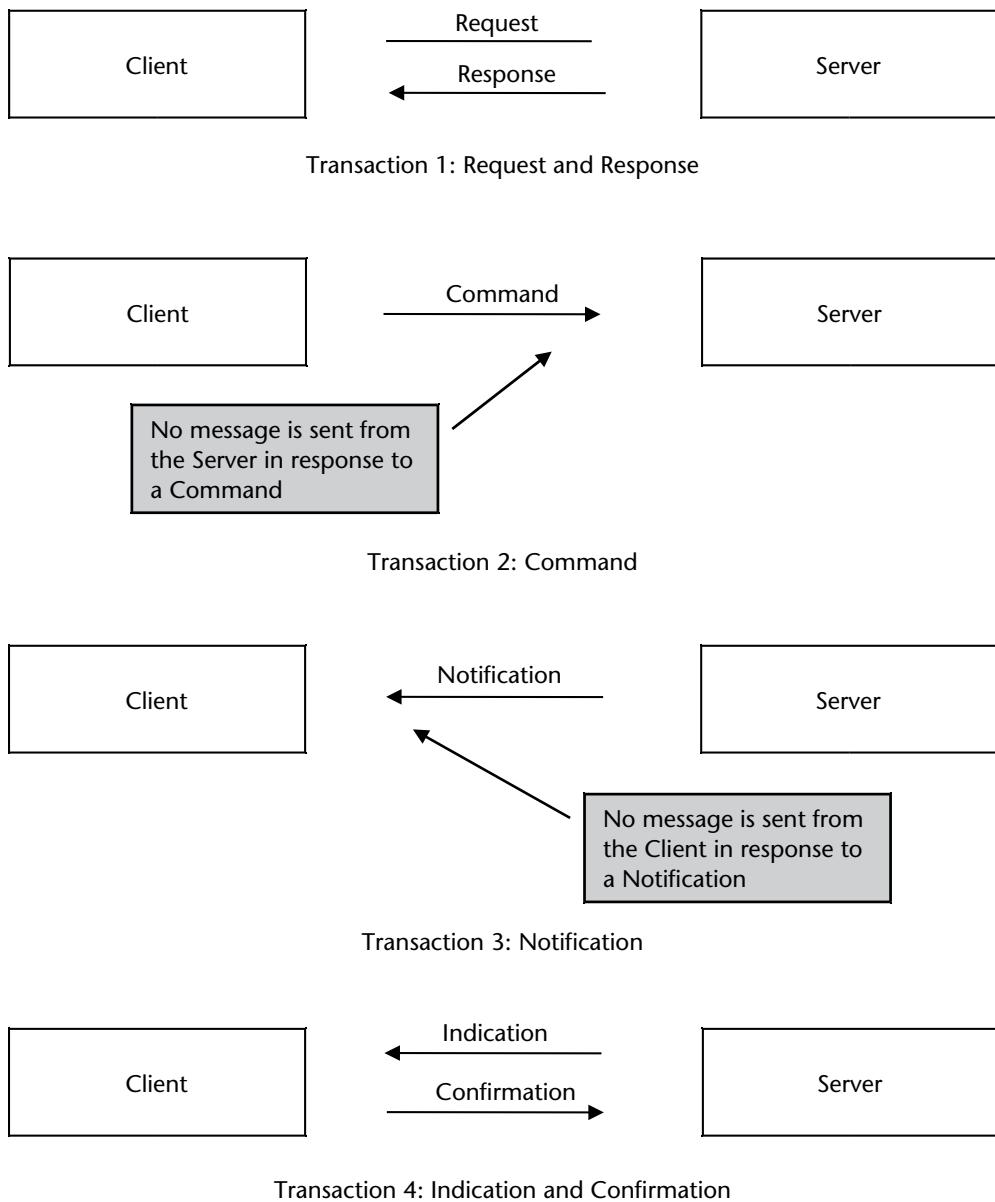


Figure 13.6 GATT Client and Server Transactions.

13.3 Attributes

The structure of an attribute is defined by the ATT protocol as explained in Chapter 12. It is shown in Figure 13.8.

The Attribute Handle is used to uniquely refer to an attribute. It ranges from 0x0001 to 0xFFFF. The attributes are arranged in increasing order of attribute handles. It is not mandatory for the attribute handles to be contiguous. There may be gaps in the numbering of the attribute handles.

The Attribute Type describes the type of an attribute. It is in the form of a 16-bit UUID or 128-bit UUID.

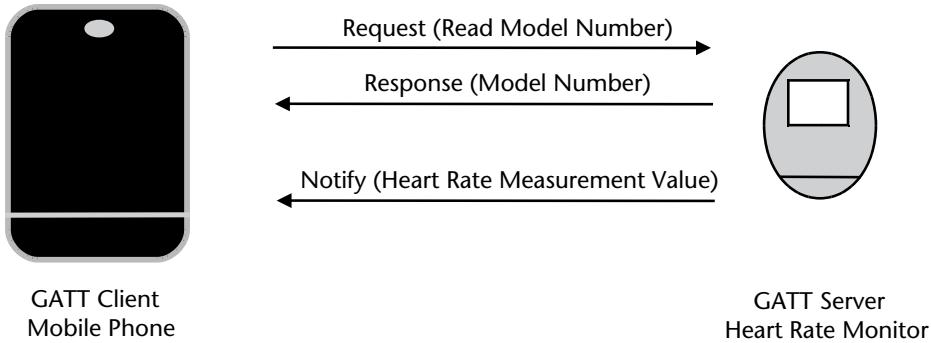


Figure 13.7 GATT Client and Server Example.

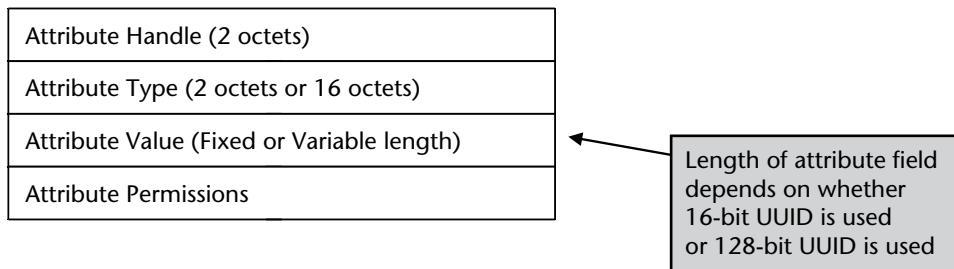


Figure 13.8 Attribute structure.

The Attribute Permissions are defined by either the GATT profile or a higher layer profile or application. This defines whether an attribute can be read or written or whether it requires authentication or authorization. As shown in Figure 13.10, the attributes containing list of services and characteristics supported by a device have the following permissions:

- Read-Only.
- No Authentication required.
- No Authorization required.

The permissions of the Characteristic Value Declaration or the Characteristic Descriptor Declaration are not set by the GATT profile. Rather these are set by the higher layer profile or the implementation. For example, the Health Thermometer Service defines the following permissions for the characteristics:

- Temperature Measurement—No Permissions required.
- Measurement Interval—No Permissions required for reading, Authentication required for writing.

This means that any device can read the measurement interval but the device needs to authenticate itself if it wants to write this characteristic.

13.3.1 Attribute Caching

In order to save power, GATT allows reducing the number of transactions between the client and server by supporting caching of attributes. The clients can discover the set of attributes on the server and store this information. The clients can use these values during the lifetime of a connection and also across connections. This means that once a client disconnects, it can again use the information it had discovered about the attributes when it connects again to the server. This saves both a significant number of packet exchanges and a lot of time that would otherwise have been spent in discovering the attributes again.

The devices in which the services can change over a period of time define a Service Changed Characteristic. This characteristic supports indication. If any of the services on the device changes (added, modified, or removed), the server sends an indication. Three scenarios are possible:

1. One or more clients are connected to the server: The indication is sent by the server to all the connected client(s).
2. One or more clients are bonded but not connected: The indication is sent by the server whenever the client(s) connect next time.
3. The clients are not bonded or connected: The attribute cache is considered to be valid only during the connection. So the clients should do an attribute discovery on connection if the server supports Service Changed Characteristic.

This is shown in Figure 13.9.

The Service Changed Indication contains the range of Attribute Handles that may have changed and for which the attribute cache is no longer valid. The Service Changed Characteristic is explained later in this chapter.

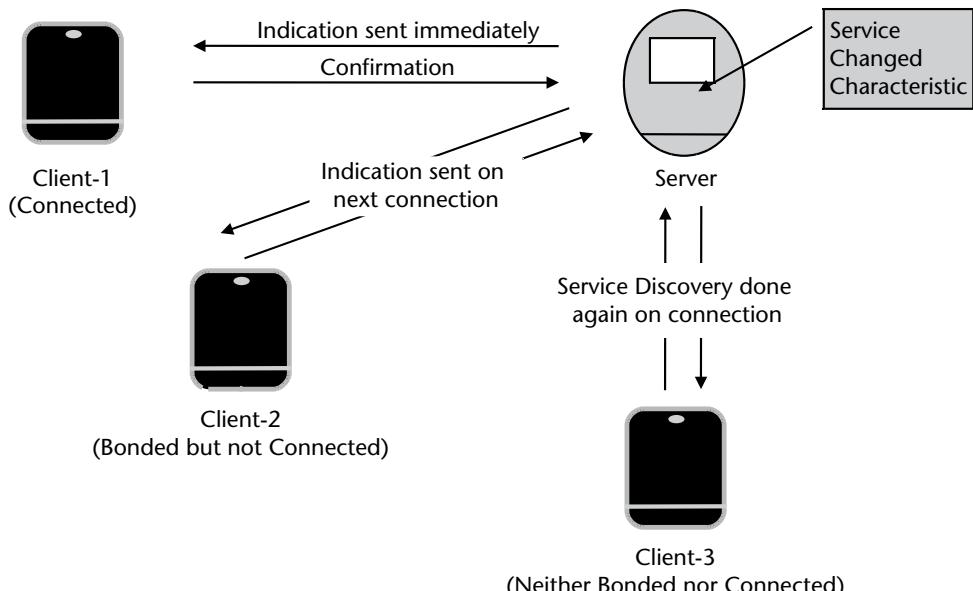


Figure 13.9 Attribute caching.

13.3.2 Attribute Grouping

ATT protocol provides support for attributes and procedures for accessing those services. GATT defines a structure using the attributes and groups the attributes into three types:

- <<Primary Service>>
- <<Secondary Service>>
- <<Characteristic>>

Depending on the implementation, a device may implement the groups that are required for that particular implementation. GATT provides procedures for accessing the services and characteristics.

13.3.3 Notification and Indication

Notifications and Indications are another power saving mechanism introduced by the LE specification. The power consumption of devices which use Notification and Indication procedures is generally much lower than the scenario where the same functionality is implemented using Read procedures.

As an analogy, consider the interrupt and polling mechanisms in the microprocessor world. If a peripheral like keyboard is connected to a microprocessor, the microprocessor can get the keystrokes from the keyboard using two mechanisms:

- Polling: The microprocessor reads the keyboard registers periodically to see if a key has been pressed.
- Interrupt: The keyboard sends an interrupt whenever the user presses a key.

If the microprocessor uses polling mechanism, it may need to read the keyboard registers as frequently as once in 10 milliseconds in order to ensure that keystrokes are not lost. If the user is away for several hours, this would mean that microprocessor will continue generating read transactions without fetching any meaningful data leading to unnecessary power consumption.

If the microprocessor uses the interrupt mechanism, it will be informed by the keyboard whenever a key is pressed and the microprocessor can do the appropriate processing only when the key is pressed. This would lead to significant power savings.

The Notification and Indication mechanisms are comparable to interrupts. Whenever the LE server has to inform the LE client(s) that it has some data to send, it can send a notification or indication and the client can do the appropriate processing only when the key is pressed.

Some examples are:

- The Health Thermometer Profile uses Indication for the Temperature Measurement characteristic. Whenever it has a temperature measurement to send to the clients, it sends an Indication.

- The Heart Rate Monitor Profile uses Notification for Heart Rate Measurement characteristic. Whenever it has a heart rate measurement to send to the clients, it sends a Notification.

The main difference between Notification and Indication is that with Notification there is no acknowledgement sent by the client to the server while in the case of Indication the client sends a Confirmation to the server so that the server is informed that the client has received the Indication.

13.4 Service Definition

As explained earlier, a service is defined by its service definition. A service definition contains three parts in the following order:

1. Service Declaration (Mandatory).
2. Include Definitions (Optional: Zero or More).
3. Characteristic Definitions (Optional: Zero or More).

The structure of the service definition is shown in Figure 13.10 and explained in the following sections.

13.4.1 Service Declaration

The definition of a service starts with a service declaration. The service declaration is the mandatory part of a service definition. This is an attribute with:

- Attribute Type = UUID of <>Primary Service>> or <>Secondary Service>>
- Attribute Value = UUID of service. This is called Service UUID.
- Attribute Permissions = Read Only, No Authentication, No Authorization.

13.4.2 Include Definition

An include definition is used to reference other services. A service definition may contain zero or more include definitions. Each include definition contains only one include declaration. The include declaration is an attribute with:

- Attribute Type = UUID of <>Include>>
- Attribute Value = Attribute handle of included service, end group handle, and service UUID.
- Attribute Permissions = Read Only, No Authentication, No Authorization.

13.4.3 Characteristic Definition

A characteristic definition is used to define a characteristic within a service. It contains a value as well as information about the value. A service definition may contain

Service Definition

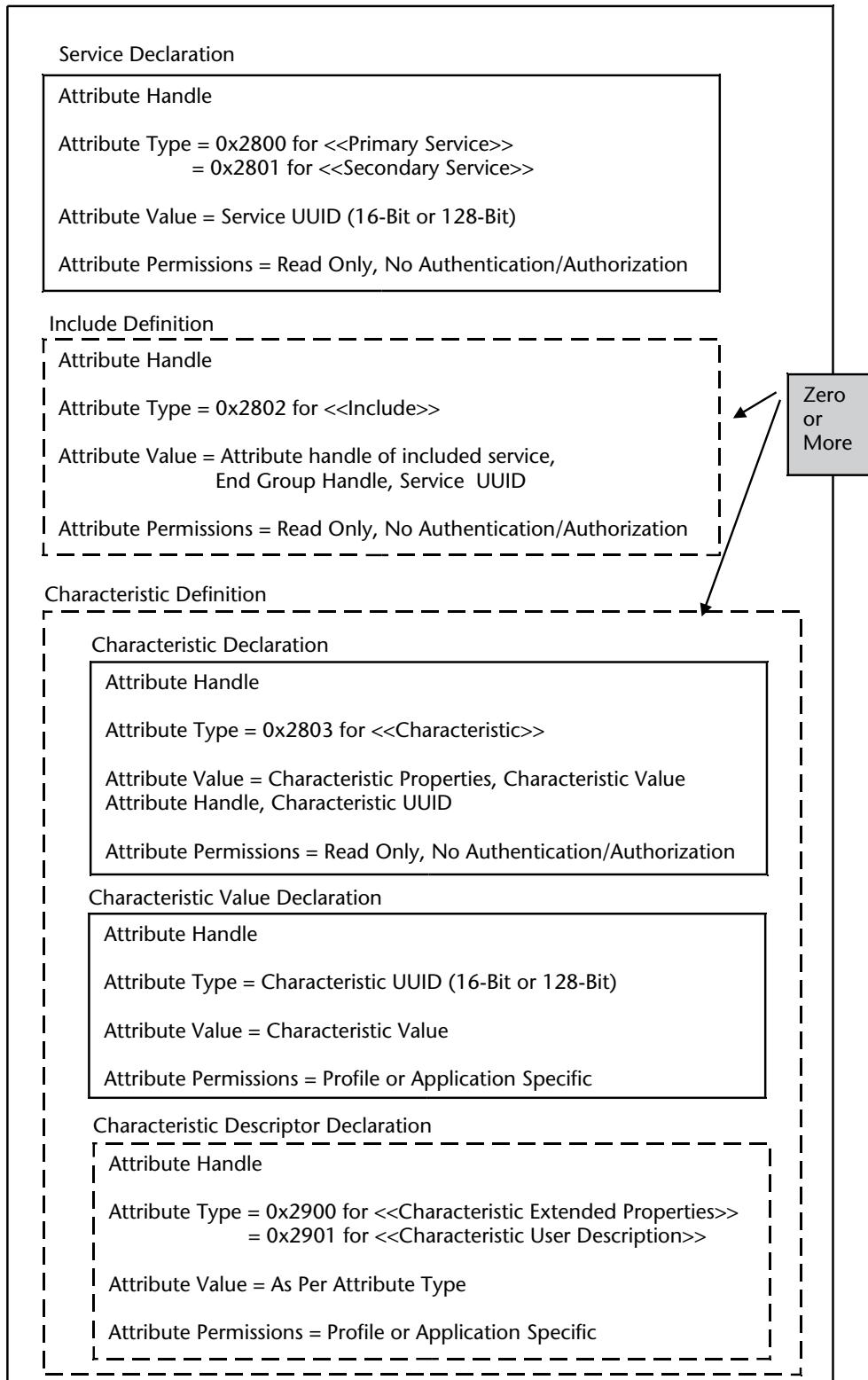


Figure 13.10 Service definition.

zero or more characteristic definition depending on the number of characteristics it supports. Each characteristic definition includes three parts in the following order:

1. Characteristic Declaration (Mandatory): Information about the characteristic.
2. Characteristic Value Declaration (Mandatory): Value of the characteristic.
3. Characteristic Descriptor Declaration (Optional: Zero or More): Additional information.

A service may contain both mandatory and optional characteristic definitions. The mandatory definitions are located before the optional ones.

13.4.3.1 Characteristic Declaration

A characteristic declaration is used to declare the information about a characteristic like the characteristic's UUID, properties, etc. It is mandatory to include it within a characteristic definition. It is an attribute with:

- Attribute Type = UUID for <<Characteristic>>
- Attribute Value = The Attribute Value containing:
 - a. *Characteristic Properties*: This is a bit field that determines how the Characteristic Value can be used. For example, whether the value can be broadcast, read, written, notified or indicated. For example, a Read Only characteristic has this value set to 0x02 and a broadcast characteristic has this value set to 0x01.
 - b. *Characteristic Value Attribute Handle*: This is the Attribute Handle of the Attribute that contains the Characteristic Value. Characteristic Value is explained in the next section.
 - c. *Characteristic UUID*: This is a 16-bit or 128-bit to describe the type of Characteristic Value.
- Attribute Permissions = Read Only, No Authentication, No Authorization.

The significance of various bits in the Characteristic Properties bit field is shown in Table 13.1. For example if the value of properties is 0x08, then it would mean that the Characteristic Value can be read and written.

13.4.3.2 Characteristic Value Declaration

A Characteristic Value Declaration contains the value of the characteristic. It is mandatory to include it within a characteristic definition. This is an attribute with:

- Attribute Type = Same as the UUID provided in Characteristic Declaration.
- Attribute Value = Characteristic Value.
- Attribute Permissions = Higher layer profile or implementation specific.

It may be noted that for the previous attributes, the permissions were defined by GATT to be Read Only, No Authentication and No Authorization while for

Table 13.1 Bit-Fields for Characteristic Properties

<i>Properties</i>	<i>Value</i>	<i>Description</i>
Broadcast	0x01	Broadcast of Characteristic value permitted.
Read	0x02	Read permitted
Write Without Response	0x04	Write Without Response permitted
Write	0x08	Write permitted
Notify	0x10	Notifications permitted
Indicate	0x20	Indications permitted.
Authenticated Signed Writes	0x40	Signed write permitted
Extended Properties	0x80	Additional properties are defined in the Characteristics Extended Properties Descriptor.

the Characteristic Value the permissions are defined by the higher layer profile or implementation. This is because GATT defines that any client can read the list of services and characteristics supported by a device, but, whether a client can read or write the value of a characteristic is left to the discretion of the higher layer profile or implementation.

13.4.3.3 Characteristic Descriptor Declaration

The Characteristic Descriptors are used to provide additional information about the Characteristic Value. These are optional and a Characteristic Definition may contain more than one of these. A characteristic descriptor is an attribute with:

- Attribute Type = UUID for the additional information provided by this Characteristic Descriptor.
- Attribute Value = Value of the additional information provided by this Characteristic Descriptor.
- Attribute Permissions = Either defined by GATT or Higher layer profile or implementation specific.

For example a Characteristic Descriptor Declaration may be used to provide a user friendly string about the description of a Characteristic Value. If a Characteristic Value provides the temperature then this descriptor could provide a string saying “Room Temperature” or “Celcius”. These strings could be used for preparing user interfaces on the client side.

This descriptor is also used to specify if the value can be indicated, notified, or broadcast. For example, the Temperature Measurement Characteristic in the Health Thermometer Service has the properties of “Indicate”. This characteristic also has an associated Client Characteristic Configuration Descriptor which can be configured for indication. So once a temperature measurement is available and this descriptor is configured for indication, an indication is sent to the peer device.

The Characteristic Descriptor could also be used to specify the format of the Characteristic Value. For example, whether the value is 1-bit, 8-bit, 16-bit, signed or unsigned, a UTF-8 string or UTF-16 string, or whether it is a floating point value.

13.5 Configured Broadcast

A client can configure the server to broadcast certain characteristic values in advertising data. This is done by the client setting the broadcast configuration bit in the client characteristic configuration.

13.6 GATT Features

The GATT profile defines eleven features. Each feature is mapped into procedures and subprocedures which use the procedures defined by the ATT protocol.

A summary of the features and the corresponding procedures and sub-procedures is shown in Table 13.2. These are described in detail in the following sections.

13.6.1 Server Configuration

The server configuration feature is used to configure the parameters of the ATT protocol. It has only one subprocedure related to configuring the ATT_MTU size.

13.6.1.1 Exchange MTU

The exchange MTU subprocedure is initiated only once during a connection by the client to set the ATT_MTU to be used for the connection.

Table 13.2 GATT Features and Procedures

S. No	Feature	Subprocedures
1	Server Configuration	Exchange MTU
2	Primary Service Discovery	Discover All Primary Services Discover Primary Services By Service UUID
3	Relationship Discovery	Find Included Services
4	Characteristic Discovery	Discover All Characteristics of a Service Discovery Characteristic by UUID
5	Characteristic Descriptor Discovery	Discover All Characteristic Descriptors
6	Characteristic Value Read	Read Characteristic Value Read Using Characteristic UUID Read Long Characteristic Values Read Multiple Characteristic Values
7	Characteristic Value Write	Write Without Response Signed Write Without Response Write Characteristic Value Write Long Characteristic Values Characteristic Values Reliable Writes
8	Characteristic Value Notification	Notifications
9	Characteristic Value Indication	Indications
10	Characteristic Descriptor Value Read	Read Characteristic Descriptors Read Long Characteristic Descriptors
11	Characteristic Descriptor Value Write	Write Characteristic Descriptors Write Long Characteristic Descriptors

The default value of ATT_MTU is 23 octets for LE. The exchange MTU request is used by the client if it wants to use an ATT_MTU bigger than this value. A bigger value will finally be used after the request and response messages are exchanged and if both the client and the server support that bigger value. This is shown in Figure 13.11.

This is especially useful where a client may have bigger receive buffers compared to the server. For example, if the client is a mobile phone and the server is a thermometer. The mobile phone may send an MTU size of 300 octets in the Exchange MTU Request. If the thermometer can support an Rx MTU of only 40 bytes, it will send an Exchange MTU Response with 40 bytes. After these messages are exchanged, both the thermometer and mobile phone will use an MTU of 40 bytes. If the thermometer sent an Error Response, then both will use the default MTU of 23 bytes.

13.6.2 Primary Service Discovery

The Primary Service Discovery procedure is used by the client to discover the primary services on the server. It can either use the Discover All Primary Services sub-procedure or Discover Primary Services by Service UUID sub-procedure.

13.6.2.1 Discover All Primary Services

This subprocedure is used by the client to discover all the primary services on the server. It uses the ATT Read By Group Type Request with the following parameters:

- Attribute Type: UUID for <>Primary Service>>.
- Starting Handle: 0x0001.
- Ending Handle: 0xFFFF.

The server either sends the ATT Read By Group Type Response or an Error Response. If the server sends the Read By Group Type Response, then it contains the following parameters:

- Length: Size of each attribute data.

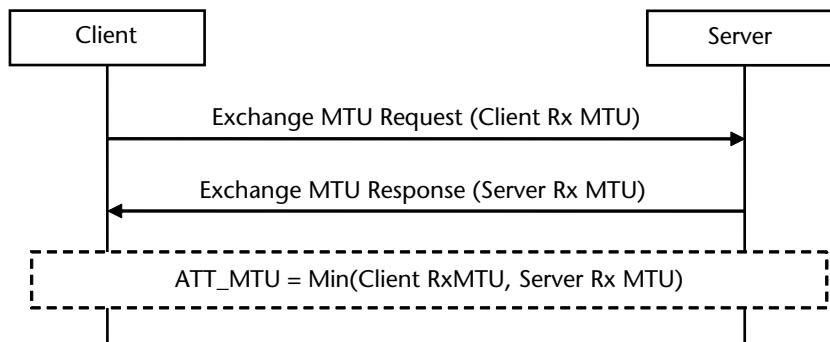


Figure 13.11 Exchange MTU.

- Attribute Data List: This contains the following:
 - Attribute Handle: Handle of the service declaration.
 - End Group Handle: Handle of last attribute within the service definition.
 - Attribute Value: Service UUID of a service supported by the server.

It is possible that not all attributes are returned in one response. In that case, the End Group Handle in the response would be less than the Ending Handle given in the request. The client can then send another Read By Group Type Request by modifying the Starting Handle to the received End Group Handle + 1. This subprocedure is completed when the server returns an Error Response with the error code set to Attribute Not Found.

This is shown in Figure 13.12. The client sends the Read By Group Type Request message to the server by setting the Starting Handle to 0x0001 and Ending Handle to 0xFFFF to get the attribute handles within the entire possible range

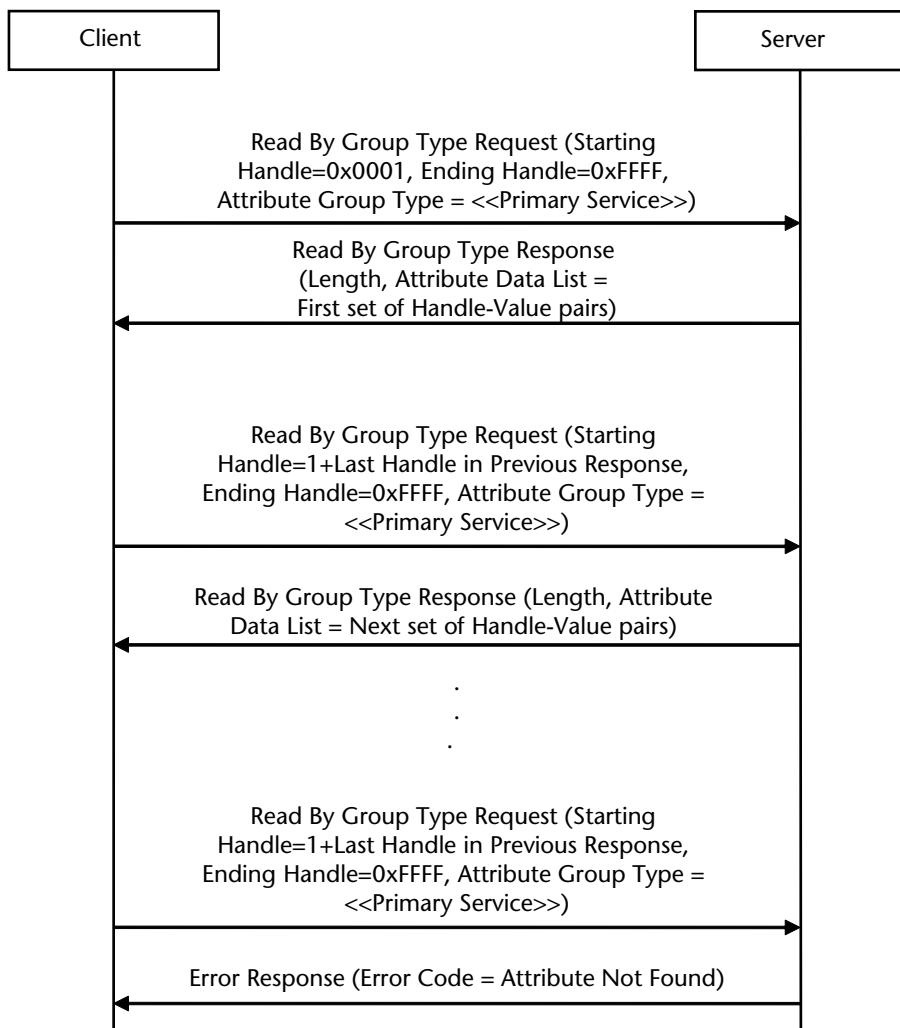


Figure 13.12 Discover All Primary Services.

of attribute handles. The server responds with the first set of Handle-Value pairs depending on the number of pairs that can be accommodated within one Read By Group Type Response.

The client then sends the next Read By Group Type Request by adjusting the Starting Handle to retrieve the next set of Handle-Value Pairs. The server responds with the next set of Handle Value pairs till no more remain as per the Starting Handle specified by the client. At that time the server returns an Error Response. This indicates to the client that all the handles have been retrieved.

Figure 13.13 shows a message sequence chart of the air logs captured when doing a Discover All Primary Services procedure with an LE server. A few things may be noted:

- In the first Read By Group Type Request, the client tries to read all handles from 1 to 65535.
- The server responds with three handle value pairs:
 - Handles 1 to 7 (Service 1).
 - Handles 16 to 19 (Service 2).
 - Handles 80 to 82 (Service 3) (This got truncated in the screen shot).
- The client increments the Last Handle received in previous response by 1 and does a second Read By Group Type Request, this time reading from 83 to 65535
- The server responds with two handle value pairs:

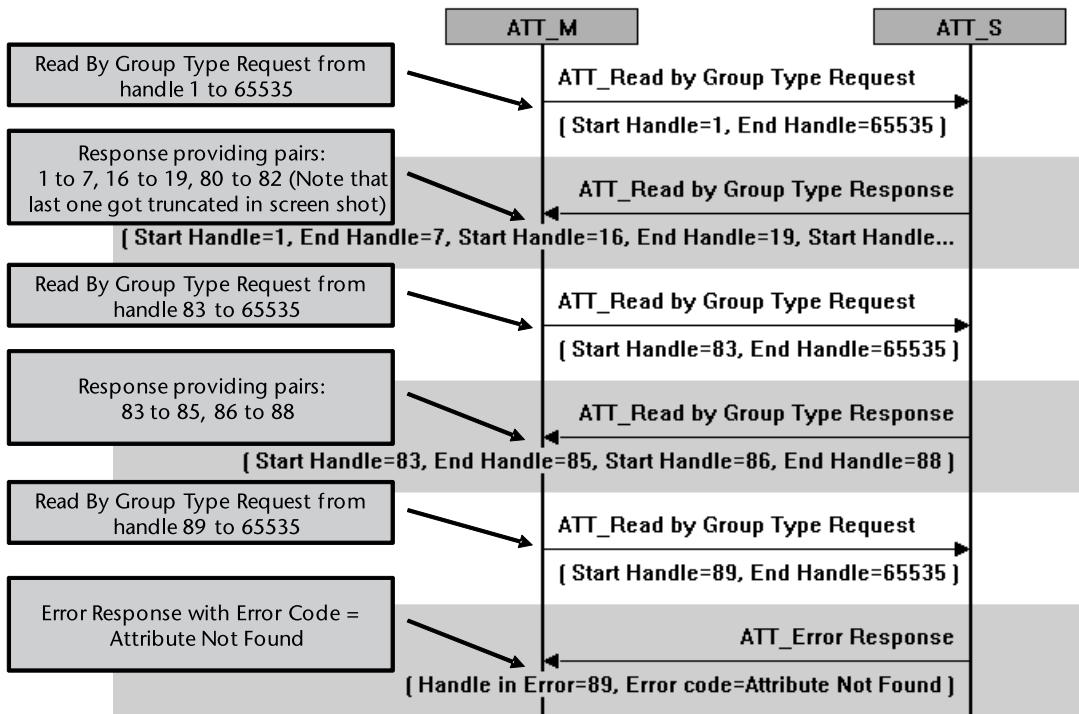


Figure 13.13 MSC generated from air capture of Discover All Primary Services.

- Handles 83 to 85 (Service 4).
- Handles 86 to 88. (Service 5).
- The client increments the Last Handle received in previous response by 1 and does a third Read By Group Type Request, this time reading from 89 to 65535.
- The server responds with an Error Response with the Error Code = Attribute Not Found.

This indicates to the client that it has received all handle value pairs. The Discover All Primary Services procedure is considered completed by the client. It was able to retrieve a total of five services from the server.

13.6.2.2 Discover Primary Service By Service UUID

This subprocedure is used by the client to discover a primary service on the server if the Service UUID is known. As shown in Figure 13.10, the Service UUID is a part of the service declaration.

It uses the ATT Find By Type Value Request with the following parameters:

- Attribute Type: UUID for <>Primary Service>>.
- Attribute Value: Service UUID of the Primary service to search (16-bit or 128-bit).
- Starting Handle: 0x0001.
- Ending Handle: 0xFFFF.

The server either sends the ATT Find By Type Value Response or an Error Response.

If the server sends the Find By Type Value Response, then it contains the following parameters:

- List of Attribute Handle Ranges.
 - Starting Handle of service definition.
 - Ending Handle of the service definition.

It is possible that not all attributes are returned in one response. In that case, the End Group Handle in the response would be less than the Ending Handle given in the request. The client can then send another Find By Type Value Request by modifying the Starting Handle to the received Ending Handle + 1. This subprocedure is completed when the server returns an Error Response with the error code set to Attribute Not Found.

This is shown in Figure 13.14. The client sends the Find By Type Value Request message to the server by setting the Starting Handle to 0x0001, Ending Handle to 0xFFFF, and Attribute Value to the Service UUID to be searched. The server responds with the first set of Handle Information List containing the Starting Handle and Ending Handle of a particular service definition.

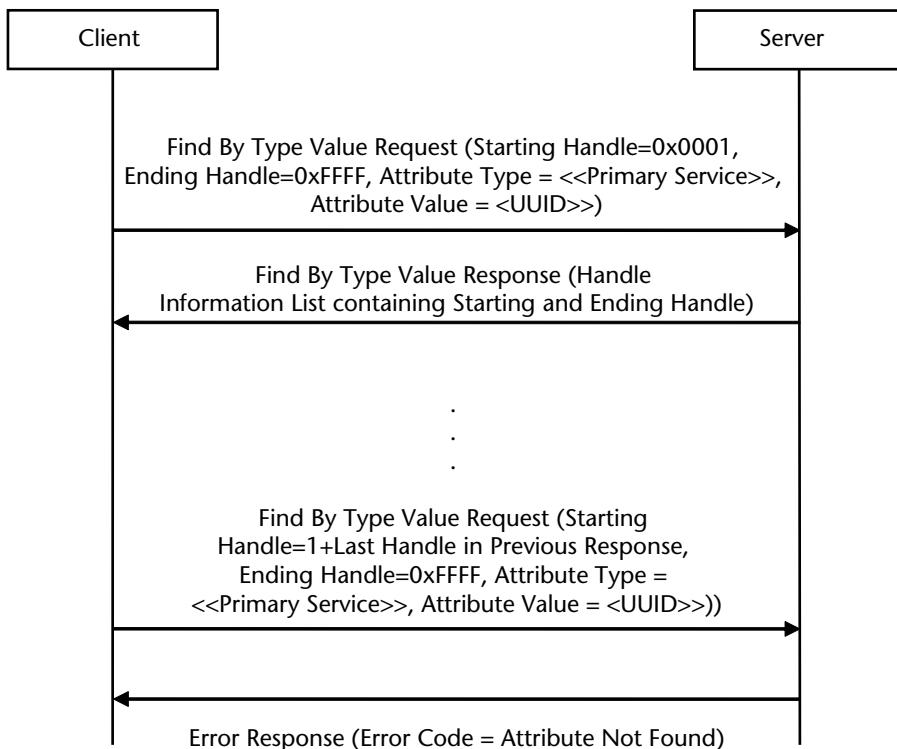


Figure 13.14 Discover Primary Service by Service UUID.

The client then sends the next Find By Type Value Request by adjusting the Starting Handle to retrieve the next set of Handle Information List. The server responds with the next set of Handle Information List until none remain per the Starting Handle specified by the client. At that time the server returns an Error Response. This indicates to the client that all the handles have been retrieved.

13.6.3 Relationship Discovery

The Relationship Discovery procedure is used by the client to discover service relationships of the services. This category contains one subprocedure: Find Included Services.

13.6.3.1 Find Included Services

This subprocedure is used by the client to discover the include service declarations within a service definition on the server. As shown in Figure 13.10, a service definition may contain zero or more include service declarations to specify which other services are included by the service.

It uses the ATT Read By Type Request with the following parameters:

- Attribute Type: UUID for <<Include>>.
 - Starting Handle: Starting Handle of the specified service.

- Ending Handle: Ending Handle of the specified service.

The Starting Handle and the Ending Handle specify the values returned in the Primary Service Discovery Procedure. The server either sends the ATT Read By Type Response or an Error Response.

If the server sends the Read By Type Response, then it contains the following parameters:

- Length: Size of the handle-value pairs.
- Attribute Data List: Handle-value pairs containing Attribute Handle and Attribute Value. The Attribute Value contains:
 - Attribute Handle: Handle of the included service declaration.
 - End Group Handle: Handle of last attribute within the included service declaration.
 - UUID: Service UUID.
 - If the Service UUID is 16-bit UUID, then it is also returned in the response.
 - If the UUID is 128-bit, then the ATT Read Request is used later on with the Attribute Handle parameter to retrieve the 128-bit UUID.

Not all handle-value pairs are necessarily returned in one response. In that case, the Attribute Handle in the response would be less than the Ending Handle given in the request. The client can then send another Read By Type Request by modifying the Starting Handle to the last received Attribute Handle + 1. This sub-procedure is completed when the server returns an Error Response with the error code set to Attribute Not Found.

