IEEE Standards

802.15.1™

**IEEE Standard for Information technology—
Telecommunications and information exchange between systems—
Local and metropolitan area networks—
Specific requirements**

Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)

IEEE Computer Society

Sponsored by the
LAN/MAN Standards Committee

IEEE

Published by
The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

14 June 2002

Print: SH94963
PDF: SS94963

Recognized as an

American National Standard (ANSI)

IEEE Std 802.15.1™-2002

**IEEE Standard for Information technology—
Telecommunications and information exchange between systems—
Local and metropolitan area networks—
Specific requirements—**

Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)

Sponsor

**LAN/MAN Standards Committee
of the
IEEE Computer Society**

Approved 15 April 2002

IEEE-SA Standards Board

Approved 26 July 2002

American National Standards Institute

Abstract: The lower transport layers [(Logical Link Control and Adaptation Protocol (L2CAP), Link Manager Protocol (LMP), baseband and radio] of the Bluetooth™ wireless technology are defined. Bluetooth is an industry specification for short-range radio frequency (RF)-based connectivity for portable personal devices. The IEEE 802.15.1 Task Group has reviewed and provided a standard adaptation of the Bluetooth specification (version 1.1) medium access control (MAC) (L2CAP, LMP, and baseband) and physical layer (PHY) (radio). Also specified is a clause on service access points (SAPs), which includes a logical link control (LLC)-MAC interface for the ISO/IEC 8802-2 LLC. A normative annex is included that provides a Protocol Implementation Conformance Statement (PICS) proforma, and a informative high-level behavioral ITU-T Z.100 specification and description language (SDL) model for an integrated Bluetooth MAC sublayer are also specified.
Keywords: ad hoc network, Bluetooth, Bluetooth wireless technology, circuit switching, FH-CDMA, frequency-hopping code division multiple access, mobile, mobility, nomadic, packet switching, piconet, radio, radio frequency, scatternet, short-range, ubiquitous computing and communications, wearables, wireless, wireless personal area network, WPAN

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Print: ISBN 0-7381-3068-0 SH94963
PDF: ISBN 0-7381-3069-9 SS94963

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "**AS IS**."

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

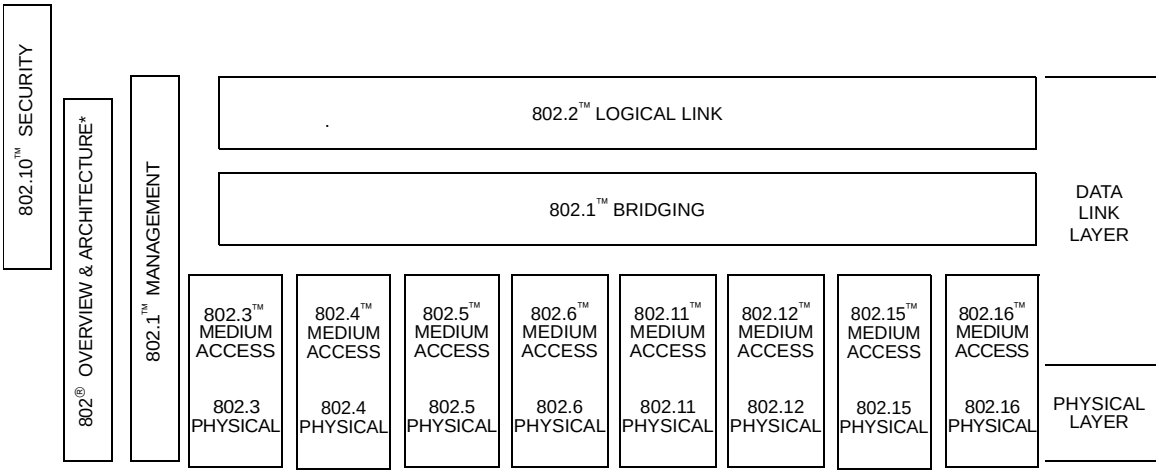
Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Introduction

[This introduction is not part of IEEE Std 802.15.1-2002, IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs).]

This standard is part of a family of standards for local and metropolitan area networks. The relationship between the standard and other members of the family is shown below. (The numbers in the figure refer to IEEE standards numbers.[1]) This standard focuses only on the medium access control (MAC) and physical layer (PHY) of an 802.15.1 wireless personal area network (WPAN).



* Formerly IEEE Std 802.1A™.

This family of standards deals with the Physical and Data Link Layers as defined by the International Organization for Standardization (ISO) Open Systems Interconnection Basic Reference Model (ISO/IEC 7498-1:1994). The access standards define several types of medium access technologies and associated physical media, each appropriate for particular applications or system objectives. Other types are under investigation.

The standards defining the technologies noted above are as follows:

- IEEE Std 802:[2, 3] — *Overview and Architecture*. This standard provides an overview to the family of IEEE 802 Standards. This document forms part of the IEEE Std 802.1 scope of work.
- IEEE Std 802.1B™ and 802.1K™ [ISO/IEC 15802-2]: — *LAN/MAN Management*. Defines an Open Systems Interconnection (OSI) management-compatible architecture, and services and protocol elements for use in a LAN/MAN environment for performing remote management.
- IEEE Std 802.1D™ — *Media Access Control* (*MAC) Bridges*. Specifies an architecture and protocol for the [ISO/IEC 15802-3]: interconnection of IEEE 802 LANs below the MAC service

[1]The IEEE Standards referred to in the above figure and list are trademarks owned by the Institute of Electrical and Electronics Engineers, Incorporated.

[2]IEEE and 802 are registered trademarks in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Incorporated.

[3]The IEEE 802 Architecture and Overview Specification, originally known as IEEE Std 802.1A, has been renumbered as IEEE Std 802. This has been done to accommodate recognition of the base standard in a family of standards. References to IEEE Std 802.1A should be considered as references to IEEE Std 802.

• IEEE Std 802.1E™ [ISO/IEC 15802-4]:	*System Load Protocol*. Specifies a set of services and protocol for those aspects of management concerned with the loading of systems on IEEE 802 LANs.
• IEEE Std 802.1F™	*Common Definitions and Procedures for IEEE 802 Management Information*.
• IEEE Std 802.1G™ [ISO/IEC 15802-5]:	*Remote Media Access Control (MAC) Bridging*. Specifies extensions for the interconnection, using non-LAN systems communication technologies, of geographically separated IEEE 802 LANs below the level of the logical link control protocol.
• IEEE Std 802.1H™	*Recommended Practice for Media Access Control (MAC)*
[ISO/IEC TR 11802-5]	*Bridging of Ethernet V2.0 in IEEE 802 Local Area Networks*.
• IEEE Std 802.1Q™	*Virtual Bridged Local Area Networks*. Defines an architecture for Virtual Bridged LANs, the services provided in Virtual Bridged LANs, and the protocols and algorithms involved in the provision of those services.
• IEEE Std 802.2 [ISO/IEC 8802-2]:	*Logical Link Control*.
• IEEE Std 802.3 [ISO/IEC 8802-3]:	*CSMA/CD Access Method and Physical Layer Specifications*.
• IEEE Std 802.4 [ISO/IEC 8802-4]:	*Token Bus Access Method and Physical Layer Specifications.*
• IEEE Std 802.5 [ISO/IEC 8802-5]:	*Token Ring Access Method and Physical Layer Specifications*.
• IEEE Std 802.6 [ISO/IEC 8802-6]:	*Distributed Queue Dual Bus Access Method and Physical Layer Specifications.*
• IEEE Std 802.10:	*Interoperable LAN/MAN Security.* Currently approved: Secure Data Exchange (SDE).
• IEEE Std 802.11: [ISO/IEC 8802-11]	*Wireless LAN Medium Access Control (MAC) Sublayer and Physical Layer Specifications.*
• IEEE Std 802.12: [ISO/IEC 8802-12]	*Demand Priority Access Method, Physical Layer and Repeater Specification*.
• IEEE Std 802.15:	*Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for: Wireless Personal Area Networks.*
• IEEE Std 802.16:	*Air Interface for Fixed Broadband Wireless Access Systems*.
• IEEE Std 802.17™:	*Resilient Packet Ring Access Method and Physical Layer Specifications.*

In addition to the family of standards, the following is a recommended practice for a common physical layer technology:

• IEEE Std 802.7™:	*IEEE Recommended Practice for Broadband Local Area Networks*.