This is shown in Figure 13.15. The client sends the Read By Type Request message to the server by setting the Starting Handle and Ending Handle to the attribute handles that were received in the primary service discovery procedure and the Type set to <>Include<>. The server responds with the first set of Handle-Value pairs of the included services.

The client then sends the next Read By Type Request by adjusting the Starting Handle to retrieve the next set of Handle-Value Pairs. The server responds with the next set of Handle Value pairs until none remain per the Starting Handle specified by the client. At that time the server returns an Error Response. This indicates to the client that all handles for the included services have been retrieved.

Figure 13.16 extends the sample air logs that were shown in Figure 13.13. During Discover All Primary Services, the client found five services on the server. This example shows that the client tries to do a Find Included Services for each of these five services using the Read By Type Request. Each of the requests returns an Error Response. This means that there are no included services on the server.

13.6.4 Characteristic Discovery

The Characteristic Discovery procedure is used by the client to discover the characteristic definitions included in a service on the server. As shown in Figure 13.10, a service may contain zero or more characteristic definitions. This category contains

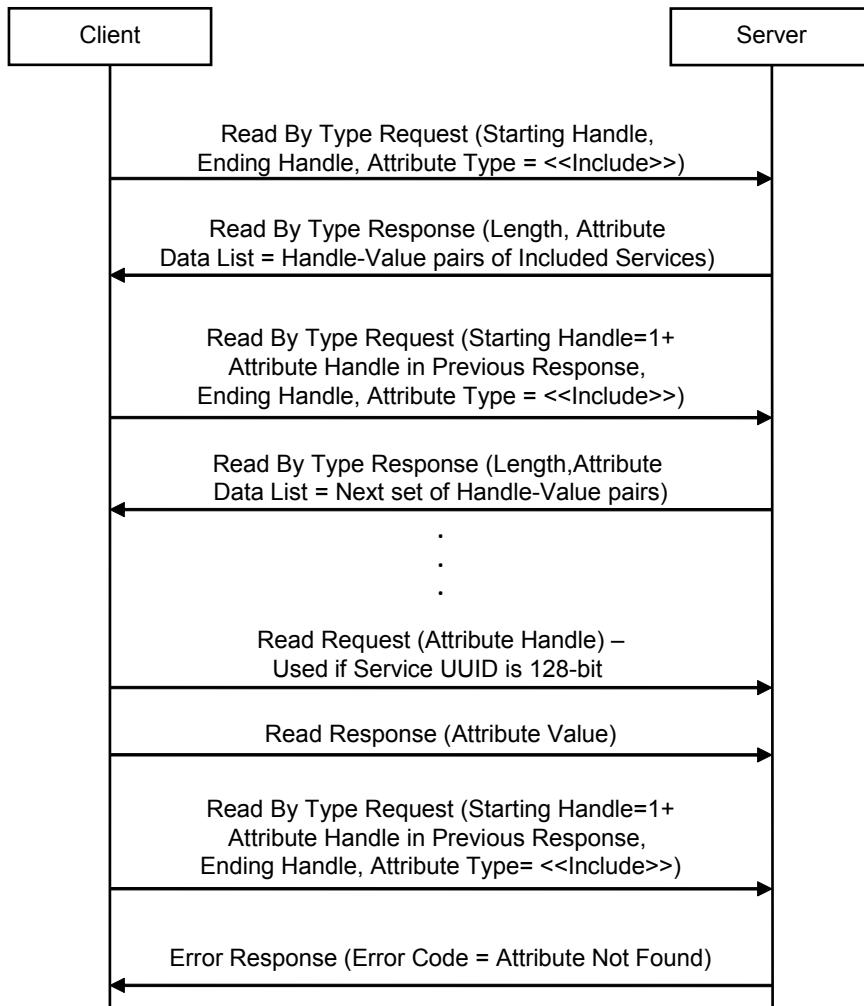


Figure 13.15 Find Included Services.

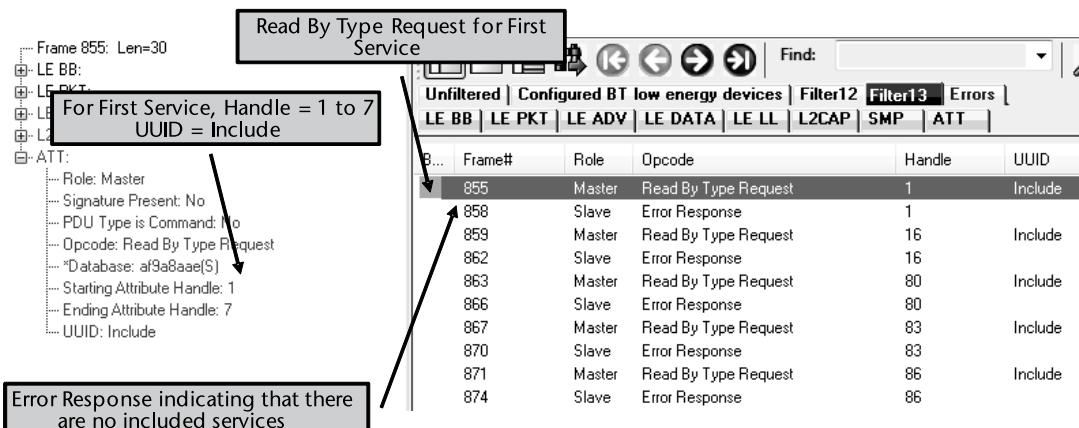


Figure 13.16 Example air log of Find Included Services.

two subprocedures: Discover All Characteristics of a Service and Discover Characteristics by UUID.

13.6.4.1 Discover All Characteristics of a Service

This subprocedure is used by the client to discover all characteristic declarations within a service definition. This is quite similar to the Find Included Services subprocedure. The main difference is that in this case the Attribute Type parameter is set to <>Characteristic>> while in the Find Included Services subprocedure it was set to <>Include>>.

This subprocedure uses the ATT Read By Type Request with the following parameters:

- Attribute Type: UUID for <>Characteristic>>.
- Starting Handle: Starting Handle of the specified service.
- Ending Handle: Ending Handle of the specified service.

The Starting Handle and the Ending Handle specify the values returned in the Primary Service Discovery Procedure. The server either sends the ATT Read By Type Response or an Error Response.

If the server sends the Read By Type Response, then it contains the following parameters:

- Length: Size of the handle-value pairs.
- Attribute Data List: handle-value pairs containing Attribute Handle and Attribute Value.
 - Attribute Handle: Handle of the characteristic declaration.
 - Attribute Value: The Attribute Value contains Characteristic Properties, Characteristic Value Handle, Characteristic UUID.

It is possible that not all handle-value pairs are returned in one response. In that case, the Attribute Handle in the response would be less than the Ending Handle given in the request. The client can then send another Read By Type Request by modifying the Starting Handle to the last received Attribute Handle + 1. This subprocedure is completed when the server returns an Error Response with the error code set to Attribute Not Found.

This is shown in Figure 13.17. The client sends the Read By Type Request message to the server by setting the Starting Handle and Ending Handle to the attribute handles received in the primary service discovery procedure and the Type set to <>Characteristic>>. The server responds with the first set of Handle-Value pairs of the characteristic declarations in the service definition.

The client then sends the next Read By Type Request by adjusting the Starting Handle to retrieve the next set of Handle-Value Pairs. The server responds with the next set of Handle Value pairs until no more remain as per the Starting Handle specified by the client. At that time the server returns an Error Response. This in-

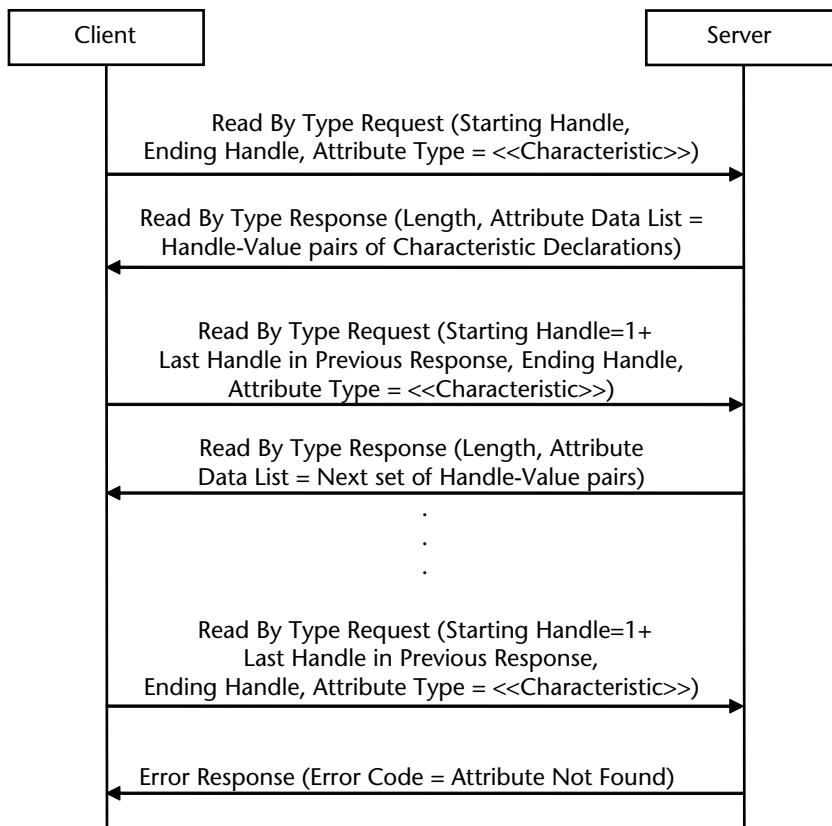


Figure 13.17 Discover All Characteristics of a service.

dicates to the client that all handles for the characteristic declarations have been retrieved.

Figure 13.18 extends the sample air logs that were shown in Figure 13.13 and Figure 13.16. During Discover All Primary Services, the client found five services on the server. The first service had the starting handle as 1 and the ending handle as 7. This example shows that the client tries to do a Discover All Characteristics of a Service procedure for the first service.

The right side of the figure shows the following:

- Frame #875: First Read By Type Request by the client with UUID = Characteristic, Handle = 1 to 7.
- Frame #878: Response by the server with the first set of characteristics (Handles 2 and 4).
- Frame #879: Second Read By Type Request by the client with UUID = Characteristic, Handle = 5 to 7.
- Frame #882: Read By Type Response with the second set of characteristics (Handle 6).
- Frame #883: Third Read By Type Request by the client with UUID = Characteristic, Handle = 7.

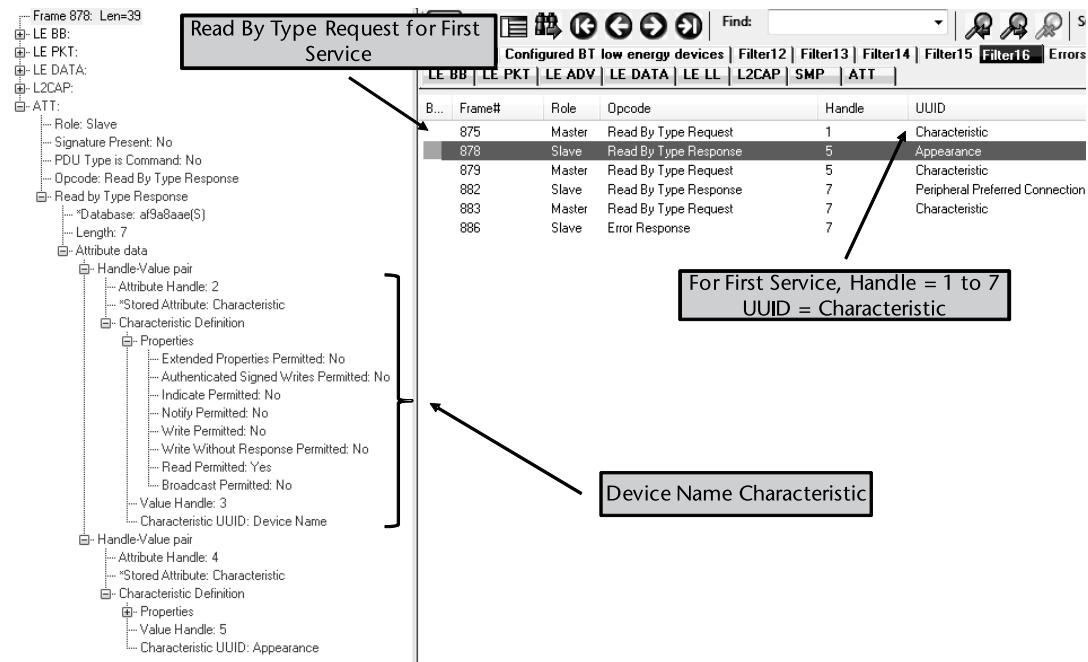


Figure 13.18 Example air log of Discover All Characteristics of a service.

- Frame #886: Error Response by the server to indicate that there are no more characteristics in this service.

The left side of the figure shows the characteristics received in the first response from the server. The following may be observed:

- The first characteristic is Device Name with Attribute Handle 2.
- The Attribute value is stored at Attribute Handle 3 (Indicated by Value Handle).
- Read operation is permitted on the characteristic. All other operations are not permitted.

13.6.4.2 Discover Characteristics by UUID

This subprocedure is used by the client to discover the characteristic declarations within a service definition when the characteristic UUID is known. This subprocedure is quite similar to the Discover All Characteristics of a Service subprocedure. In this case, once the characteristics are received from the server, they are checked to see if the UUID matches the UUID that was requested. If it matches, then the characteristic is considered to be found. In both the cases, whether the characteristic is found or not, the procedure continues to search the remaining characteristics until all characteristics have been searched.

It uses the ATT Read By Type Request with the following parameters:

- Attribute Type: UUID for <<Characteristic>>.

- Starting Handle: Starting Handle of the specified service.
- Ending Handle: Ending Handle of the specified service.

The Starting Handle and the Ending Handle specify the values that were returned in the Primary Service Discovery Procedure. The server either sends the ATT Read By Type Response or an Error Response.

If the server sends the Read By Type Response, then it contains the following parameters:

- Length: Size of the handle-value pairs.
- Attribute Data List: handle-value pairs containing Attribute Handle and Attribute Value.
 - Attribute Handle: Handle of the characteristic declaration.
 - Attribute Value: The Attribute Value contains Characteristic Properties, Characteristic Value Handle, Characteristic UUID.

The Attribute Value in each of the handle-value pairs is checked to see if it matches the Characteristic UUID that was requested. If it matches, then the characteristic is considered to be found.

It is possible that not all handle-value pairs are returned in one response. In that case, the Attribute Handle in the response would be less than the Ending Handle given in the request. The client can then send another Read By Type Request by modifying the Starting Handle to the last received Attribute Handle + 1. This subprocedure is completed when the server returns an Error Response with the error code set to Attribute Not Found.

This is shown in Figure 13.19. The client sends the Read By Type Request message to the server by setting the Starting Handle and Ending Handle to the attribute handles received in the primary service discovery procedure and the Type set to <>Characteristic<>. The server responds with the first set of Handle-Value pairs of the characteristic declarations in the service definition. The client checks those handle-value pairs to see if Characteristic UUID matches the one that was provided. If it matches then the characteristic is considered to be found.

The client then sends the next Read By Type Request by adjusting the Starting Handle to retrieve the next set of Handle-Value Pairs. The server responds with the next set of Handle Value pairs till no more remain per the Starting Handle specified by the client. At that time the server returns an Error Response. This indicates to the client that all handles for the characteristic declarations have been retrieved.

13.6.5 Characteristic Descriptor Discovery

The Characteristic Discovery procedure is used by the client to discover the characteristic descriptors of a characteristic. As shown in Figure 13.10, the characteristic descriptor is an optional component of the characteristic definition. This category contains one subprocedure: Discover All Characteristic Descriptors.

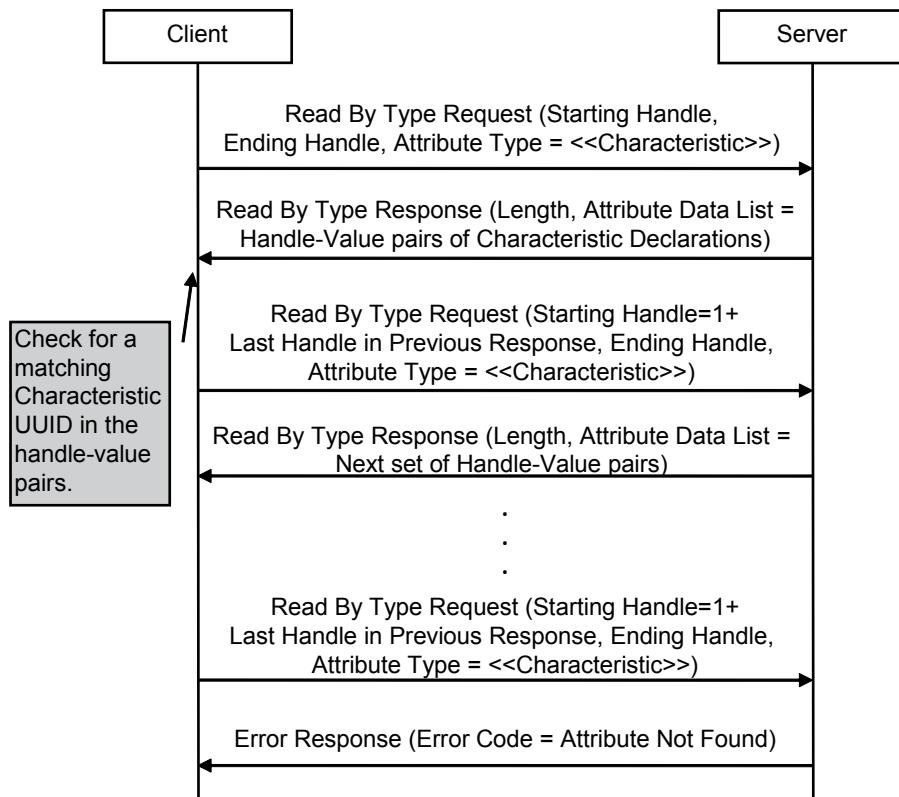


Figure 13.19 Discover Characteristics by UUID.

13.6.5.1 Discover All Characteristic Descriptors

This subprocedure is used by the client to discover all the characteristic descriptors within a characteristic definition.

It uses the ATT Find Information Request with the following parameters:

- Starting Handle: Starting Handle of the specified characteristic value + 1.
- Ending Handle: Ending Handle of the specified characteristic.

The server either sends the ATT Find Information Response or an Error Response. If the server sends the Find Information Response, then it contains the following parameters:

- Format: Specifies whether the handles are 16-bit or 128-bit.
- Attribute Data List: handle-value pairs containing Attribute Handle and Attribute Value.
 - Attribute Handle: Handle of the characteristic descriptor declaration.
 - Attribute Value: Characteristic Descriptor UUID.

It is possible that not all handle-value pairs are returned in one response. In that case, the Attribute Handle in the response would be less than the Ending

Handle given in the request. The client can then send another Find Information Request by modifying the Starting Handle to the last received Attribute Handle + 1. This subprocedure is completed when the server returns an Error Response with the error code set to Attribute Not Found.

This is shown in Figure 13.20. The client sends the Find Information Request message to the server by setting the Starting Handle and Ending Handle. The server responds with the first set of Handle-Value pairs of the characteristic descriptors in the characteristic definition.

The client then sends the next Find Information Request by adjusting the Starting Handle to retrieve the next set of Handle-Value Pairs. The server responds with the next set of Handle Value pairs until none remain per the Starting Handle specified by the client. At that time the server returns an Error Response. This indicates to the client that all handles for the characteristic descriptors have been retrieved.

13.6.6 Characteristic Value Read

The Characteristic Value Read procedure is used by the client to read a Characteristic Value from the server. This category contains four sub-procedures:

- Read Characteristic Value.
- Read Long Characteristic Values.
- Read Using Characteristic UUID.
- Read Multiple Characteristic Values.

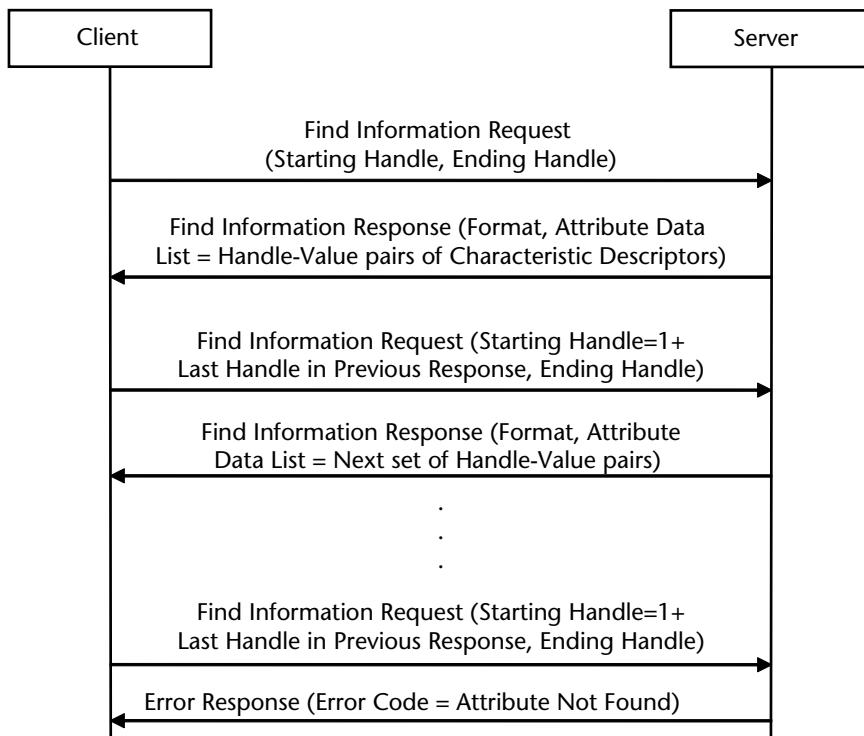


Figure 13.20 Discover All Characteristic descriptors.

13.6.6.1 Read Characteristic Value

This subprocedure is used by the client to read a Characteristic Value from the server. It requires a Characteristic Value Handle as an input. It uses the ATT Read Request with the following parameter:

- Attribute Handle: Characteristic Value Handle (This handle was retrieved in the Characteristic Discovery Procedure).

The server either sends the ATT Read Response or an Error Response. The server can send an Error Response if the read operation is not permitted on the Characteristic Value or if the authentication, authorization, or encryption size is insufficient. Note from Figure 13.10 that the permissions for the Characteristic Value are set by the profile.

If the server sends the Read Response, then it contains the following parameter:

- Attribute Value: Characteristic Value.

If the length of the characteristic value is more than ATT_MTU-1, then only ATT_MTU-1 bytes are returned and the remaining bytes may be read using the Read Long Characteristic Value procedure. This is shown in Figure 13.21.

13.6.6.2 Read Long Characteristic Value

This subprocedure is very similar to the Read Characteristic Value Procedure explained in previous section. It is used when the length of the Characteristic Value is longer than that can be sent in a single Read Response message.

It uses the ATT Read Blob Request with the following parameters:

- Attribute Handle: Characteristic Value Handle (This handle was retrieved in the Characteristic Discovery Procedure).
- Value Offset: Offset from where the Attribute Value is to be read.

The server either sends the ATT Read Blob Response or an Error Response.

The server can send an Error Response if the read operation is not permitted on the Characteristic Value or if the authentication, authorization, or encryption size is insufficient. Note from Figure 13.10 that the permissions for the Characteristic

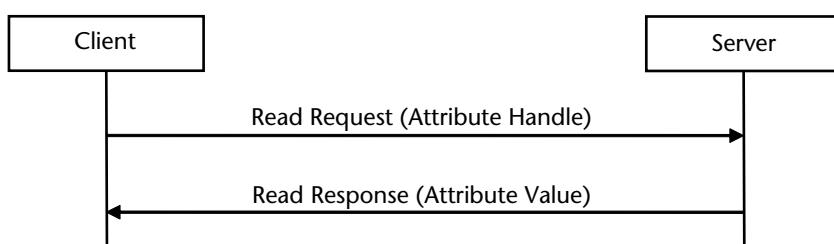


Figure 13.21 Read Characteristic Value.

Value are set by the profile. An error is also returned if the Characteristic Value is shorter than ATT_MTU-1 bytes.

If the server sends the Read Blob Response, then it contains the following parameter:

- Part Attribute Value: Characteristic Value bytes starting at the specified Value Offset.

It is possible to read the first part of the Attribute Value using Read Request and the remaining parts using Read Blob Request. This subprocedure is shown in Figure 13.22.

13.6.6.3 Read Using Characteristic UUID

This subprocedure is used by the client to read a Characteristic Value from the server when it knows the Characteristic UUID but does not know the Characteristic Value Handle.

It uses the ATT Read By Type Request with the following parameters:

- Starting Handle: Starting Handle.
- Ending Handle: Ending Handle.
- Attribute Type: Characteristic UUID.

The Starting Handle and Ending Handle are typically the handle range of the service in which the characteristic is located. The server either sends the ATT Read By Type Response or an Error Response.

If the server sends the Read By Type Response, then it contains the following parameters:

- Attribute Handle: Characteristic Value Handle.
- Attribute Value: Characteristic Value.

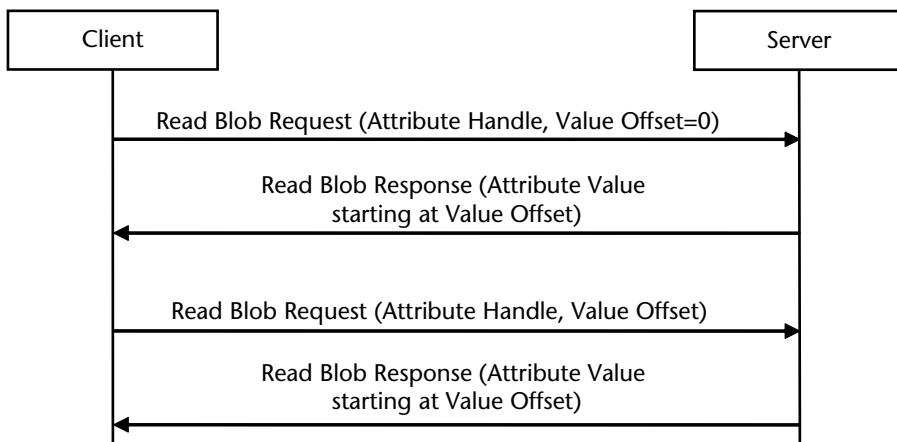


Figure 13.22 Read Long Characteristic Value.

This is shown in Figure 13.23.

13.6.6.4 Read Multiple Characteristic Values

This subprocessure is used by the client to read multiple Characteristic Values from the server when it knows the Characteristic Value Handles. This is very similar to the Read Characteristic Value subprocessure.

It uses the ATT Read Multiple Request with the following parameters:

- Set Of Handles: List of Characteristic Value Handles for which the Values are to be retrieved.

The server either sends the ATT Read Multiple Response or an Error Response. The server can send an Error Response if the read operation is not permitted on any of the Characteristic Value or if the authentication, authorization or encryption size is insufficient. Note from Figure 13.10 that the permissions for the Characteristic Value are set by the higher layer profile or application.

If the server sends the Read Multiple Response, then it contains the following parameters:

- Set Of Value: List of Characteristic Values.

This is shown in Figure 13.24.

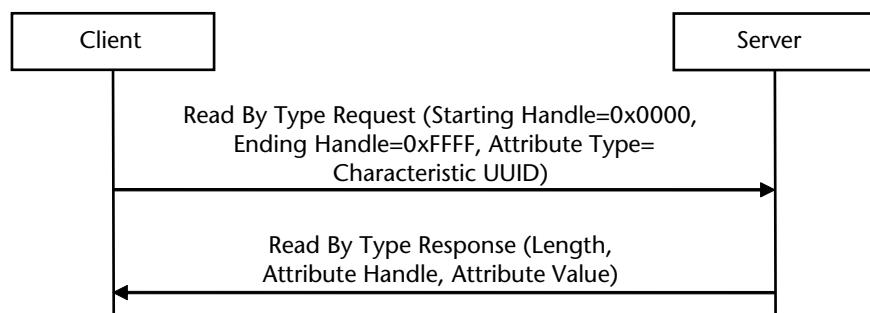


Figure 13.23 Read Using Characteristic UUID.

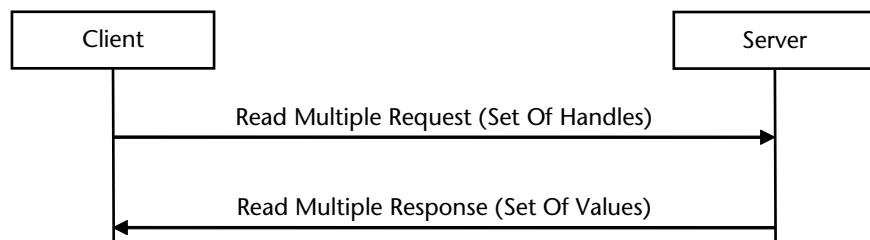


Figure 13.24 Read Multiple Characteristic Values.

13.6.7 Characteristic Value Write

The Characteristic Value Write procedure is used by the client to write a Characteristic Value to the server. This category contains five sub-procedures:

- Write Without Response.
- Signed Write Without Response.
- Write Characteristic Value.
- Write Long Characteristic Value.
- Reliable Writes.

13.6.7.1 Write Without Response

This subprocedure is used by the client to write a Characteristic Value to the server when it does not need an acknowledgment that the write has been successful. This is the simplest procedure that a client can use to write a Characteristic Value to the server.

It uses the ATT Write Command with the following parameters:

- Attribute Handle: Characteristic Value Handle (This handle was retrieved in the Characteristic Discovery Procedure).
- Attribute Value: Characteristic Value to be written.

Since this is an ATT command, there is no response from the server. (Note that the server responds to ATT requests but not ATT commands). This is shown in Figure 13.25.

13.6.7.2 Signed Write Without Response

This subprocedure is used by the client to write a Characteristic Value to the server when it does not need an acknowledgment that the write has been successful. It is similar to the Write Without Response procedure. The main enhancement is that the Attribute Value is signed by the client with an authentication signature. This authentication signature is verified by the server before writing the value. It uses the ATT Write Command with the following parameters:

- Attribute Handle: Characteristic Value Handle (This handle was retrieved in the Characteristic Discovery Procedure).



Figure 13.25 Write Without Response.

- Attribute Value: Characteristic Value that is authenticated by signing the value.

Since this is an ATT command, there is no response from the server. (Note that the server responds to ATT requests but not ATT commands).

This subprocedure requires the client and server to share a bond (defined in GAP profile). The server verifies the authentication signature to check the authenticity of the client before writing the value. This is shown in Figure 13.26.

13.6.7.3 Write Characteristic Value

This subprocedure is used by the client to write a Characteristic Value to the server. In this subprocedure the client receives an acknowledgment from the server once the value is written. It uses the ATT Write Request with the following parameters:

- Attribute Handle: Characteristic Value Handle (This handle was retrieved in the Characteristic Discovery Procedure).
- Attribute Value: Characteristic Value to be written.

The server either sends the ATT Write Response or an Error Response.

The server can send an Error Response if the write operation is not permitted on the Characteristic Value or if the authentication, authorization, or encryption size is insufficient. Note from Figure 13.10 that the permissions for the Characteristic Value are set by the higher layer profile or application. If the server sends the Write Response, then it indicates that the value has been successfully written. This is shown in Figure 13.27.

13.6.7.4 Write Long Characteristic Value

This subprocedure is used by the client to write a Characteristic Value to the server when the length of the Characteristic Value is longer than that can be written in a single Write Request. It splits the Characteristic Value into parts and uses multiple ATT Prepare Write Requests to queue the Characteristic Value parts on the server. Then it uses the Execute Write Request to request the server to write all the queued parts.

It uses the ATT Prepare Write Request with the following parameters to queue the Characteristic Value to write after splitting the Characteristic Value into several parts.

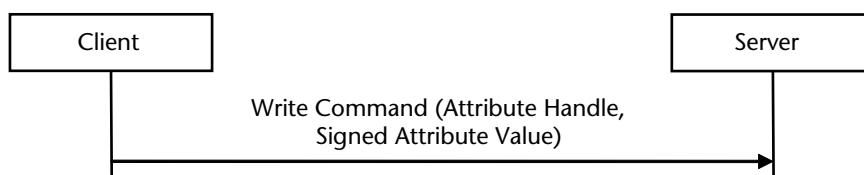


Figure 13.26 Signed Write Without Response.

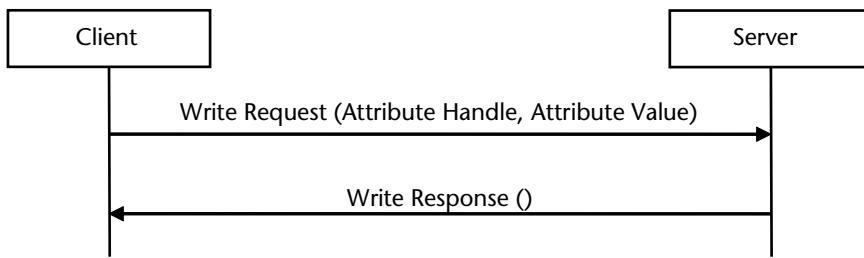


Figure 13.27 Write Characteristic Value.

- Attribute Handle: Characteristic Value Handle (This handle was retrieved in the Characteristic Discovery Procedure).
- Value Offset: The offset of the first octet to be written.
- Part Attribute Value: The octets of the Characteristic Value starting at Value Offset.

Once all the parts are queued, the subprocedure uses the ATT Execute Write Request with the following parameters to write all the queued parts.

- Flags = 0x01 to write all the queued parts.

The server can send an Error Response if the write operation is not permitted on the Characteristic Value or if the authentication, authorization, or encryption size is insufficient.

This is shown in Figure 13.28. The client splits the Characteristic Value into several parts as per the ATT_MTU and then sends the subsequent parts in the Prepare Write Request after adjusting the Value Offset. The server queues the various parts that it receives. After sending all the parts, the client finally sends the Execute Write Request to write all the queued parts.

One point to note is that, even though the Prepare Write Response from the server contains the same parameters sent in the Prepare Write Request, it is not mandatory for the client to verify that the server has sent back the same values. There is a possibility that some values may have been corrupted during sending from the client to the server (the client does not cross check the received values to see if those were corrupted). The next section describes the Reliable Writes subprocedure where it is mandatory for the client to check if the server returned the same values in response that it had sent in request.

13.6.7.5 Reliable Writes

This subprocedure is used by the client to write a Characteristic Value to the server when it requires assurance that the correct Characteristic Value is going to be written by the server. It is also used when multiple values are to be written together in one single operation. For example, this procedure is used to ensure that a set of values is written by a client together in a particular order without any other values being written by another client in between.

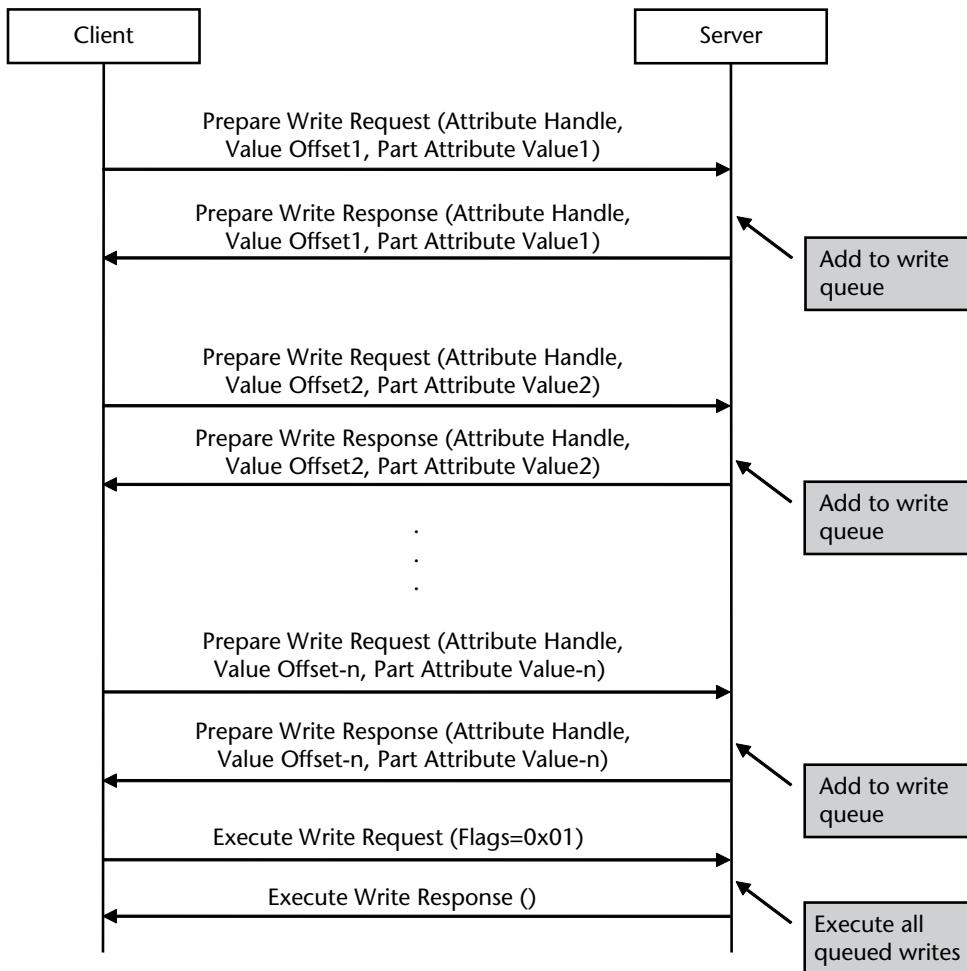


Figure 13.28 Write Long Characteristic Value.

This procedure requires the server to send back the same Characteristic Value which the client had sent to it so that the client can verify that the server had received the value correctly. It uses the ATT Prepare Write Request with the following parameters to queue the Characteristic Value to write after splitting the Characteristic Value into several parts.

- Attribute Handle: Characteristic Value Handle (This handle was retrieved in the Characteristic Discovery Procedure).
 - Value Offset: The offset of the first octet to be written.
 - Part Attribute Value: The octets of the Characteristic Value starting at Value Offset.

The subsequent ATT Prepare Write Request messages may also send different Attribute Handles if several Characteristic Values are to be written together. It then

uses the ATT Execute Write Request with the following parameters to write all the queued parts.

- Flags = 0x01 to write all the queued parts.

The server can send an Error Response if the write operation is not permitted on any of the Characteristic Values or if the authentication, authorization, or encryption size is insufficient. The server can also send an Error Response if the queue is full. This procedure is shown in Figure 13.29. The client sends the first Characteristic Value to be written in the Prepare Write Request. The server queues the value and also returns the same value in Prepare Write Response. The client verifies the received value with the value it sent to check if the server indeed received the value correctly. After that the client may either send other parts of the same Characteristic Value or another Characteristic Value in the subsequent Prepare Write Requests. Once it has sent all the Characteristic Values, it finally sends the Execute Write Request. On receipt of the Execute Write Request, the

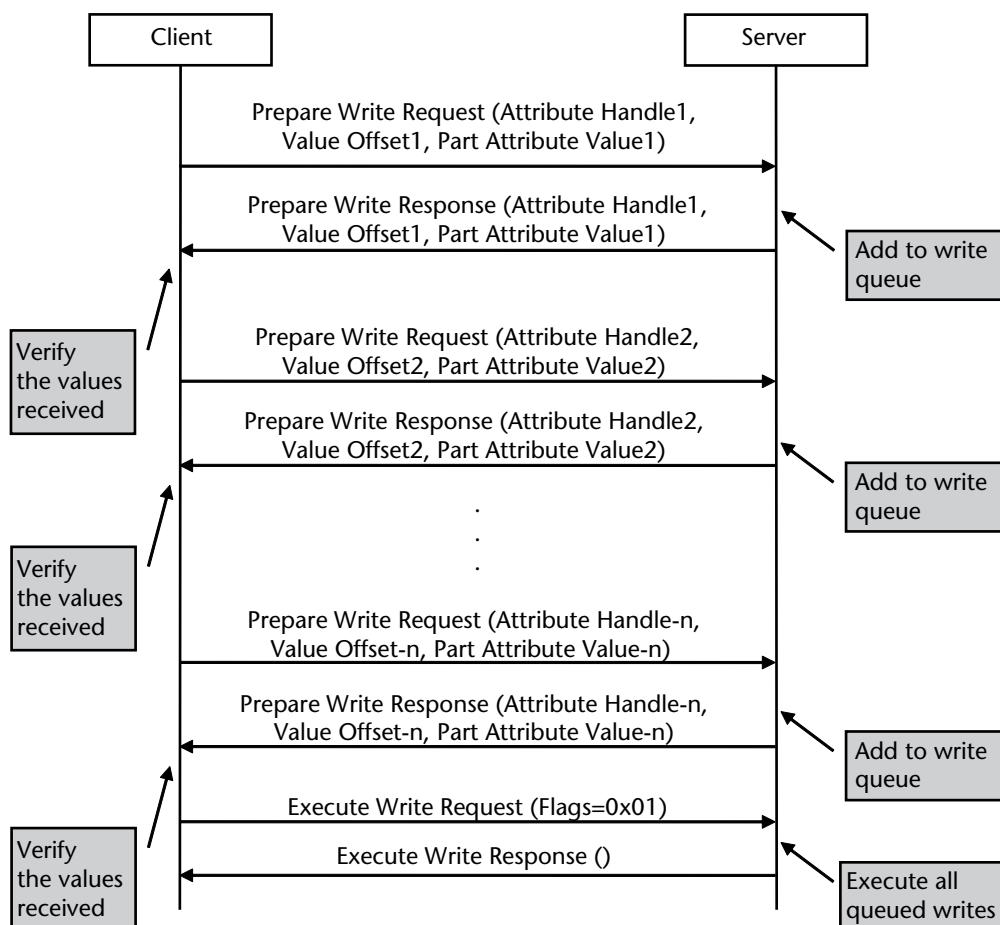


Figure 13.29 Reliable Writes.

server writes all the values that it had queued up in the same order in which it had received the requests.

13.6.8 Characteristic Value Notification

The Characteristic Value Notification procedure is used by the server to notify a Characteristic Value to the client. Whether a Characteristic Value can be notified is defined by the profile. This category contains one subprocedure: Notifications.

13.6.8.1 Notifications

This subprocedure is used by the server to notify a Characteristic Value to the client without expecting any acknowledgment from the client. It uses the ATT Handle Value Notification with the following parameters:

- Attribute Handle: Characteristic Value Handle that is to be notified.
- Attribute Value: Characteristic Value.

Since this is an ATT notification, there is no response from the client. (Note that the client responds to ATT indications but not ATT notifications). This is shown in Figure 13.30.

13.6.9 Characteristic Value Indication

The Characteristic Value Indication procedure is used by the server to indicate a Characteristic Value to the client. Whether a Characteristic Value can be indicated is defined by the profile. This category contains one sub-procedure: Indications.

13.6.9.1 Indications

This subprocedure is used by the server to indicate a Characteristic Value to the client. The client responds with a confirmation. It uses the ATT Handle Value Indication with the following parameters:

- Attribute Handle: Characteristic Value Handle that is to be indicated.
- Attribute Value: Characteristic Value.

The client responds with a Handle Value Confirmation to confirm that it has received the indication. This is shown in Figure 13.31.

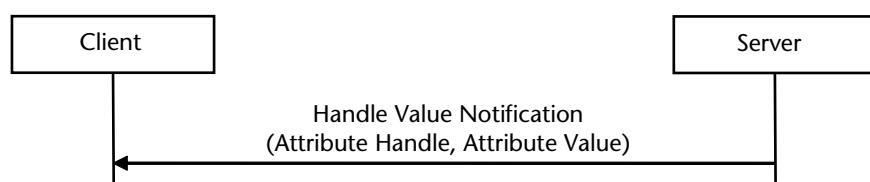


Figure 13.30 Notifications.

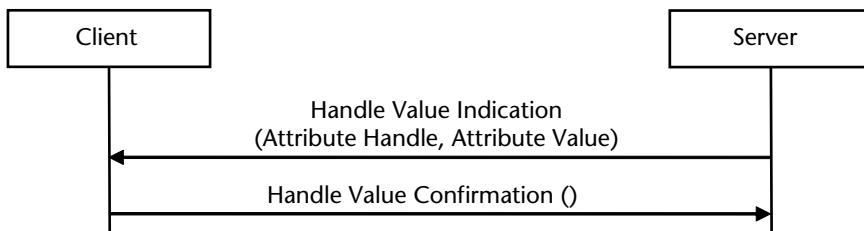


Figure 13.31 Indications.

13.6.10 Characteristic Descriptors

The Characteristic Descriptors procedure is used by the client to read and write the Characteristic Descriptors on the server. All these subprocedures require the handle of the Characteristic Descriptor Declaration. This handle may be retrieved using the Characteristic Descriptor Discovery Procedure which was described earlier.

This category contains four subprocedures:

- Read Characteristic Descriptors.
- Read Long Characteristic Descriptors.
- Write Characteristic Descriptors.
- Write Long Characteristic Descriptors.

13.6.10.1 Read Characteristic Descriptors

This subprocedure is used by the client to read a Characteristic Descriptor from the server. The client needs to have the Attribute Handle of the Characteristic Descriptor Declaration to use this subprocedure. It uses the ATT Read Request with the following parameters:

- Attribute Handle: Characteristic Descriptor Handle. (This handle was retrieved in the Characteristic Descriptor Discovery Procedure.)

The server either sends the ATT Read Response or an Error Response. The server can send an Error Response if the read operation is not permitted on the Attribute Value or if the authentication, authorization, or encryption size is insufficient. Note from Figure 13.10 that the permissions for the Attribute Value are set by the higher layer profile or application.

If the server sends the Read Response, then it contains the following parameter:

- Attribute Value: Profile specific value that is stored in the Characteristic Descriptor Declaration.

If the length of the characteristic value is more than ATT_MTU-1, then only ATT_MTU-1 bytes are returned and the remaining bytes may be read using the Read Long Characteristic Descriptor procedure. This is shown in Figure 13.32.

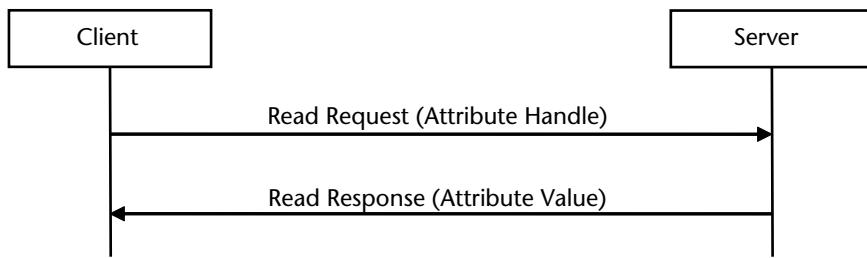


Figure 13.32 Read Characteristic Descriptor.

13.6.10.2 Read Long Characteristic Descriptor

This subprocess is very similar to the Read Characteristic Descriptor Procedure explained in previous section. It is used when the length of the Attribute Value is longer than that can be sent in a single Read Response message. It uses the ATT Read Blob Request with the following parameters:

- Attribute Handle: Characteristic Descriptor Handle (This handle was retrieved in the Characteristic Descriptor Discovery Procedure.)
- Value Offset: Offset from where the Attribute Value is to be read.

The server either sends the ATT Read Blob Response or an Error Response. The server can send an Error Response if the read operation is not permitted on the Attribute Value or if the authentication, authorization, or encryption size is insufficient. Note from Figure 13.10 that the permissions for the Attribute Value are set by the higher layer profile or application. An error is also returned if the Attribute Value is shorter than ATT_MTU-1 bytes.

If the server sends the Read Blob Response, then it contains the following parameter:

- Part Attribute Value: Attribute Value byte starting at the specified Value Offset.

It is possible to read the first part of the Attribute Value using Read Request and the remaining parts using Read Blob Request. This is shown in Figure 13.33.

13.6.10.3 Write Characteristic Descriptors

This subprocess is used by the client to write an Attribute Value into a Characteristic Descriptor on the server. In this subprocess the client receives an acknowledgment from the server once the value is written.

It uses the ATT Write Request with the following parameters:

- Attribute Handle: Characteristic Descriptor Handle (this handle was retrieved in the Characteristic Descriptor Discovery Procedure).
- Attribute Value: Attribute Value to be written into the Characteristic Descriptor.

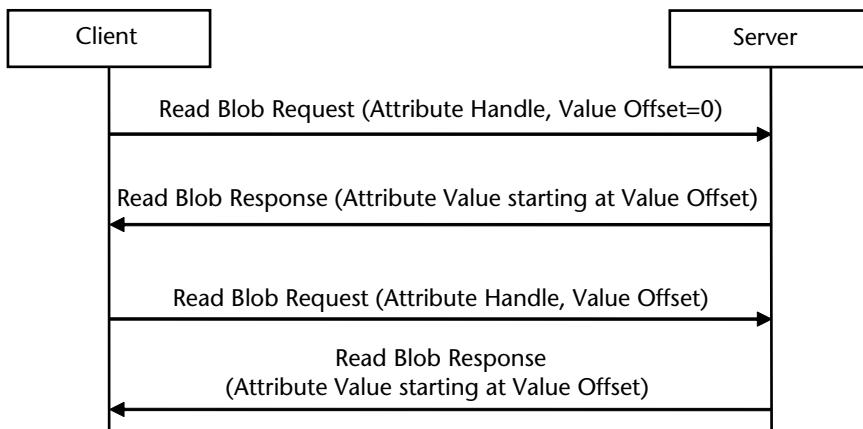


Figure 13.33 Read Long Characteristic Descriptor.

The server either sends the ATT Write Response or an Error Response. The server can send an Error Response if the write operation is not permitted on the Attribute Value or if the authentication, authorization, or encryption size is insufficient. Note from Figure 13.10 that the permissions for the Attribute Value are set by the profile. An Error Response may also be returned if the value being written is invalid or of the incorrect size.

If the server sends the Write Response, then it indicates that the value has been successfully written.

This is shown in Figure 13.34.

13.6.10.4 Write Long Characteristic Descriptors

This subprocess is used by the client to write an Attribute Value into a Characteristic Descriptor on the server when the length of the Attribute Value is longer than that can be written in a single Write Request. It splits the Attribute Value into parts and uses multiple ATT Prepare Write Requests to queue the Attribute Value parts on the server. Then it uses the Execute Write Request to request the server to write all the queued parts. It uses the ATT Prepare Write Request with the following parameters to queue the Attribute Value to write after splitting the Attribute Value into several parts:

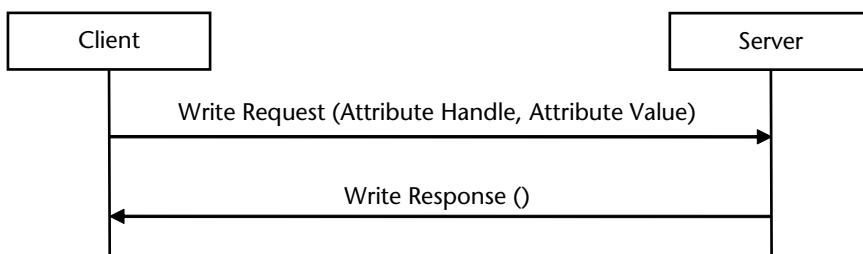


Figure 13.34 Write Characteristic Descriptor.

- Attribute Handle: Characteristic Descriptor Handle (this handle was retrieved in the Characteristic Descriptor Discovery Procedure).
- Value Offset: The offset of the first octet to be written.
- Part Attribute Value: The octets of the Attribute Value starting at Value Offset.

It then uses the ATT Execute Write Request with the following parameters to write all the queued parts.

- Flags = 0x01 to write all the queued parts.

The server can send an Error Response if the write operation is not permitted on the Attribute Value or if the authentication, authorization, or encryption size is insufficient. Note from Figure 13.10 that the permissions for the Attribute Value are set by the higher layer profile or application. An Error Response may also be returned if the value being written is invalid or of the incorrect size.

This is shown in Figure 13.35. The client splits the Attribute Value into several parts as per the ATT_MTU and then sends the subsequent parts in the Prepare Write Request after adjusting the Value Offset. The server queues the various parts that it receives. After sending all the parts, the client finally sends the Execute Write Request to write all the queued parts.

One point to note is that, even though the Prepare Write Response from the server contains the same parameters that were sent in the Prepare Write Request, it is not mandatory for the client to verify if the server has sent back the same values.

13.7 Timeouts

All GATT procedures use a timeout mechanism to ensure that the procedure does not wait infinitely for a response from the remote side. If a timeout happens, it is assumed that the link has gone down and no further GATT procedures are performed. Further GATT procedures are performed only after a new ATT bearer has been established.

13.8 GATT Service

A server may expose a GATT service in addition to the profile specific services. If the GATT service is exposed by the server then it is exposed as a primary service with service UUID set to <>Generic Attribute Profile<>. GATT specification defines only one characteristic that can be contained in this service:

- Service Changed Characteristic.

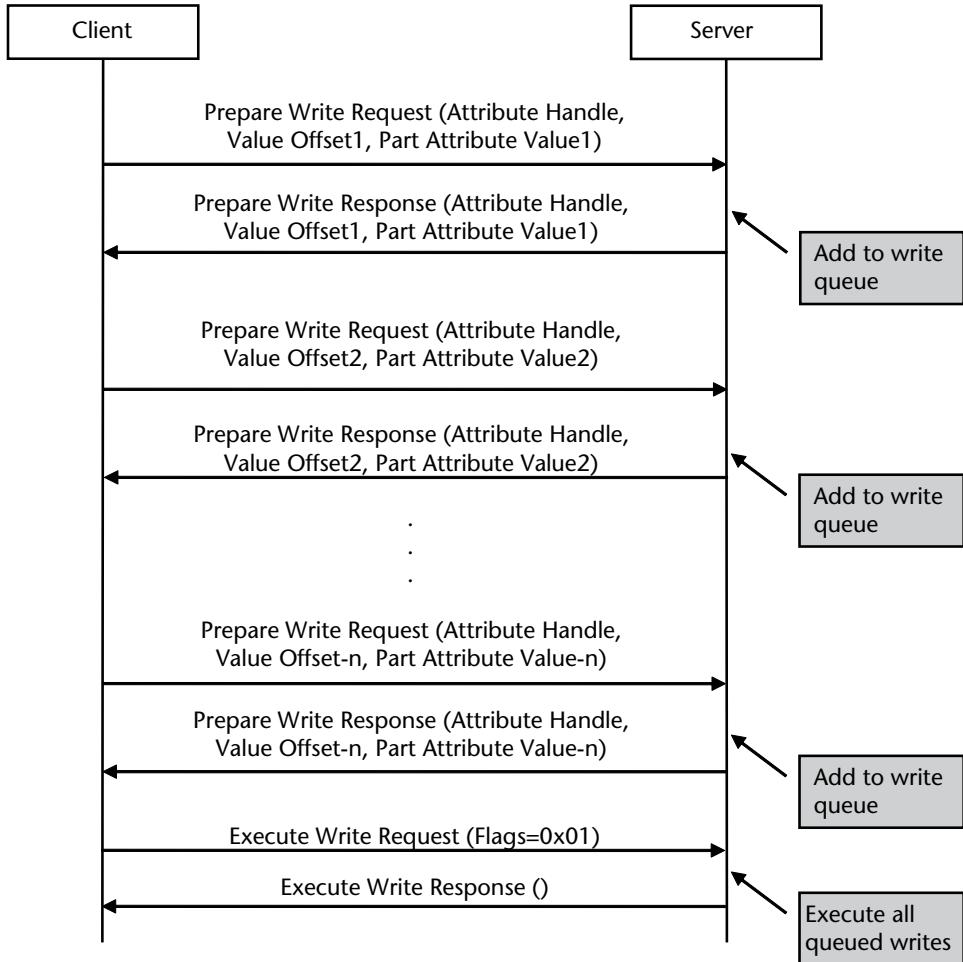


Figure 13.35 Write Long Characteristic Descriptor.

13.8.1 Service Changed Characteristic

The service changed characteristic is used by the server to indicate to the clients that the services on the server have changed. This can happen when new services are added, services are removed, or modified on the server. This characteristic is implemented by the server only if the list of services supported by the device can change over the lifetime of a device. Otherwise this characteristic is not implemented. The list of services supported by a device can change, for example, after a firmware upgrade.

The clients that are bonded to the server are informed about the changed services when they reconnect to the server. This was shown in Figure 13.9. The definition of the service changed characteristic is shown in Figure 13.36.

The Characteristic Value Declaration contains two handles 0xAAAA and 0xB BBBB. These indicate the starting and ending handle of the attribute handles that have changed.

Service Definition**Service Declaration**

Attribute Handle

Attribute Type = 0x2800 for <<Primary Service>>

Attribute Value = Service UUID (16-Bit or 128-Bit)

Attribute Permissions = Read Only, No Authentication/Authorization

Characteristic Definition**Characteristic Declaration**

Attribute Handle

Attribute Type = 0x2803 (UUID for <<Characteristic>>)

Attribute Value

Characteristic Properties = 0x26

Characteristic Value Attribute Handle,
Characteristic UUID = 0x2A05 <<Service Changed>>

Attribute Permissions = No Authentication/Authorization

Characteristic Value Declaration

Attribute Handle

Attribute Type = 0x2A05 (UUID for <<Service Changed>>)

Attribute Value

0xAAAA – Start of Affected Attribute Handle Range
0xB BBBB – End of Affected Attribute Handle Range

Attribute Permissions = No Authentication, No Authorization,
Not Readable, Not Writable.

Figure 13.36 Service Changed Service.

The <<Service Changed>> characteristic is implemented as a control-point attribute. It is configured to be indicated. The Characteristic Properties field within the Characteristic Declaration is set to 0x26. This is a bit mask which has the following bits set:

- 0x20 – Indicate: This characteristic value can be indicated.
- 0x02 – Read: This value can be read.
- 0x04 – Write Without Response: This value can be written using the Write Without Response procedure.

13.9 Security Considerations

13.9.1 Authentication and Authorization Requirements

As shown in Figure 13.10, each attribute or characteristic defined by GATT profile has its own associated set of authentication and authorization requirements. These are defined independently for each of the attribute or characteristic and could be one of the following:

- Specified by the GATT profile.
- Specified by a higher layer profile.
- Specified by the application.

The authentication and authorization procedures will be explained in Chapter 14. As a thumb rule, the list of services and characteristics supported by a device is considered to have the following permissions:

- Read-Only.
- No Authentication required.
- No Authorization required.

This means that the service declaration, include definition and characteristic declaration are considered read-only and other devices can at least retrieve these without getting an error. The characteristics could be either read-only or write-only or read-write. If a client, for example, tries to access a write-only attribute using one of the read procedures, it gets an Error Response.

13.10 Summary

The GATT profile uses the attributes defined by the ATT protocol as building blocks and defines a hierarchy using those services. This hierarchy organizes the attributes into profiles which may contain services which may in turn contain characteristics. Based on the functionality, similar attributes are grouped together. This provides a very effective and powerful mechanism for devices to exchange information.

GATT provides the facility to discover the services of the remote devices (similar to the services provided by SDP in the BR/EDR world). It then provides facilities to discover the characteristics and included services that are contained in the service. Finally it provides mechanisms to read, write, notify and indicate characteristics.

The GATT based profile architecture abstracts all complexity related to the hierarchy and access of attributes so that the profiles on top are very simple.

Bibliography

[1] Bluetooth Core Specification 4.0 <http://www.bluetooth.org>.

Generic Access Profile

14.1 Introduction

The Generic Access Profile (GAP) defines the base functionality common to all Bluetooth devices. This includes modes of device and generic procedures related to discovery of the devices in vicinity, connecting to devices, and security. GAP is mandatory to be implemented for all devices that support Bluetooth. The position of GAP in the LE Protocol Stack is shown in Figure 14.1.

In general the profiles help to ensure interoperability between devices from different vendors. This is one of the strong points of Bluetooth technology compared to many other wireless technologies. Bluetooth devices from one vendor can seamlessly work with devices from other vendors provided they have some profiles in common.

The profiles also recommend the terminology to be used at the user interface level so that the end user gets a similar look and feel when using devices from different vendors. For example, GAP recommends the terminology to be used for representing various Bluetooth parameters (Bluetooth Device Address, Device Name, Passkey, etc.) at the user interface level. So, when asking for a user to enter the PIN key, GAP recommends the user interface to refer to it as “Bluetooth Passkey”.

Before going further, it is highly recommended to read the sections related to GAP in Chapter 4 to get a good background. This chapter describes the LE specific parts of GAP. GAP defines the basic requirements of an LE device to include at least the following:

- Physical Layer;
- Link Layer;
- L2CAP;
- Security Manager;
- ATT;
- GATT.

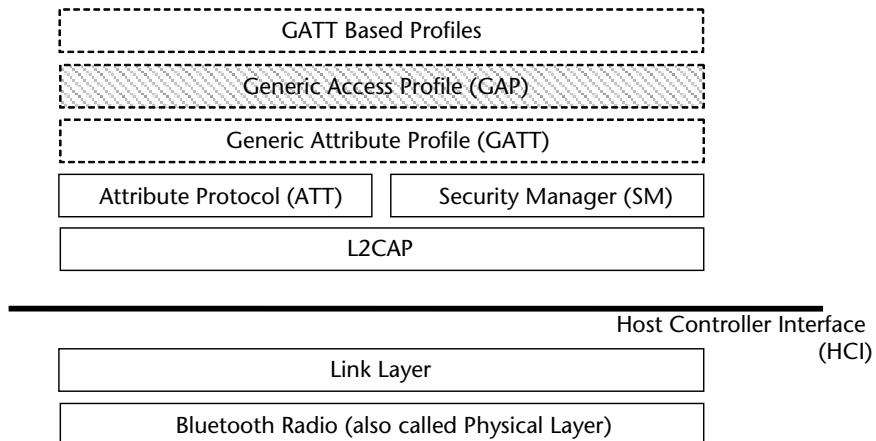


Figure 14.1 GAP in LE Protocol Stack.

14.2 Roles

GAP defines the following four roles for LE devices.

1. Broadcaster;
2. Observer;
3. Peripheral;
4. Central.

A device may operate in multiple GAP roles at the same time provided the link layer supports this.

14.2.1 Broadcaster Role

A device that is operating in the broadcaster role transmits advertising events periodically. This role is optimized for transmitter only applications. The device may not contain a receiver to optimize the memory footprint and cost. Devices supporting this role use the link layer advertising events to broadcast data. This is shown in Figure 14.2.

14.2.2 Observer Role

A device that is operating in the observer role receives the advertising events. The observer role is optimized for receiver only applications. The device may not contain a transmitter to optimize the memory footprint and cost. The devices supporting these roles receive the data that is sent in the link layer advertisement events. This is shown in Figure 14.2.

14.2.3 Peripheral Role

The device in the Peripheral role is the one that accepts a connection request from another device. This role is optimized for devices that support a single connection



Figure 14.2 Broadcast and Observer roles.

and act as a Slave in a piconet. A device needs to have both a transmitter and a receiver to support this role. This is shown in Figure 14.3.

14.2.4 Central Role

The device in the Central role is the one that initiates establishment of a physical connection. The Central role supports multiple connections and is the initiator of connections. This device acts as a Master in a piconet and may connect to more than one Slave. This is shown in Figure 14.3.

14.3 Representation of Bluetooth Parameters

GAP states requirements about the generic terms that should be used on the user interface level. These are useful not only when designing user interfaces but also in user manuals, documentation, advertising, etc. This helps to ensure a uniform user experience irrespective of the vendor who builds the device or the application.

14.3.1 Bluetooth Device Address

A Bluetooth Device Address is used to identify a Bluetooth device. An LE-only device can either use a public address or a random address. Public Addresses and Random Addresses were explained in Chapter 8. The term to be used on the user interface level is defined by GAP as “Bluetooth Device Address”. It is represented as 12 hexadecimal characters which may or may not be separated into subparts by

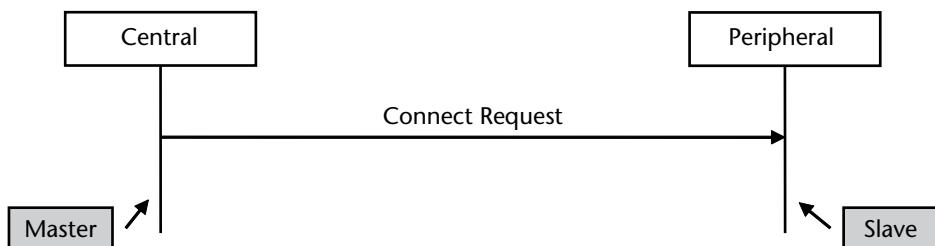


Figure 14.3 Peripheral and Central roles.

the colon symbol. So the Bluetooth Device Address could be represented as 00AAB-BCCDDEE or 00:AA:BB:CC:DD:EE. If a device supports dual mode (Both BR/EDR and LE), then it uses the same Bluetooth Device Address (also referred to as BD_ADDR) on both the BR/EDR physical channel and LE physical channel.

14.3.2 Bluetooth Device Name

Since it is difficult to refer to devices using a 48-bit address at a user level, the Bluetooth devices expose a user friendly name. This is a character string which is easier to remember, refer to, and distinguish one device from another. For Example ‘BT Thermometer’, ‘SmartPhone’, ‘Deskjet Printer’.

For a BR/EDR device, the name of the remote device can be fetched using HCI_Remote_Name_Request command. It may also be returned in the Extended Inquiry Result Event if an HCI_Inquiry was done after setting the Inquiry_Mode to Extended Inquiry Result Format.

For an LE only device the name is stored in the Device Name Characteristic. This Characteristic can be read using the GATT procedures. (Note that in the case of BR/EDR, the procedures to fetch the name of the remote device were defined at the link manager level. In the case of LE, they are placed much higher, at the GATT level. This is another step towards making the link layer much simpler in case of LE). The name is up to 248 octets in length and is UTF-8 encoded. It is Null terminated if the length is less than 248 octets. The term to be used on the UI level is “Bluetooth Device Name”.

14.3.3 Bluetooth Passkey

The Bluetooth Passkey is used to authenticate two Bluetooth devices during the pairing process. For LE devices, the Bluetooth Passkey is a 6-digit numerical value and is represented in the integer range 000000 to 999999. The term to be used on the UI level is “Bluetooth Passkey”. Similar to BR/EDR, a device could either have a fixed passkey or it may be entered by the user. The devices that have no mechanism to enter the passkey (no keypad) would use a fixed passkey.

14.3.4 Bluetooth Class of Device

The Class of Device is valid for only BR/EDR devices and not used for LE-only devices.

14.3.5 Pairing—Authentication and Bonding

Pairing is performed by the Security Manager as explained in Chapter 11. If the user initiates the pairing procedure to create a bond (trusted relationship) between the two devices, then the procedure is referred to as Bonding. If the user is requested to enter the Passkey as a part of the connection establishment procedure, then the procedure is referred to as Authentication.

14.4 Advertising and Scan Response Data Format

The Advertiser can send data to the devices that are scanning using the various types of advertising events. In addition if the Scanner requests for additional information then the Advertiser can provide additional data using the SCAN_RSP advertising packets. This is shown in Figure 14.4.

The Advertising Data is sent in the AdvData field of the following packets:

- Connectable Undirected (ADV_IND).
- Non-Connectable Directed (ADV_NONCONN_IND).
- Scannable Undirected (ADV_SCAN_IND).

The Scan Response Data is sent in the ScanRspData field of the SCAN_RSP packets. These packets were explained in Chapter 8. Note from Chapter 8 that the Connectable Directed (ADV_DIRECT_IND) packets do not contain any host data. Both the AdvData and ScanRspData are 0 to 31 octets in length. These contain a sequence of Advertising Data (AD) structures. The AD structure is shown in Figure 14.5.

The AD Structure contains three fields:

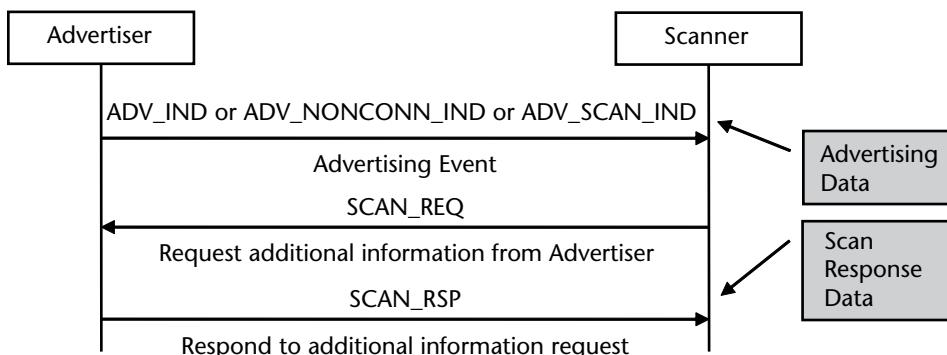


Figure 14.4 Advertising Events containing Advertising and Scan Response Data.

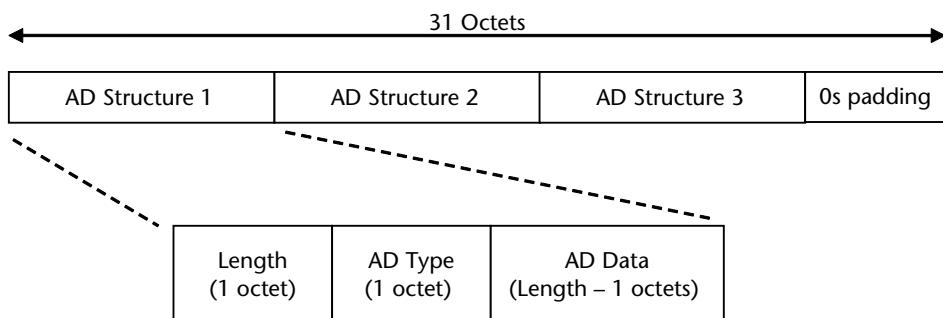


Figure 14.5 Format of Advertising Data and Scan Response Data.

1. Length.
2. AD Type.
3. AD Data.

The AD Type field is used to specify the type of data contained in AD Data. The different values of AD Type field are listed in the Bluetooth Assigned Numbers Document. Some of the commonly used AD Type fields are explained below.

14.4.1 Local Name (AD Type = 0x08 or 0x09)

This is used to specify the device name. If the device name is too long, then a shortened version can be provided in this AD Type and the full name can be read from the device name characteristic. The AD Type field indicates the local name as follows:

- 0x08: Shortened local name.
- 0x09: Complete local name (If it can fit the AdvData).

14.4.2 Flags (AD Type = 0x01)

The Flag AD Type is a bit map used to provide information about the device. The various values are shown in Table 14.1. The value can be a combination of these values. For example a value of 0x06 would indicate:

- LE General Discoverable Mode.
- BR/EDR not supported.

14.4.3 Manufacturer Specific Data (AD Type = 0xFF)

This is used to specify the company identifier and any manufacturer specific information.

Figure 14.6 shows an example of ADV_IND packets that contain AdvData. In this example, the ADV_IND packet contains a payload of 23 bytes, out of which Advertiser Address is 6 bytes and AdvData is 17 bytes.

The AdvData contains two AD structures:

Table 14.1 Flags Contained in AdvData.

<i>Value</i>	<i>Information</i>
0x01	LE Limited Discoverable Mode
0x02	LE General Discoverable Mode
0x04	BR/EDR Not Supported
0x08	Simultaneous LE and BR/EDR to the same device capable (controller)
0x10	Simultaneous LE and BR/EDR to the same device capable (host)

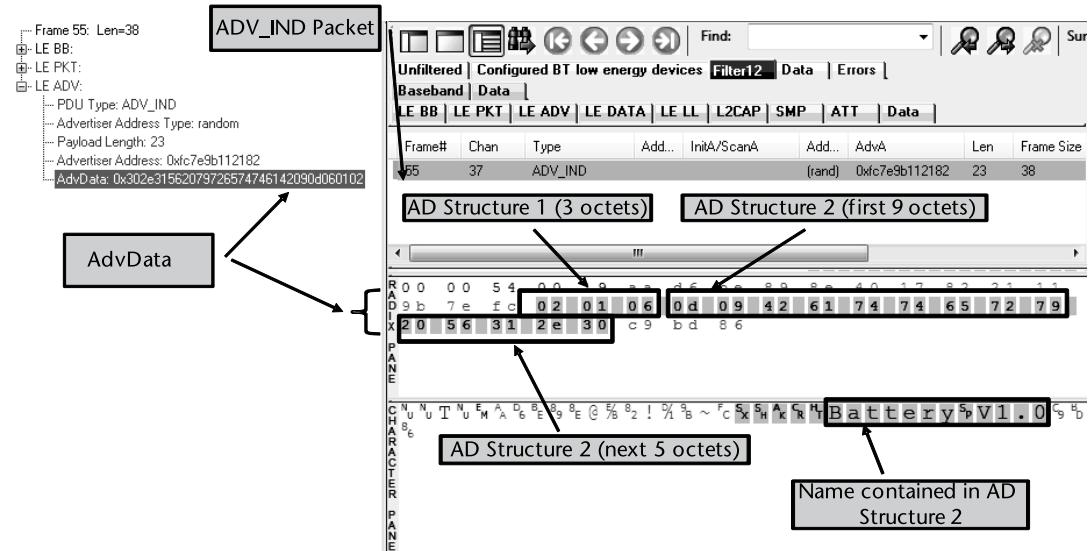


Figure 14.6 Example of AdvData contained in ADV_IND packets.

- AD Structure 1: 3 octets: It contains [02 01 06]. This is decoded as follows:
 - Length = 02.
 - AD Type = 01 (Flags).
 - AD Data = 06. This means:
 - The device is currently in LE Generate Discoverable Mode.
 - The device does not support BR/EDR.
- AD Structure 2: 14 octets. It contains [0d 09 42 61 74 74 65 72 79 20 56 31 2e 30]. This is decoded as follows:
 - Length = 0d.
 - AD Type = 09 (complete local name).
 - AD Data = 42 61 74 74 65 72 79 20 56 31 2e 30.
 - When converted to ascii characters, this means “Battery V1.0.”

14.5 GAP Characteristics

GAP defines certain characteristics to provide further information about the device using the GAP service. There can be only one instance of the GAP service on a device. The Service UUID of the GAP service is <>Generic Access Profile>>. Some of the commonly used GAP characteristics are explained below.

14.5.1 Device Name Characteristic

The Device Name Characteristic contains the name of the device as a UTF-8 encoded string. It can be 0 to 248 octets long. It has the following fields:

- Attribute Type: 0x2A00 – UUID for <>Device Name>>.
- Attribute Value: Device Name: 0 to 248 octets in length.

The air log for ATT Read Response for reading the Device Name Characteristic is shown in Figure 14.7. As shown in the figure, the Master executes the following steps to read the device name:

1. Frame #283: The Master sends a Read Request to read the Characteristic for Attribute Handle 2. (UUID = Characteristic).
2. Frame #286: The Slave sends a Read Response indicating that the Characteristic contains Device Name and the Attribute Handle is 3.
3. Frame #289: The Master sends a Read Request to read the Device Name (UUID = Device Name).
4. Frame #292: The Slave sends a Read Response which contains the Device Name.

14.5.2 Appearance Characteristic

The appearance characteristic is used by the discovering devices to associate an icon (or string) to this device. The various values of the appearance characteristic are defined in the Bluetooth Assigned Numbers document. This can be compared to the Class of Device (CoD) in the case of BR/EDR. The remote devices use the CoD parameter to display an icon for the device (for example, icon of a headset, mobile phone or printer). In the case of LE, the remote devices could fetch the appearance characteristic to find the type of the device and then display an icon or any other user interface that is appropriate for that device. It has the following fields:

- Attribute Type: 0x2A01 – UUID for <>Appearance>>
- Attribute Value: Appearance (as per values defined in Bluetooth Assigned Numbers): 2 octets in length.