The reader of this standard is urged to become familiar with the complete family of standards.

Conformance test methodology

Products built based on this standard are considered to conform to (or be compliant with) this standard if they pass the Bluetooth™ qualification program as set forth by the Bluetooth Special Interest Group (SIG), Inc. More information can be found at http://ieee802.org/15/Bluetooth/.

IEEE Std 802.15.1-2002

IEEE Std 802.15.1-2002 has been derived from the Bluetooth specifications (version 1.1) (see 2.4). This standard provides the reader with the lower layers of the Bluetooth specifications or MAC and PHY:

— The radio frequency (RF) layer, specifying the radio parameters.

— The baseband layer, specifying the lower level operations at the bit and packet levels, i.e., forward error correction (FEC) operations, encryption, cyclic redundancy check (CRC) calculations, and automatic repeat request (ARQ) protocol.

— The link manager (LM) layer, specifying connection establishment and release, authentication, connection and release of synchronous connection-oriented (SCO) and asynchronous connectionless (ACL) channels, traffic scheduling, link supervision, and power management tasks.

— The Logical Link Control and Adaptation Protocol (L2CAP) layer, which has been introduced to form an interface between standard data transport protocols and the Bluetooth protocol. It handles the multiplexing of higher layer protocols and the segmentation and reassembly (SAR) of large packets.

Additionally, the IEEE has conducted numerous ballots and provided all comments back to the Bluetooth SIG, Inc. as errata. These errata have in turn been reviewed and in some cases adopted into the Bluetooth specification.

This standard contains state-of-the-art material. The area covered by this standard is undergoing evolution. Revisions are anticipated to this standard within the next few years to clarify existing material, to correct possible errors, and to incorporate new related material.

There is a body of generally available industry information quantifying the level of over-the-air coexistence between 802.15.1 and 802.11. Additional information may be obtained from the anticipated IEEE 802.15.2™ Recommended Practice for Coexistence of Wireless Personal Area Networks with Other Wireless Devices Operating in Unlicensed Frequency Bands (available in draft form as P802.15.2 at the time of this publication) that deals specifically with the coexistence issue.[3]

Information on the current revision state of this and other IEEE 802 standards may be obtained from:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331 USA

[3]This IEEE standards project was not approved by the IEEE-SA Standards Board at the time this publication went to press. For information about obtaining a draft, contact the IEEE (http://standards.ieee.org). Upon approval of IEEE P802.15.2 by the IEEE-SA Standards Board, this draft will be superceded by the approved standard and published as IEEE Std 802.15.2-2003. Approval is expected in early 2003.

Participants

The following is a list of the participants who were members of the IEEE 802.15 Working Group, the task group (TG) leadership, and the Task Group 1 Editorial Team when the IEEE Std 802.15.1-2002 was approved:

Robert F. Heile, *Chair*
Ian C. Gifford, *Vice Chair*
James D. Allen, *Vice Chair*

Thomas M. Siep, *Editor-in-Chief*
Patrick W. Kinney, *Secretary*
Michael D. McInnis, *Assistant Secretary and Editor*

Stephen J. Shellhammer, *TG2 Chair*
John R. Barr, *TG3 Chair*
Robert F. Heile, *TG4 Chair (Acting)*

Ian C. Gifford, *TG1 Chair and Technical Editor*
Chatschik Bisdikian, *TG1 Vice Chair and Technical Editor*
Thomas M. Siep, *TG1 Editor-in-Chief*
Michael D. McInnis, *TG1 Secretary and Technical Editor*
David E. Cypher, *TG1 SDL Technical Editor*
Michael T. Camp, *TG1 Technical Editor*
Fujio Watanabe, *TG1 Technical Editor*

Enrique Aguado
Masaaki Akahane
Richard Alfvin
James D. Allen
David Archer
Arun Arunachalam
John R. Barr
Edul Batliwala
Alan Bien
Chatschik Bisdikian
Timothy J. Blaney
Monique Bourgeois
Ed Callaway
Michael T. Camp
Boaz Carmeli
Michael Carrafiello
Hung-Kun Chen
James Chen
Kwang-Cheng Chen
Todor Cooklev
Wm. Caldwell Crosswy
David E. Cypher
Anand Ganesh Dabak
Michael Derby
Darrell Diem
Mary DuVal
Michael Dydyk
Richard Eckard
Nick Evans
Atul Garg
Ian C. Gifford
James Gilb
Ms. Nada Golmie
Rajugopal Gubbi
Jose Gutierrez
Yasuo Harada
Allen Heberling
Robert F. Heile
Barry Herold
Mark Hinman
Masaki Hoshina
Bob Huang
Katsumi Ishii
Phil Jamieson
Jeyhan Karaoguz
Stuart Kerry
Patrick W. Kinney
Bruce P. Kraemer
Jim Lansford
Yunxin Li
Jie Liang
Stanley Ling
James Little
Kevin Marquess
John McCorkle
Daniel R. McGlynn
Michael D. McInnis
Vinay Mitter
Akira Miura
Peter Murray
Wayne Music
Marco Naeve
Mohammed Nafie
Erwin R. Noble
Arto Palin
Gregory Parks
Gregg Rasor
Ivan Reede
Carlos A. Rios
Benno Ritter
Richard Roberts
Martin Rofheart
Chandos Rypinski
Juha Salokannel
Timothy Schmidl
Mark Schrader
Michael Seals
Stephen J. Shellhammer
Bill Shvodian
Thomas M. Siep
Carl Stevenson
Katsumi Takaoka
Teik-Kheong Tan
Jerry Thrasher
Bijan Treister
Robert E. Van Dyck
Ritesh Vishwakarma
Barry Volinskey
Fujio Watanabe
Richard Wilson
Song-Lin Young
Amos Young

Major contributions were received from the following individuals:

BSIG PM chairs
James Kardach
Francis Truntzer

BSIG P802.15.1/D0.5 reviewers
Jaap Haartsen
Jon Inouye
Patrick Kane
David Moore
Thomas Müller

Dan Sonnerstam (editor)
Cathy Hughes (editor)

IEEE Clause 7
Part A/Radio
Steve Williams
Todor V. Cooklov
Poul Hove Kristensen
Kurt B. Fischer
Kevin D. Marquess
Troy Beukema
Brian Gaucher
Jeff Schiffer
James P. Gilb
Rich L. Ditch
Paul Burgess
Olaf Joeressen
Thomas Müller
Arto T. Palin
Steven J. Shellhammer
Sven Mattisson
Lars Nord (section owner)
Anders Svensson
Mary A. DuVal
Allen Hotari

IEEE Clause 8
Part B/Baseband
Kevin D. Marquess
Chatschik Bisdikian
Kris Fleming
James P. Gilb
David E. Cypher
Nada Golmie
Olaf Joeressen
Thomas Müller
Charlie Mellone
Harmke de Groot
Terry Bourk
Steven J. Shellhammer
Jaap Haartsen
Henrik Hedlund (section owner)
Tobias Melin
Joakim Persson
Mary A. DuVal
Onn Haran
Thomas M. Siep
Ayse Findikli

IEEE Clause 9
Part C/Link Manager Protocol
Kim Schneider
Toru Aihara
Chatschik Bisdikian
Kris Fleming
David E. Cypher
Thomas Busse
Julien Corthial
Olaf Joeressen
Thomas Müller
Dong Nguyen
Harmke de Groot
Terry Bourk
Johannes Elg
Jaap Haartsen
Tobias Melin (section owner)
Mary A. DuVal
Onn Haran
John Mersh

IEEE Clause 10
Part D/L2CAP
Jon Burgess
Paul Moran
Doug Kogan
Kevin D. Marquess
Toru Aihara
Chatschik Bisdikian
Kris Fleming
Uma Gadamsetty
Robert Hunter
Jon Inouye (section owner)
Steve C. Lo
Chunrong Zhu
Sergey Solyanik
David E. Cypher
Nada Golmie
Thomas Busse
Rauno Makinen
Thomas Müller
Petri Nykänen
Peter Ollikainen
Petri O. Nurminen
Johannes Elg
Jaap Haartsen
Elco Nijboer
Ingemar Nilsson
Stefan Runesson
Gerrit Slot
Johan Sörensen
Goran Svennarp
Mary A. DuVal
Thomas M. Siep
Kinoshita Katsuhiro

IEEE Clause 11
Part H:1/HCI
Todor Cooklev
Toru Aihara
Chatschik Bisdikian
Nathan Lee
Akihiko Mizutani
Les Cline
Bailey Cross
Kris Fleming
Robert Hunter
Jon Inouye
Srikanth Kambhatla
Steve Lo
Vijay Suthar
Bruce P. Kraemer
Greg Muchnik
David E. Cypher
Thomas Busse
Julien Courthial
Thomas Müller
Dong Nguyen
Jürgen Schnitzler
Fujio Watanabe
Christian Zechlin
Johannes Elg
Christian Johansson (section owner)
Patrik Lundin
Tobias Melin
Mary A. DuVal
Thomas M. Siep
Masahiro Tada
John Mersh

IEEE Annex A
Bluetooth PICS
Stefan Agnani
David E. Cypher
Fujio Watanabe

IEEE Annex B
David E. Cypher
Allen D. Heberling
Hans Roelofs
Yunming Song

IEEE Annex C
Part K:1/GAP
Ken Morley
Chatschik Bisdikian
Jon Inouye
Brian Redding
David E. Cypher
Stephane Bouet
Thomas Müller
Martin Roter
Johannes Elg
Patric Lind (section owner)
Erik Slotboom
Johan Sörensen

IEEE Annex D
Appendix VII/Optional Paging Scheme
Olaf Joeressen
Thomas Müller
Markus Schetelig
Fujio Watanabe
Jaap Haartsen (section owner)
Joakim Persson
Ayse Findikli
(continued)

Major contributions were received from the following individuals: *(continued)*

IEEE Annex E
Part I:1/Bluetooth Test Mode
Jeffrey Schiffer
David E. Cypher
Daniel Bencak
Arno Kefenbaum
Thomas Müller (section owner)
Roland Schmale
Fujio Watanabe
Stefan Agnani
Mårten Mattsson
Tobias Melin
Lars Nord
Fredrik Töörn
John Mersh
Ayse Findikli

IEEE Annex F
Appendix VI/Baseband Timers
David E. Cypher
Jaap Haartsen (section owner)
Joakim Persson
Ayse Findikli

IEEE Annex G
Appendix IX/MSCs
Todor Cooklev
Toru Aihara
Chatschik Bisdikian
Nathan Lee
Kris Fleming
Greg Muchnik
David E. Cypher
Thomas Busse

IEEE "must-to-shall" reviewers
Chatschik Bisdikian
Thomas W. Baker
Thomas Müller
Lars Nord
Henrik Hedlund
John Mersh
Thomas M. Siep
Christian Johansson
Patric Lind

Mentors
Richard C. Braley
Jim Carlo
Simon C. Ellis
Harald Blaatand "Bluetooth" II

The following members of the balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Toru Aihara
John R. Barr
Chatschik Bisdikian
James T. Carlo
Bruce J. Currivan
Vern A. Dubendorf
Mary A. DuVal
Kurt B. Fischer
Michael A. Fischer
Avraham Freedman
Ian C. Gifford
Simon Harrison
Vic Hayes
Robert F. Heile
James Ivers
Stuart J. Kerry
Brian G. Kiernan
Patrick W. Kinney
Gregory Luri
Roger B. Marks
Peter Martini
Michael D. McInnis
Marco Naeve
Robert O'Hara
Roger Pandanda
Jon W. Rosdahl
Thomas M. Siep
Carl R. Stevenson
John Viaplana
Fujio Watanabe
Don Wright

When the IEEE-SA Standards Board approved this standard on 15 April 2002, it had the following membership:

James T. Carlo, *Chair*
James H. Gurney, *Vice Chair*
Judith Gorman, *Secretary*

Sid Bennett
H. Stephen Berger
Clyde R. Camp
Richard DeBlasio
Harold E. Epstein
Julian Forster*
Howard M. Frazier
Toshio Fukuda
Arnold M. Greenspan
Raymond Hapeman
Donald M. Heirman
Richard H. Hulett
Lowell G. Johnson
Joseph L. Koepfinger*
Peter H. Lips
Nader Mehravari
Daleep C. Mohla
William J. Moylan
Malcolm V. Thaden
Geoffrey O. Thompson
Howard L. Wolfman
Don Wright

*Member Emeritus

Also included is the following nonvoting IEEE-SA Standards Board liaison:

Alan Cookson, *NIST Representative*
Satish K. Aggarwal, *NRC Representative*

Jennifer McClain Longman
IEEE Standards Project Editor

Contents

List of Figures

List of Tables

IEEE Standard for Information technology—
Telecommunications and information exchange between systems—
Local and metropolitan area networks—
Specific requirements

Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)

1. Overview

Wireless personal area networks (WPANs) are used to convey information over short distances among a private-intimate group of participant devices. Unlike a wireless local area network (WLAN), a connection made through a WPAN involves little or no infrastructure or direct connectivity to the world outside the link. This allows small, power-efficient, inexpensive solutions to be implemented for a wide range of devices.

1.1 Scope

This standard defines physical layer (PHY) and medium access control (MAC) specifications for wireless connectivity with fixed, portable, and moving devices within or entering a personal operating space (POS). A goal of the IEEE 802.15.1™ Task Group is to achieve a level of interoperability that could allow the transfer of data between a WPAN device and an IEEE 802.11™ device.

A POS is the space about a person or object that typically extends up to 10 m in all directions and envelops the person whether stationary or in motion. This standard has been developed to ensure coexistence with all IEEE 802.11 networks.

1.2 WPAN definition

The term *WPAN* in this standard refers specifically to a wireless personal area network as used in this standard.

Specifically, this standard:

- — Describes the functions and services required by an IEEE 802.15.1 device to operate within ad hoc networks.
- — Describes the MAC procedures to support the asynchronous connectionless (ACL) and synchronous connection-oriented (SCO) link delivery services:
  - — The baseband layer, specifying the lower level operations at the bit and packet levels, e.g., forward error correction (FEC) operations, encryption, cyclic redundancy check (CRC) calculations, Automatic Repeat Request (ARQ) Protocol.
  - — The link manager (LM) layer, specifying connection establishment and release, authentication, connection and release of SCO and ACL channels, traffic scheduling, link supervision, and power management tasks.

- The Logical Link Control and Adaptation Protocol (L2CAP) layer, forming an interface between standard data transport protocols and the Bluetooth™ protocol. It handles the multiplexing of higher layer protocols and the segmentation and reassembly (SAR) of large packets. The data stream crosses the LM layer, where packet scheduling on the ACL channel takes place. The audio stream is directly mapped on an SCO channel and bypasses the LM layer. The LM layer, though, is involved in the establishment of the SCO link. Between the LM layer and the application, control messages are exchanged in order to configure the Bluetooth transceiver for the considered application.

— Describes the 2.4 GHz industrial, scientific, medical (ISM) band PHY signaling techniques and interface functions that are controlled by the IEEE 802.15.1 MAC. Requirements are defined for two reasons:

  - To provide compatibility between the radios used in the system.
  - To define the quality of the system.

Above the L2CAP layer may reside the Serial Cable Emulation Protocol based on ETSI TS 07.10 (RFCOMM), Service Discovery Protocol (SDP), Telephone Control Protocol specification (TCS), voice-quality channels for audio and telephony, and other network protocols.

2. References

The following standards and specifications contain provisions that, through references in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards and specifications are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the references listed below.