The air log for ATT Read Response for reading the Appearance Characteristic is shown in Figure 14.8. As shown in the figure, the Master executes the following steps to read the Appearance:

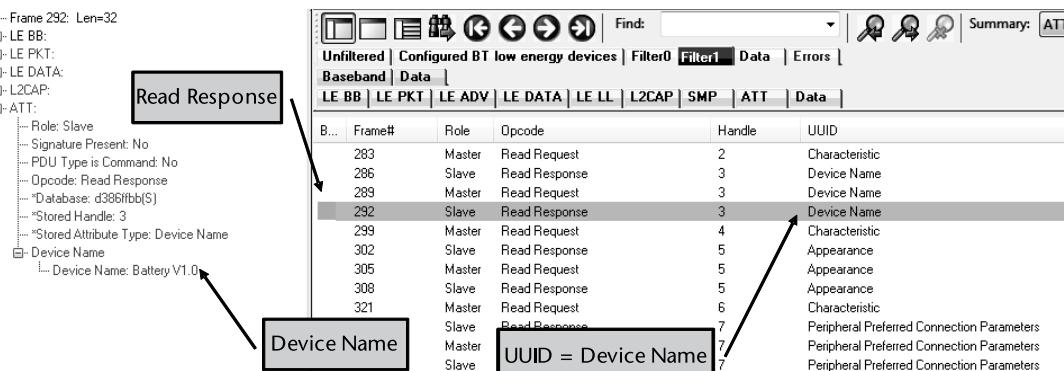


Figure 14.7 Device Name Characteristic.

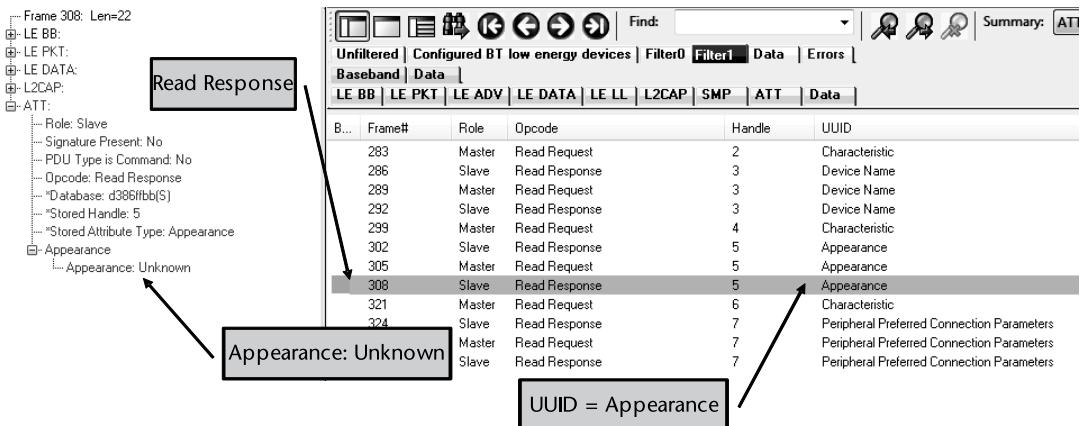


Figure 14.8 Appearance Characteristic.

1. Frame #299: The Master sends a Read Request to read the Characteristic for Attribute Handle 4. (UUID = Characteristic).
2. Frame #302: The Slave sends a Read Response indicating that the Characteristic contains Appearance and the Attribute Handle is 5.
3. Frame #305: The Master sends a Read Request to read the Appearance (UUID = Appearance).
4. Frame #308: The Slave responds with the Appearance characteristic. It contains the value “Unknown Appearance”.

14.5.3 Peripheral Privacy Flag Characteristic

The Peripheral Privacy Flag Characteristic defines whether privacy is currently in use within this device or not.

Setting this flag to 1 indicates that privacy is enabled in this device. It has the following fields:

- Attribute Type: 0x2A02 – UUID for <>Peripheral Privacy Flag>>
- Attribute Value: Peripheral Privacy Flag: 1 octet in length
 - 0x00: Indicates that privacy is disabled in this device.
 - 0x01: Indicates that privacy is enabled in this device.
 - Remaining values are reserved.

The Peripheral may implement this characteristic as only readable or readable/writable. If this characteristic is writable, then the central device can remotely enable or disable privacy on this device.

14.5.4 Reconnection Address Characteristic

The Reconnection Address is used to store the address to connect to next time. It may be exchanged between the two devices at each connection. If the device changes its address, then it can provide the new reconnection address to the peer

device so that the peer device knows which address to connect to the next time. It has the following fields:

- Attribute Type: 0x2A03 – UUID for <<Reconnection Address>>
- Attribute Value: Reconnection Address.

14.5.5 Peripheral Preferred Connection Parameters Characteristic

The peripheral preferred connection parameters (PPCP) characteristic contains the various preferred connection parameters of a peripheral. This includes parameters like minimum and maximum connection interval, Slave latency, and connection supervision timeout multiplier. It has the following fields:

- Attribute Type: 0x2A04 – UUID for <<Peripheral Preferred Connection Parameters>>
- Attribute Value: Peripheral Preferred Connection Parameter: 8 octets in length.

The air log for ATT Read Response for reading the PPCP Characteristic is shown in Figure 14.9. As shown in the figure, the Master executes the following steps to read the PPCP:

1. Frame #321: The Master sends a Read Request to read the Characteristic for Attribute Handle 6. (UUID = Characteristic).
2. Frame #324: The Slave sends a Read Response indicating that the Characteristic contains PPCP and the Attribute Handle is 7.
3. Frame #325: The Master sends a Read Request to read the Appearance (UUID = PPCP).
4. Frame #328: The Slave responds with the PPCP characteristic. It contains the 8-octets: “ff ff ff ff 00 00 ff ff”. The meaning of these octets is shown in Table 14.2.

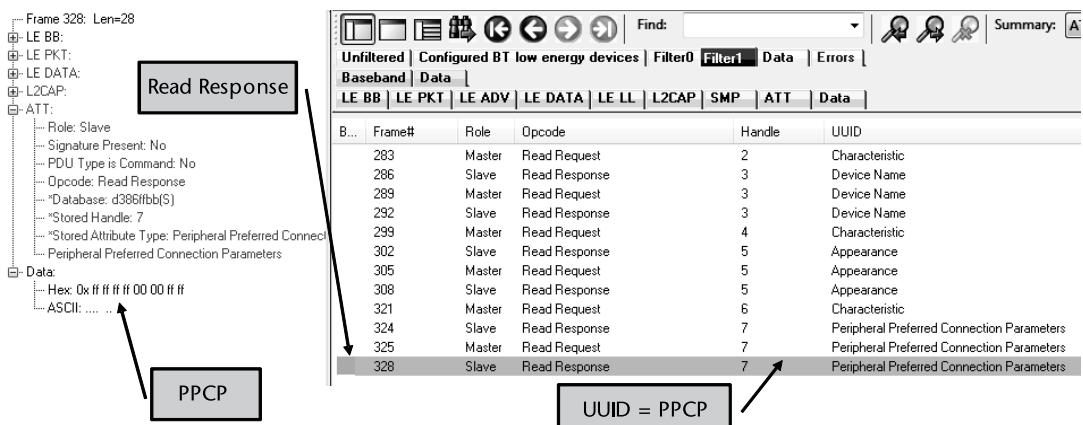


Figure 14.9 Peripheral Preferred Connection Parameters Characteristic.

Table 14.2 Peripheral Preferred Connection Parameters Data

Name	Size	Value	Meaning
Minimum Connection Interval	2	ff ff	No specific minimum
Maximum Connection Interval	2	ff ff	No specific maximum
Slave Latency	2	00 00	Slave latency of the connection in terms of number of connection events
Connection Supervision Timeout Multiplier	2	ff ff	No specific value requested

14.6 Operational Modes and Procedures

GAP defines the various modes in which an LE device can be in as well as the procedures that can be carried out. It broadly defines the following four modes and procedures:

1. Broadcast mode and observation procedure.
2. Discovery modes and procedures.
3. Connection modes and procedures.
4. Bonding modes and procedures.

14.6.1 Broadcast Mode and Observation Procedure

The broadcast mode is the simplest mode which allows an LE device to send data to other LE devices using advertising events. The device that sends the data is termed as broadcaster and the device that receives the data is termed as observer.

14.6.1.1 Broadcast Mode

The broadcaster sends data in either nonconnectable undirected (ADV_NONCONN_IND) or scannable undirected advertising events (ADV_SCAN_IND). These events were explained in Chapter 8. Both these events permit sending data from the Advertiser to the Scanner but do not permit the Scanner to connect to the Advertiser. Since there is no acknowledgement from any device, the data sent in broadcast mode is considered unreliable.

14.6.1.2 Observation Procedure

The observation procedure is used by the observer to receive data from the broadcaster. The device can use either passive scanning or active scanning to receive the advertising events. As explained in Chapter 8, in the passive scanning mode the device just receives the advertising data while in the active scanning mode the device may also request some additional information from the Advertiser using SCAN_REQ.

14.6.2 Discovery Modes and Procedures

The discovery modes and procedures are used to discover other devices in the vicinity. There are three discovery modes and three discovery procedures in this category. The discovery modes are implemented by the device acting in the Peripheral role. (These are excluded for device acting in the Central role since the Central device does the discovery and is not the one that is to be discovered) The three discovery modes are shown in Table 14.3.

The discovery procedures are implemented by the device to discover the devices in the vicinity. Out of these the first two are implemented by the device acting in the Central role while the third can be implemented by both the Central and Peripheral device. The three discovery procedures are shown in Table 14.4.

14.6.2.1 Nondiscoverable Mode

A device in nondiscoverable mode cannot be discovered by other devices. In this mode either the device does not send any advertising packets or if it sends advertising packets then it does not set the LE General Discoverable Mode and LE Limited Discoverable Mode flags in the AD type.

14.6.2.2 Limited Discoverable Mode

A device configured in limited discoverable mode can be discovered by other devices for only a limited period of time. An example of this could be a device which sends advertisements for a limited period of time when the user presses a button on it. A device in limited discoverable mode can send three types of events:

- Nonconnectable advertising events (ADV_NONCONN_IND).
- Scannable undirected advertising events (ADV_SCAN_IND).
- Connectable undirected advertising events (ADV_IND).

Table 14.3 Discovery Modes

S. No.	Mode	Mandatory/Optional
1	Nondiscoverable Mode	Mandatory for Peripheral. Excluded for Central.
2	Limited Discoverable Mode	Optional for Peripheral. Excluded for Central.
3	General Discoverable Mode	Mandatory if Limited Discoverable mode is not implemented, else optional. Excluded for Central.

Table 14.4 Discovery Procedures

S. No.	Procedure	Mandatory/Optional
1	Limited Discovery Procedure	Optional for Central. Excluded for Peripheral
2	General Discovery Procedure	Mandatory for Central. Excluded for Peripheral
3	Name Discovery Procedure	Optional for both Peripheral and Central. Can be invoked by either of these.

A device in limited discoverable mode sets the LE Limited Discoverable Mode flag to one and LE General Discoverable Mode flag to zero in the AD Type. The Limited Discoverable Mode is shown in Figure 14.10.

14.6.2.3 Limited Discovery Procedure

A device that is performing limited discovery procedure receives the data from the devices in limited discoverable mode. The host receives the advertising packets from all the devices and checks the AD Type field in the advertisement packets. If the AD Type field is present and has the LE Limited Discoverable flag set to one then it adds that device to the list of devices found during the limited discovery procedure.

Besides the flag, the advertising data may contain other AD Types like local name, service UUIDs, etc. This data may also be used by the host for other procedures. The Limited Discovery Procedure is shown in Figure 14.10.

14.6.2.4 General Discoverable Mode

A device configured in general discoverable mode can be discovered by other devices that are performing the general discovery procedure. This is typically used when the device wishes to remain in the discoverable mode for a long period of time. A device in general discoverable mode can send three types of events:

- Nonconnectable advertising events (ADV_NONCONN_IND).
- Scannable undirected advertising events (ADV_SCAN_IND).
- Connectable undirected advertising events (ADV_IND).

A device in general discoverable mode sets the LE Limited Discoverable Mode flag to zero and LE General Discoverable Mode flag to one in the AD Type. The General Discoverable Mode is shown in Figure 14.11.

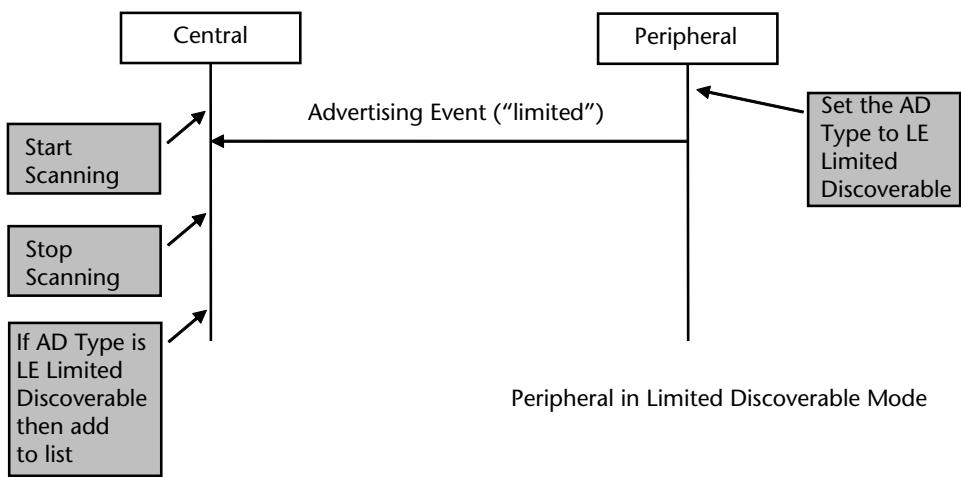


Figure 14.10 Limited Discoverable Mode and Limited Discovery Procedure.

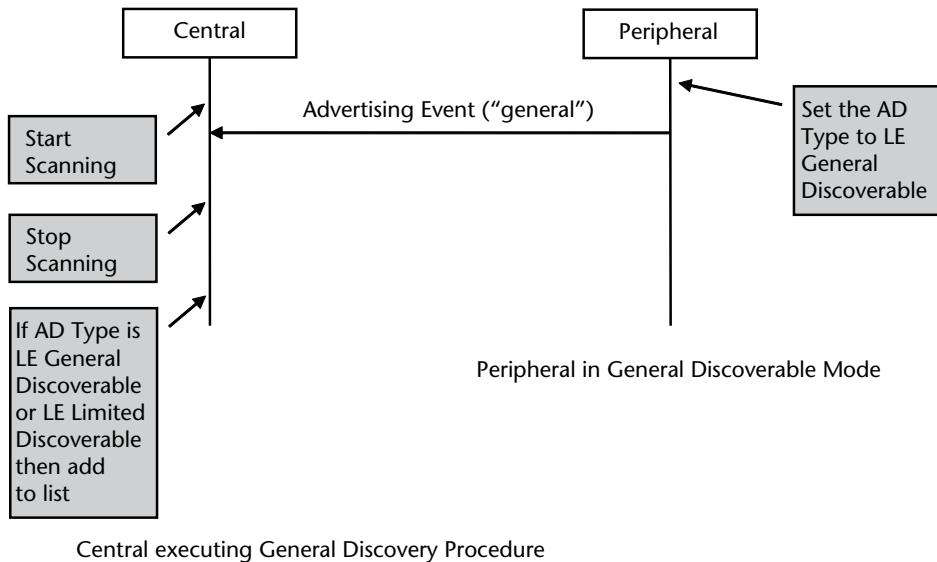


Figure 14.11 General Discoverable Mode and General Discovery Procedure.

14.6.2.5 General Discovery Procedure

A device that is performing general discovery procedure receives the data from the devices in general discoverable mode or limited discoverable mode. The host receives the advertising packets from all the devices and checks the AD Type field in the advertisement packets. If the AD Type field is present and has either the LE Limited Discoverable flag or LE General Discoverable flag set to one then it adds that device to the list of devices found during the general discovery procedure.

Besides the flag, the advertising data may contain other AD Types like local name, service UUIDs etc. This data may also be used by the host for other procedures. The General Discovery Procedure is shown in Figure 14.11.

14.6.2.6 Name Discovery Procedure

The name discovery procedure is used to obtain the Bluetooth Device Name of a remote device. It can be invoked by either the device in Central role or the device in Peripheral role. The Bluetooth name of the LE device is present at two places:

1. In the Advertising Event if the AD Type in the AD structure is set to Local Name.
 - This name is acquired during the general discovery procedure or limited discovery procedure.
 - It is possible that the full name may not be provided in the advertising events since the advertising events are limited to 31 octets. In such cases a shortened name may be provided in the advertising event.
2. Device Name Characteristic.

If the complete device name is not acquired using step (1) above, then the name discovery procedure may be performed.

The name discovery procedure uses the following steps:

1. Establish a connection using the connection establishment procedures.
2. Read the Device Name Characteristic using the GATT procedure Read Using Characteristic UUID.
3. Terminate the connection if it's not needed any longer.

The Name Discovery Procedure is shown in Figure 14.12

14.6.3 Connection Modes and Procedures

The connection modes and procedures are used to establish a connection. There are three connection modes and six connection procedures in this category. The connection modes are entered by the device in the Peripheral role and the discovery procedures are initiated by the device in Central role. The terminate connection

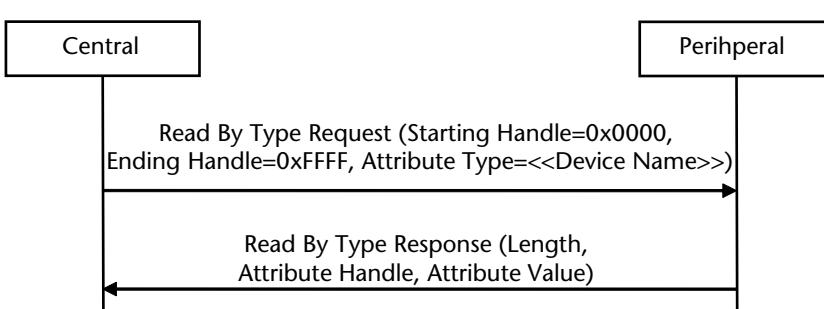
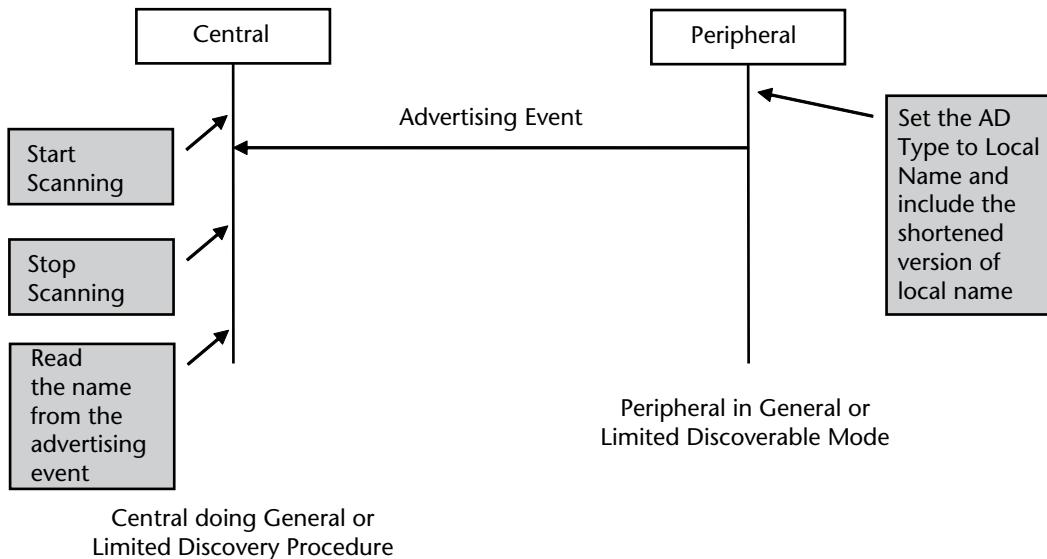


Figure 14.12 Name Discovery Procedure.

procedure and update connection parameter procedure can be initiated by both the Peripheral and Central. The three connection modes are shown in Table 14.5.

The six discovery procedures are shown in Table 14.6.

14.6.3.1 Nonconnectable Mode

In the Nonconnectable Mode, the device does not allow connection establishment. This mode is entered by the Peripheral device when it does not intend to allow the other devices to connect to it.

The Peripheral can send following two types of events in this mode:

1. Nonconnectable undirected advertising events.
2. Scannable undirected advertising events.

As explained in Chapter 8, both these events don't allow the remote device to establish a connection. The Scannable undirected advertising event allows the remote device to get additional information from the Peripheral.

14.6.3.2 Directed Connectable Mode

In the Directed Connectable Mode, the device allows connections from only a specified peer device. In this mode the Peripheral devices uses the directed connectable advertising events (ADV_DIRECT_IND). It specifies the address of the peer device that is allowed to connect to it in the directed connectable advertising event.

The peer device which is acting as a Central can establish a connection with it. After a connection is established, the Central device becomes the Master and the Peripheral device becomes the Slave. Once the connection is established, the device exits the Directed Connectable Mode and enters the Nonconnectable Mode. This mode is shown in Figure 14.13.

Table 14.5 Connection Modes (All Modes Excluded for Broadcaster and Observer except Nonconnectable Mode)

S. No.	Mode	Mandatory/Optional
1	Nonconnectable Mode	Mandatory for Peripheral, Broadcaster and Observer. Excluded for Central.
2	Directed Connectable Mode	Optional for Peripheral. Excluded for all other roles.
3	Undirected Connectable Mode	Mandatory for Peripheral. Excluded for all other roles.

Table 14.6 Discovery Procedures (All Procedures Excluded for Broadcaster and Observer)

S. No.	Procedure	Mandatory/Optional
1	Auto Connection Establishment Procedure	Optional for Central. Excluded for Peripheral
2	Direct Connection Establishment Procedure	Mandatory for Central. Excluded for Peripheral
3	General Connection Establishment Procedure	Mandatory for Central if Privacy feature is supported, else optional. Excluded for Peripheral
4	Selective Connection Establishment Procedure	Optional for Central. Excluded for Peripheral
5	Connection Parameter Update Procedure	Mandatory for Central. Optional for Peripheral
6	Terminate Connection Procedure	Mandatory for Central and Peripheral

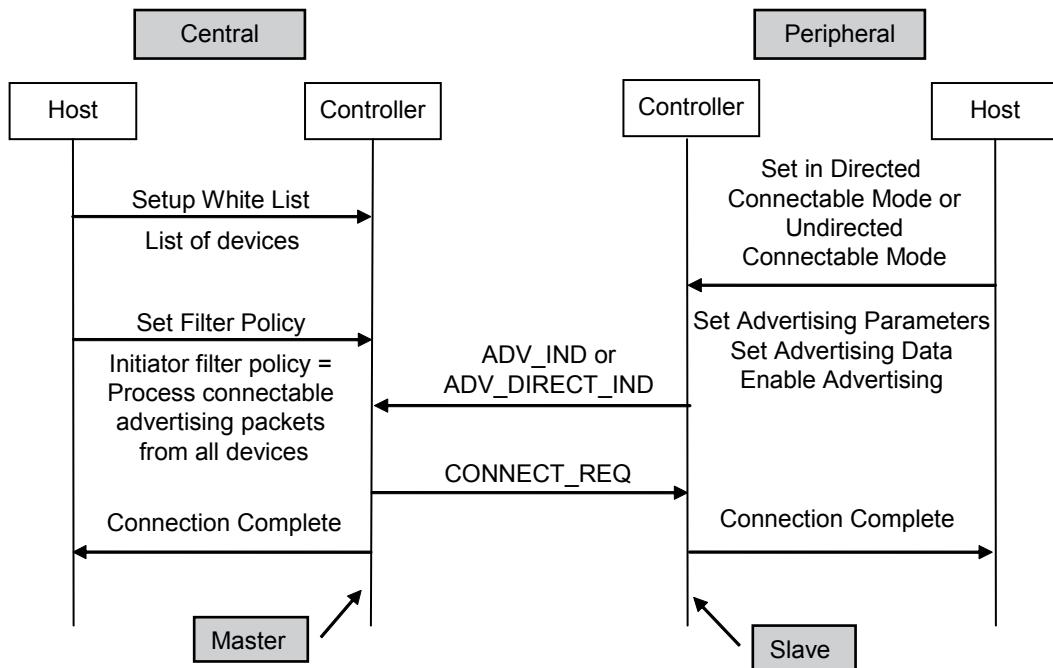


Figure 14.13 Directed Connectable Mode, Undirected Connectable Mode, and Auto Connection Establishment Procedure.

14.6.3.3 Undirected Connectable Mode

In the Undirected Connectable Mode, the device allows connections from any peer device. In this mode the Peripheral devices uses the undirected connectable advertising events (ADV_IND).

The peer device which is acting as a Central can establish a connection with it. After a connection is established, the Central device becomes the Master and the Peripheral device becomes the Slave. Once the connection is established, the device exits the Undirected Connectable Mode and enters the Nonconnectable Mode. This mode is shown in Figure 14.13.

14.6.3.4 Auto Connection Establishment Procedure

In Auto Connection Establishment Procedure the host of the Central configures the controller to autonomously establish a connection with a device in the directed connectable mode or undirected connectable mode.

The list of devices to which the controller can establish a connection autonomously is specified using a White List. The initiator White List is used for this purpose. If the peer device address matches one of the addresses stored in the White List, then the controller establishes a connection autonomously. (White List was explained in Chapter 8). This is shown in Figure 14.13.

14.6.3.5 Direct Connection Establishment Procedure

In Direct Connection Establishment Procedure the host of the Central establishes a connection to a particular peer device. White Lists are not used in this procedure.

The host simply specifies the address of the peer device to connect to when initiating the connection establishment. This is shown in Figure 14.14.

14.6.3.6 General Connection Establishment Procedure

In General Connection Establishment Procedure the host of the Central configures the controller to connect to a particular device that may be in either directed connectable mode or undirected connectable mode.

In this procedure, the host of the Central broadly performs the following steps:

1. The host first scans for advertising packets to find out the list of devices that are present in the vicinity and can be connected to.
2. After getting the list of devices that are present, the host selects one of the devices to which the connection is to be made (for example, by displaying a UI to the user to select the device to connect to).
3. The host then uses the direct connection establishment procedure to establish a connection to the selected device. (The direct connection establishment procedure was explained earlier).

White Lists are not used in this procedure since the host selects the device to connect to instead of giving a list of device to connect to autonomously. This is shown in Figure 14.15.

14.6.3.7 Selective Connection Establishment Procedure

In Selective Connection Establishment Procedure the host of the Central configures the controller to connect to one of the devices in a set of devices specified in a White List.

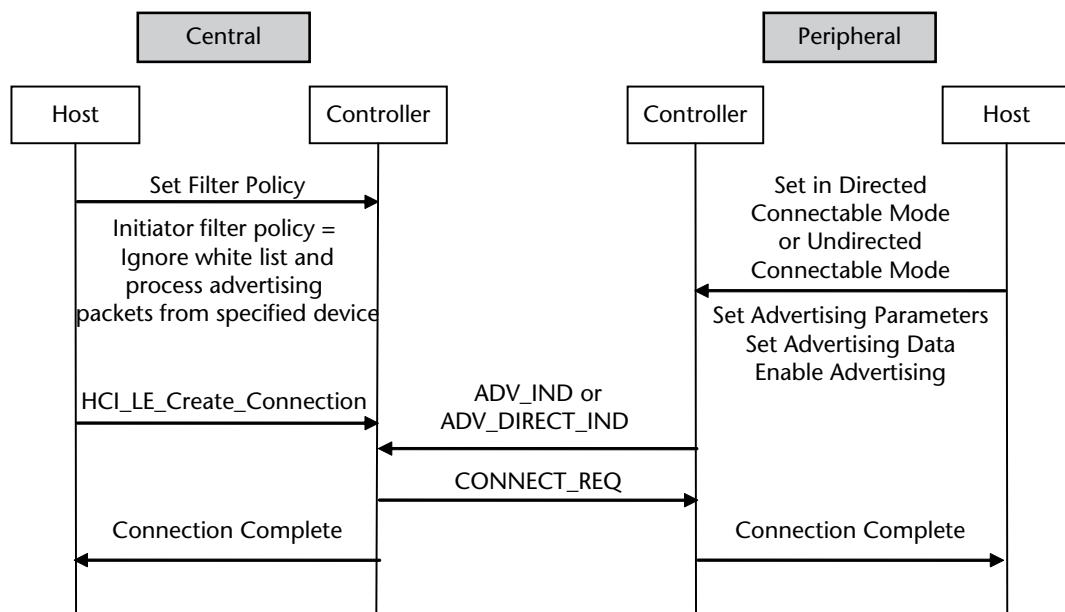


Figure 14.14 Direct Connection Establishment Procedure.

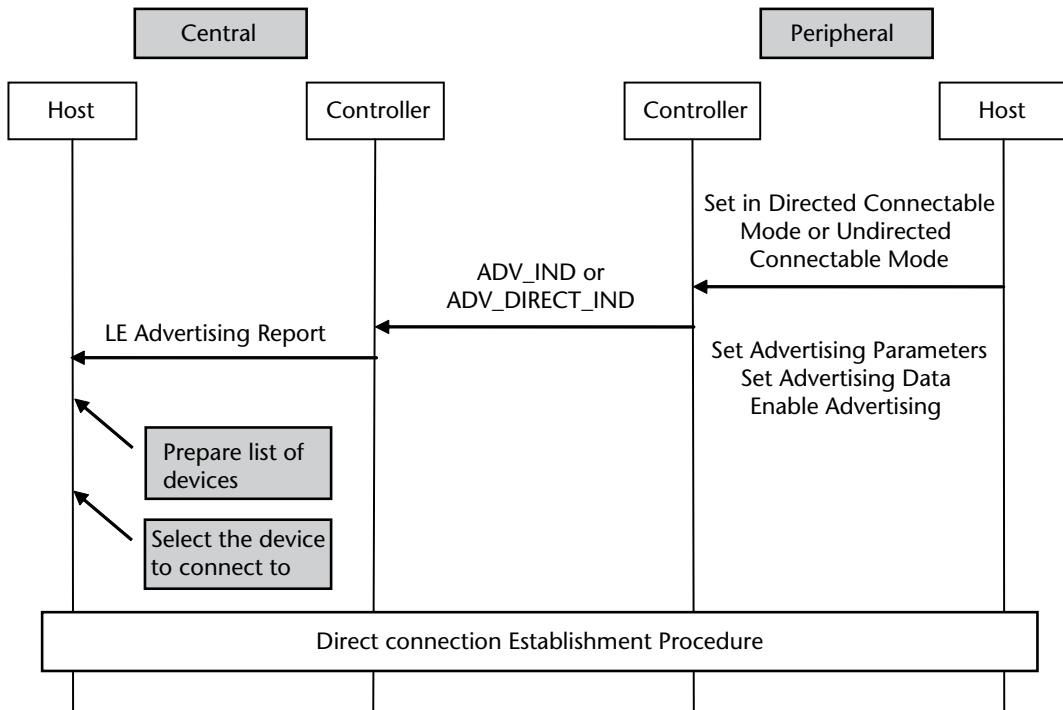


Figure 14.15 General Connection Establishment Procedure.

In this procedure, the host of the Central broadly performs the following steps:

1. The host writes the list of devices that it may connect to into the White List.
2. The host sets the scanner filter policy so that it gets the advertising packets from only the devices in the white list.
3. The host starts scanning.
4. When one of the devices in the white list is discovered, the host stops scanning and initiates a connection to the discovered device using direct connection establishment procedure. (The direct connection establishment procedure was explained earlier).

This is shown in Figure 14.16.

14.6.3.8 Connection Parameter Update Procedure

The Connection Parameter Update Procedure is used by either the Peripheral or Central to update the link layer parameters of the current connection.

There are two possible scenarios:

1. Connection Parameter Update initiated by the Central. In this scenario the link layer connection update procedure is used.
2. Connection Parameter Update initiated by the Peripheral. In this scenario the L2CAP connection parameter update procedure is used.

Both the scenarios are shown in Figure 14.17.

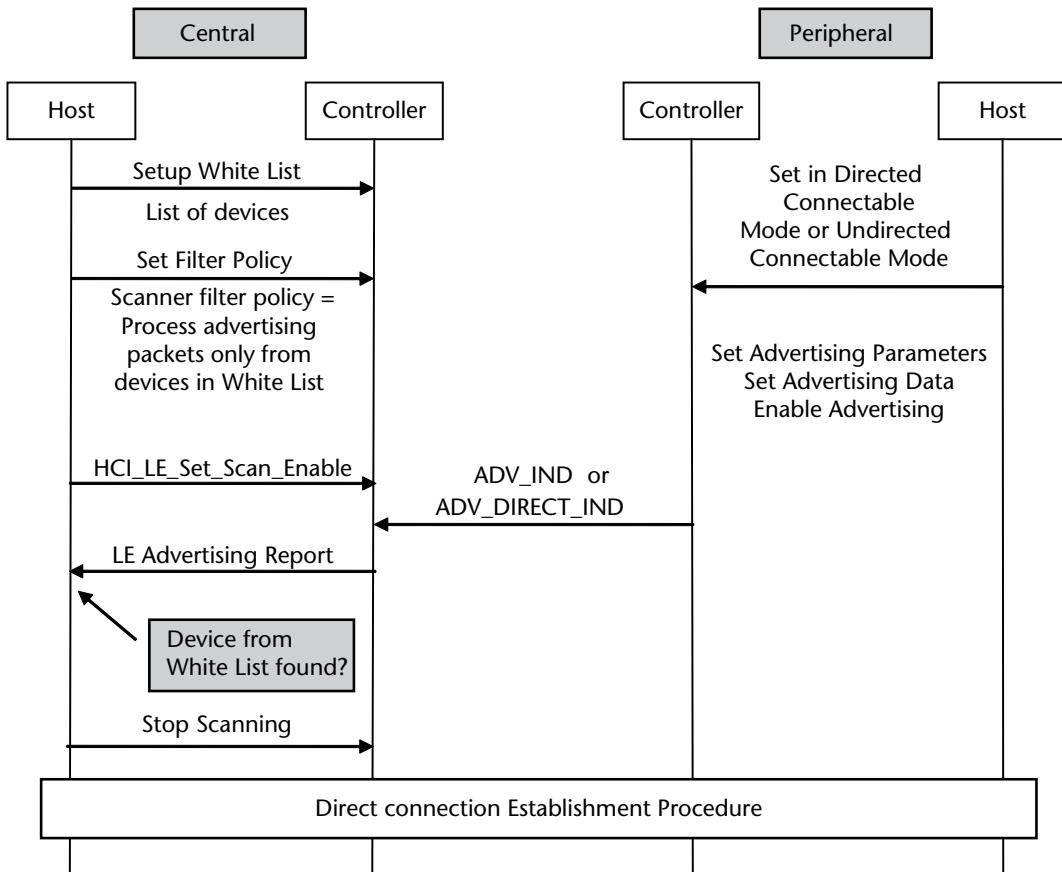


Figure 14.16 Selective Connection Establishment Procedure.

14.6.3.9 Terminate Connection Procedure

The Terminate Connection Procedure is used by either the Peripheral or Central to terminate the current connection. This procedure is shown in Figure 14.18.

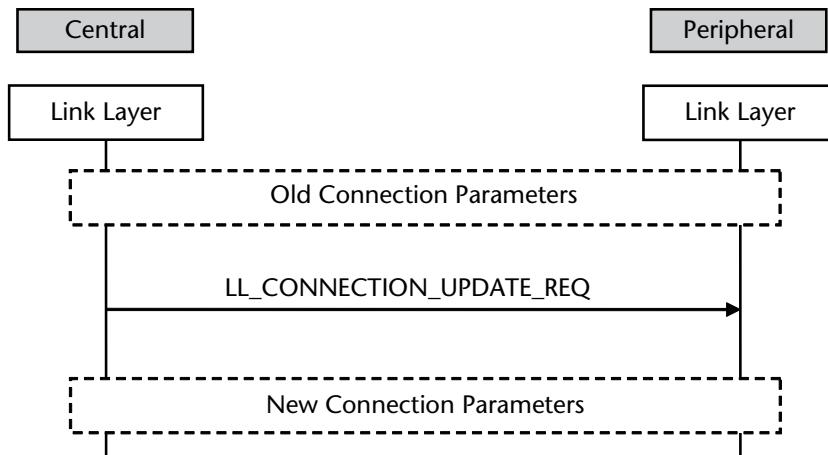
14.6.4 Bonding Modes and Procedures

The bonding procedure allows two devices to create a trusted relationship between them. There are two bondable modes and one bonding procedure in this category. The bondable modes are implemented by both the device acting in Central role and Peripheral role. The two bondable modes are shown in Table 14.7.

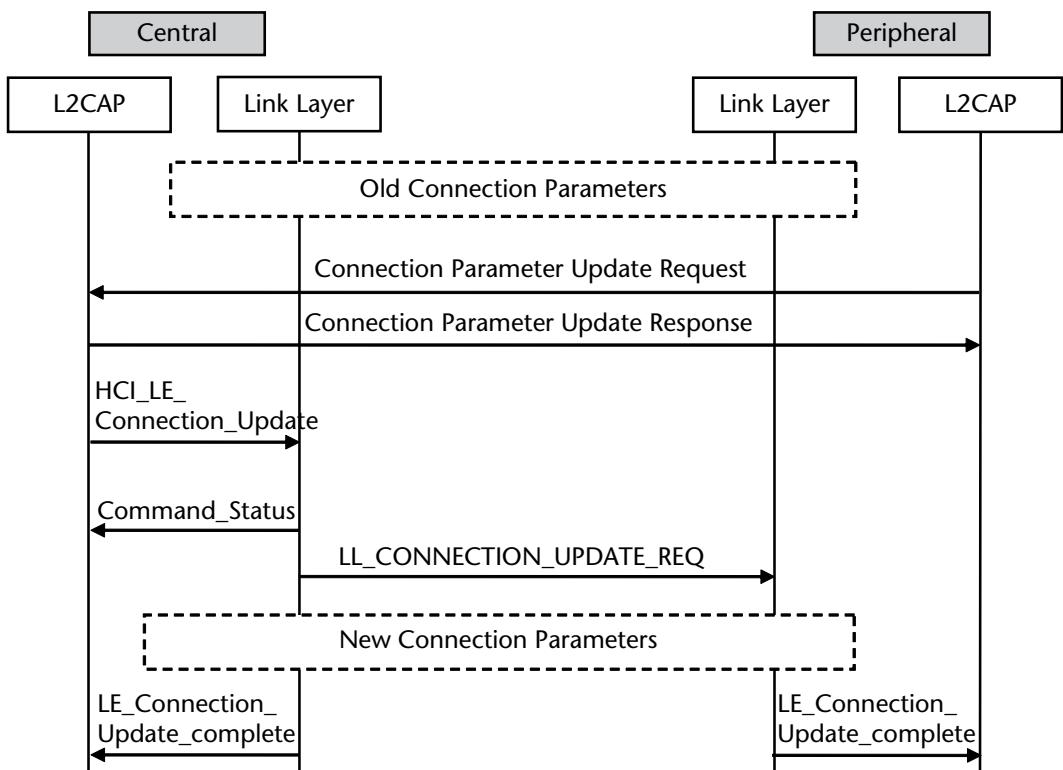
The bonding procedure is implemented by both the device acting in Central role and Peripheral role. The one bonding procedure is shown in Table 14.8.

14.6.4.1 Nonbondable Mode

In this mode, a device does not allow a bond to be created with a peer device.



Scenario 1: Connection Parameter Update Procedure Initiated by Central



Scenario 2: Connection Parameter Update Procedure Initiated by Peripheral

Figure 14.17 Connection Parameter Update Procedure.

14.6.4.2 Bondable Mode

In this mode, a device allows a bond to be created with a peer device. This is shown in Figure 14.19.

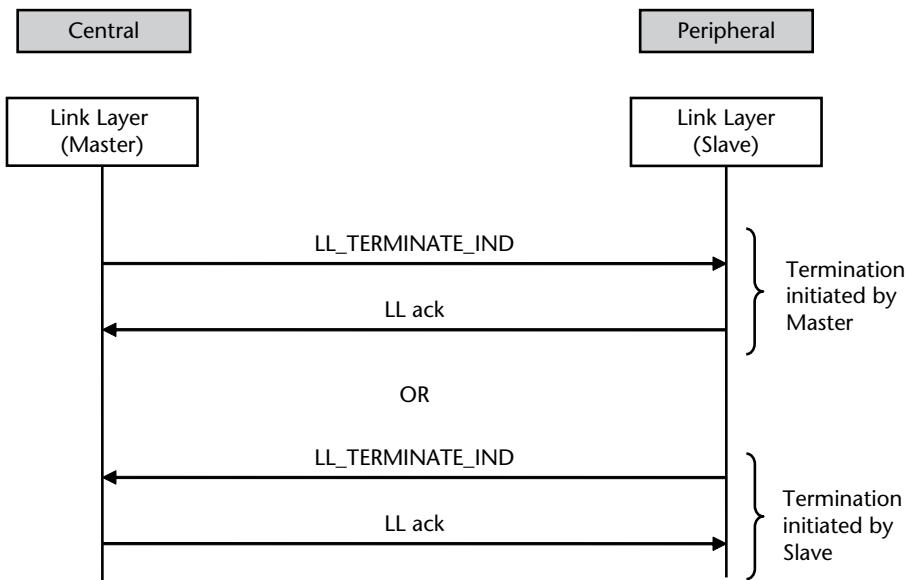


Figure 14.18 Terminate Connection Procedure.

14.6.4.3 Bonding Procedure

The Bonding Procedure is performed, for example, when a device needs to access a service on a peer device that requires bonding. It is initiated by the Security Manager of the Central. If the peer device is in Bondable Mode, the two devices exchange and store the bonding information in the security database. This is shown in Figure 14.19.

14.7 Security

Implementation of security is optional for LE devices. It is implemented if the access to any of the services is to be protected.

Once the LE connection is done, the Security Manager of each device will specify the security mode to be used for further transactions. A device may impose

Table 14.7 Bonding Modes

S. No.	Mode	Mandatory/Optional
1	Non-Bondable Mode	Mandatory for Peripheral and Central
2	Bondable Mode	Optional for Peripheral and Central

Table 14.8 Bonding Procedure

S. No.	Procedure	Mandatory/Optional
1	Bonding Procedure	Optional for Peripheral and Central

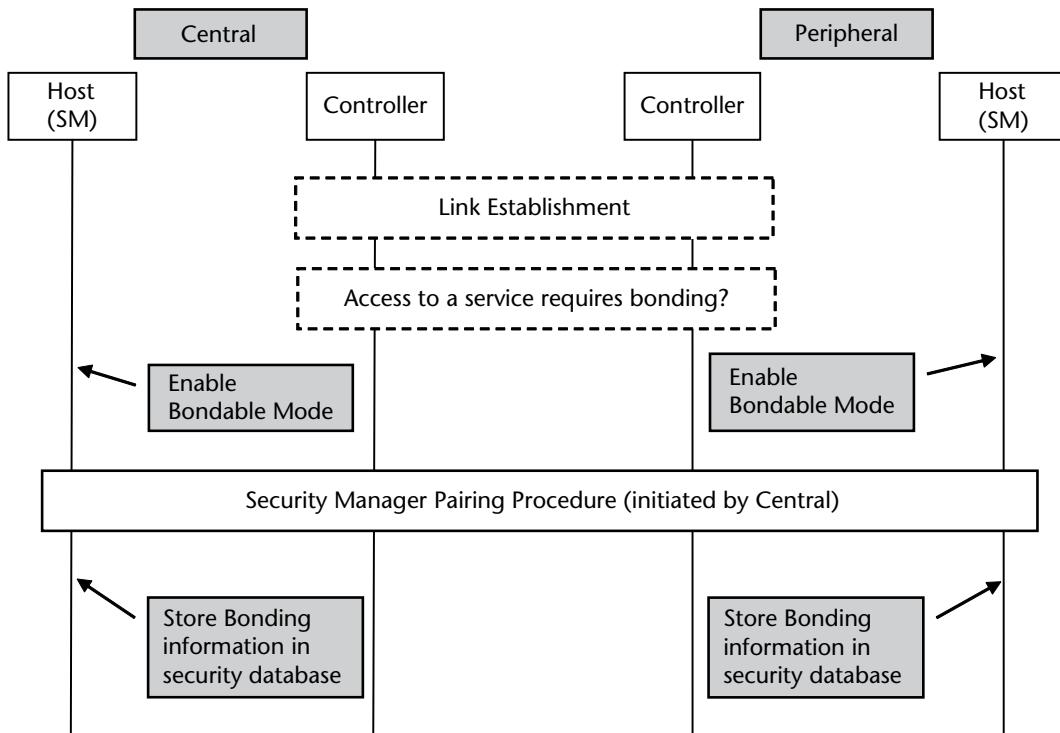


Figure 14.19 Bonding Procedure.

security requirements either at a device level or at a service level. In general, if an application requires security, it will specify the requirements to the Security Manager so that the Security Manager can enforce the correct level of security.

The LE specification defines two modes and four procedures with respect to security.

The two security modes are shown in Table 14.9.

The four security procedures are shown in Table 14.10.

14.7.1 LE Security Mode 1

LE Security Mode 1 has three security levels with increasing security levels:

1. Level 1: No security (No authentication and no encryption).
2. Level 2: Unauthenticated pairing with encryption.
3. Level 3: Authenticated pairing with encryption.

A device with higher level of security also satisfied the security requirements for lower levels. So a device in Level 2 satisfies the requirements for Level 1 and a device in Level 3 satisfies the requirements for Level 2 and Level 1.

Table 14.9 Security Modes (All Modes are Excluded for Broadcaster and Observer)

S. No.	Mode	Mandatory/Optional
1	LE Security Mode 1	Optional for Peripheral and Central
2	LE Security Mode 2	Optional for Peripheral and Central

Table 14.10 Security Procedures (All Procedures are Excluded for Broadcaster and Observer)

S. No.	Mode	Mandatory/Optional
1	Authentication Procedure	Optional for Peripheral and Central
2	Authorization Procedure	Optional for Peripheral and Central
3	Connection Data Signing	Optional for Peripheral and Central
4	Authenticate Signed Data Procedure	Optional for Peripheral and Central

14.7.2 LE Security Mode 2

LE Security Mode is used for connection based data signing. It has two security levels:

1. Level 1: Unauthenticated pairing with data signing.
2. Level 2: Authenticated pairing with data signing.

Data signing is explained later in this chapter.

14.7.3 Authentication Procedure

The authentication procedure is initiated after a connection has been established. It covers both LE Security Mode 1 and LE Security Mode 2.

There are two types of pairing:

1. Authenticated pairing: This requires pairing performed with authentication set to “MITM protection”.
2. Unauthenticated pairing: This requires pairing performed with authentication set to “No MITM protection”.

These were explained in Chapter 11.

The authentication procedure describes the step to be performed in two scenarios:

1. Receiving a service request: Each service on the server side has security settings associated with it. Before replying to a service request, the server checks to see if the current level of security is sufficient to allow access to that particular service. If it is not, the server replies with the appropriate error code.
2. Initiating a service request: While initiating a service request, the client compares the level of security required to access a service and the current level of security. If the current level of security is insufficient, it initiates the appropriate procedures like pairing or encryption.

14.7.4 Authorization Procedure

Authorization procedure involves getting a confirmation from the user to continue with a security procedure. This may be done after a successful authentication.

14.7.5 Encryption Procedure

A connection may be encrypted by either the Central (Master) or the Peripheral (Slave). This involves invoking the appropriate Security Manager procedures from the Master or Slave side:

1. Initiation from the Slave side: The Slave device may initiate security by sending a Security Request command to the Master. This was explained in Chapter 11.
2. Initiation from the Master side: The Master device may initiate the setup of encrypted session. This was explained in Chapter 11.

14.7.6 Data Signing

Data signing is used in LE Security Mode 2 when there is a requirement to send authenticated data between two devices on an unencrypted connection. The data in the data PDU is signed and the signature is appended to the data PDU itself using the Connection Data Signing Procedure. On the peer device, the signature is verified using the Authenticate Signed Data procedure. This is shown in Figure 14.20.

14.7.7 Privacy Feature

Since many of the LE devices are supposed to be carried by people (for example, shoes, watch, heart rate sensor), tracking those devices would allow tracking a person by tracking the transmissions from these devices. This could compromise the privacy of a person.

The privacy feature is used to prevent tracking of devices over a period of time. This is done by changing the Bluetooth device address frequently. It is optional to implement this feature in LE devices.

If a Peripheral device supports the privacy feature, then it exposes two characteristics:

1. Peripheral Privacy Flag (Mandatory to support privacy): Privacy is enabled when this flag is set to 1.
2. Reconnection Address Characteristic (Optional): If this characteristic is present, then it can be used in the directed connectable mode to include the address of the Central device in the directed connectable advertising packets (ADV_DIRECT_IND). It may also allow only packets from the device whose address matches the reconnection address. This address can be stored in the White List to reduce power consumption by filtering packets from other devices.

When a Central device connects to a Peripheral which supports privacy and exposes the Reconnection Address Characteristic, then it writes a new reconnection

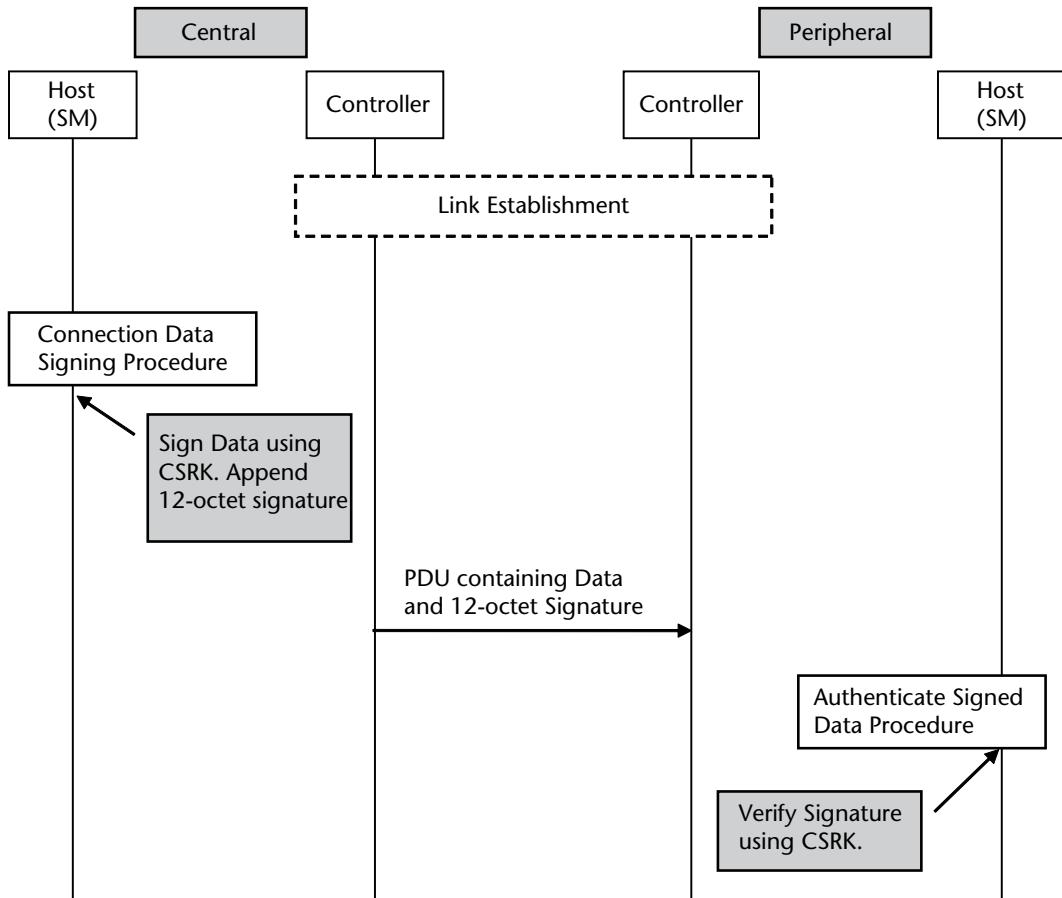


Figure 14.20 Data Signing.

address in this characteristic every time it makes a connection to it. It may write the same reconnection address to its own White List to allow packets from only this device the next time.

14.7.8 Random Device Address

The Random Address was explained in Chapter 8. It is a privacy feature of LE where the device can hide its real address and use a random address which can change over time. So the real address is not revealed at any time. This helps to ensure that a device cannot be tracked.

The Generic Access Profile defines the random address to be of two types:

1. **Static Address:** A device may choose to initialize its static address to a new value after each power cycle but cannot change it while it is still powered. If the device changes its static address, the peer device will not be able to connect to it with the old address that they may have stored.
2. **Private Address:** The private address may further be of following two types:

- a. Non-resolvable private address: The peer device can never discover the real address.
- b. Resolvable private address: The peer device can derive the real address using the random address and the link key of the connection.

The different types of device addresses for LE devices are shown in Figure 14.21.

14.8 Summary

The Generic Access Profile defines the modes that a device can be in as well as the generic procedures related to discovering devices, discovering the names of devices, connecting to devices, and security.

GAP tries to maintain cohesiveness in terms of naming conventions and modes as far as possible between BR/EDR and LE. This helps to ensure a uniform user experience when the user is connecting to either an LE device or a BR/EDR device. Of course the internal procedures behind the scenes are different between BR/EDR and LE. So even though the procedure to fetch the name from the remote device is different for BR/EDR and LE, GAP defines the same term “Bluetooth Device Name” to be used at the UI level.

This chapter completed the explanation of the LE protocol stack and mandatory profiles. The next chapter will focus on GATT-based profiles which use the services provided by GAP and GATT to support the use cases that were discussed in Chapter 1.

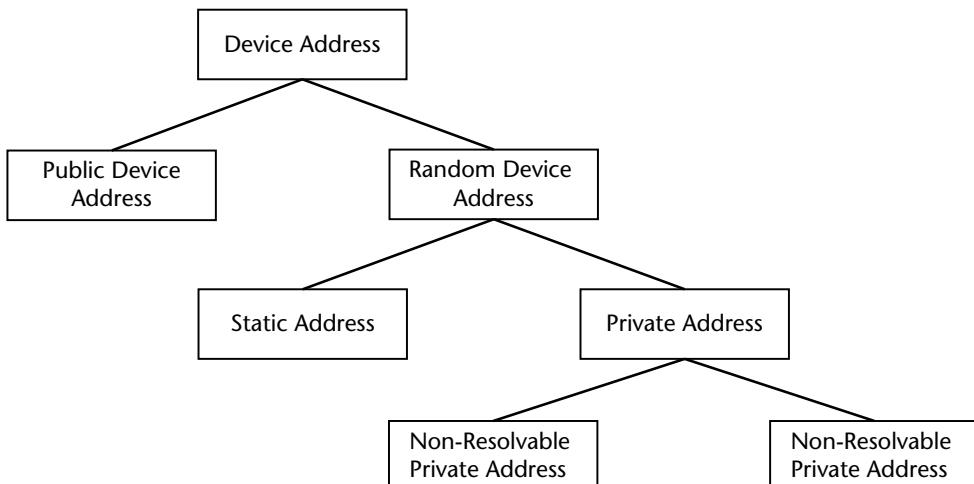


Figure 14.21 Types of Bluetooth Device Addresses for LE devices.

Bibliography

Bluetooth Core Specification 4.0 <http://www.bluetooth.org>.

Bluetooth Assigned Numbers, <https://www.bluetooth.org/assigned-numbers>.

GATT-Based Profiles

15.1 Introduction

The LE Architecture was introduced in Chapter 6. Though it looks similar to the BR/EDR architecture, the LE profiles are much simpler. A major reason for this is that LE introduced the concept of GATT-based profiles. Most of the common functionality needed by all the LE profiles is moved into the ATT protocol and GATT profile. The profiles on top of GATT use the services provided by GATT and only need to implement the minimum items needed to support that specific function. The simplicity of the GATT-based profiles and services can also be seen in the brevity of some of the specification documents (Some are only ten pages and several of the documents are around twenty pages.)

As an example, the battery profile just defines the characteristics that are exposed by a device to provide information about the battery. The generic procedures to access those characteristics are defined by the ATT protocol and GATT profile. This makes the battery profile very simple.

This chapter introduces some of the commonly used GATT-based services and profiles. The location of GATT-based profiles in the LE Protocol Stack is shown in Figure 15.1.

The Bluetooth SIG has defined a versatile list of GATT-based services and profiles to address different scenarios where LE devices can be used. There are currently more than 20 profiles already defined.

15.2 Profile, Services, and Characteristics

The GATT-based profile architecture was explained in detail in Chapter 13. LE introduces the concept of services and profiles that are defined independently of each other in separate specifications.

A service can be considered to be a data structure used to describe a particular function or feature. It is a collection of characteristics and describes what a device does. All services are implemented by a server and accessed by one or more remote devices (acting as clients).

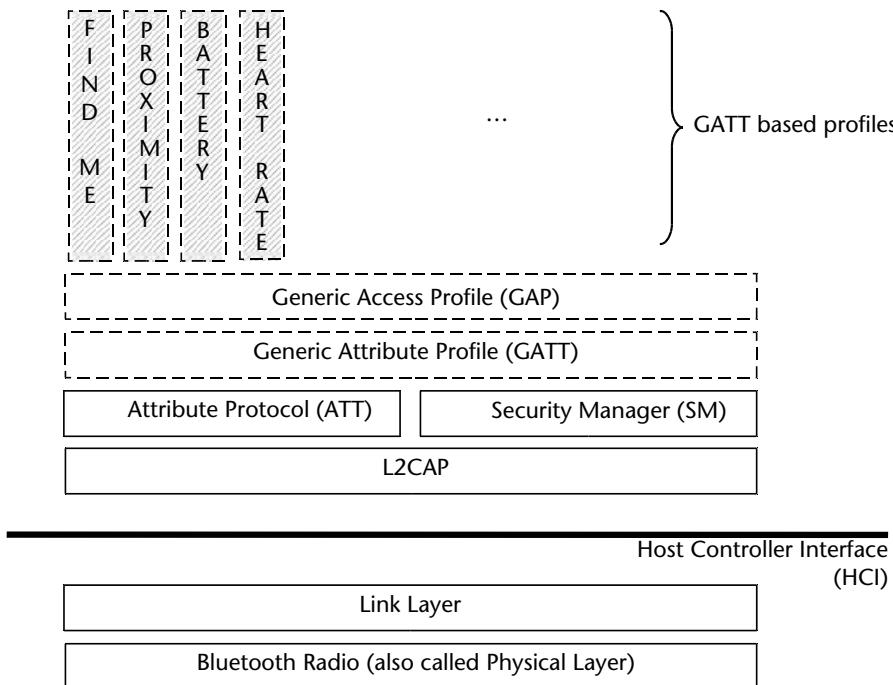


Figure 15.1 GATT-based Profiles in LE Protocol Stack.

A profile can include one or more services. It is possible for several profiles to use the same service. For example the Device Information Service may be used by several profiles to provide useful information about the device (like manufacturer name, model number, etc.). Figure 15.2 shows the relationship between Profiles, Services and Characteristics.

- A Device may support one or more profiles.
 - A Profile may support zero or more services. (It may not contain any services as well for some of the profile roles.)
 - Each service may contain one or more characteristics.
 - The characteristics are the data values that can be read, written, indicated or notified.

The most important thing to note here is that this framework allows a remote device to read or write data or register notifications or indications for that data. This data is contained in the characteristics. The LE profiles are broadly focused on providing access to some particular data. Each profile finally exposes certain data in the form of characteristics which can be accessed by remote devices. This data could be:

- The temperature that the LE temperature sensor wants to report.
- The alert level that has been set so that the device can be alerted on some specific condition.

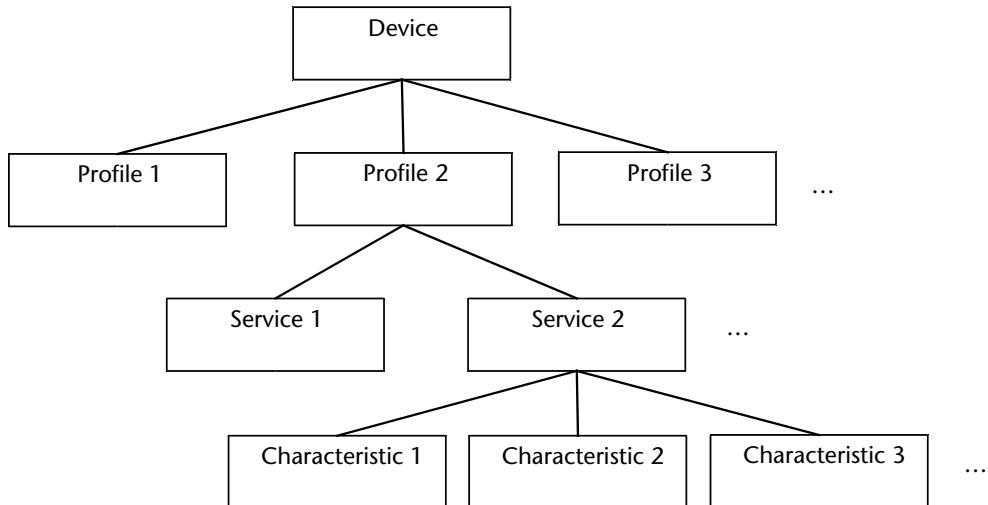


Figure 15.2 Profiles, Services and Characteristics.

- The heartbeat data.
- The status of a battery.

Note that this is in contrast to BR/EDR profiles. For example the Handsfree or A2DP profiles cannot really be considered to be data centric. They enable voice and audio to be transferred on top of a Bluetooth connection.

15.3 Immediate Alert Service (IAS)

The Immediate Alert Service is a very simple service which allows a remote device to write an alert into it. When the remote device writes an alert into it, the device may take some specific action like flashing an LED, sounding a buzzer, etc.

One example might be finding a misplaced key fob. The user may press a button on the mobile phone which would lead to an alert being generated by the key fob. The key fob could, for example, start buzzing. This could be used to locate the key fob.

15.3.1 Service Declaration

The Link Loss Service is instantiated as a <<Primary Service>>. The Service UUID is set to <<Immediate Alert>>.

15.3.2 Service Characteristics

The Immediate Alert Service exposes only one characteristic which is mandatory.

1. Alert Level (Write Without Response): The remote devices can write the alert level using the GATT Write Without Response procedure.

The Alert Level is a control point characteristic. (Note from Chapter 12 that control point characteristics are the ones which cannot be read. These attributes can only be written, notified, or indicated).

The remote device can set the Alert Level to one of the following:

1. No Alert: The device does not alert.
2. Mild Alert: The device alerts.
3. High Alert: The device alerts in the strongest possible way.

On writing a Mild Alert or High Alert, the device continues to alert until one of the following conditions occur:

1. An implementation specific timeout occurs
2. User takes some action on the device like pressing a button to acknowledge and stop the alert.
3. A new alert level is written.
4. The physical link is disconnected.

The Immediate Alert Service is shown in Figure 15.3.

In summary, from a use case scenario perspective, this service can be used to set an alert in the remote LE device so that the LE device can take appropriate action.

15.4 Find Me Profile (FMP)

The Find Me Profile supports the function to allow users to find misplaced devices. A button is pressed on a device which causes an alert to be raised on a peer device.

15.4.1 Roles

Find Me Profile defines the following two roles:

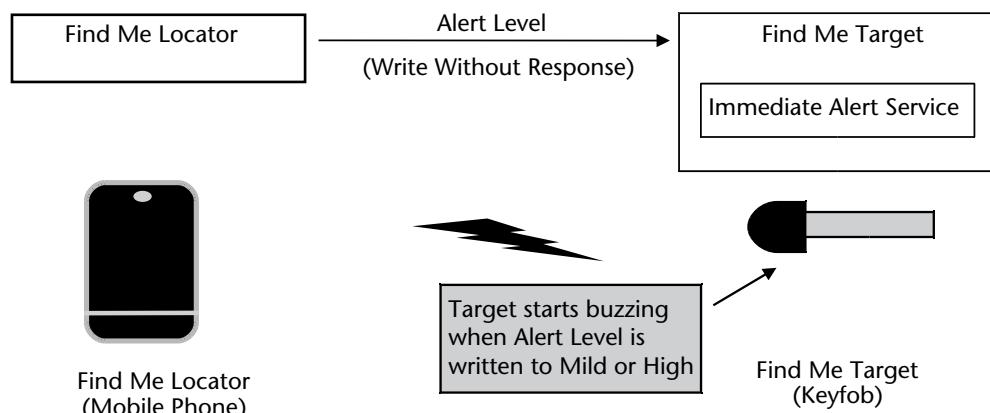


Figure 15.3 Immediate Alert Service and Find Me Profile roles.

1. Find Me Locator (GATT Client): This is the device on which the button is pressed. When a button is pressed, it writes the specific Alert Level into the Alert Level characteristic of the Find Me Target.
2. Find Me Target (GATT Server): This is the device on which an alert is raised.

The Find Me Locator uses the GATT service discovery procedures to discover the Immediate Alert Service on the Target.

The profile does not impose any restrictions on which of the two devices should act as GAP Central or Peripheral. Either of the devices can act in the GAP Central role and the peer device acts in the GAP Peripheral role. The Central device does the discovery and connection establishment with the Peripheral device. The Find Me Profile roles are shown in Figure 15.3.

15.5 Link Loss Service (LLS)

The Link Loss Service is a very simple service which allows an alert to be raised when the connection to a remote device is lost. When the link is lost, the device may take some specific action like flashing an LED, sounding a buzzer, locking the device, etc. This is useful when an alert is needed if a device moves out of range or comes into range.

One example of this service is when a user may use it between a watch and a mobile phone. If the user forgets a mobile phone and walks away (or if the mobile phone is stolen), the watch would raise an alert.

Another example could be a playground or a shopping mall where a child and parent may wear a watch (or any other device) supporting this service. If the child moves out of the range, then the connection would be lost and the parent would get an alert on their watch.

15.5.1 Service Declaration

The Link Loss Service is instantiated as a <>Primary Service<>. The Service UUID is set to <>Link Loss<>.

15.5.2 Service Characteristics

The Link Loss Service exposes only one characteristic which is mandatory:

1. Alert Level (Read, Write): The remote devices can read or write the alert level using the GATT Read Characteristic Value and Write Characteristic Value procedures.

The Alert Level can be set to one of the following:

1. No Alert: The device does not alert.
2. Mild Alert: The device alerts on loss of a link.
3. High Alert: The device alerts in the strongest possible way on loss of a link.

When the service is implemented in a device and the connection is lost, the device starts alerting at the specified alert level. The alert continues till one of the following occurs:

1. An implementation specific timeout occurs.
2. The user takes some action on the device like pressing a button to acknowledge and stop the alert.
3. The physical link is reconnected.

The Link Loss Service is shown in Figure 15.4.

15.6 Transmit Power Service (TPS)

The Transmit Power Service is a very simple service which exposes the device's transmit power level when the device is in connected mode.

One example of this service is when the user wishes to know the distance between two devices. The greater the distance, the greater would be the transmit power level. So depending on the transmit power level, the user can correlate the distance between the two devices.

Another example of this service could be a connection between the user's watch and a computer. As soon as the user comes near the computer, the computer automatically unlocks itself and when the user moves away from the computer, the computer locks itself. This can be done based on the transmit power level between the watch and the computer to calculate the distance between the two.

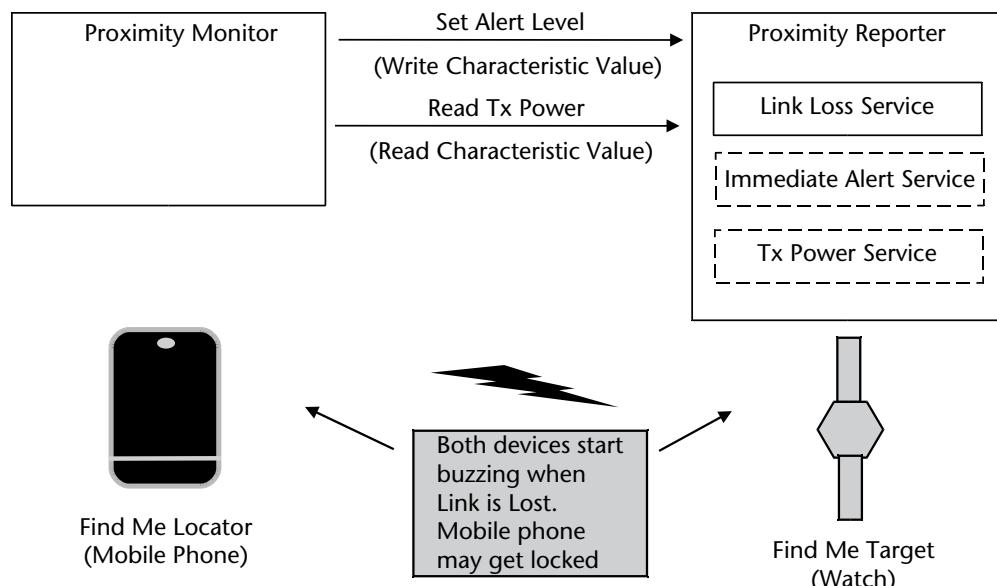


Figure 15.4 Immediate Alert Service, Link Loss Service, Tx Power Service and Proximity Profile roles.

15.6.1 Service Declaration

The Transmit Power Service is instantiated as a <<Primary Service>>. The Service UUID is set to <<Tx Power>>.

15.6.2 Service Characteristics

The Transmit Power Service exposes only one characteristic which is mandatory:

1. Tx Power Level (Read): The remote devices can read the Tx Power Level using the GATT Read Characteristic Value procedure.

The Tx Power Service is shown in Figure 15.4.

15.7 Proximity Profile (PXP)

The Proximity Profile supports the use of monitoring the proximity (distance) between two devices. It supports the following scenarios:

1. If a device moves far away and the connection drops, then an alert is generated.
2. If a device moves far away and the path loss increases above a certain value, then an alert is generated.

15.7.1 Roles

Proximity Profile defines the following two roles:

1. Proximity Monitor (GATT Client): The Monitor can read the Tx Power of the remote device and set the alert levels for immediate alert and link loss using GATT procedures.
2. Proximity Reporter (GATT Server): The Reporter exposes the characteristics of Immediate Alert Service, Link Loss Service and Tx Power Service.

Generally the Proximity Reporter is an LE-only device which acts as a GAP Peripheral. The Proximity Monitor could be a dual-mode or LE-only device which acts as GAP Central. The Proximity Profile roles are shown in Figure 15.4. The services shown in dashed boxes are optional while the services shown in solid boxes are mandatory.

15.8 Battery Service (BAS)

The Battery Service is a very simple service to provide information about the battery to remote devices. The remote devices may either read the battery level or be notified when the battery level has changed. One example of this service would be to get a notification on the mobile phone if the battery of a thermometer is about to run down.

15.8.1 Service Declaration

The Service UUID is set to <>Battery Service>>.

15.8.2 Service Characteristics

The Battery Service exposes only one characteristic which is mandatory:

1. **Battery Level (Read, Notify):** The remote devices can read the level using the GATT Read Characteristic Value or be notified when the battery level changes. The notification property is optional.

The Battery Level is denoted as a percentage from 0% to 100%. 0% represents a battery that is fully discharged and 100% denotes a battery that is fully charged. In order to save battery power, instead of polling the battery level periodically, the client can also configure the server to send a notification when the battery level changes.

15.9 Device Information Service (DIS)

The Device Information Service is used to provide manufacturer information about a device. For example, this service can be used to provide the manufacturer name, model number, serial number, etc.

15.9.1 Service Declaration

The Device Information Service is instantiated as a <>Primary Service>>. The Service UUID is set to <>Device Information>>.

15.9.2 Service Characteristics

The Device Information Service exposes the following characteristics. It is mandatory to support any one of these characteristics.

1. **Manufacturer Name String:** The name of the manufacturer of the device.
2. **Model Number String:** Model Number.
3. **Serial Number String:** Serial Number of the particular device.
4. **Hardware Revision String:** Revision of hardware in the device.
5. **Firmware Revision String:** Revision of firmware in the device.
6. **Software Revision String:** Revision of software in the device.
7. **System ID:** This represents a structure that contains an Organizationally Unique Identifier (OUI) followed by a manufacturer-defined identifier.
8. **IEEE 11073-20601 Regulatory Certification Data List:** Regulatory and Certification information about the product. The IEEE 11073-20601 specification defines a common framework for making an abstract model of personal health data so that this data can be exchanged and interpreted between health devices and computer systems.

All these characteristics are read only and can be read using the GATT Characteristic Value Read procedure.

15.10 Current Time Service (CTS)

The Current Time Service is used to provide the current time to remote devices. One example of this service would be to update the time in a watch automatically when a user travels to a new time zone or when daylight savings time changes. The watch could be connected to a mobile phone and be notified whenever the user travels to a new time zone.

15.10.1 Service Declaration

The Current Time Service is instantiated as a <>Primary Service>>. The Service UUID is set to <>Current Time Service>>.

15.10.2 Service Characteristics

The Current Time Service exposes the following three characteristics. Out of these, the first characteristic is mandatory and the remaining two are optional.

1. Current Time (Read, Notify): This characteristic provides the current date and time of the server device. This characteristic can be read or can be configured to be notified.
2. Local Time Information (Read): This characteristic provides the information like time zone and day light saving offset.
3. Reference Time Information (Read): This characteristic provides information about the reference time source from which the time was obtained. For example, this provides information on how accurate the time source is.

15.11 Health Thermometer Service (HTS)

The Health Thermometer Service is used to provide information from the thermometer like temperature, temperature type, etc. This service can be used in healthcare and fitness applications.

15.11.1 Service Declaration

The Device Information Service is instantiated as a <>Primary Service>>. The Service UUID is set to <>Health Thermometer Service >>.

15.11.2 Service Characteristics

The Health Thermometer Service exposes the following characteristics. Some of the characteristics also expose a corresponding Client Characteristic Configuration

Descriptor or a Valid Range Descriptor. Out of these the first service (Temperature Measurement) along with the corresponding Client Characteristic Configuration Descriptor are mandatory.

1. Temperature Measurement (Indicate): Used to indicate a temperature measurement to the peer device. The least significant bit is used to indicate whether the temperature is in Celsius or Fahrenheit.
 - a. Client Characteristic Configuration Descriptor: Used to configure the Temperature Measurement Characteristic (for example, to enable indication).
2. Temperature Type (Read): This is used to specify the location of the human body where the temperature is measured.
3. Intermediate Temperature (Notify): This is used to send intermediate values to a device while the temperature measurement is still in progress. The interval at which this is sent could typically vary from 0.25 seconds to 2 seconds.
 - a. Client Characteristic Configuration Descriptor: Used to configure the Intermediate Temperature Characteristic.
4. Measurement Interval (Read): This is used to set the interval between two successive measurements.
 - a. Client Characteristic Configuration Descriptor: Used to configure the Measurement Interval Characteristic.
 - b. Valid Range Descriptor: This provides the supported range of measurement interval values.

This service is shown in Figure 15.5.