2.1 IEEE documents

IEEE Std 802®-2001, IEEE Standards for Local and Metropolitan Area Networks: Overview and Architecture.[3, 4]

2.2 ISO documents

ISO/IEC 3309:1993 Information technology—Telecommunications and information exchange between systems—High-level data link control (HDLC) procedures—Frame structure.[5]

ISO/IEC 7498-1:1994, Information technology—Open Systems Interconnection—Basic Reference Model: The Basic Model.

ISO/IEC 8802-2:1998, Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 2: Logical link control.

ISO/IEC 10039:1991, Information technology—Open Systems Interconnection—Local Area Networks—Medium Access Control (MAC) Service Definition.

ISO/IEC 15802-1:1995, Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Common specifications—Part 1: Medium Access Control (MAC) service definition.

2.3 ITU documents

ITU-T Recommendation G.711 (11/88), Pulse code modulation (PCM) of voice frequencies.[6]

ITU-T Recommendation O.150 (05/96), Digital test patterns for performance measurements on digital transmission equipment.

ITU-T Recommendation O.153 (10/92), Basic parameters for the measurement of error performance at bit rates below the primary rate.

ITU-T Recommendation X.200 (07/94), Information technology—Open systems interconnection—Basic reference model: The basic model.

---

[3]IEEE and 802 are registered trademarks in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Incorporated.

[4]IEEE publications are available from the Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (http://standards.ieee.org/).

[5]ISO/IEC publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse (http://www.iso.ch/). ISO/IEC publications are also available in the United States from Global Engineering Documents, 15 Inverness Way East, Englewood, Colorado 80112, USA (http://global.ihs.com/). Electronic copies are available in the United States from the American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (http://www.ansi.org/).

[6]ITU-T publications are available from the International Telecommunications Union, Place des Nations, CH-1211, Geneva 20, Switzerland/Suisse (http://www.itu.int/).

ITU-T Recommendation Z.100 (11/99), Programming languages—Specification and description language (SDL).

2.4 Bluetooth documents[7]

2.4.1 Bluetooth Core Specification Volume 1

Bluetooth Special Interest Group, “Core Specification Version 1.1,” Specification of the Bluetooth System, February 22, 2001. [Bluetooth_11_Specifications_Book.pdf]

2.4.2 Bluetooth Profiles Specification Volume 2

Bluetooth Special Interest Group, “Profiles Specification Version 1.1,” Specification of the Bluetooth System, February 22, 2001. [Bluetooth_11_Profiles_Book.pdf]

2.4.3 Bluetooth Assigned Numbers

Bluetooth Special Interest Group, “Bluetooth Assigned Numbers Version 1.1,” Specification of the Bluetooth System, February 22, 2001. [Bluetooth_11_Assigned_Numbers.pdf]

2.4.4 Bluetooth continuous variable slope delta (CVSD) encoded test signal

Bluetooth Special Interest Group, “Bluetooth CVSD encoded test signal Version 2.1,” A test signal file providing the digital CVSD encoded test signal as referred to in the Bluetooth Specification, February 22, 2001. [BluetoothSpecFiles_new_tar.gz]

2.4.5 Bluetooth Personal Area Networking Profile

Bluetooth Special Interest Group, “Bluetooth Personal Area Networking Profile Revision 0.95a,” June 26, 2001. [PAN-Profile.pdf]

2.4.6 Bluetooth Network Encapsulation Protocol (BNEP) Specification

Bluetooth Special Interest Group, “Bluetooth Network Encapsulation Protocol (BNEP) Specification Revision 0.95a,” June 12, 2001. [BNEP.pdf]

2.5 Other documents

Infrared Data Association, IrDA Object Exchange Protocol (IrOBEX), Version 1.2, April 1999.[8]

Internet Engineering Task Force, “A Proposed Flow Specification,” RFC 1363, September 1992.[9]

Internet Engineering Task Force, “The Point-to-Point Protocol (PPP),” RFC 1661, July 1994.

[7]Bluetooth documents are available from: http://www.bluetooth.com/dev/specifications.asp or via the IEEE website: http://ieee802.org/15/Bluetooth/.

[8]IrDA documents are available from http://www.irda.org/.

[9]Internet Engineering Task Force documents are available from http://www.ietf.org/.

3. Definitions

For the purposes of this standard, the following terms and definitions apply. *IEEE 100™, The Authoritative Dictionary of IEEE Standards Terms*, Seventh Edition [B4][9], should be referenced for terms not defined in this clause.

3.1 ad hoc network: A network typically created in a spontaneous manner. The principal characteristic of an ad hoc network is its limited temporal and spatial extent.

3.2 asynchronous connectionless (ACL) link: A point-to-multipoint link between the master and all the slaves participating on the piconet. In the slots not reserved for the synchronous connection-oriented (SCO) links, the master can establish an ACL link on a per-slot basis to any slave, including the slaves already engaged in an SCO link.

3.3 attachment gateway (AG): A communications node with at least two communication interfaces, one of which is a Bluetooth interface and one of which is an interface to another network. An AG is used to attach a Bluetooth wireless personal area network (WPAN) to the other network. In particular, an 802 local area network (LAN) AG attaches a Bluetooth WPAN to an 802 LAN, while a public switched telephone network (PSTN) AG attaches a Bluetooth WPAN to the PSTN.

3.4 authenticated device: A Bluetooth device whose identity has been verified during the lifetime of the current link, based on the authentication procedure.

3.5 authentication: A generic procedure based on Link Manager Protocol (LMP) authentication if a link key exists or on LMP-pairing if no link key exists.

3.6 authorization: A procedure where a user of a Bluetooth device grants a specific (remote) Bluetooth device access to a specific service. Authorization implies that the identity of the remote device can be verified through authentication.

3.7 authorize: The act of granting a specific Bluetooth device access to a specific service. It may be based on user confirmation or on the existence of a trusted relationship.

3.8 Bluetooth baseband: The layer that specifies the medium access control and physical layer procedures to support the exchange of real-time voice, data information streams, and ad hoc networking between Bluetooth units.

3.9 Bluetooth channel: A channel that is divided into time slots in which each slot corresponds to a radio frequency (RF) hop frequency. Consecutive hops correspond to different RF hop frequencies and occur at a nominal hop rate of 1600 hops/s. These consecutive hops follow a pseudo-random hopping sequence, hopping through either a 79 or 23 RF channel set.

3.10 Bluetooth host controller interface (HCI): A command interface to the baseband controller and link manager and access to hardware status and control registers. This interface provides a uniform method of accessing the Bluetooth baseband capabilities.

3.11 Bluetooth host: A computing device, peripheral, cellular telephone, 802 local area network attachment gateway (AG), public switched telephone network AG, etc. A Bluetooth host attached to a Bluetooth unit may also communicate with other Bluetooth hosts attached to their Bluetooth units.

[9]The numbers in brackets correspond to the numbers in the bibliography in Annex H.

3.12 Bluetooth unit: A voice and data circuit equipment for a short-range wireless communication link. It allows voice and data communications between Bluetooth hosts.

3.13 Bluetooth: A wireless communication link, operating in the unlicensed industrial, scientific, medical band at 2.4 GHz using a frequency-hopping transceiver. It allows real-time voice and data communications between Bluetooth hosts. The link protocol is based on time slots.

3.14 bond: A relation between two Bluetooth devices defined by creating, exchanging, and storing a common link key. The bond is created through the bonding or Link Manager Protocol-pairing procedures.

3.15 bonding: A dedicated procedure for performing the first authentication, where a common link key is created and stored for future use.

3.16 channel establishment: A procedure for establishing a channel on the Logical Link Control and Adaptation Protocol level.

3.17 channel: A logical connection on the Logical Link Control and Adaptation Protocol level between two devices serving a single application or higher layer protocol.

3.18 connect (to service): To establish a connection to a service. If not already done, this procedure includes the establishment of a physical link, link, and channel.

3.19 connectable device: A Bluetooth device in range that will respond to a page.

3.20 connecting: A phase in the communication between devices when a connection between them is being established. (Connecting phase follows after the link establishment phase is completed.)