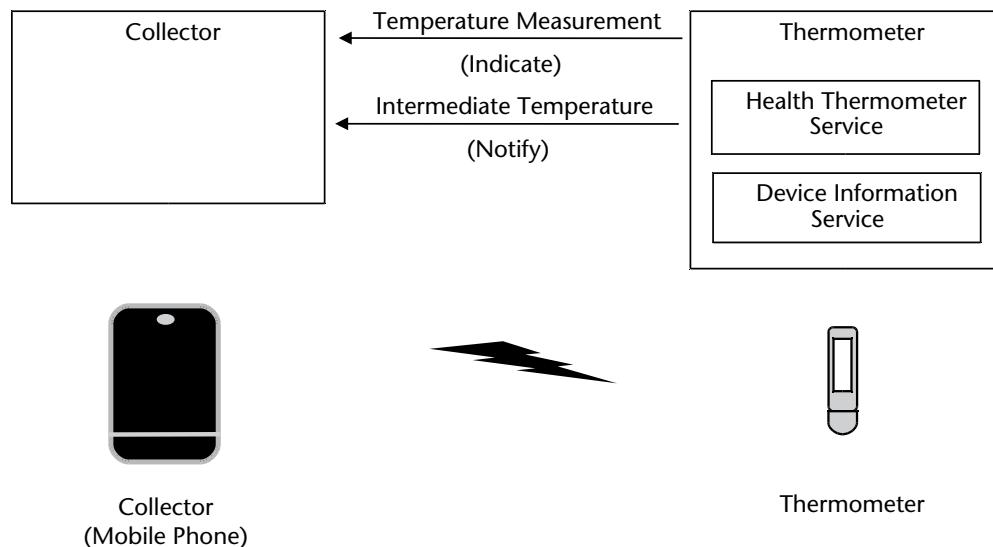


Figure 15.5 Health Thermometer Service and Health Thermometer Profile.

15.12 Health Thermometer Profile (HTP)

The Health Thermometer Profile allows a device to interact with a thermometer sensor that exposes the Health Thermometer Service.

15.12.1 Roles

The Health Thermometer Profile defines the following two roles:

1. Thermometer (GATT Server): The thermometer is the device that performs the temperature measurement and informs the collector.
2. Collector (GATT Client): The collector is the device that receives the data from the thermometer.

The Thermometer implements the GAP Peripheral role. The collector implements the GAP Central role. The Thermometer Profile roles are shown in Figure 15.5. The Thermometer role includes two services—Health Thermometer Service and Device Information Service. Both the services are mandatory.

15.13 Blood Pressure Service (BPS)

The Blood Pressure Service is used to provide information about the blood pressure and other related information. This service can be used in healthcare applications.

15.13.1 Service Declaration

The Service UUID is set to <>Blood Pressure Service >>.

15.13.2 Service Characteristics

The Blood Pressure Service exposes the following characteristics. Some of the characteristics also expose a corresponding Client Characteristic Configuration Descriptor. Out of these the first service (Blood Pressure Measurement) along with the corresponding Client Characteristic Configuration Descriptor and the Blood Pressure Feature are mandatory.

1. Blood Pressure Measurement (Indicate): Used to indicate a blood pressure measurement to the peer device. It may have additional fields like the pulse rate, Systolic and Diastolic pressure, etc.
 - a. Client Characteristic Configuration Descriptor: Used to configure the Blood Pressure Measurement Characteristic.
2. Intermediate Cuff Pressure (Notify): This is used to send the intermediate values while the measurement is still in progress.
 - a. Client Characteristic Configuration Descriptor: Used to configure the Intermediate Cuff Pressure Characteristic.
3. Blood Pressure Feature (Read): This is used to describe the supported features of the Blood Pressure Sensor.

The Blood Pressure Service is shown in Figure 15.6.

15.14 Blood Pressure Profile (BLP)

The Blood Pressure Profile allows a device to interact with a blood pressure sensor that exposes the Blood Pressure Service.

15.14.1 Roles

The Blood Pressure Profile defines the following two roles:

1. Blood Pressure Sensor (GATT Server): The blood pressure sensor is the device that performs the blood pressure measurement and informs the collector.
2. Collector (GATT Client): The collector is the device that receives the data from the blood pressure sensor.

The Blood Pressure Sensor implements the GAP Peripheral role. The collector implements the GAP Central role. The Blood Pressure Profile roles are shown in Figure 15.6. The Blood Pressure Sensor role includes two services—Blood Pressure Service and Device Information Service. Both the services are mandatory.

15.15 Health, Sports and Fitness Profiles

Similar to the services and profiles described above, LE includes several other profiles for health, sports and fitness applications. These include the following:

1. Glucose Service (GLS): Exposes the glucose and other data from a glucose sensor.

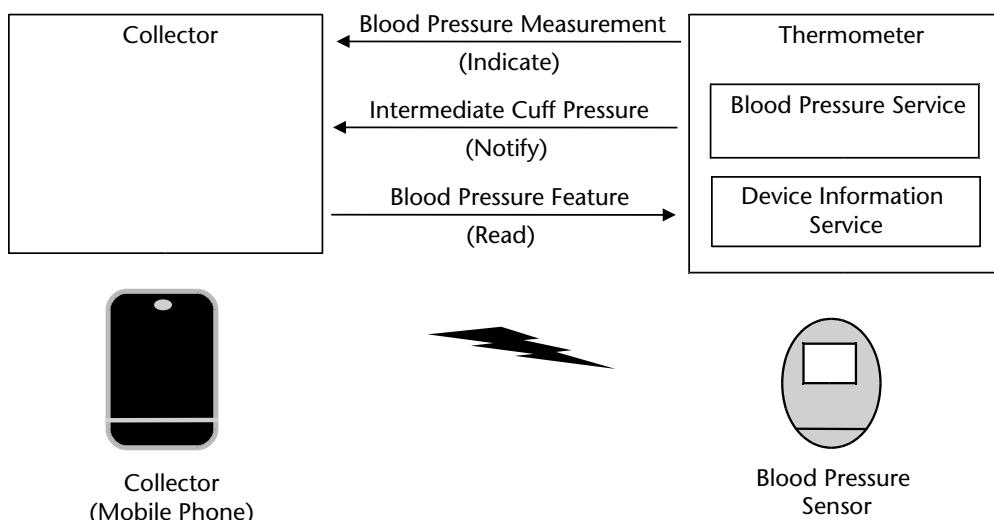


Figure 15.6 Blood Pressure Service and Blood Pressure Profile.

2. Glucose Profile (GLP): Allows a device to interact with a glucose sensor device.
3. Heart Rate Service (HRS): Exposes the heart rate and other data from a heart rate sensor.
4. Heart Rate Profile (HRP): Allows a collector to interact with a heart rate sensor device.
5. Cycling Speed and Cadence Service (Service): Exposes the cycling speed and other information to be used in sports and fitness applications.
6. Cycling Speed and Cadence Profile (CSCP): Allows a collector to interact with a cycling speed and cadence sensor.

15.16 Other Services and Profiles

There are several other profiles and services that LE specifies for use in different applications. These include the following:

1. Alert Notification Service (ANS): This service exposes information such as count of new alerts, count of unread alerts, different types of alerts etc. This information can be passed, for example, from a mobile phone to a watch so that the user can see this information on the watch itself instead of pulling out the mobile phone from the pocket or briefcase.
2. Alert Notification Profile (ANP): This profile enables a client to receive different types of alerts from a server device.
3. Phone Alert Status Service (PASS): This service exposes the phone alert status and ringer settings. It could be used, for example, to display an alert on a watch when there is an incoming call on the mobile. Besides this, it could also be used to set the phone to silent mode or some other mode from the watch itself.
4. Phone Alert Status Profile (PASP): This profile enables a client to receive the phone alert from a server.
5. HID Service: The HID service exposes data like HID reports which can be used between HID hosts and HID devices. For example, an LE-based keyboard or mouse could use this service to send reports to a computer whenever a user presses a key on the keyboard or moves the mouse.
6. HID over GATT (HOGP): This profile enables HID devices to interact with HID hosts using the GATT profile over an LE transport. (Note that BR/EDR specification also defines an HID profile which defines HID over a BR/EDR transport).

15.17 Practical Examples

This section explains a practical use of the Proximity profile with the help of air logs. It shows the communication between a dual mode device and an LE sensor supporting proximity profile.

Table 15.1 Services Supported by LE Sensor

S. No.	Service Name	Starting Attribute Handle	Ending Attribute Handle
1	Generic Access Profile	1	7
2	Generic Attribute Profile	16	19
3	Link Loss Alert	80	82
4	Immediate Alert	83	85
5	Tx Power	86	88

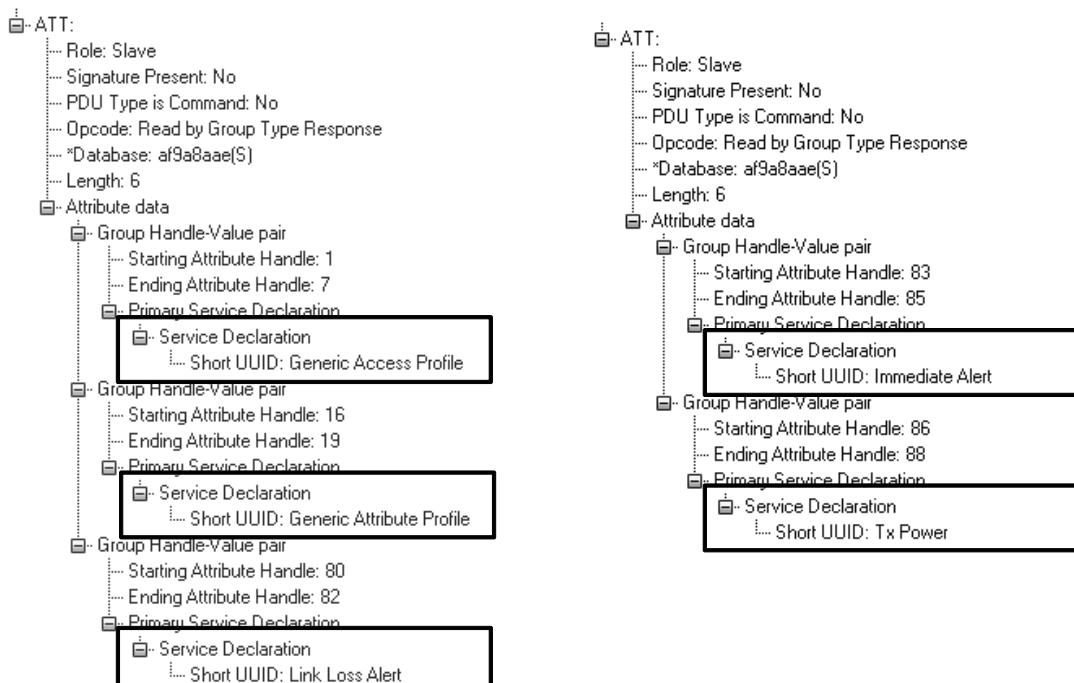
Step 1: Discover All Services of the LE sensor

As a first step, the dual mode device discovers the services of the LE sensor using GATT procedure “Discover All Services”. This procedure was explained in Chapter 13. The resulting list of services is shown in Figure 15.7. It shows that the LE sensor contains the five services shown in Table 15.1.

Step 2: Discover All Characteristics of the Alert Level Service

Once all the services are discovered, the dual mode device goes on to discover all characteristics of each of those services using the GATT “Discover All Characteristics of a Service” procedure. Figure 15.8 shows the characteristics of the Alert Level Service. It contains one characteristic as follows:

- Alert Level Characteristic: Value Handle 85.

**Figure 15.7** Discover All Services.

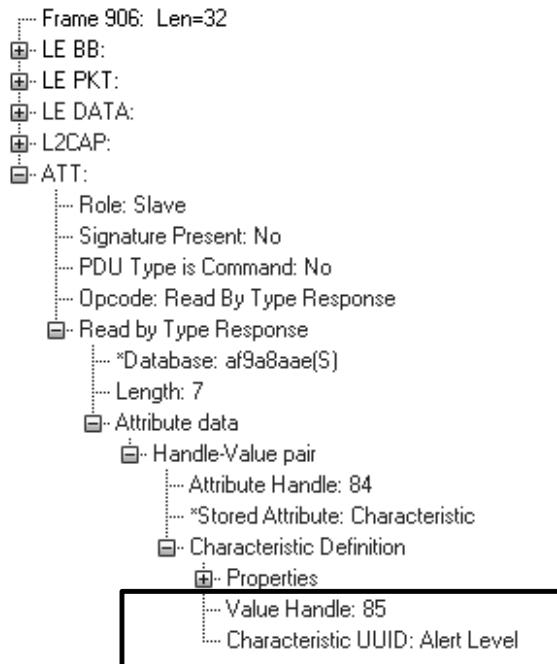


Figure 15.8 Discover All Characteristics of the Alert Level Service.

Step 3: Read or Write the Characteristics

The final step executed by the profile is to write the alert level characteristic. Figure 15.9 shows two operations:

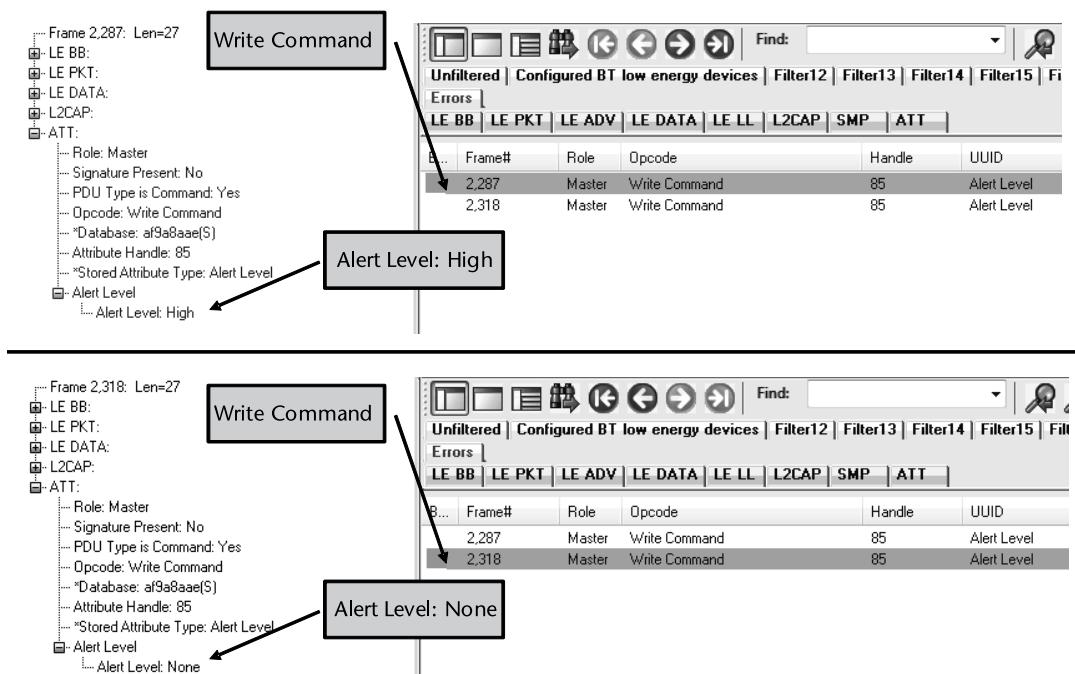


Figure 15.9 Read or Write Alert Level Characteristic.

1. The top part (a) shows writing Alert Level to high. Once this is written, the device starts buzzing.
2. The bottom part (b) shows writing Alert Level to None. Once this is written, the device stops buzzing.

15.18 Summary

The GATT-based LE profiles are much simpler than most of the BR/EDR profiles. As of writing of this book, already 20 profiles have been defined by the Bluetooth SIG. Each profile is expected to cater to some specific use case scenario. The use cases of LE were explained in Chapter 1. This chapter provided details on how those use cases get implemented using the GATT-based LE profiles.

Bibliography

Bluetooth Core Specification 4.0 <http://www.bluetooth.org>.

Bluetooth Profiles Specifications <http://www.bluetooth.org>.

Bluetooth Assigned Numbers, <https://www.bluetooth.org/assigned-numbers>.

IEEE Std 11073-20601™- 2008 Health Informatics, Personal Health Device Communication, Application Profile, Optimized Exchange Protocol, version 1.0 or later.

Developing LE Applications

16.1 Introduction

In Chapter 5, we looked at some practical exercises that can be done on an Ubuntu system using the BlueZ stack to understand the various Bluetooth operations.

In this chapter, we will look at how to build an LE application based on the BlueZ stack. We will start with basic operations like setting up the development environment, enabling the Bluetooth adapter, advertising, and scanning. After that various GATT operations like discovering services, characteristics and reading and writing characteristics will be explained. Finally this chapter will explain the steps needed to make a real world application—an application to find lost keys.

Broadly this chapter will show examples of the various concepts that were explained in previous chapters and how various components come together to make a complete real world application. Besides the various procedures, this chapter will also explain what happens inside the protocol stack when those procedures are executed. In particular, this chapter will go into the transactions happening at the ATT level as well as the HCI level.

The BlueZ stack running in an Ubuntu environment has been selected for the examples due to several reasons:

1. This environment is easy to setup. The Ubuntu operating system can be downloaded and is pretty straightforward to install.
2. This environment is quite inexpensive. Only a couple of PCs are needed along with Bluetooth dongles and application development can be started.
3. The latest versions of BlueZ provide in-built support for LE. This means that the developer can directly focus on writing the application instead of first looking at how to bring up the various protocol stack layers.
4. BlueZ is open source. This means that the source code is available for reference at any time to see how exactly the different components are implemented. It can, for example, be used to see how the *gatttool* interfaces with the BlueZ stack to carry out various GATT operations.
5. Once the developer has made some enhancements to the BlueZ stack it can also be contributed back to the community so that others can also benefit from those.

6. BlueZ provides a powerful *hcidump* tool which can be used for analysis of the various procedures and what happens internally when those procedures are invoked. BlueZ also provides several utilities and sample examples which can be used for reference.
7. Even though the end environment may not be Linux- or BlueZ-based, these examples and sample source code will be very useful in understanding how a typical LE implementation works. This can help in quick ramp-up and result in speedier application development in the target environment.
8. BlueZ is a powerful environment to use as a peer device for testing since it supports both the LE only and dual mode roles. For example, if the developer is developing an LE device (Bluetooth Smart), then the BlueZ environment can be used as a Bluetooth Smart Ready device for testing that LE device. On the other hand if the developer is developing a Bluetooth Smart Ready device or application, the BlueZ environment can be used as a Bluetooth Smart device.

This chapter will also explain some of the tips and tricks that can be useful while developing and debugging an LE application. Some of the tools that can be used for assistance are also explained.

16.2 Ingredients

You will need the following:

1. Two PCs running any flavor of Linux (Preferably a current one to ensure it includes support for LE). For the purpose of examples, Ubuntu 12.10 is used here though any other Linux system will serve the purpose provided it's not too old. These two PCs will be referred to by the following names:
 - a. *lepc*: This is the PC that will run the LE side of the sample applications.
 - b. *dualpc*: This is the PC that will run the Dual mode side of the sample applications.
2. A couple of LE devices. If you search the internet you may find development kits from some vendors which can be used for developing LE applications. Some LE devices have also started appearing in the market and these are expected to grow rapidly. For the purpose of examples in this chapter, we will use a couple of PTS dongles and attach them via the USB interface to each of the Ubuntu PCs. The PTS dongle is available for purchase from the Bluetooth SIG website.

16.2.1 Installing hcidump

BlueZ comes with a very useful tool called *hcidump* which can be used to find out the packets being exchanged on the HCI interface. By default this tool is not installed on the Ubuntu system. The message shown in Figure 16.1 is received on invoking *hcidump* if this tool is not installed.

There are two methods for installing *hcidump*:

```
dualpc# hcidump  
The program 'hcidump' is currently not installed. You can install it by  
typing:  
apt-get install bluez-hcidump
```

Figure 16.1 Invoking hcidump.

Method 1: The following command may be given to download and install it.

```
apt-get install bluez-hcidump
```

This should install hcidump on the system.

Method 2: The source code of the hcidump command may be downloaded from the bluez website (Reference [3]) and built. This method is explained in detail here.

The commands needed to build hcidump are shown in Figure 16.2. Some of the output of the commands is removed for ease of understanding.

```
dualpc# gunzip bluez-hcidump-2.3.tar.gz  
dualpc#  
  
dualpc# tar xvf bluez-hcidump-2.3.tar  
bluez-hcidump-2.3/  
.  
.  
.  
bluez-hcidump-2.3/README  
dualpc#  
  
dualpc# cd bluez-hcidump-2.3  
dualpc#  
dualpc# ./configure  
checking for ...  
.  
.  
.  
dualpc#  
  
dualpc# make  
make -no-print-directory all-am  
  CC      src/bpasniff.o  
.  
.  
.  
dualpc#  
  
dualpc# make install  
test -z ...  
.  
.  
.  
dualpc#
```

Figure 16.2 Building hcidump.

The following are the steps needed to build hcidump.

1. Unpack the source code and prepare to build.

```
gunzip bluez-hcidump-2.3.tar.gz
tar xvf bluez-hcidump-2.3.tar
cd bluez-hcidump-2.3.tar
```

2. Configure the source code:

```
./configure
```

3. Build and Install

```
./make
./make install
```

Note that before doing a make install, root privileges may be required. This can be done by executing the command:

```
sudo su
```

16.2.2 Basic Bluetooth operations

Before trying the examples provided in this chapter, the two LE dongles will need to be connected to each of the two PCs: *lepc* and *dualpc*. Note that some of the hciconfig commands are supported only if the user has root privileges. So it's better to do the following before trying out the commands mentioned in this chapter:

```
sudo su
```

16.2.2.1 Invoking hcidump

It will be useful to start hcidump before giving any of the commands mentioned in the following section. This tool will show the various commands given on the HCI interface and the events received. Apart from the commands and events this tool also decodes the parameters of the commands and events. It also shows the higher layer transactions for ACL data packets. For example, for ACL data packets, this tool shows the transactions that happen at the ATT protocol layer. Figure 16.3 shows how to invoke hcidump. (The *-i* option is used to specify the HCI interface for which the tool should display the various commands and events. This will be explained shortly).

```
dualpc# hcidump -i hci1 -X
HCI sniffer - Bluetooth packet analyzer ver 2.3
device: hci1 snap_len: 1028 filter: 0xffffffff
```

Figure 16.3 Invoking hcidump.

The best way to use this tool is to execute it in a separate terminal window so that all commands and events can be seen in that separate window and these don't garble the display of commands and events on the main window.

16.2.2.2 Accessing the LE device

Once a dongle (also known as Bluetooth adapter) is connected to a PC, it should be accessible through an *hci* interface. The *hci* interfaces are referred to by BlueZ as *hci0*, *hci1* and so on, depending on the number of Bluetooth adapters connected to the PC.

The *hciconfig* command can be used to configure the HCI adapter(s). The following command is used to find information about all HCI adapter(s) attached to the *lepc* PC:

```
hciconfig -a
```

The output of this command is shown in Figure 16.4. The following things may be observed:

```
lepc# hciconfig -a
hci0: Type: BR/EDR Bus: USB
      BD Address: 08:ED:B9:DE:52:FB  ACL MTU: 8192:128  SCO MTU: 64:128
      UP RUNNING PSCAN
      RX bytes:1314 acl:0 sco:0 events:30 errors:0
      TX bytes:606 acl:0 sco:0 commands:30 errors:0
      Features: 0xff 0xff 0x8f 0xfe 0x83 0xe1 0x08 0x80
      Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
      Link policy: RSWITCH HOLD SNIFF PARK
      Link mode: SLAVE ACCEPT
      Name: 'ubuntu-0'
      Class: 0x6e0100
      Service Classes: Networking, Rendering, Capturing, Audio, Telephony
      Device Class: Computer, Uncategorized
      HCI Version: 2.1 (0x4) Revision: 0x100
      LMP Version: 2.1 (0x4) Subversion: 0x100
      Manufacturer: not assigned (6502)

hci1: Type: BR/EDR Bus: USB
      BD Address: 00:1B:DC:05:C8:D6  ACL MTU: 310:10  SCO MTU: 64:8
      UP RUNNING PSCAN
      RX bytes:1063 acl:0 sco:0 events:31 errors:0
      TX bytes:849 acl:0 sco:0 commands:30 errors:0
      Features: 0xff 0xff 0x8f 0x8e 0xd8 0x1f 0x5b 0x87
      Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
      Link policy: RSWITCH HOLD SNIFF PARK
      Link mode: SLAVE ACCEPT
      Name: 'ubuntu-1'
      Class: 0x6e0100
      Service Classes: Networking, Rendering, Capturing, Audio, Telephony
      Device Class: Computer, Uncategorized
      HCI Version: 4.0 (0x6) Revision: 0x1d86
      LMP Version: 4.0 (0x6) Subversion: 0x1d86
      Manufacturer: Cambridge Silicon Radio (10)
```

Figure 16.4 Get information about all Bluetooth devices connected to the PC.

1. The PC has two Bluetooth adapters attached to it on hci0 and hci1.
2. The information about hci0 is as follows:
 - a. BD_ADDR is 08:ED:B9:DE:52:FB.
 - b. The ACL MTU size is 8192 bytes and the device has 128 buffers.
 - c. The SCO MTU size is 64 bytes and the device has 128 buffers.
 - d. The bitmap indicating features supported by the device is: ff ff 8f fe 83 e1 08 80. (This indicates the set of features that the link manager supports out of the feature mask definition.)
 - e. The name of the device is ‘ubuntu-0.’
 - f. The Class of Device (CoD) is 0x6e0100. This means that it supports the following: Device Class: Computer, Uncategorized; Service Class: Networking, Rendering, Capturing, Audio, Telephony.
 - g. The LMP version is 2.1. This means that this device support Bluetooth 2.1 specification.
3. The information about hci1 is as follows:
 - a. BD_ADDR is 00:1B:DC:05:B5:B3.
 - b. The ACL MTU size is 310 bytes and the device has 10 buffers.
 - c. The SCO MTU size is 64 bytes and the device has 8 buffers.
 - d. The bitmap indicating features supported by the device is: ff ff 8f 7e d8 1f 5b 87 (This indicates the set of features that the link manager supports out of the feature mask definition.)
 - e. The name of the device is ‘ubuntu-1.’
 - f. The Class of Device (CoD) is 0x6e0100. This means that it supports the following: Device Class: Computer, Uncategorized; Service Class: Networking, Rendering, Capturing, Audio, Telephony.
 - g. The LMP version is 4.0. This means that this device support Bluetooth 4.0 specification.

This indicates that hci0 is a BR/EDR device and hci1 is a dual mode device.

16.2.2.3 Opening and Closing HCI interface

The HCI interface may be opened and closed by the hciconfig command. By default the HCI interface is open when Ubuntu boots up. The command to open the HCI interface is:

```
hciconfig hci1 up
```

The command to close the HCI interface is:

```
hciconfig hci1 down
```

These are shown in Figure 16.5.

```
dualpc# hciconfig hci1 down
dualpc# hciconfig hci1 up
dualpc#
```

Figure 16.5 Opening and Closing HCI interface.

The hcidump of the previous command that was used to bring up the HCI interface (hciconfig hci1 up) throws up several interesting bits of information. This is shown in Figure 16.6. Some of the lines of output are removed for easier understanding.

Some of the key points to note are:

1. The HCI Reset command is given at the beginning to reset the controller and bring it to a known state.
2. The Read Local Version Information command is used to find out the version of Bluetooth specification supported by the controller.
3. The LE Read Buffer Size command is used to read the number of LE buffers. This command returns 0 as the length of the LE packets. This means that the controller is using shared ACL buffers for LE and BR/EDR. The number of shared buffers in the controller is read by the HCI Read Buffer Size command. This was explained in Chapter 9.
4. The Write LE Host Supported command is used to inform the controller that the host supports LE.

16.2.2.4 Link Manager Supported States

The various states of the link manager were explained in Chapter 8. BlueZ supports the following command to get the list of states supported by the link manager:

```
hciconfig hci1 lestates
```

Note that this command takes the HCI adapter as an argument. In this case, the LE dongle is attached on hci1 interface. So the command is used with the first argument as hci1.

The output of this command is shown in Figure 16.7. The device in this case supports the following LM states:

- Non-connectable Advertising State.
- Scannable Advertising State.
- Connectable Advertising State.

There are many more possible states that the device supports. The complete list is shown in Figure 16.7.

The commands and events exchanged on the HCI interface as shown in Figure 16.8. It shows that the HCI command LE_Read_Supported_States is used to get the list of supported states. This command was explained in Chapter 9.

```

< HCI Command: Reset (0x03|0x0003) plen 0
> HCI Event: Command Complete (0x0e) plen 4
    Reset (0x03|0x0003) ncmd 1
    status 0x00
< HCI Command: Read Local Supported Features (0x04|0x0003) plen 0
> HCI Event: Command Complete (0x0e) plen 12
    Read Local Supported Features (0x04|0x0003) ncmd 1
    status 0x00
    Features: 0xff 0xff 0x8f 0x7e 0xd8 0x1f 0x5b 0x87
< HCI Command: Read Local Version Information (0x04|0x0001) plen 0
> HCI Event: Command Complete (0x0e) plen 12
    Read Local Version Information (0x04|0x0001) ncmd 1
    status 0x00
    HCI Version: 4.0 (0x6) HCI Revision: 0x1d86
    LMP Version: 4.0 (0x6) LMP Subversion: 0x1d86
    Manufacturer: Cambridge Silicon Radio (10)
< HCI Command: Read Buffer Size (0x04|0x0005) plen 0
> HCI Event: Command Complete (0x0e) plen 11
    Read Buffer Size (0x04|0x0005) ncmd 1
    status 0x00
    ACL MTU 310:10 SCO MTU 64:8
< HCI Command: Read BD ADDR (0x04|0x0009) plen 0
> HCI Event: Command Complete (0x0e) plen 10
    Read BD ADDR (0x04|0x0009) ncmd 1
    status 0x00 bdaddr 00:1B:DC:05:B5:B3
< HCI Command: Set Event Mask (0x03|0x0001) plen 8
    Mask: 0xffffffffffffbfff07f8bf3d
> HCI Event: Command Complete (0x0e) plen 4
    Set Event Mask (0x03|0x0001) ncmd 1
    status 0x00
< HCI Command: Read Local Supported Commands (0x04|0x0002) plen 0
> HCI Event: Command Complete (0x0e) plen 68
    Read Local Supported Commands (0x04|0x0002) ncmd 1
    status 0x00
    Commands: ffffffff03fefffffffffffff30fe8fe3ff783ff1c00000061f7ffff7f
< HCI Command: Write LE Host Supported (0x03|0x006d) plen 2
    0000: 01 01
    ..
> HCI Event: Command Complete (0x0e) plen 4
    Write LE Host Supported (0x03|0x006d) ncmd 1
    0000: 00
    ..
< HCI Command: LE Read Buffer Size (0x08|0x0002) plen 0
> HCI Event: Command Complete (0x0e) plen 7
    LE Read Buffer Size (0x08|0x0002) ncmd 1
    status 0x00 pktlen 0x0000 maxpkt 0x00
< HCI Command: Write Simple Pairing Mode (0x03|0x0056) plen 1
    mode 0x01
> HCI Event: Command Complete (0x0e) plen 4
    Write Simple Pairing Mode (0x03|0x0056) ncmd 1
    status 0x00
    ..
< HCI Command: Write LE Host Supported (0x03|0x006d) plen 2
    0000: 01 01
    ..
> HCI Event: Command Complete (0x0e) plen 4
    Write LE Host Supported (0x03|0x006d) ncmd 1
    0000: 00
    ..

```

Figure 16.6 HCI transactions during hciconfig hci1 up.

16.3 Advertising and Scanning

The advertising and scanning procedures were explained in Chapter 8.

```
lepc# hciconfig hci1 lestates
Supported link layer states:
    YES Non-connectable Advertising State
    YES Scannable Advertising State
    YES Connectable Advertising State
    YES Directed Advertising State
    YES Passive Scanning State
    YES Active Scanning State
    YES Initiating State/Connection State in Master Role
    YES Connection State in the Slave Role
    YES Non-connectable Advertising State and Passive Scanning State
combination
    YES Scannable Advertising State and Passive Scanning State combination
    YES Connectable Advertising State and Passive Scanning State
combination
    YES Directed Advertising State and Passive Scanning State combination
    YES Non-connectable Advertising State and Active Scanning State
combination
    YES Scannable Advertising State and Active Scanning State combination
    YES Connectable Advertising State and Active Scanning State
combination
    YES Directed Advertising State and Active Scanning State combination
    YES Non-connectable Advertising State and Initiating State combination
    YES Scannable Advertising State and Initiating State combination
    YES Non-connectable Advertising State and Master Role combination
    YES Scannable Advertising State and Master Role combination
    YES Non-connectable Advertising State and Slave Role combination
    YES Scannable Advertising State and Slave Role combination
    NO  Passive Scanning State and Initiating State combination
    NO  Active Scanning State and Initiating State combination
    YES Passive Scanning State and Master Role combination
    YES Active Scanning State and Master Role combination
    YES Passive Scanning State and Slave Role combination
    YES Active Scanning State and Slave Role combination
    YES Initiating State and Master Role combination/Master Role and
Master Role combination
```

Figure 16.7 Retrieving the list of LM states supported.

```
dualpc# hcidump -i hci1 -X
HCI sniffer - Bluetooth packet analyzer ver 2.3
device: hci1 snap_len: 1028 filter: 0xffffffff
< HCI Command: LE Read Supported States (0x08|0x001c) plen 0
> HCI Event: Command Complete (0x0e) plen 12
  LE Read Supported States (0x08|0x001c) ncmd 1
    0000: 00 ff ff 3f 1f 00 00 00 00 ...?.....
```

Figure 16.8 HCI transactions for getting LE states.

The following command is used to enable advertising on the PC *lepc*.

```
hciconfig hci1 leadv
```

The following command is used for scanning on PC *dualpc*.

```
hcitool -i hci1 lescan
```

The advertising command is shown in Figure 16.9. The corresponding output of hcidump command is also shown in Figure 16.9. As may be noted, the HCI command LE_Set_Advertise_Enable is used to enable advertising.

The command for scanning is shown in Figure 16.10. The BD_ADDR for lepc is 00:1B:DC:05:C8:D6. When *dualpc* does a scanning procedure, this device is shown amongst the list of devices found.

The corresponding output of hcidump command is also shown in Figure 16.10. As may be noted, the HCI commands LE_Set_Scan_Parameters and LE_Set_Scan_Enable are used to enable scanning. The remote devices are informed by the controller by sending LE_Advertising_Report events.

16.4 Creating a Connection

Creating an LE connection is quite simple. Once the *lepc* is in advertising mode (as shown in previous section) the following command can be given to create an LE connection by *dualpc*:

```
hcitool -i hcil lecc BD_ADDR_OF_lepc
```

This is shown in Figure 16.11. It may be noted that on successful connection this command returns the connection handle. For example, in this case, the connection handle assigned to the LE connection is 39.

16.5 GATT Operations

16.5.1 Enable GATT Functionality on Server

BlueZ on ubuntu includes a GATT server which provides support for FindMe and Proximity profiles. It includes the following services:

1. Generic Access Service.
2. Generic Attribute Service.

```
lepc# hciconfig hcil leadv
lepc#
```

Advertising on lepc

```
lepc# hcidump -i hcil -X
HCI sniffer - Bluetooth packet analyzer ver 2.3
device: hcil snap_len: 1028 filter: 0xffffffff
< HCI Command: LE Set Advertise Enable (0x08|0x000a) plen 1
  0000: 01
> HCI Event: Command Complete (0x0e) plen 4
    LE Set Advertise Enable (0x08|0x000a) ncmd 1
      status 0x00
```

HCI commands and events for advertising

Figure 16.9 Advertising.

```
dualpc# hcitool -i hcil lescan
LE Scan ...
00:1B:DC:05:C8:D6 (unknown)
00:1B:DC:05:C8:D6 (unknown)
00:1B:DC:05:C8:D6 (unknown)
00:1B:DC:05:C8:D6 (unknown)
00:1B:DC:05:C8:D6 (unknown)
00:1B:DC:05:C8:D6 (unknown)
^C
dualpc#
```

Scanning on dualpc

```
< HCI Command: LE Set Scan Parameters (0x08|0x000b) plen 7
  type 0x01 (active)
  interval 10.000ms window 10.000ms
  own address: 0x00 (Public) policy: All
> HCI Event: Command Complete (0x0e) plen 4
  LE Set Scan Parameters (0x08|0x000b) ncmd 1
  status 0x00
< HCI Command: LE Set Scan Enable (0x08|0x000c) plen 2
  value 0x01 (scanning enabled)
  filter duplicates 0x01 (enabled)
> HCI Event: Command Complete (0x0e) plen 4
  LE Set Scan Enable (0x08|0x000c) ncmd 1
  status 0x00
> HCI Event: LE Meta Event (0x3e) plen 12
  LE Advertising Report
    ADV_IND - Connectable undirected advertising (0)
    bdaddr 00:1B:DC:05:C8:D6 (Public)
    RSSI: -39
> HCI Event: LE Meta Event (0x3e) plen 12
  LE Advertising Report
    SCAN_RSP - Scan Response (4)
    bdaddr 00:1B:DC:05:C8:D6 (Public)
    RSSI: -39
```

HCI commands and events for scanning

Figure 16.10 Scanning.

```
dualpc# hcitool -i hcil lecc 00:1B:DC:05:C8:D6
Connection handle 39
```

Creating connection from dualpc to lepc

```
< HCI Command: LE Create Connection (0x08|0x000d) plen 25
  bdaddr 00:1B:DC:05:C8:D6 type 0
> HCI Event: Command Status (0x0f) plen 4
  LE Create Connection (0x08|0x000d) status 0x00 ncmb 1
> HCI Event: LE Meta Event (0x3e) plen 19
  LE Connection Complete
  status 0x00 handle 39, role master
  bdaddr 00:1B:DC:05:C8:D6 (Public)
```

HCI commands and events for creating a connection

Figure 16.11 Creating an LE connection.

3. Link Loss Service.
4. Immediate Alert Service.
5. Tx Power Service.

This section configures *lepc* as a GATT server. To do this, the GATT server needs to be enabled. The steps to do this are as follows:

1. Set *EnableGatt* to *true* in */etc/bluetooth/main.conf*.
2. Kill the Bluetooth daemon by giving the command.

```
sudo killall bluetoothd
```

3. Start Bluetooth daemon again by giving the command.

```
sudo /usr/sbin/bluetoothd
```

16.5.2 Execute GATT Procedures from the Client

This section configures *dualpc* as the GATT client and uses the *gatttool* application. The *gatttool* application can be used to send a single request to the GATT layer or it can also be used in an interactive mode where a series of requests can be sent interactively. In the interactive mode, *gatttool* comes up with a command prompt where the requests can be given. This section will explain how to run the *gatttool* in the interactive mode.

The syntax of the *gatttool* application is:

```
gatttool [option...]
```

It can be invoked as follows:

```
gatttool -i hciX -I -b BD_ADDR_OF_REMOTE_DEVICE
```

where:

- I *hciX* specifies the HCI Interface (local adapter interface)
- b *BD_ADDR_OF_REMOTE_DEVICE* specifies the *BD_ADDR* of the remote device
- I specifies to use the interactive mode

In this example, the *gatttool* command is run as follows:

```
gatttool -i hci1 -I -b 00:1B:DC:05:C8:D6
```

The *-I* switch is used to run *gatttool* in interactive mode so that further commands can be given at the *gatttool* command prompt. Once this command is executed, the *gatttool* shows its own prompt where further commands can be given. The prompt shown by *gatttool* is shown in Figure 16.12.

```
dualpc# gatttool -i hci1 -I -b 00:1B:DC:05:C8:D6
[    ] [00:1B:DC:05:C8:D6] [LE]>
```

Figure 16.12 Running gatttool in interactive mode.

16.5.2.1 Connecting from the Client

The command to create a connection is:

```
connect
```

Once this command is given, the display changes as shown in Figure 16.13. Note that the prompt in the first square bracket has changed to CON to indicate that GATT client is now connected to a GATT server.

16.5.2.2 Getting the Primary Services

The command to get the primary services is:

```
primary
```

The output of this command is shown in Figure 16.14. The output shows the following for each primary service:

- Attribute Handle;
- End Group Handle;

```
[    ] [00:1B:DC:05:C8:D6] [LE]> connect
[CON] [00:1B:DC:05:C8:D6] [LE]>
```

Figure 16.13 Connecting from the client.

```
[CON] [00:1B:DC:05:C8:D6] [LE]> primary
[CON] [00:1B:DC:05:C8:D6] [LE]>
attr handle: 0x0001, end grp handle: 0x0008 uuid: 00001800-0000-1000-8000-
00805f9b34fb
attr handle: 0x0009, end grp handle: 0x000b uuid: 00001803-0000-1000-8000-
00805f9b34fb
attr handle: 0x000c, end grp handle: 0x000f uuid: 00001804-0000-1000-8000-
00805f9b34fb
attr handle: 0x0010, end grp handle: 0x0010 uuid: 00001801-0000-1000-8000-
00805f9b34fb
attr handle: 0x0011, end grp handle: 0x0013 uuid: 00001802-0000-1000-8000-
00805f9b34fb
attr handle: 0x0014, end grp handle: 0x0017 uuid: 0000a002-0000-1000-8000-
00805f9b34fb
attr handle: 0x0022, end grp handle: 0x002c uuid: 0000a004-0000-1000-8000-
00805f9b34fb
attr handle: 0xffffa, end grp handle: 0xffffe uuid: feee74dc-a8de-3196-1149-
d43596c00a4f
[CON] [00:1B:DC:05:C8:D6] [LE]>
```

Figure 16.14 Getting the primary services.

- UUID of that service (128-bit UUID of the service).

It indicates the primary services as shown in Table 16.1. Note that the service UUID has been converted from 128-bit to 16-bit. As explained in Chapter 12, a 16-bit UUID can be derived from the 128-bit bit UUID for the UUIDs that are already defined by the Bluetooth SIG. It is shown as xxxx in the 128-bit representation when the representation is in the form of 0000xxxx-0000-1000-8000-00805F9B34FB.

The service UUIDs are mapped to the Service Names by referring to the Bluetooth Assigned Numbers for GATT (see Reference [4]) and GATT based Service UUIDs (see Reference [5]).

The transactions that are initiated on the ATT level for getting the primary services can be captured by using the hcidump tool. The output of the hcidump tool for getting the primary services is shown in Figure 16.15. The following points may be noted:

1. The ATT Read_By_Group_Type request is used to read the services. It is executed with the following parameters:
 - a. Start Handle: 0x0001.
 - b. Ending Handle: 0xffff.
 - c. UUID: 0x2800 (This means that <>Primary Service>> is requested).
2. All ATT methods use the ACL packets to send and receive data.
3. The ATT Read_By_Group_Type response returns the first three services with the End Group Handle of the last service being 0x0f.
4. The ATT Read_By_Group_Type request is used again to read the next set of services. It is executed with the following parameters:
 - a. Start Handle: 0x0010.
 - b. Ending Handle: 0xffff.
 - c. UUID: 0x2800 (This means that <>Primary Service>>) is requested.
5. This continues till an Error Response is received from the remote side.

16.5.2.3 Getting the Characteristics

The command to get the characteristics that are included in the service is:

Table 16.1 Primary Services on the GATT Server

S. No	Service UUID	Service Name	Starting Handle	Ending Handle
1	1800	<>Generic Access Service>>	0x0001	0x0008
2	1803	<>Link Loss Service>>	0x0009	0x000b
3	1804	<>Tx Power Service>>	0x000c	0x000f
4	1801	<>Generic Attribute Service>>	0x0010	0x0010
5	1802	<>Immediate Alert Service>>	0x0011	0x0013
6	a002		0x0014	0x0017
7	a004		0x0022	0x002c
8	User define 128-bit	User defined service	0xffffa	0xffffe

```

dualpc# hcidump -i hcil -X
HCI sniffer - Bluetooth packet analyzer ver 2.3
device: hcil snap_len: 1028 filter: 0xffffffff
< ACL data: handle 39 flags 0x00 dlen 11
    ATT: Read By Group req (0x10)
        start 0x0001, end 0xffff
        type-uuid 0x2800
> HCI Event: Number of Completed Packets (0x13) plen 5
    handle 39 packets 1
> ACL data: handle 39 flags 0x02 dlen 24
    ATT: Read By Group resp (0x11)
        attr handle 0x0001, end group handle 0x0008
        value 0x00 0x18
        attr handle 0x0009, end group handle 0x000b
        value 0x03 0x18
        attr handle 0x000c, end group handle 0x000f
        value 0x04 0x18
< ACL data: handle 39 flags 0x00 dlen 11
    ATT: Read By Group req (0x10)
        start 0x0010, end 0xffff
        type-uuid 0x2800
> HCI Event: Number of Completed Packets (0x13) plen 5
    handle 39 packets 1
> ACL data: handle 39 flags 0x02 dlen 24
    ATT: Read By Group resp (0x11)
        attr handle 0x0010, end group handle 0x0010
        value 0x01 0x18
        attr handle 0x0011, end group handle 0x0013
        value 0x02 0x18
        attr handle 0x0014, end group handle 0x0017
        value 0x02 0xa0

```

Figure 16.15 ATT procedures during GATT primary service discovery.

```
characteristics start_handle end_handle
```

The output of this command is shown in Figure 16.16.

Table 16.2 shows the following characteristics for service <>Generic Access Profile>> (UUID=0x1800).

```

[CON] [00:1B:DC:05:C8:D6] [LE]> characteristics 0x0001 0x0008
[CON] [00:1B:DC:05:C8:D6] [LE]>
handle: 0x0004, char properties: 0x02, char value handle: 0x0006, uuid:
00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0007, char properties: 0x02, char value handle: 0x0008, uuid:
00002a01-0000-1000-8000-00805f9b34fb
[CON] [00:1B:DC:05:C8:D6] [LE]>

```

Figure 16.16 Getting the characteristics of a service.

Table 16.2 Characteristics of Generic Access Profile

S. No	Characteristic UUID	Characteristic Name	Characteristic Handle	Characteristic Properties	Characteristic Value Handle
1	2a00	<>Device Name>>	0x0004	0x02	0x0006
2	2a01	<>Appearance Characteristic>>	0x0007	0x02	0x0008

Note that the Characteristic UUID is mapped to Characteristic Name in the Bluetooth Assigned Numbers. (See Reference [6]). Similarly, the characteristics of each of the service can also be retrieved. Figure 16.17 shows the commands used to retrieve the characteristics of all the services. The meaning of each of the characteristics is shown in Table 16.3.

A few points worth noting in the table are:

1. The Generic Access Service contains two characteristics—Device Name and Appearance Characteristic. Both these services were explained in detail in Chapter 14.
2. The Generic Attribute Service does not contain any characteristic in this example. It can only contain one characteristic—Service Changed Characteristic which was explained in Chapter 13. In this particular device this characteristic is missing. This means that the list of services supported by this device cannot change. In general, if this service is absent for an LE only implementation, this would have given information to the clients that the services cannot change over the entire lifetime of the device.

```
[CON] [00:1B:DC:05:C8:D6] [LE]> characteristics 0x0001 0x0008
[CON] [00:1B:DC:05:C8:D6] [LE]>
handle: 0x0004, char properties: 0x02, char value handle: 0x0006, uuid:
00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0007, char properties: 0x02, char value handle: 0x0008, uuid:
00002a01-0000-1000-8000-00805f9b34fb
[CON] [00:1B:DC:05:C8:D6] [LE]> characteristics 0x0009 0x000b
[CON] [00:1B:DC:05:C8:D6] [LE]>
handle: 0x000a, char properties: 0x0a, char value handle: 0x000b, uuid:
00002a06-0000-1000-8000-00805f9b34fb
[CON] [00:1B:DC:05:C8:D6] [LE]> characteristics 0x000c 0x000f
[CON] [00:1B:DC:05:C8:D6] [LE]>
handle: 0x000d, char properties: 0x12, char value handle: 0x000e, uuid:
00002a07-0000-1000-8000-00805f9b34fb
[CON] [00:1B:DC:05:C8:D6] [LE]> characteristics 0x0010 0x0010
[CON] [00:1B:DC:05:C8:D6] [LE]>
[CON] [00:1B:DC:05:C8:D6] [LE]> characteristics 0x0010 0x0010
[CON] [00:1B:DC:05:C8:D6] [LE]>
[CON] [00:1B:DC:05:C8:D6] [LE]> characteristics 0x0011 0x0013
[CON] [00:1B:DC:05:C8:D6] [LE]>
handle: 0x0012, char properties: 0x04, char value handle: 0x0013, uuid:
00002a06-0000-1000-8000-00805f9b34fb
[CON] [00:1B:DC:05:C8:D6] [LE]> characteristics 0x0014 0x0017
[CON] [00:1B:DC:05:C8:D6] [LE]>
handle: 0x0015, char properties: 0x12, char value handle: 0x0016, uuid:
0000a003-0000-1000-8000-00805f9b34fb
[CON] [00:1B:DC:05:C8:D6] [LE]> characteristics 0x0022 0x002c
[CON] [00:1B:DC:05:C8:D6] [LE]>
handle: 0x0025, char properties: 0x02, char value handle: 0x0026, uuid:
0000a006-0000-1000-8000-00805f9b34fb
handle: 0x0029, char properties: 0x02, char value handle: 0x002a, uuid:
0000a009-0000-1000-8000-00805f9b34fb
[CON] [00:1B:DC:05:C8:D6] [LE]> characteristics 0xffffa 0xffffe
[CON] [00:1B:DC:05:C8:D6] [LE]>
handle: 0xfffffc, char properties: 0x02, char value handle: 0xffffd, uuid:
e9258cle-8962-c4b6-0b45-2c9018f28880
[CON] [00:1B:DC:05:C8:D6] [LE]>
```

Figure 16.17 Getting the characteristics of all services.

Table 16.3 List of All Characteristics within the Primary Services

S. No	Characteristic Handle	Characteristic UUID	Characteristic Name	Characteristic Properties	Characteristic Value Related Handle
<i>Generic Access Service [0x0001 – 0x0008]</i>					
1	0x0004	2a00	<<Device Name>>	0x02	0x0006
2	0x0007	2a01	<<Appearance Characteristic>>	0x02	0x0008
<i>Link Loss Service [0x0009 – 0x000b]</i>					
3	0x000a	2a06	<<Alert Level>>	0x0a	0x000b
<i>Tx Power Service [0x000c – 0x000f]</i>					
4	0x000d	2a07	<<Tx Power Level>>	0x12	0x000e
<i>Generic Attribute Service [0x0010 – 0x0010] – No Characteristics</i>					
<i>Immediate Alert Service [0x0011 – 0x0013]</i>					
5	0x0012	2a06	<<Alert Level>>	0x04	0x0013
<i>a002 [0x0014 – 0x0017]</i>					
6	0x0015	a003		0x12	0x0016
<i>a004 [0x0022 – 0x002c]</i>					
7	0x0025	a006		0x02	0x0026
8	0x0029	a009		0x02	0x002a
<i>fee74dc-a8de-3196-1149-d43596c00a4f [0xffffa – 0xffffe]</i>					
9	0xffffc	User defined 128-bit UUID		0x02	0xffffd

3. The meaning of the different bits in the bit-map of characteristic values was explained in Chapter 13. For example:
 - a. 0x02 means that only read is permitted on the Characteristic Value.
 - b. 0x4 means that only write without response is permitted.
 - c. 0x0a means that the Characteristic Value can be read and written.
 - d. 0x12 means that it can be read and notified.
4. There are two instances of <<Alert Level>> characteristic:
 - a. First time as a part of Link Loss Service where it has permissions of 0x0a (read, write permitted).
 - b. Second time as a part of Immediate Alert Service where it has permissions of 0x04 (It can only be written). This was also explained in Chapter 15.
5. There are certain user defined UUIDs apart from the ones defined in Bluetooth Assigned Numbers.
6. The handle to the Characteristic Value is provided in the last column. This is the handle that would be used to access the characteristic provided there are sufficient permissions as per Characteristic Properties.

The transactions that are initiated on the ATT level for getting the characteristics are shown in Figure 16.15.

The following points may be noted:

1. The ATT Read_By_Type request is used to read the characteristics. It is executed with the following parameters:
 - a. Start Handle: 0x0001.
 - b. Ending Handle: 0x0008 (Handles that were received for the first service).
 - c. UUID: 0x2803 (This is the UUID for means that <>Characteristic>>).
2. The ATT Read_By_Type response returns the two characteristics.

16.5.3 Reading and Writing Characteristics

Now that the handles for all the characteristics values and characteristic permissions are available, the next step is to read and write these characteristics.

16.5.3.1 Reading and Writing Alert Level in Link Loss Service

This section will show how to read and write the Alert Level Characteristic in the Link Loss Service. This has an attribute handle of 0x000b and has Characteristic Properties of 0x0a. This means that this Characteristic Value can be read and written. This is in line with the description of Link Loss Service in Chapter 15.

The read and write operations are shown in Figure 16.19. The following may be noted:

1. The syntax of the command to read Characteristic Value is:

```
dualpc # hcidump -i hcii -X
HCI sniffer - Bluetooth packet analyzer ver 2.3
device: hcii snap len: 1028 filter: 0xffffffff
< ACL data: handle 39 flags 0x00 dlen 11
    ATT: Read By Type req (0x08)
        start 0x0001, end 0x0008
        type-uuid 0x2803
> HCI Event: Number of Completed Packets (0x13) plen 5
    handle 39 packets 1
> ACL data: handle 39 flags 0x02 dlen 20
    ATT: Read By Type resp (0x09)
        length: 7
        handle 0x0004, value 0x02 0x06 0x00 0x00 0x00 0x2a
        handle 0x0007, value 0x02 0x08 0x00 0x01 0x2a
```

Figure 16.18 ATT procedures for reading characteristics.

```
[CON] [00:1B:DC:05:C8:D6] [LE]> char-write-cmd 0x000b 0x01
[CON] [00:1B:DC:05:C8:D6] [LE]> char-read-hnd 0x000b
Characteristic value/descriptor: 01

[CON] [00:1B:DC:05:C8:D6] [LE]> char-write-cmd 0x000b 0x02
[CON] [00:1B:DC:05:C8:D6] [LE]> char-read-hnd 0x000b
Characteristic value/descriptor: 02
```

Figure 16.19 Reading and writing the Alert Level Characteristic in the Link Loss Service.

```
char-read-hnd <handle> [offset]
```

2. The syntax of the command to write Characteristic Value is:

```
char-write-cmd <handle> <new value>
```

3. The first command writes the value 0x01 to the Alert Level Characteristic. The command is:

```
char-write-cmd 0x000b 0x01
```

4. The second command reads the value of the Alert Level Characteristic. The command is:

```
char-read-hnd 0x000b
```

5. This returns the same value that was written. This confirms that the value was written correctly.
6. The next two commands do the same write and read operations with a value of 0x02.

The transactions that are initiated on the ATT level for reading and writing characteristics are shown in Figure 16.20. The following points may be noted:

1. The ATT Write command is used to write the characteristics. It is executed with the following parameters:
 - a. Handle: 0x000b.
 - b. Value: 0x0001.
2. The ATT Read request is used to read a characteristic. It is executed with the following parameters:
 - a. Handle: 0x000b.

```
< ACL data: handle 39 flags 0x00 dlen 9
  ATT: Write cmd (0x52)
        handle 0x000b value 0x00 0x01
> HCI Event: Number of Completed Packets (0x13) plen 5
  handle 39 packets 1


< ACL data: handle 39 flags 0x00 dlen 7
  ATT: Read req (0x0a)
        handle 0x000b
> HCI Event: Number of Completed Packets (0x13) plen 5
  handle 39 packets 1
> ACL data: handle 39 flags 0x02 dlen 6
  ATT: Read resp (0x0b)
        0000: 01
```

Figure 16.20 ATT procedures for reading and writing characteristics.

3. The remote side responds with a ATT Read response. It contains the following parameters:
 - a. Value: 0x0001.

16.5.3.2 Writing Alert Level in Immediate Alert Service

This section will show how to read and write the Alert Level Characteristic in the Immediate Alert Service. This has an attribute handle of 0x0013 and has Characteristic Properties of 0x04. This means that this Characteristic Value can only be written. This is in line with the description of Immediate Alert Service in Chapter 15.

The write operations are shown in Figure 16.21. The following may be noted:

1. The first command writes the value 0x01 to the Alert Level Characteristic. The command is:

```
char-write-cmd 0x0013 0x01
```

2. The second command writes the value of 0x02 to the Alert Level Characteristic. The command is:

```
char-write-cmd 0x0013 0x01
```

3. The third command writes the value of 0x00 to the Alert Level Characteristic. The command is:

```
char-write-cmd 0x0013 0x00
```

16.6 Disconnecting

16.6.1 Disconnecting the GATT Connection

The GATT connection can be disconnected by the following command:

```
disconnect
```

This is shown in Figure 16.22. Note that the CON message disappears in the second line after giving the disconnect command. This indicates that the connection is no longer there. The final command given is quit to come out the gatttool interactive interface.

```
[CON] [00:1B:DC:05:C8:D6] [LE]> char-write-cmd 0x0013 0x01
[CON] [00:1B:DC:05:C8:D6] [LE]>
[CON] [00:1B:DC:05:C8:D6] [LE]> char-write-cmd 0x0013 0x02
[CON] [00:1B:DC:05:C8:D6] [LE]>
[CON] [00:1B:DC:05:C8:D6] [LE]> char-write-cmd 0x0013 0x00
[CON] [00:1B:DC:05:C8:D6] [LE]>
```

Figure 16.21 Writing the Alert Level Characteristic in the Immediate Alert Service.

```
[CON] [00:1B:DC:05:C8:D6] [LE]> disconnect
[ ] [00:1B:DC:05:C8:D6] [LE]> quit
dualpc#
```

Figure 16.22 Disconnecting the GATT connection.

16.6.2 Disconnecting the LE Connection

The existing LE connection can be disconnected by the following command on *dualpc*:

```
hcitool -i hci1 ledc connection_handle
```

This is shown in Figure 16.23. The connection handle is that same that was returned while creating the LE connection in Figure 16.11. The HCI Disconnect command is used to terminate the connection.

16.7 Real-World Application—Find Lost Keys

The previous sections provided all the components that are needed to make a real world application to find lost keys. These components can be used as building blocks to make a full real world LE application.

An example of such an application can be one used to find lost keys (or any other lost devices). The example of this was explained in Chapter 1. Such an application will use the Find Me profile which was explained in Chapter 15.

The broad steps this application will need to perform are as follows:

1. Search for LE devices in the vicinity.
2. Select the device that has been lost.
3. Create a connection to the device.
4. Create a GATT connection.
5. Get the primary services.
6. Check if it contains the Immediate Alert Service.

```
dualpc# hcitool -i hci1 ledc 39
dualpc#
```

Disconnecting from dualpc

```
< HCI Command: Disconnect (0x01|0x0006) plen 3
    handle 39 reason 0x13
    Reason: Remote User Terminated Connection
> HCI Event: Command Status (0x0f) plen 4
    Disconnect (0x01|0x0006) status 0x00 ncmd 1
> HCI Event: Disconn Complete (0x05) plen 4
    status 0x00 handle 39 reason 0x16
    Reason: Connection Terminated by Local Host
```

HCI commands and events for disconnection

Figure 16.23 Disconnecting the LE connection.

7. If it contains the Immediate Alert Service, then find the characteristics contained in this service.
8. This will return the Characteristic Value Handle for the Alert Level characteristic.
9. Write a “High Alert” to the Characteristic Value handle.
10. The key fob should start raising an alert (Audio or Visual indication).
11. Disconnect the GATT and LE connection.

The commands to be sent for each of these steps (along with the parameters) were explained in previous sections. These can be put together into a script or a program to perform all the steps. Applications can also use services provided by BlueZ through the *D-Bus* interface. *D-bus* is a message bus system that can be used for interprocess communication. The applications can be written in either Python or C and are simple and straightforward to implement.

16.8 Debugging LE Applications

16.8.1 Logging the HCI Interface

The first and most common mechanism used for debugging Bluetooth applications is logging the packets exchanged on the HCI interface. This includes commands sent from the host to the controller and the events that are sent back by the controller.

BlueZ includes the *hcidump* tool which provides the facility to print command and event packets exchanged on the HCI interface. A similar tool may be available on the system on which the LE application is developed. If such a tool is not available, adding the facility to log the packets may be as easy as dumping (or printing) the packets that are sent and received.

At a very preliminary level, the packets sent and received can be analyzed to check if the transactions initiated by the application or the remote side were correct. For example, if the host sends an LE connection command, the packets at the HCI level can be checked to see if the command packet was sent with the correct values and to the correct device.

At a bit more advanced level, the packets can be decoded to print the friendly names of the commands along with meaning of each of the arguments that were passed. Besides this, the higher layer protocol data can also be decoded and used for debugging instead of just confining to the HCI interface. This facility is provided by the *hcidump* tool (examples of this were shown earlier.) For example, the ATT level transactions at the time of discovering services of the remote device were shown in Figure 16.15. The source code for *hcidump* is also available from the BlueZ website. This could be used as a reference for implementation in other environments as well.

At a still more advanced level, the packets could be converted to a format which a sniffer tool can open so that the packets can directly be opened in the sniffer tool. The sniffer can provide more detailed analysis of the packets which can be useful for debugging. One such format is the *BT-Snoop* file format. The conversion from the raw send/receive packets to the BT-Snoop format is straightforward and can be done using a simple script.

16.8.2 Air Sniffer

Air sniffer is a very powerful tool to understand what is happening on the air, and to quickly find the root cause of any problems. It provides several features including:

1. Analysis of the various packets that are sent over the air. The packets can be analyzed at various levels starting from the baseband layer up to the protocol layers. The various transactions along with the parameters are displayed in an easy to understand format.
2. The profile level data may also be extracted for separate analysis.
3. Display of the used/unused channels.
4. Message sequence charts of the various transactions.
5. Facility to save logs so that these can be reopened for further analysis later on.

These are only some of the features provided by the air sniffers. Most of the sniffers available in the market provide several other advanced features that can simplify the LE development a great deal.

16.8.3 Peer Devices and Interoperability Testing

Bluetooth communications require at least two devices to interact with each other. In many of the scenarios, the application writer or device manufacturer may be focusing only on one of the roles. For example, if an LE key fob is being made, then the application writer needs to focus on the key fob functionality only which may include the Immediate Alert Service. Though the peer functionality is also important from a testing perspective, the application writer may not need to spend time on writing that functionality from scratch. Instead of that, a peer device which already has this functionality can be used. In this particular example, a Linux based PC can be used as a peer device to test all the functionality of the key fob. A good selection of peer device can help in speedier development of applications and also to ensure that the device is exhaustively tested before releasing to the market.

Once the device is functioning well, the set of peer devices can be increased for getting further testing coverage. This is known as *interoperability testing* where the device is tested with several other devices to ensure that it works well with all those devices. It will be explained in detail in the next chapter.

16.8.4 Profile Tuning Suite (PTS)

The Profile Tuning Suite is a powerful, PC-based, black box testing tool provided by the Bluetooth SIG. It can be used during product development, testing, and qualification stages. It has support for most of the BR/EDR profiles and several LE profiles as well. It is semi-automated which leads to reduction of testing time and also ensuring that novice users can ramp-up in using this tool faster. It provides an ample amount of debugging information along with message sequence charts (MSCs) which can be useful for debugging the LE implementation. The PTS tool will be covered in further detail in the next chapter.

16.9 Disclaimer

The examples provided here are only for educational purposes to illustrate the various LE operations. These may not work or may have unpredictable results. So you are advised to use them at your own risk. To make them suitable for commercial needs several enhancements, error checks, and exhaustive testing would be required.

16.10 Summary

The previous chapters explained the various components of the LE architecture including the Link Manager, HCI, L2CAP, ATT, GATT, GAP, and GATT-based profiles. This chapter illustrated how these various components come together for an end-to-end use case like finding lost keys.

Besides the commands and events needed to perform various operations, this chapter also showed the various transactions that are happening ‘behind the scenes’ on the HCI level and ATT level. This would give a good overview on how the various layers interact with each other. Lastly this chapter also covered some of the tools and techniques that could be useful during development for debugging the LE applications. The next chapter will cover another aspect about developing LE applications, which is the testing and qualification of these applications.

References

- [1] Bluetooth Core Specification 4.0 <http://www.bluetooth.org>.
- [2] Bluetooth SIG, Specifications of the Bluetooth System, Profiles <http://www.bluetooth.org>.
- [3] BlueZ website (<http://www.bluez.org>).
- [4] Bluetooth Assigned Numbers for Generic Attribute Profile (<http://www.bluetooth.org/Technical/AssignedNumbers/Generic-Attribute-Profile.htm>).
- [5] Bluetooth Assigned Numbers for GATT-based service UUIDs (<http://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx>).
- [6] Bluetooth Assigned Numbers for Characteristic Descriptions (<http://developer.bluetooth.org/gatt/characteristics/Pages/default.aspx>).

Testing and Qualification

17.1 Introduction

One of the strong points about the Bluetooth SIG besides the specification, is that it provides a full-fledged qualification program along with tools for testing and qualifying devices. The test tools include the Bluetooth test specifications, test vectors, as well as software to test the Bluetooth implementation.

It is mandatory for all devices to undergo Bluetooth qualification before these can be sold in the market. The qualification program is designed so that a majority of the testing can be done by the developers on their own.

In addition, the Bluetooth SIG regularly conducts events where developers from all over the world can come together and test their products.

17.2 Need for Bluetooth Qualification

Bluetooth qualification ensures that the device conforms to the Bluetooth specifications. It assures product quality, conformance to the standards, and is an important mechanism to ensure interoperability between devices from different vendors. Bluetooth qualification is one of the major reasons behind the huge success of Bluetooth technology. When consumers buy Bluetooth products, they are assured that these products will work with other Bluetooth products in the market.

Bluetooth qualification allows a manufacturer to use the Bluetooth trademark and logos and is the precondition of the intellectual property license for use of Bluetooth wireless technology. This is mandatory before the product can be sold in the market.

First and foremost, the device manufacturer needs to sign up for Bluetooth SIG membership. SIG membership provides access to Bluetooth resources and specifications to build Bluetooth products and the license to use the Bluetooth trademark and logos. The membership is open to all companies that are wishing to develop, market, and promote Bluetooth products.

Once a product has been qualified, it gets listed on the SIG website among the Bluetooth Qualified Products List and is assigned a QDID (Qualification Design ID). Anybody can look up the Bluetooth website to check the set of features that

the product was qualified against by getting the listing details of the QDID of that product. (In fact a QDID can be bought from the SIG at the product concept stage and can be used throughout the product development lifecycle to define features and test plans). The Bluetooth SIG has a comprehensive and well defined qualification program which provides step-by-step guide to the various requirements and phases during qualification.

17.3 What Is Interoperability?

In simple terms, interoperability means that a device from one vendor should work seamlessly with device from another vendor. So, for example, a mobile phone from one vendor should work with any headset, car kit, thermometer, or glucose meter irrespective of who manufacturers them.

For the consumers, this means that they don't get restricted to one particular vendor for all their Bluetooth products. They can buy Bluetooth products just about anywhere in the world without worrying whether those products will be compatible with their existing Bluetooth devices. The consumers buying decisions can focus more on the product appeal, features, and price than on the technical jargon and compatibility concerns.

For the developers this means ensuring full compliance to the specifications. The specifications are written in such a manner that if these are properly complied to, then interoperability is inherently guaranteed. The profiles define very clear roles for the devices that need to interact along with how each device is expected to behave. The specifications even define the terms to be used at the User Interface (UI) level for a uniform user experience. So if a heart rate monitor has to interact with a laptop, the roles and responsibilities of both the heart rate monitor and application running on the laptop are clearly documented.

17.4 Development Resources and Events

Bluetooth SIG hosts several events throughout the year in different parts of the world. These include UnPlugFest, Round Table meetings, All Hands Meetings, Automotive events, and several more. Besides this, the Bluetooth SIG hosts mailing lists, working groups, and discussion forums and provides ample of resources to assist during product development and testing.

Bluetooth SIG has working groups and SIG committees where people can follow what is happening as well as actively contribute to developing the technology further.

17.4.1 UnPlugFest (UPF)

UnPlugFests are interoperability testing events organized by the Bluetooth SIG three times a year, once in each region (Asia, North America, and Europe). The participants can register for the event on the SIG website after paying the participation fee. During registration, the participants register the platform and provide information

on the set of tests that they plan to run. The set of tests are broadly divided into three categories:

1. Category-1: This includes the tests for the Bluetooth Lower layers (below HCI) including RF, Baseband and link manager.
2. Category-2: This includes the test cases for different protocol layers—SDP, RFCOMM, TCS, BNEP, AVDTP, AVCTP, ATT, GATT, GAP, SM, etc
3. Category-3: This includes the tests at a profile level. Such tests usually involve the complete system including the application. The testing includes both BR/EDR profiles as well as GATT-based profiles and services for LE.

In addition, there are special testing categories which focus on things like multi-profile test suites.

During the event participants are allocated test slots with other participants depending on Category/Profiles selected at the time of registration. During the test slot the participants test and debug the functionality of the profiles that they had selected against a device from another participant. Once a slot is over, they move to the next slot with a different participant. This way they get an opportunity to test their implementation with several other implementations. There are also special debug slots which allow the developers to retest bug fixes or issues found during regular testing hours.

Certain events also feature robustness testing where the ability to withstand scenarios like buffer overflow, receiving corrupt packets, misbehavior of the peer device, etc. can be tested to ensure that the product will not break under extreme conditions. The event serves as an excellent mechanism for interoperability testing. This is particularly useful in the case of new protocols and profiles where the number of available off-the-shelf devices may be very limited.

17.4.2 Automotive Events

The use of Bluetooth devices in automotive applications is increasing. There are several profiles like HandsFree, A2DP, AVRCP, MAP, PBAP, SAP, Proximity which can be used in automobiles. The automotive event is specially focused to cater to the increasing need of testing mobile devices with automotive accessories.

In addition there are several private car kits events organized by various groups where interoperability testing for car kits can be carried out. These events are either invitation-only or open for registration. The details of the forthcoming events can be looked up on the Bluetooth SIG website.

17.4.3 SIG Device Library

The Bluetooth SIG maintains a library of devices that can be used for interoperability testing. The SIG has a program where developers can donate their devices to the SIG with those devices then becoming part of the device library. Other developers can then use those devices for interoperability testing. This library is continuously growing with the addition of new devices that support the latest protocols and profiles. This is useful especially when off-the-shelf devices are not available in the market for testing.

17.4.4 Profile Tuning Suite (PTS)

Profile Tuning Suite (PTS) is a PC-based test tool that is provided by the Bluetooth SIG for testing the Bluetooth implementation. PTS helps to automate the testing environment which helps both in getting results faster and getting more accurate results. Execution of PTS tool requires a PTS radio module (USB Dongle) that is attached to the PC. After that the PTS software can be run on the PC to test the Implementation Under Test (IUT). The PTS radio module and the PTS software can be ordered from the Bluetooth SIG website. A typical PTS setup is shown in Figure 17.1.

PTS supports testing most of the full range of Bluetooth Protocols and Profiles including LE profiles. Depending on the role that is being tested, PTS acts as an initiator or acceptor. During testing some operations may also need to be performed on the IUT (for example, accept/reject an incoming connection, initiate/accept file transfer, etc). The IUT needs to support these operations in order to utilize the full power of this tool.

PTS is very useful during development and debugging because it provides complete message sequence charts and debug logs of the communication with the IUT. It also allows the developer to generate and save XML based test reports. Once the tests pass, these reports can be used during the qualification process as an evidence of compliance with the specification. PTS includes informative messages at each step on the operations that need to be performed (e.g., initiate a pairing from the remote device). This allows even new users to quickly understand the tool and start using it effectively.

17.5 Bluetooth Qualification Program

The purpose of the Bluetooth Qualification Program is to promote interoperability, verify conformance to the Bluetooth specifications, enforce compliance, grant IP license, and recognize members who meet a high standard of testing. One of the reasons for the huge success of the Bluetooth technology is this qualification program. Once a user buys a Bluetooth certified product, he or she is assured that his product will be interoperable with other Bluetooth certified products.

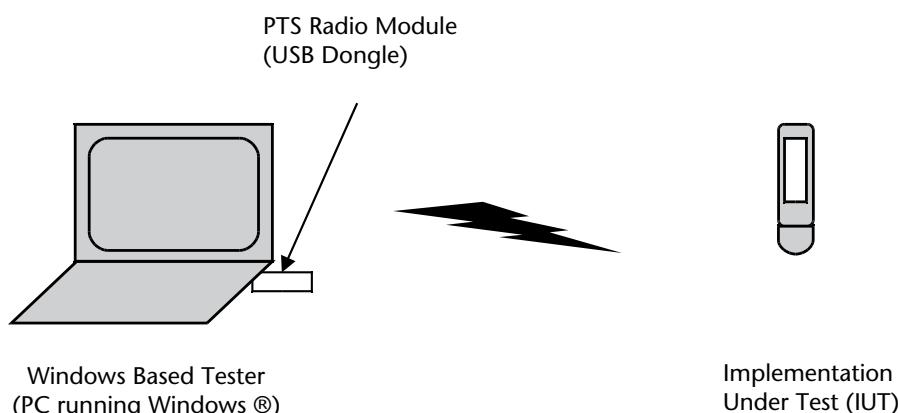


Figure 17.1 PTS setup.

The qualification process encompasses creating the test plans based on the features supported by the device, executing the test cases, and then providing evidence to prove that the test cases are running correctly. The test cases involve a set of mandatory and optional tests for each component (protocol stack layers, profiles, etc). The mandatory test cases define the bare minimum scenarios which need to be passed. On top of that, based on the features supported by the product, additional optional tests can be executed.

The main focus is to ensure compliance which includes not only the end Pass/Fail status but also adherence to the message sequence charts to ensure that all the steps are in the correct sequence. It is mandatory for a device to undergo Bluetooth qualification before being sold in the market. Once a device is qualified, it gets listed on the Bluetooth website as a part of End Product Listing (EPL).

The Bluetooth SIG provides an online interface where the developers can create their own test plans depending on the features supported by the device and then submit the test results. The broad outline of the qualification process is shown in Figure 17.2. A product undergoes four broad phases before it can be qualified:

1. Concept: This entails logging on the Bluetooth Qualification page and creating a project by entering the project name and expected qualification date. Bluetooth SIG provides several resources like online tutorials, training material, white papers, specification documents, etc. to assist during this phase.
2. Prototype: This phase includes defining which Bluetooth protocol and profile features the end product will support and generating a test plan based on those features. During this phase, the member starts building a design based on the concept that was defined in the previous phase. There are various events like UnPlugFests which can be used to test the prototype. Tools like PTS can also be used for testing the prototype.
3. Testing: This includes testing the prototype using tools like PTS and submitting the test results. Any bugs found during this phase are fixed and retesting is done to ensure that there are no side effects of the fixes.
4. Qualification: This includes signing a declaration of compliance and paying the requisite certification fee to get the product listed on the Bluetooth website. Qualification includes obtaining IP license protection and permissions to use the Bluetooth trademark and logos.

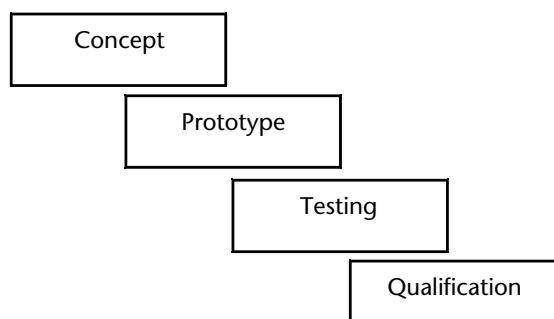


Figure 17.2 Overview of the Bluetooth Qualification Program.

The qualification process can be carried out by the device manufacturers solely or with assistance from third parties known as Bluetooth Qualification Test Facility (BQTF). Some of the tests require involvement of a BQTF during the qualification process.

17.6 Test Categories

The test cases for Bluetooth qualification have been split into 4 categories. Each category has a well-defined set of requirements on the test equipment used for running the test cases and the evidence needed to prove that the test cases are passing.