3.21 connection establishment: A procedure for creating a connection mapped onto a channel.

3.22 connection: A connection between two peer applications or higher layer protocols mapped onto a channel.

3.23 coverage area: The area where two Bluetooth units can exchange messages with acceptable quality and performance.

3.24 creation of a secure connection: A procedure of establishing a connection, including authentication and encryption.

3.25 creation of a trusted relationship: A procedure where the remote device is marked as a trusted device. This procedure includes storing a common link key for future authentication and pairing (if the link key is not available).

3.26 device discovery: A procedure for retrieving the Bluetooth device address, clock, class-of-device field, and used paging scheme from discoverable devices.

3.27 discoverable device: A Bluetooth device in range that will respond to an inquiry (normally in addition to responding to a page).

3.28 idle (or in idle mode): As seen from a remote device, the condition of a Bluetooth device when no link is established between the remote device and the Bluetooth device.

3.29 inquiry: A message sent by a Bluetooth unit to discover the other Bluetooth units that are within the coverage area. The Bluetooth units that capture inquiry messages may send a response to the inquiring Bluetooth unit. The response contains information about the Bluetooth unit itself and its Bluetooth host.

3.30 isochronous user (UI) channel: A channel used for time-bounded information, e.g., compressed audio (asynchronous connectionless link).

3.31 known device: A Bluetooth device for which at least the Bluetooth device address is stored.

3.32 link establishment: A procedure for establishing a link on the Link Manager Protocol (LMP) level. A link is established when both devices have agreed that LMP setup is completed.

3.33 link: An asynchronous connectionless link.

3.34 Link-Manager-Protocol (LMP) authentication: A LMP-level procedure for verifying the identity of a remote device. The procedure is based on a challenge-response mechanism using a random number, a secret key, and the Bluetooth device address of the noninitiating device. The secret key can be a previously exchanged link key.

3.35 Link-Manager-Protocol (LMP) pairing: A procedure that authenticates two devices, based on a personal identification number (PIN), and subsequently creates a common link key that can be used as a basis for a trusted relationship or a (single) secure connection. The procedure consists of the following steps: creation of an initialization key (based on a random number and a PIN), creation and exchange of a common link key, and LMP-authentication based on the common link key.

3.36 logical channel: The different types of channels on a physical link.

3.37 mode: A set of directives that define how a device responds to certain events.

3.38 name discovery: A procedure for retrieving the user-friendly name (i.e., the Bluetooth device name) of a connectable device.

3.39 packet: Format of aggregated bits that can be transmitted in one, three, or five time slots.

3.40 page: A baseband substate where a device transmits page trains and processes any eventual responses to the page trains. Also, the transmission by a device of page trains containing the device access code of the device to which the physical link is requested.

3.41 page scan: A baseband substate where a device listens for page trains. Also, the listening by a device for page trains containing its own device access code.

3.42 paging: Messages sent out by a Bluetooth unit to set up a communication link to another Bluetooth unit that is active within the coverage area.

3.43 paired device: A Bluetooth device with which a link key has been exchanged (either before connection establishment was requested or during connecting phase).

3.44 physical channel: Synchronized radio frequency hopping sequence in a piconet.

3.45 physical link: A baseband-level connection between two devices. The connection is established using paging. A physical link comprises a sequence of transmission slots on a physical channel alternating between master and slave transmission slots.

3.46 piconet: The Bluetooth units sharing a common channel.

3.47 pre-paired device: A Bluetooth device with which a link key was exchanged and stored before link establishment.

3.48 RFCOMM client: An application that requests a connection to another application (i.e., RFCOMM server).

3.49 RFCOMM initiator: The device initiating the RFCOMM session, i.e., setting up RFCOMM channel on Logical Link Control and Adaptation Protocol and starting RFCOMM multiplexing with the set asynchronous balanced mode command on data link connection identifier set to 0.

3.50 RFCOMM server: An application that waits for a connection from an RFCOMM client on another device. What happens after such a connection is established is out of the scope of this definition.

3.51 RFCOMM server channel: A subfield of the TS 07.10 data link connection identifier number. This abstraction is used to allow both server and client applications to reside on both sides of an RFCOMM session.

3.52 scatternet: Two or more piconets co-located in the same area with interpiconet communication.

3.53 specification and description language (SDL): A modern, high-level programming language. It is object-oriented, formal, textual, and graphical. SDL is intended for the description of complex, event-driven, real-time, and communication systems.

3.54 service discovery: Procedures for querying and browsing for services offered by or through another Bluetooth device.

3.55 silent: As seen from a remote device, the condition of a Bluetooth device if it does not respond to inquiries made by the remote device. A device may be silent due to being nondiscoverable or due to baseband congestion while being discovered.

3.56 synchronous connection-oriented (SCO) link: A point-to-point link between a master and a single slave in the piconet. The master maintains the SCO link by using reserved slots at regular intervals.

3.57 time slot: A part of the physical channel. Each time slot is 625 μs long.

3.58 trusted device: A paired device that is explicitly marked as trusted.

3.59 trusting: The marking of a paired device as trusted. Trust marking can be done by the user or done by the device automatically after a successful pairing.

3.60 unknown device: A Bluetooth device for which no information (e.g., Bluetooth device address, link key) is stored.

3.61 unpaired device: A Bluetooth device for which no exchanged link key was available before connection establishment was requested.

4. Acronyms and abbreviations

ac	alternating current
ACK	acknowledgment
ACL	asynchronous connectionless (link)
ACO	authenticated ciphering offset
AES	advanced encryption standard
AG	attachment gateway
AM_ADDR	active member address
AR_ADDR	access request address
ARIB	Association of Radio Industries and Businesses
ARQ	automatic repeat request
ARQN	automatic repeat request negative
ASN.1	abstract syntax notation one
BB	baseband
BCH	Bose-Chaudhuri-Hocquenghem
BD_ADDR	Bluetooth device address
BER	bit error rate
BNEP	Bluetooth Network Encapsulation Protocol
BQA	Bluetooth qualification administrator
BSIG	Bluetooth Special Interest Group, Inc.
BT	bandwidth time product (i.e., B*T)
CAC	channel access code
CC	call control
CDMA	code divison multiple access
CID	channel identifier
CL	connectionless
COD	class of device
CODEC	coder decoder
COF	ciphering offset number
CRC	cyclic redundancy check
CSMA/CD	carrier sense multiple access with collision detection
CVSD	continuous variable slope delta (modulation)
DAC	device access code
dc	direct current
DCE	data communication equipment
DCI	default check initialization
DCID	destination channel identifier
DH	data-high rate
DIAC	dedicated inquiry access code
DLC	data link control
DLCI	data link connection identifier
DLL	data link layer
DM	data-medium rate
DQPSK	differential quadrature phase shift keying
DSAP	destination address field
DTE	data terminal equipment
DTMF	dual tone multiple frequency

DUT	device under test
DV	data - voice
ED	energy detection
EIFS	extended interframe space
ERTX	expanded response timeout expired
ETC	extreme test conditions
ETSI	European Telecommunications Standards Institute
FC	frame control
FCC	Federal Communications Commission
FCS	frame check sequence
FEC	forward error correction
FER	frame error rate
FH	frequency hopping
FHS	frequency hop synchronization
FHSS	frequency hopping spread spectrum
FIFO	first in first out
FSK	frequency shift keying
FW	firmware
GAP	generic access profile
GEOP	generic object exchange profile
GFSK	Gaussian frequency shift keying
GIAC	general inquiry access code
GM	group management
HA	host application software using Bluetooth
HC	host controller
HCI	host controller interface
HEC	header error check
HID	human interface device
HPC	hand-held personal computer
HV	high-quality voice
HW	hardware
IAC	inquiry access code
ICS	implementation conformance statement
ICV	integrity check value
ID	identity or identifier
IDU	interface data unit
IETF	Internet Engineering Task Force
IP	Internet Protocol
IrDA	Infrared Data Association
IrMC	infrared mobile communications
ISDN	integrated services digital networks
ISM	industrial, scientific, medical
IUT	implementation under test
IV	initialization vector
L_CH	logical channel
L2CA	logical link control and adaptation
L2CAP	Logical Link Control and Adaptation Protocol
LAN	local area network