- Category A test cases focus on areas such as RF, Baseband, Link Manager, and HCI. It is mandatory to perform these test cases at a SIG authorized test facility (BQTF or BRTF: These will be introduced shortly).
- Category B test cases can be tested at the member's end using a member defined test setup as per the test requirements specified in the test specifications. The member can submit evidence of passing these test cases and does not necessarily take the test equipment to a test facility. This evidence includes information about the test setup, execution, and results.
- Category C test cases can be tested at the member's end using a test setup similar to Category B. For these test cases, the member just has to provide a declaration of the test performed and test results. Details regarding the test setup and execution are not needed.
- Category D test cases are informative and optional. These can be run on member-defined test equipment and no evidence needs to be submitted.

17.6.1 BQTF, BRTF

A Bluetooth Qualification Test Facility (BQTF) and a Bluetooth Reference Test Facility (BRTF) are testing labs that are formally recognized by the Bluetooth SIG as competent to perform the Bluetooth qualification conformance tests. The main difference between the two is that a BRTF is authorized to do testing for the member's own company while BQTF provides testing services to all member companies.

The “Category A” tests require special validated and commercially available test equipment. Some of these tests including the tests in the areas like RF, Baseband, Link Manager, and HCI are required to be performed at a BQTF or BRTF.

In addition to these tests, BQTF also provides assistance during the qualification process including qualifying the complete protocol stack and profiles. These facilities provide services beyond qualification such as more exhaustive testing of the device, interoperability testing, etc.

17.6.2 BQE

Bluetooth Qualification Expert (BQE) is an individual who is recognized by the Bluetooth SIG to provide qualification related services to the SIG members. The

BQE is a subject matter expert for all matters related to qualification including understanding the qualification process, and requirements for qualification. A BQE service includes checking declarations and documents against the qualification and test requirements, reviewing all test reports, and providing assistance to get the product listed.

17.6.3 Test Documents

Bluetooth SIG provides a comprehensive set of documents to test each protocol and profile. These are explained briefly below.

17.6.3.1 Test Specifications

The Bluetooth test specifications describe the test cases in detail including information about the purpose, message sequence charts and verdicts for each of the test cases. These provide sufficient information to the members to carry out the tests on their own and to find out whether the test cases pass or fail. These help the members to ensure that all test cases are passing before they start a formal qualification cycle of their product.

The test cases include testing for both valid behavior and invalid behavior.

- Valid Behavior (BV) tests are used to ensure that the implementation works properly when it receives a valid message.
- Invalid Behavior (BI) tests are used to test whether the implementation works properly even when it receives an invalid message.

17.6.3.2 Test Case Reference List (TCRL)

The TCRL is a reference document that contains the complete list of test cases for each protocol or profile. This is a living document that is continually upgraded with new test cases added, old/invalid/obsolete test cases removed, and updates including newer versions of the specifications. It is a big spreadsheet which has tabs for different entities. Each tab provides the list of test cases that are defined for that entity.

17.6.3.3 Implementation Conformance Statement (ICS)

The ICS provides a standard means to define the Bluetooth capabilities supported by a product. It is a template which contains the list of test cases and whether those test cases are Mandatory, Optional or Conditional. The member can fill in this statement to indicate the capabilities of the implementation to be qualified.

17.6.3.4 Implementation Extra Information for Testing (IXIT)

The IXIT typically contains information about the test environment including physical setup, conditions for running the test cases, configuration parameters, etc.

17.6.4 Prequalification

Prequalification refers to the internal tests that a vendor does before going for qualification. This process involves exhaustive testing of the device against the set of test cases planned to be finally run during the qualification round. If any bugs are found, those are fixed. The tests are then rerun to ensure there are no side effects of the fixes. The output of the prequalification phase is a complete set of passing results which indicate that the device is now ready for qualification.

17.7 Summary

Bluetooth SIG has a very comprehensive qualification program to ensure that a product meets all requirements before it can be sold in the market. The Bluetooth SIG also offers tools and services which are very helpful during the development of the product to help improve product interoperability and smooth qualification. These tools help in debugging the product to reduce both the development cost and cycle time.

The Bluetooth SIG also conducts regular events where the developers can participate to test their product. These events are particularly helpful to test implementations where the peer devices are not easily available.

Besides Bluetooth qualification, since the Bluetooth devices use ISM band radios, the Bluetooth products may also need to pass regulatory tests to ensure that they comply with the government regulations for transmission in the ISM band. Different countries may have different requirements about these regulatory tests.

Bibliography

Information available on the Bluetooth SIG website. <http://www.bluetooth.org>.
Bluetooth Qualification Program Reference Document (PRD).

Glossary of Acronyms and Important Terms

ACL	Asynchronous Connection Oriented. (Due to legacy reasons it is abbreviated as ACL)
ACL-C	ACL Control
ACL-U	ACL User
AD	Advertising Data
AES	Advanced Encryption Standard
AFH	Adaptive Frequency Hopping
AM_ADDR	Active Member Address
AMP	Alternate MAC PHY
ANP	Alert Notification Profile
ANS	Alert Notification Service
ARQ	Acknowledgement/Repeat Request
ARQN	Unnumbered ARQ
ATT	Attribute Protocol
AVCTP	Audio/Video Control Transport Protocol
AVDTP	Audio/Video Distribution Transport Protocol
BAS	Battery Service
BB	Baseband
BD_ADDR	Bluetooth Device Address
BER	Bit Error Rate
BLE	Bluetooth Low Energy. Also referred to as LE.
BLP	Blood Pressure Profile
BLS	Blood Pressure Service
BT	Bandwidth Time
CT	Controller (Used in context of AVCTP)
CAC	Channel Access Code
CLK	Clock
CODEC	Coder Decoder
CRC	Cyclic Redundancy Check
CSRK	Connection Signature Resolving Key
CVSD	Continuous Variable Slope Delta Modulation
DAC	Device Access Code
DH	Data—High Rate

DIAC	Dedicated Inquiry Access Code
DIS	Device Information Service
DIV	Diversifier
DM	Data—Medium Rate
DPSK	Differential Phase Shift Keying
DV	Data-Voice Packet
ECDH	Elliptic Curve Diffie Hellman
EDIV	Encrypted Diversifier
EDR	Enhanced Data Rate
EIR	Extended Inquiry Response
ER	Encryption Root
eSCO	Extended Synchronous Connection Oriented
ETSI	European Telecommunications Standards Institute
FCC	Federal Communications Commission
FCS	Frame Check Sequence
FDMA	Frequency Division Multiple Access
FEC	Forward Error Correction Code
FHS	Frequency Hop Synchronization
FHSS	Frequency Hopping Spread Spectrum
FM	Frequency Modulation
FMP	Find Me Profile
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GFSK	Gaussian Frequency Shift Keying
GIAC	General Inquiry Access Code
GLP	Glucose Profile
GLS	Glucose Service
HCI	Host Controller Interface
HIDS	HID Service
HOGP	HID Over GATT Profile
HRP	Heart Rate Profile
HRS	Heart Rate Service
HTP	Health Thermometer Profile
HTS	Health Thermometer Service
HV	High Quality Voice
IAC	Inquiry Access Code
IAS	Immediate Alert Service
ID	Identity
IEEE	Institute of Electronic and Electrical Engineers
IETF	Internet Engineering Task Force
IR	Identify Root
IrDA	Infrared Data Association (http://www.irda.org)
IRK	Identity Resolution Key
IrOBEX	OBject EXchange protocol that is maintained by the Infrared Data Association (IrDA)
ISM	Industrial, Scientific and Medical
ISO	International Organization for Standardization
IUT	Implementation Under Test

IV	Initialization Vector
L2CAP	Logical Link Control and Adaptation Protocol
LAP	Lower Address Part
LC	Link Control or Link Controller
LE	Bluetooth Low Energy
LIAC	Limited Inquiry Access Code
LLCP	Link Layer Control Protocol
LLID	Link Layer ID
LLS	Link Loss Service
LM	Link Manager
LMP	Link Manager Protocol
LSB	Least Significant Bit
LSTO	Link Supervision Timeout
LTK	Long Term Key
MMI	Man Machine Interface
MIC	Message Integrity Check
MITM	Man In The Middle
MMI	Man Machine Interface
MSB	Most Significant Bit
MSC	Message Sequence Chart
MTU	Maximum Transmission Unit
NAK	Negative Acknowledgement
NESN	Next Expected Sequence Number
NIST	National Institute of Standards and Technology
NFC	Near Field Communication
OBEX	OBject EXchange
OCF	OpCode Command Field
OGF	OpCode Group Field
OOB	Out of Band
OUI	Organizationally Unique Identifier
PCB	Printed Circuit Board
PIN	Personal Identification Number
PDU	Protocol Data Unit
PM_ADDR	Parked Member Address
POS	Point Of Sale
PPM	Parts Per Million
PSK	Phase Shift Keying
PSM	Protocol/Service Multiplexer
PXP	Proximity Profile
QOS	Quality Of Service
RAND	Random Number
RF	Radio Frequency
RFC	Request For Comments
RFU	Reserved for Future Use
RSSI	Received Signal Strength Indication
RTC	Real Time Clock
SAR	Segmentation And Reassembly
SCO	Synchronous Connection Oriented

SDP	Service Discovery Protocol
SEP	Stream End Point (Used in context of AVDTP)
SEQN	Sequential Numbering Scheme
SM	Security Manager
SMP	Security Manager Protocol
SN	Sequence Number
SRK	Signature Resolving Key
SSP	Secure Simple Pairing
STK	Short Term Key
TDD	Time Division Duplex
TDMA	Time Division Multiple Access
TG	Target (Used in context of AVCTP)
TK	Temporary Key
TPS	Transmit Power Service
UART	Universal Asynchronous Receiver Transmitter
UI	User Interface
TX	Transmit
USB	Universal Serial Bus
UTF-8	8-bit UCS/Unicode Transformation Format
UUID	Universally Unique Identifier

About the Author

Naresh Gupta holds a bachelor of engineering degree in computer engineering from Delhi College of Engineering, one of the premier institutes in India, along with an MBA in operations. He has close to two decades of industry experience, including more than 11 years of experience with Bluetooth technology.

Naresh started working on Bluetooth in 2001 as a part of HCL Technologies when he was requested to lead a project related to Bluetooth protocol stack. This project involved development of the complete Bluetooth protocol stack from scratch and porting it on embedded operating systems like microITRON, VxWorks, and Windows CE.

Thereafter he moved to STMicroelectronics and ST-Ericsson, where he led development activities related to connectivity technologies including Bluetooth and Bluetooth Low Energy. He and his team were involved with implementing the latest evolutions in wireless connectivity technologies. He worked with STMicroelectronics and ST-Ericsson for close to 9 years. He is currently in a senior management role at another multinational technology company.

Naresh has been actively involved in delivering guest lectures, conducting semester courses, and organizing workshops at some of the premier educational institutes in India. He has traveled extensively across the globe to conduct Bluetooth training programs and workshops. He has filed several patents on wireless technologies and published papers in international journals.

You may find him on <http://in.linkedin.com/in/guptanaresh>.

Index

A

3GPP (3rd Generation Partnership Program), 4, 14, 109, 118
Acceptor
 Generic Audio Video Distribution Profile (GAVDP), 113
 Profile Tuning Suite (PTS), 376,
Access Address, 139, 161, 175
Asynchronous Connection Oriented (ACL)
 ACL, ACL-C, ACL-U, Logical Link, Channel, 40, 44, 45, 56, 61, 80, 81, 83, 84, 86, 105, 113, 114
Buffers, Buffer Size, 67, 68, 188, 190, 204, 355
Data Packet, 47, 48, 62, 63, 66, 69, 70, 86, 186, 187, 201, 257, 352, 362, 363
Logical Transport, 41, 42, 53, 55, 58
Packet Types, 46, 49,
Active Scanning, 172–173, 197, 315, 357
Adaptive Frequency Hopping (AFH), 20, 36–40, 58, 76, 156, 178
Adopted Protocols, 22, 23, 79, 93, 96
Advanced Audio Distribution Profile (A2DP), 45, 88, 92, 99, 101, 114, 335, 375
Advertiser, 160
Advertiser Address (AdvA), 168–171, 173, 175, 310
Advertising Data (AD, AdvData), 168, 189, 275, 309
Advertising, 149, 154, 158, 160, 356, 196, 214, 317, 356, 358
Advertising Channel PDU, 162
Advertising Channels, 136, 143, 155–157, 160, 173
Advertising Channel Index, 155, 156
Advertising Events, 157, 158, 168, 172, 306, 307, 309, 315, 316, 318–320

Advertising Enable, Disable, 192, 196, 323, 358
Advertising Filter Policy, 182, 195
Advertising Indication, 167
Advertising Parameters, 192, 195, 196, 199, 323
Advertising PDU, Advertising Packets, Format, 160, 163, 316
Advertising Physical Channel, *See* Advertising Channels
Advertising Report, Advertising Packets, 65, 159, 193, 321–323, 358
Advertising State, 151, 152, 153, 166, 167, 357
Air Sniffer, 87, 88, 227, 371
Alert Notification Profile (ANP), 260, 345
Alert Notification Service (ANS), 345
Anchor Point, 158, 159, 207
ANT, ANT+, 2, 3, 12
Appearance Characteristic, 312, 313, 363, 364, 365
Architecture
 Asymmetrical, *See* Asymmetrical Architecture
 Bluetooth, 21–24
 Data Transport, 40
 GATT Based, 261, 262
 LE, 140–142
Association Models, 72
Asymmetrical Architecture, 212
Atomic Operations, 237
ATT_MTU, 241, 245, 246, 275, 276, 289, 290, 294, 298, 299, 301
Attribute
 Caching, 269
 Client, 237, 238
 Definition, 267, 268
 Grouping, 237, 270

- Attribute (continued)
 Handle, 234, 235, 237, 267, 268
 Permissions, 234, 235, 268, 273
 Server, 237, 238
 Structure, 234, 268
 Type, 234, 235, 237, 267, 278
 Value, 89, 90, 234, 236
- Attribute Protocol (ATT), 233, 237
- Attribute Protocol PDU, 238
- Audio Gateway (AG), 109
- Audio/Video Control Transport Protocol (AVCTP), 97
- Audio/Video Distribution Transport Protocol (AVDTP), 98
- Audio/Video Remote Control Profile (AVRCP), 116
- Authenticated MITM Protection, 215
- Authentication, 60, 101, 106, 194, 211, 235, 236, 304, 308, 327
- Authentication Permissions, 235
- Authentication Procedure, 328
- Authentication Signature, 238, 239, 251, 292
- Authorization, 268, 304
- Authorization Procedure, 328
- Authorization Permissions, 236
- Automatic Gain Control (AGC), 161
- Automotive Events, 375
- B**
- Baseband Controller, 35
- Basic L2CAP Mode, 82, 205
- Battery Service (BAS), 339, 340
- Bit Stream Processing, 165
- Blood Pressure Profile (BLP), 344
- Blood Pressure Service (BPS), 343
- Bluetooth Architecture, *See* Architecture
- Bluetooth Clock, 27, 28
- Bluetooth Device Address (BD_ADDR), 26, 35, 64, 74, 75, 102, 153, 227, 307
- Bluetooth Device Inquiry, 29, 66, 74, 104, 121
- Bluetooth Device Name, 26, 27, 102, 105, 121, 308, 310, 311
- Bluetooth Passkey, 59, 102, 305, 308
- Bluetooth PIN, *See* Bluetooth Passkey
- Bluetooth Qualification Expert (BQE), 378
- Bluetooth Qualification Program, 21, 376
- Bluetooth Qualification Test Facility (BQTF), 378
- Bluetooth Radio, 33, 101
- Bluetooth Reference Test Facility (BRTF), 378
- Bluetooth SIG, 4, 17, 18, 373
- Bluetooth Smart, 134
- Bluetooth Smart Marks, 133
- Bluetooth Smart Ready, 134
- BlueZ, 119, 349
- Body Area Network (BAN), 3, 19
- Bondable Mode(s), 103, 324–327
- Bonding, 103, 105, 308
- Bonding Modes, 326
- Bonding Modes and Procedures, 324, 326
- BQE, *See* Bluetooth Qualification Expert
- BQTF, *See* Bluetooth Qualification Test Facility
- Broadcast Mode, 315
- Broadcaster, 306, 315, 320
- Browsing Services, 90, 121
- BRTF, *See* Bluetooth Reference Test Facility
- C**
- CEN, *See* European Commission for Standardization
- Central, 307
- Channel Access Code (CAC), 46
- Channel Classification, 38, 189, 194
- Channel Establishment, 105, 106
- Channel Identifier (CID), 81–83, 87, 203, 204, 222
- Channel Map, 38, 48, 156, 189, 194
- Channel Map Update Procedure, 178
- Channel Multiplexing, 83, 204
- Characteristic, 237, 261–265, 311, 333–335, 362
- Characteristic Declaration, 265, 273
- Characteristic Definition, 265, 271
- Characteristic Descriptor Discovery, 286
- Characteristic Descriptors, 265, 266, 298
- Characteristic Discovery, 281
- Characteristic Properties, 273, 274
- Characteristic UUID, 273
- Characteristic Value Attribute Handle, 273
- Characteristic Value Declaration, 273

Characteristic Value Indication, 275, 297
Characteristic Value Notification, 275, 297
Characteristic Value Read, 275, 288
Characteristic Value Write, 275, 292
Characteristic Values Reliable Writes, 275
Class of Device (CoD), 27, 102, 312, 354
Clock, *See* Bluetooth Clock
Command Reject, 87, 203, 207
Command Type Methods, 250
Comparison between BR/EDR and LE, 141
Configured Broadcast, 275
Confirm value generation, 216
Confirmation, 238, 239, 252
Confirmation Type Methods, 252
Connectable Directed, 163, 169, 172
Connectable Undirected, 163, 167, 309, 316
Connection Establishment, 51, 57, 75, 106, 198, 320–324
Connection Establishment Substates, 51, 52
Connection Event Interval (connInterval), 158, 207
Connection Events, 158
Connection Modes and Procedures, 318, 319
Connection Parameter Update Procedure, 323, 325
Connection Parameter Update Request, 87, 206, 207
Connection Parameter Update Response, 87, 206, 207
Connection Setup, 29, 193, 204
Connection Signature Resolving Key (CSRK), 221
Connection State, 51, 153, 166, 174, 193
Connection Update Procedure, 177, 323
Control Point Attributes, 236, 251, 264, 303, 336
Controller Configuration, 192
Controller Flow Control, 188
Controller Information, 191
Controller, AVCTP, 97
Core Protocols, 22, 79
CRC, 161, 165, 175
Credit Based Flow Control, 95
Cryptographic Functions, 216
Current Time Service (CTS), 341

D

Data Channel PDU, 158, 162, 163, 175
Data Physical Channel, 155, 159, 160
Data Rates, 2, 3, 28, 143
Data Rates and Coverage, 2, 3
Data Signing, 251, 328–330
Data Transport Architecture, 40
Defragmentation, 204
Device Access Code (DAC), 46, 48
Device Address, 26, 102, 153, 307, 330, 331
Device Discovery, 101, 105, 189, 193
Device Discovery Substates, 51, 52
Device Filtering, 181
Device Information Service (DIS), 261, 263, 340
Device Name Characteristic, 265, 308, 311, 312, 318, 319
Device Setup, 188
Discover All Characteristic Descriptors, 275, 287
Discover All Characteristics of a Service, 275, 283, 346
Discover All Primary Services, 275, 276, 278
Discover Primary Services By Service UUID, 275, 276
Discoverable Mode, 103, 310, 311, 316–319
Discovering Devices, 121, 155
Discover Characteristics by UUID, 275, 285
Discovery Modes, 315, 316
Discovery Modes and Procedures, 315, 316
DM1 Packet, 48, 49, 53, 54, 56
Dual Mode Devices, 102, 132, 139, 141, 153

E

Enabling and Disabling Bluetooth, 120
Encrypted Diversifier (EDIV), 220, 221
Encryption, 60, 71, 165, 189, 194, 195, 226, 327
Encryption Information, 223, 226, 230
Encryption Key Size, 219, 223, 236
Encryption Pause Procedure, 177, 179, 180
Encryption Procedure, 179, 329
Encryption Restart Procedure, 179
Encryption Start Procedure, 177, 179
End Product Listing (EPL), 377

- Enhanced Retransmission Mode, 82, 205
eSCO, *See* Extended Synchronous Connection Oriented
 eSCO Logical Transport, 41–44, 46, 57
 European Telecommunications Standards Institute (ETSI), 5
 European Commission for Standardization (CEN), 4
 Exchange MTU, 253, 275, 276
 Exchange MTU Request, 240, 241
 Exchange MTU Response, 240, 241
 Execute Write Request, 248–250, 294–296, 300–302
 Execute Write Response, 248–250, 294–296
 Extended Synchronous Connection Oriented (eSCO), 20, 42–47, 50, 63, 69, 109, 114, 137
- F**
- Feature Exchange Procedure, 177, 180, 181, 218
 Feature Set, 180
 Federal Communications Commission (FCC), 5
 FHS Packet, 48, 49, 54
 FHSS (Frequency Hopping Spread Spectrum), 21
 File Transfer Profile (FTP), 112
 Find By Type Value Request, 242, 279
 Find By Type Value Response, 243, 279
 Find Included Services, 280–282
 Find Information Request, 241, 287, 288
 Find Information Response, 242, 287, 288
 Find Me Profile (FMP), 260, 262, 263, 264, 336, 337, 369
 Flow Control Mode, 82, 205
 Fragmentation, 203, 204
 Frequency Band, 34, 135, 145, 155
 Frequency Hopping, 20, 21, 33, 35–38, 135, 145, 156
 FTP, *See* File Transfer Protocol
- G**
- GAP, *See* Generic Access Profile
 GAP Characteristics, 311–313
 GATT, *See* Generic Attribute Profile
- GATT-based Architecture, 134, 140, 141, 260–262, 333
 GATT Features, 275
 GATT operations, 358
 GATT Service, 301
 General Discoverable Mode, 103, 310, 316–318
 General Inquiry, 104
 General Inquiry Access Code (GIAC), 104
 Generic Access Profile (GAP), 100, 142, 260, 305, 346
 Generic Attribute Profile (GATT), 140, 141, 259, 346
 Generic Audio/Video Distribution Profile (GAVDP), 113
 Generic Object Exchange Profile (GOEP), 110
 Glucose Profile (GLP), 345
 Glucose Service (GLS), 344
 GOEP (Generic Object Exchange Profile), 101, 107, 110, 112
 Grouping of Attribute Handles, 237
 GSM, 2, 3
- H**
- Handle Information List, 243, 279, 280
 Hands-Free (HF), 28, 92, 107–109
 Hands-Free Profile, 107, 109
 HCI, Host Controller Interface
 HCI ACL Data Packet, 62, 63, 69, 86, 186, 187, 190, 201
 HCI Command, 60, 63, 66, 70
 HCI Command Packets, 61, 62, 69, 70, 186
 HCI Commands and Events, 187, 189
 HCI Event, 60–62, 64
 HCI Event Packet, 69, 70, 186
 HCI Flow Control, 65
 HCI Packet Types, 61, 186
 HCI Transport Layer, 68
 HCI UART Interface, 69, 70
 Headset Profile, 107, 108
 Health Thermometer Profile (HTP), 260, 261, 270, 343
 Health Thermometer Service (HTS), 234, 268, 274, 341
 Health, Sports and Fitness Profiles, 344

Heart Rate Monitor Profile (HRP), 260, 263, 271, 345
Heart Rate Service (HRS), 263, 264, 266, 345
HID, *See* Human Interface Devices
HID over GATT (HOGP), 345
HID Service, 345
History (Bluetooth Specification), 19
Hop Increment, 156, 175, 178
Host Controller Interface (HCI), 23, 24, 26, 60, 185
Host Flow Control, 189, 191
Human Interface Devices (HID), 23, 84, 345

I

ID Packet, 48, 53
Identity Information, 223, 226, 227
Identity Resolution Key (IRK), 220, 221
Idle Mode Procedures, 104
IEEE, *See* Institute of Electrical and Electronics Engineers
IEEE 802.15, 19, 20
IEEE Word Usage, 30
Immediate Alert Service (IAS), 236, 256, 262–265, 335, 336–339, 360
Implementation Conformance Statement (ICS), 379
Implementation Extra Information for Testing (IXIT), 379
Include Definition, 264, 265, 271, 272
Indication, 238–240, 253, 266, 267, 269–271, 274, 297, 342
Indication Type Methods, 252, 253
Industry, Scientific and Medical (ISM) Band, 20, 21, 33, 34, 36, 135, 140, 145, 156
Infrared Data Association (IrDA), 3, 17, 23, 96
Initiating PDU, 162, 163
Initiating State, 151, 153, 162, 166, 167, 173
Initiator, 75, 76, 113, 152, 153, 163, 166, 172, 227, 376
Initiator Address (InitA), 169, 175
Initiator Key Distribution, 223, 224, 228
Initiator Filter Policy, 183, 195, 321, 322
Inquiry, *See* Bluetooth Device Inquiry
Inquiry Access Code (IAC), 46, 48, 104
Inquiry Scan, 35, 40, 51–53, 74, 104

Institute of Electrical and Electronics Engineers (IEEE), 5, 19, 23, 26, 27, 30, 32, 153, 340
Interference with WiFi, 155
Interoperability, 4, 18, 21, 22, 96, 100, 305, 371, 373, 374, 375, 376
Interoperability Testing, 371, 374, 375, 378
Invalid Behavior (BI), 379
International Organization for Standardization (ISO), 5, 13
International Telecommunications Union (ITU), 5
IrDA, *See* Infrared Data Association
ISM Band, *See* Industrial, Scientific and Medical
ISO, *See* International Organization for Standardization
ITU, *See* International Telecommunications Union

J

Just Works, 72, 73, 106, 214, 215, 218–220, 224, 229

K

Key generation, 211

L

L2CAP, *See* Logical Link Control and Adaptation Protocol
L2CAP Features, 83, 203
L2CAP Modes of Operation, 82
L2CAP Parameters, 205
L2CAP PDU, 87, 201, 202, 206
L2CAP Signaling, 86, 87, 203, 206
LE Architecture, *See* Architecture
LE Meta Event, 65, 186, 187, 359
LE Security Mode 1, 327, 328
LE Security Mode 2, 328, 329
LE Timeline, 149
LE Topology, 160
Limited Discoverable Mode, 103, 104, 310, 316–319
Limited Inquiry, 103, 104
Link Control Packets, 47, 48

Link Controller, *See* Baseband Controller

Link Controller States, 50, 52

Link Information, 189, 194

Link Layer, 137, 151-184

Link Layer Control Procedures, 176, 177

Link Layer Control Protocol (LLCP), 176

Link Layer Packet Format, 160, 161

Link Layer States, 151, 166-168

Link Loss Service (LLS), 337

Link Manager, 56, 75, 101

Link Manager Protocol (LMP), 56

Link Supervision, 41, 57

Local Area Network (LAN), 3

Logical Link Control and Adaptation Protocol (L2CAP), 80-88, 96, 105, 201-209, 222, 305, 323

Logical Transport, 41, 42, 46, 53, 55, 58

Long Term Key (LTK), 179, 189, 194, 220, 221

M

MAN, *See* Metropolitan Area Network

Man In The Middle (MITM), 71, 72, 73, 106, 213, 215, 220, 222, 228, 328

Man Machine Interface (MMI), 22, 24,

Master Identification, 223, 226, 227

Master Response, 51, 52, 54

Maximum Transmission Unit (MTU), 84, 85, 87, 120, 202, 203, 205, 240, 241, 253, 275

Memory footprint, 138, 306

Message Integrity Check (MIC), 163

Method Type, 238

Metropolitan Area Network (MAN), 3

MMI, *See* Man Machine Interface

Modulation Characteristics, 147

More Data (MD), 164

Mostly Off Technology, 136, 149

N

Narrow Band Speech, 43

Near Field Communication (NFC), 3, 6, 13, 73

Next Expected Sequence Number (NESN), 164

NFC, *See* Near Field Communication

No security, 106, 215, 216, 327

Nonbondable Mode, 103, 324, 326

Nonconnectable Directed, 163, 309

Nonconnected States, 167

Nondiscoverable Mode, 103, 316

Nonresolvable Private Address, 154, 331

Notification, 233, 235, 238-240, 267, 270, 271, 297

Notification Type Methods, 252

NULL Packet, 48, 49, 52, 57, 137, 139, 161

O

OBEX, *See* Object Exchange Protocol

OBEX Client and Server, 97

OBEX Operations, 96, 97

Object Exchange Protocol (OBEX), 96

Object Push Profile (OPP), 111

Observation Procedure, 315

Observer, 306, 307, 315, 320

OOB Authentication Data, 215, 219, 220

OpCode, 56, 61, 62, 70, 186, 187, 238, 239

Opcode Command Field (OCF), 61, 62, 187

Opcode Group Field (OGF), 61, 62, 186, 187

OPP, *See* Object Push Protocol

Organizationally Unique Identifier (OUI), 27, 153, 340

Out of Band (OOB), 73, 215, 218, 219, 220, 222

Output Power, 21, 33, 34, 147

P

Packet Format

BR/EDR, 46

HCI, 62, 63, 187

LE, 138, 160, 161

LE L2CAP, 205

Page, 30, 50, 51, 52

Page Scan, 35, 40, 51, 52, 75, 104, 129

Pairing, 56, 59, 103, 106, 214, 217, 225, 229, 308

Pairing Confirm, 224, 225

Pairing Failed, 225

Pairing Feature Exchange, 217, 222

Pairing Methods, 214, 215

Pairing Random, 224, 225

Pairing Request, 222, 223, 228

Pairing Response, 223, 228

- Pairing, Simple Secure, *See* Secure Simple Pairing
PAN, *See* Personal Area Network
Parameter Total Length, 62, 70, 186, 187
Park State, 51, 55, 57
Passive Eavesdropping, 73, 212, 213, 215
Passive Eavesdropping Protection, 71, 215
Passive Scanning, 171, 172, 196, 197, 315
Passkey Entry, 72, 73, 106, 214, 215, 218, 219, 220, 224
PDU and SDU, 201, 202
Periodic Inquiry, 75
Peripheral, 306, 307, 316–321, 323–329
Peripheral Preferred Connection Parameters (PPCP), 314, 315
Peripheral Privacy Flag Characteristic, 313
Personal Area Network (PAN), 3, 17
Phone Alert Status Profile (PASP), 345
Phone Alert Status Service (PASS), 345
Physical Channel, 35, 36, 39, 40, 154–157, 160, 161
Physical Link, 40, 41, 57, 85, 189, 194, 203, 336, 338
Piconet, 21, 29, 30, 35–37, 40, 159, 160, 166
Piconet Channel, Basic, Adapted, Inquiry Scan, Page Scan, 40, 49
POLL Packet, 48, 52
Power Class, 33, 34
Preamble, 161
Prepare Write Request, 237, 240, 248, 293, 294, 295, 301
Prepare Write Response, 240, 249, 250, 294, 295, 301, 302
Prequalification, 380
Primary Service, 234, 237, 247, 248, 254, 255, 265, 270, 279, 301, 361
Primary Service Discovery, 275, 276, 278, 363
Privacy Feature, 154, 214, 221, 320, 329
Private Address, 154, 216, 330, 331
Profile Dependencies, 100, 101, 260
Profile Tuning Suite (PTS), 18, 371, 376
Profiles, 22–24, 80, 100, 140, 260–263, 264, 333, 345
Protocol Data Unit (PDU), 28, 82, 201
Protocol/Service Multiplexing (PSM), 83, 84, 88, 92
Proximity Profile (PXP), 183, 262, 263, 339, 345, 358
PTS, *See* Profile Tuning Suite
Public Device Address, 153, 220, 227, 331
Q
Qualification, 19, 21, 371, 373, 376–380
Qualification Design ID (QDID), 373
R
Radio, 33, 34, 57, 101, 135, 136, 138, 145, 146, 165
Radio Frequency Identification (RFID), 2, 3
Random Address, 153, 154, 162, 168, 189, 192, 214, 220, 221, 330, 331
Random Address function, 216
Read Blob Request, 240, 245, 253, 289, 290, 299
Read Blob Response, 240, 245, 299
Read By Group Type Request, 237, 240, 247, 254, 276, 277, 362
Read By Group Type Response, 240, 247, 254, 276, 277, 362
Read By Type Request, 240, 243, 244, 247, 280–287, 290, 291, 319, 366
Read By Type Response, 240, 243, 244, 247, 281–287, 290, 291, 319, 366
Read Characteristic Descriptors, 275, 298
Read Characteristic Value, 275, 288, 289, 291, 337, 338–340, 366
Read Long Characteristic Descriptors, 275, 298, 299
Read Long Characteristic Values, 275, 288, 289
Read Multiple Characteristic Values, 275, 288, 291
Read Multiple Request, 240, 246, 291
Read Multiple Response, 240, 246, 291
Read Request, 244, 245, 281, 289, 290, 298, 299, 312, 313, 367
Read Response, 240, 244, 245, 282, 289, 298, 299, 312–314, 368
Read Using Characteristic UUID, 275, 288, 290, 291, 319
Reconnection Address Characteristic, 313, 329

- Relationship Discovery, 275, 280
 Reliable Writes, 275, 292, 294, 296
 Remote Information, 189, 192
 Repeated Attempts, 219
 Request and Response Type Methods, 240
 Resolvable Private Address, 154, 216, 331
 Retransmission Mode, 82, 205
 RF Channels, 21, 140, 143, 146, 154, 155
 RFCOMM, 23, 79, 84, 88, 93–95
RFID, *See* Radio Frequency Identification
 RS-232 Nine Circuits, 95
 RS-232 Serial Interface, 95
 RTP Control Protocol (RTCP), 98
 RTP Data Transfer Protocol (RTP), 98
 RxAdd, 162, 169, 173, 175
- S**
- Scannable Undirected, 163, 167, 171, 172, 197, 309, 315–317, 320
 Scanner Filter Policy, 183, 323, 324
 Scanning PDU, 162, 163
 Scanning State, 151–153, 162, 167, 168, 170, 171
 Scatternet, 29, 30, 35–37, 40, 42, 55, 64, 137, 143, 153, 160, 166
SCO, *See* Synchronous Connection Oriented
 SCO Logical Transport, 41–44, 46, 55, 57
 SDP, *See* Service Discovery Protocol
 SDP Client and Server, 89
 SDP Transactions, 91
 Searching and Browsing Services, 90
 Secondary Service, 237, 247, 259, 265, 270–272
 Secure Simple Pairing (SSP), 20, 59, 60, 71, 72, 104, 106, 129
 Security, 58, 71, 106, 211, 215, 222, 304, 326–329
 Security Breaches, 212
 Security function, 216
 Security Manager (SM), 211, 305, 308, 326–329
 Security Manager Protocol (SMP), 203, 222, 225
 Security Modes, 106, 107, 328, 329
 Security Properties, 215
- Security Request, 219, 222, 223, 329
 Segmentation and Reassembly (SAR), 83, 84
 Sequence Number (SN), 46, 164
 Sequential Transactions, 239
 Serial Port Profile (SPP), 100, 107
 Server Configuration, 275
 Service Changed Characteristic, 269, 301–303, 364
 Service Declaration, 254, 271, 272, 277, 303, 335, 337
 Service Discovery Protocol (SDP), 79, 83, 88–93, 101, 121, 122, 126
 Short Term Key (STK), 216, 217, 224
 Short Term Key (STK) Generation, 217, 219, 224
 SIG Device Library, 375
 Signed Write Without Response, 275, 292, 293
 Single Mode, 132, 133, 134
 Single Mode versus Dual Mode Devices, 132, 133
 Slave Latency (connSlaveLatency), 159, 177, 207, 314, 315
 Slave Response, 51, 52, 54
 Sniff Mode, 52, 55, 57, 64, 136
 SNK (AVDTP Sink), 98, 114, 115
 SRC (AVDTP Source), 98, 114, 115
 Standby State, 151, 152, 153, 167, 188
 Static Address, 154, 330, 331
 STK Generation, 217, 219, 224
 Stream End Point (SEP), 99, 100, 114
 Streaming Channels, 83, 86, 113
 Streaming Mode, 82, 205
 Sub-Band Codec (SBC), 45, 99, 115, 116
 Supervision Timeout
 (connSupervisionTimeout), 159, 175, 177, 194, 207, 314, 315
 Synchronous Connection Oriented (SCO), 40–47, 50, 52, 53, 55, 56, 61, 67, 69, 109
 Synchronous Packets, 47, 48, 50
- T**
- Target, AVCTP, 97
 Temperature Measurement Characteristic, 265, 266, 270, 274, 342
 Termination Procedure, 177, 181, 182

Test Case Reference List (TCRL), 379
Test categories, 378, 379
Test Specifications, 18, 378, 379
Testing, 57, 189, 195, 350, 371
Three Phase Pairing Process, 217
Time Division Duplex (TDD), 33, 36, 38, 39, 40
Topology, 21, 29, 35, 37, 42, 140, 159, 160
Tracking, 154, 214, 329
Transmit Power Service (TPS), 338, 339
transmitWindowsOffset, 176
transmitWindowSize, 159, 176
Transport Specific Key Distribution, 217
Tx Power Service, 256, 262, 263, 265, 338, 339, 360, 362, 365
TxAdd, 162, 168–171, 173, 175

U

Unauthenticated no MITM Protection, 215
UnPlugFest (UPF), 18, 374, 377
UUID, 90, 234, 235

V

Valid Behavior (BV), 379
Version Exchange Procedure, 177, 180, 181

W

WAN, *See* Wide Area Network
White List, 181–183, 189, 191, 195, 198, 199, 321–324, 329, 330
Wide Area Network (WAN), 2
Wide Band Speech (WBS), 43
Wireless Communication, 1, 2
Write Characteristic Descriptors, 275, 298, 299
Write Characteristic Value, 275, 292–294, 337, 338, 367
Write Long Characteristic Descriptors, 275, 298, 300
Write Long Characteristic Values, 275
Write Request, 237, 240, 248
Write Response, 240, 248
Write Without Response, 274, 275, 292, 303, 335, 336, 365

Z

ZigBee, 2, 3, 6, 12