LAP	lower address part
LC	link controller
LCID	local channel identifier
LCP	Link Control Protocol
LCSS	link controller service signalling
LFSR	linear feedback shift register
LIAC	limited inquiry access code
LLC	logical link control
LM	link manager
LME	layer management entity
LMP	Link Manager Protocol
Log PCM	logarithimic pulse coded modulation
LP	Lower Layer Protocol
LPO	low-power oscillator
LSB	least significant bit
M	master or mandatory
MAC	medium access control
MAPI	messaging application procedure interface
MDF	management-defined field
MIB	management information base
MLME	MAC sublayer management entity
MMI	man-machine interface
MPDU	MAC protocol data unit
MPT	Ministry of Post and Telecommunications
MSB	most significant bit
MSC	message sequence chart
MSDU	MAC service data unit
MTU	maximum transmission unit
MUX	multiplexing sublayer a sublayer of the L2CAP layer
NAK	negative acknowledgment
NAP	nonsignificant address part
NOP	no operation
NTC	nominal test condition
O	optional
OBEX	Object Exchange Protocol
OCF	opcode command field
OGF	opcode group field
OSI	open systems interconnection
PAN	personal area network
PAR	project authorization request
PC	personal computer
PCM	pulse coded modulation
PCMCIA	Personal Computer Memory Card International Association
PCS	personal communications service
PDA	personal digital assistant
PDU	protocol data unit
PHT	pseudo-Hadamard transform
PHY	physical layer

PICS	Protocol Implementation Conformance Statement
PIN	personal identification number
PLCP	physical layer convergence procedure
PLME	physical layer management entity
PM_ADDR	parked member address
PMD	physical medium dependent
PN	pseudo-random noise
PnP	plug and play
POS	personal operating space
POTS	plain old telephone service
PPDU	PHY protocol data unit
PPP	Point-to-Point Protocol
PRBS	pseudo-random bit sequence
PRD	program reference document
PRNG	pseudo-random number generator
PS	power save
PSM	protocol/service multiplexer
PSTN	public switched telephone network
QoS	quality of service
RA	receiver address
RAND	random number
RF	radio frequency
RFC	request for comments
RFCOMM	Serial Cable Emulation Protocol based on ETSI TS 07.10
RSSI	receiver signal strength indication
RTS	request to send
RTX	response timeout expired
RX	receive or receiver
S	slave
SA	source address
SABM	set asynchronous balanced mode
SAP	service access point
SAR	segmentation and reassembly
SC	scan period
SCID	source channel identifier
SCO	synchronous connection-oriented (link)
SD	service discovery
SDDB	service discovery database
SDL	specification and description language
SDP	Service Discovery Protocol
SDU	service data unit
SEQN	sequential numbering scheme
SFD	start frame delimiter
SIFS	short inter-frame space
SIG	special interest group
SLRC	station long retry count
SME	station management entity
SQ	signal quality

SR	scan repetition
SRC	short retry count
SRES	signed response
SS	supplementary services
SSAP	source address field
SSI	signal strength indication
SSRC	station short retry count
SUT	system under test
SW	software
TA	transmitter address
TAE	terminal adapter equipment
TBD	to be defined
TBTT	target beacon transmission time
TC	test control layer for the test interface
TCI	test control interface
TCP	Transmission Control Protocol
TCP/IP	Transport Control Protocol/Internet Protocol
TCS	Telephony Control Protocol specification
TDD	time division duplex
TDMA	time division multiple access
TS	technical specification
TSF	timing synchronization function
TTP	Tiny Transport Protocol
TX	transmit or transmitter
TXE	transmit enable
UA	user asynchronous
UAP	upper address part
UART	universal asynchronous receiver transmitter
UC	user control
UDP	User Datagram Protocol
UDP/IP	User Datagram Protocol/Internet Protocol
UI	user isochronous or user interface, depending on context
URL	uniform resource locator
US	user synchronous
USB	universal serial bus
UT	upper tester
UUID	universally unique identifier
WAN	wide area network
WAP	Wireless Application Protocol
WLAN	wireless local area network
WPAN	wireless personal area network
WUG	wireless user group

5. General description

This standard is derived from the Bluetooth core, profiles, and test specifications (version 1.1) (see Siep [B13]). Portions of this standard consist of unaltered or minimally altered text of the Bluetooth specifications (see 2.4). This clause provides the general description of this standard and identifies the source of each of the subsequent clauses. The Bluetooth wireless technology is an industry specification for small form factor, low-cost wireless communication and networking between personal computers (PCs), mobile phones, and other portable devices.

5.1 IEEE and Bluetooth Special Interest Group (SIG), Inc., license agreement

The Bluetooth SIG wanted to have the IEEE adopt the Bluetooth specifications and make them a formal IEEE 802 standard. The IEEE requested and was granted a limited, nonexclusive, nontransferable license from the Bluetooth SIG to adopt or adapt and copy a portion of the Bluetooth specifications to be used as base material in IEEE Std 802.15.1-2002. An agreement in principle was struck in mid-1999, and final agreement was achieved in 2000.

Specifically, the license allows IEEE to create, publish, and distribute the standard as a stand-alone publication or as part of a collection together with other IEEE standards. The area covered by this standard is undergoing evolution. Revisions are anticipated within the next few years to clarify existing material, correct possible errors, and incorporate new related or supplement material. Information on the license or current revision state of this standard may be obtained from the Secretary, IEEE Standards Board. Information on the license or the current Bluetooth specifications should be directed to the Bluetooth SIG.

5.2 The origin of the document and layout

The first five clauses are standard IEEE introductory information. The overview, reference citation, unique definition, and acronym and abbreviation clauses are common to all IEEE 802 base standards. This clause, Clause 5, is generally used in these standards to provide guidance to the reader about the form and content of the standard. In this standard, Clause 5 is devoted to the specific relationship between this document and the original Bluetooth specifications. Clause 6 describes the interworkings and architecture of this standard. It also relates those attributes to the original Bluetooth constructs.

For the next five clauses, Clause 7 through Clause 11, the standard is derived from the Bluetooth core (Volume 1) specification (see 2.4.1). Clause 11 about host controller interface (HCI) is the only Bluetooth-derived section that has undergone significant editorial modification. These edits were applied to delete implementation-specific text as well as unrelated text. Clause 12 was added by IEEE to define the service access points (SAPs). IEEE 802-specific interfaces are documented in Clause 12. These interfaces describe how the lower layers of a Bluetooth implementation would interface with the traditional IEEE 802.2™[9] logical link control (LLC) entity.

Annex A is also a derived text and corresponds to the Bluetooth Protocol Implementation Conformance Statement (PICS) proforma, which is a separate document from the Bluetooth specifications. Annex B was added by IEEE to define the high-level behavioral specification and description language (SDL) model for an integrated WPAN Bluetooth MAC sublayer (e.g., L2CAP, LM, and baseband) based on ITU-T Recommendation Z.100 (11/99)[10]. For the next five annexes, Annex C through Annex G, the standard uses additional derivation materials from the Bluetooth core (Volume 1) and profiles (Volume 2) specifications (see 2.4.1 and 2.3) to assist the reader: Part K:1, Generic Access Profile, from Volume 2, and Appendix VII, Optional Paging Scheme; Part I:1, Bluetooth Test Mode; Appendix VI, Baseband Timers; and Appendix IX, Message Sequence Charts, from Volume 1, respectively. The last annex is the bibliography.

[9]IEEE standards are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

[10]For information on references, see Clause 2.

Figure 1 maps the IEEE Std 802.15.1-2002 clauses to the applicable portion of the Bluetooth protocol stack.

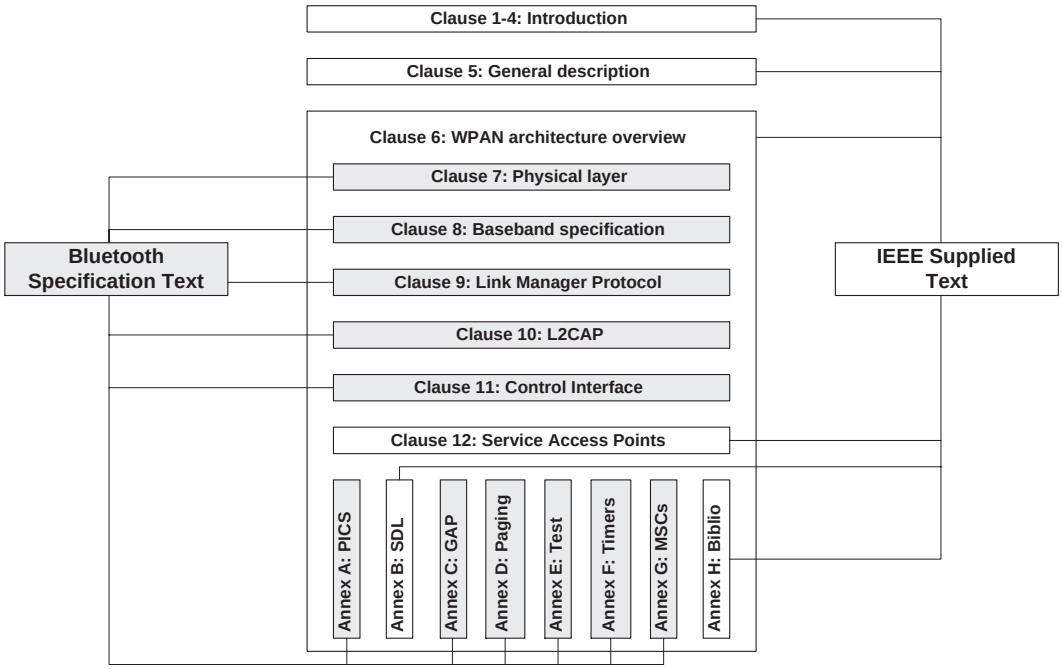


Figure 1—IEEE Std 802.15.1-2002 derived text

In Figure 1 the lines from the left-hand graphic clearly identify the shaded portions of this IEEE standard that consist of the text of the Bluetooth specifications. In most cases the shaded portions of this IEEE standard consist of unaltered or minimally altered text from the Bluetooth specifications.

Table 1 provides a detailed description of each of the clauses and annexes.

Table 1—Clause and annex descriptions

IEEE clause or annex (Bluetooth specification part)	Title	Description
Front matter	Front matter	The front matter includes mutually agreed-to copyright statement from the Bluetooth SIG and IEEE-SA.
Clause 1	Overview	Overview
Clause 2	References	The references include the derivative documents from the the Bluetooth SIG.
Clause 3	Definitions	Definitions
Clause 4	Acronyms and abbreviations	Acronyms and abbreviations
Clause 5	General description	This clause describes the source, contents, and format of the standard. The purpose is to identify the Bluetooth specification source material and make the standard easier to read for readers familiar with the Bluetooth specifications.

Table 1—Clause and annex descriptions *(continued)*

IEEE clause or annex (Bluetooth specification part)	Title	Description
Clause 6	WPAN architecture overview	The architectural clause emphasizes the large-scale separation of the system into two parts: the IEEE 802.15.1 PHY and the MAC sublayer of the data link layer (DLL).
Clause 7 (Part A)	Physical layer (PHY)	The radio clause defines the requirements for a Bluetooth transceiver operating in the unlicensed ISM band.
Clause 8 (Part B)	Baseband specification	The baseband clause describes the specifications of the Bluetooth link controller (LC) that carries out the baseband protocols and other low-level link routines.
Clause 9 (Part C)	Link Manager Protocol (LMP)	The LMP is used for link setup and control. The signals are interpreted and filtered out by the LM on the receiving side and are not propagated to higher layers.
Clause 10 (Part D)	Logical link control and adaptation protocol (L2CAP)	The Bluetooth L2CAP supports higher level protocol multiplexing, packet SAR, and the conveying of quality-of-service information. This clause also describes the protocol state machine, the packet format and composition, and a test interface required for the Bluetooth test and certification program.
Clause 11 (Part H:1)	Control interface	Control information from upper layers flows through this interface. The text of this clause was taken from the host controller interface section of the Bluetooth specification and modified to eliminate references to specific physical interfaces and their control parameters.
Clause 12	Service access point interfaces and primitives	Various entities within this standard interact in various ways. Some of these interactions are defined explicitly within this standard, via a SAP across which defined primitives are exchanged.
Annex A (Bluetooth ICS & IXIT pro forma)	Protocol implementation conformance statement (PICS proforma)	The supplier of a protocol implementation that is claimed to conform to IEEE Std 802.15.1-2002 shall complete the PICS proforma.
Annex B	Formal description of IEEE Std 802.15.1-2002 operation	SDL is the formal, object-oriented language that defines the IEEE Std 802.15.1-2002 MAC sublayer.
Annex C (V2, Part K:1)	Generic access profile	This annex defines the generic procedures related to discovery of Bluetooth devices (idle mode procedures) and link management aspects of connecting to Bluetooth devices (connecting mode procedures). It also defines procedures related to the use of different security levels. In addition, this profile includes common format requirements for parameters accessible on the user interface level.
Annex D (Appendix VII)	Optional paging schemes	For the access procedure, several paging schemes may be used. One mandatory paging scheme shall be supported by all Bluetooth devices. (This scheme is described in Clause 8.) In addition to the mandatory scheme, a Bluetooth unit may support one or more optional paging schemes. This annex contains these optional schemes.
Annex E (Part I:1)	Bluetooth test mode	This annex describes the test mode for hardware and low-level functionality tests of Bluetooth devices. The test mode includes transmitter tests (packets with constant bit patterns) and loopback tests.

Table 1—Clause and annex descriptions *(continued)*

IEEE clause or annex (Bluetooth specification part)	Title	Description
Annex F (Appendix VI)	Baseband timers	This annex contains a list of all timers defined in the baseband specification. Definitions and default values of the timers are listed. All timer values are given in slots.
Annex G (Appendix IX)	Message sequence charts (MSCs)	This annex shows examples of interworking between HCI commands and LM protocol data units (PDUs) in the form of MSCs. It helps the reader understand and correctly use the HCI commands.
Annex H	Bibliography	Bibliography

6. WPAN architecture overview

The essential components of the IEEE Std 802.15.1-2002 WPAN architecture are based on the Bluetooth specifications (see 2.4). The terms *Bluetooth WPAN* and *IEEE 802.15.1 WPAN* refer to the specific, single example of a WPAN presented in this standard.

This clause presents the IEEE 802.15.1 WPAN technology and its architecture. It also presents an overview of the major components of the Bluetooth WPAN protocol stack including a high-level summary of this standard and its relation with the Bluetooth Foundation Specification.

6.1 The WPAN communications technology

Personal electronic devices are becoming more intelligent and interactive. Many devices [e.g., notebook computers, cellular phones, personal digital assistants (PDAs), personal solid-state music players, digital cameras] have increased data capabilities. This capability allows them to retain, use, process, and communicate various amounts of information. For example, several of these personal devices have a personal information management (PIM) database maintaining personal calendars, address books, and to-do lists. PIM databases in one personal device should remain synchronized with PIM databases in other personal devices. The obvious solution for keeping these PIM databases synchronized is to interconnect and synchronize them.

Traditionally, proprietary special-purpose cables have been used to interconnect personal devices. However, many users find using these cables to be quite a frustrating and unproductive chore. Cables may get lost or damaged and they add unnecessary bulk and weight when carried around. It thus becomes quite desirable to develop connectivity solutions for interconnecting personal devices that do not require the use of cables. Because the desired solution will not involve interconnection cables, a wireless solution must be employed.

6.1.1 General requirements

The demands of the consumer market require connectivity solutions that respect the primary functionality of the personal device. For example, a communications-enabled PDA must still look and function as a PDA. Its wireless connectivity solution must not impact the PDA's form factor, weight, power requirements, cost, ease of use, or other traits in a significant way.

Personal devices used as a part of an individual's productivity management and entertainment tool set may also be needed to interact with a corporate information technology (IT) infrastructure. Further, these devices will likely be used in several of many different environments (e.g., offices, homes, in the middle of a park, industrial plants). The wireless solution for these devices should, therefore, accommodate the design and marketing requirements dictated not only by the consumer market but also by the business market.

Interconnecting personal devices is different from connecting computing devices. Typical connectivity solutions for computing devices (e.g., a WLAN connectivity solution for a notebook computer) associates the user of the device with data services available on, for instance, a corporate Ethernet-based intranet. This situation contrasts with the intimate, personal nature of a wireless connectivity solution for the personal devices associated with a particular user. The user is concerned with electronic devices in his or her possession, or in his or her vicinity, rather than to any particular geographic or network location. The term *personal area network* (PAN) was coined to describe this different kind of network connection. The untethered version of this concept is a WPAN. A WPAN can be viewed as a personal communications bubble around a person. Within this bubble, which moves as a person moves around, personal devices can connect with one another. These devices may be under the control of a single individual or several people's devices may interact with each other.

Communicating devices may not be within line of sight of each other. For this reason WPANs may employ radio frequency (RF) technologies to provide the added flexibility to communicate to hidden devices. This standard presents a WPAN using RF technology based on the Bluetooth wireless technology.

6.1.2 How WPANs differ from WLANs

At first glance the operation and objectives of a WPAN may appear to resemble the operation and objectives of a WLAN, e.g., IEEE 802.11. Both the WLAN and WPAN technologies allow a device to connect to its surrounding environment and exchange data with it over an unlicensed, wireless link. However, WLANs have been designed and are optimized for usage of transportable, computing (client) devices (e.g., notebook computers). WPAN devices are even more mobile.

The two technologies differ in three fundamental ways:

— Power levels and coverage

— Control of the media

— Lifespan of the network

6.1.2.1 Power levels and coverage

To extend the WLAN as much as possible, while minimizing the burden of multihop networks, a WLAN installation is often optimized for coverage. Typical coverage distances are about 100 m and are implemented at the expense of power consumption (typically 100 mW or more of transmit power). The added power consumption for covering larger distances has an impact on the devices participating in a WLAN: They tend to be connected to a power plug on the wall or utilize the wireless link for a relatively short time while unplugged.

A WLAN enables ease of deployment, where the much more desirable (with respect to reliability and bandwidth) cables are hard or costly to deploy. WLAN links have been designed to serve as a substitute for the physical cables of a LAN cabling infrastructure. While wireless connectivity allows for portability of the client devices, the WLAN itself is quasi-static. A client device in a WLAN is typically connected to a fixed base station, and on occasion it may roam between fixed base stations. While WLANs are much easier to deploy as compared with their wire line counterparts, they still need to be deployed and set up. Their primary orientation is reaching outward from portable devices to connect to an established infrastructure—wired or not.

WPANs are oriented to interconnect multiple mobile, personal devices. The distinction between “mobile” and “portable” devices in this discussion is that mobile devices typically operate on batteries and have a fleeting interconnection with other devices; portable devices are moved less frequently, have longer time periods of connections, and usually run from power supplied by wall sockets. A personal device may not necessarily have the need to access LAN-level data services, but access to data services on a LAN is not excluded.

In contrast to a WLAN, a WPAN trades coverage for power consumption. Through small coverage area (about 10 m), reduced power consumption (typically 1 mW of transmit power), and low power modes of operation, a WPAN can achieve sufficiently small power-consumption rates to enable portability. Several simple, power-conscious personal devices can, therefore, utilize a WPAN technology, share data, and be truly mobile.

6.1.2.2 Control of the medium

WLANs are typically wireless extensions of wire-line LANs. Like its wired counterpart, IEEE 802.3™[11] LAN (Ethernet), a WLAN's primary objective is to provide the type of infrastructure data services typically available through a LAN to the client devices. Once a client device joins a WLAN, the device typically stays connected to the LAN until the device moves away from the LAN's boundaries. These devices typically operate in an office or an industrial warehouse, a home, or similar infrastructure.

Given the high variety of personal devices that may participate in a WPAN, the WPAN technology should support applications with stringent (i.e., reserved) bandwidth requirements as well as applications with more flexible bandwidth requirements. To provide the necessary bandwidth guarantees for the various connections, the WPAN employs a controlling mechanism that regulates the transmissions of the devices in the WPAN.

WLANs employ similar coordination functions as an option, recognizing that over the large distances covered by them, it may not always be desirable to have a strict and absolute control of the media. When they do have this level of control, it is described as a "contention-free period," but it does not mean an interference-free environment. Other independently operating networks (of various technologies) may occasionally interfere with transmissions during a contention-free period. However, no contention resolution mechanisms are employed to recover from disturbed transmissions during a contention-free period. With this in mind, in IEEE 802.15.1 WPANs, all the time is contention free. This level of control is achieved by creating a relationship (in IEEE 802.15.1 WPANs, a master-slave relationship) between the devices and operating on a single, time-multiplexed, slotted system. The IEEE 802.15.1 WPAN master polls its collection of IEEE 802.15.1 WPAN slaves for transmissions, thus regulating the bandwidth assigned to them based on quality-of-service requirements that it enforces. Using small slots efficiently controls the jitter in transmissions experienced by the high-quality traffic. In addition, employing a frequency-hopping scheme with the small slots provides noise resilience from interference that may occur from other networks, including other independently operating IEEE 802.15.1 WPANs, operating over unlicensed bands.

The ad hoc nature of connectivity in a WPAN implies that devices may need to act as either a master or a slave at different times. As a result, the design objectives for the IEEE 802.15.1 WPAN technology (e.g., low cost, low power) still apply no matter whether a device implements the typical master-or-slave capability or only one of them. In all cases a master must be present for communications to occur.

Personal devices that participate in a WPAN are designed for their personal appeal and functionality. They are not designed to be members of an established networking infrastructure, even if they may connect to it when necessary. A typical WPAN device does not need to maintain a network-observable and network-controllable state. WLANs are required, for example, to maintain a management information base (MIB). As bona fide members of a larger infrastructure, this requirement is appropriate. However, with a WPAN technology, it may be inappropriate—if not impossible—for an end-to-end networking solution to be employed that is observed and controlled remotely over a network. Such end-to-end solutions can be built on top of the WPAN technology, are outside the scope of this standard, and will likely be application-specific.

6.1.2.3 Lifespan of the network

WLANs do not have an inherent or implied lifespan. They have "existence" independent of their constituent devices. If all the devices migrated out of a WLAN's coverage area and replacement units arrived, the WLAN would be said to have uninterrupted existence. This concept is not true for IEEE 802.15.1 WPANs. If the master does not participate, the network no longer exists.

[11]IEEE standards are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

In a WPAN a device creates a connection that lasts only for as long as needed and has a finite lifespan. For example, a file transfer application may cause a connection to be created only long enough to accomplish its goal. When the application terminates, the connection between the two devices may also be severed. The connections that a mobile client device creates in a WPAN are ad hoc and temporary in nature. The devices to which a personal device is connected in a WPAN at one moment may bear no resemblance to the devices to which it was previously connected or will connect in the future. For example, a notebook computer may connect with a PDA at one moment, a digital camera at another moment, and a cellular phone at yet another moment. At times, the notebook computer may be connected with any or all of these other devices. The WPAN technology must be able to support fast (i.e., in a few seconds) ad hoc connectivity with no need of predeployment of any type.

6.2 High-level view

This standard presents a WPAN that utilizes the Bluetooth wireless technology. In the text of this standard, unless otherwise stated, the term *Bluetooth WPAN* or simply *IEEE 802.15.1 WPAN* refers to a WPAN that utilizes the Bluetooth wireless technology. The term *Bluetooth wireless technology* and other similar terms are also used to further emphasize the use of this technology in the Bluetooth WPAN defined and described in this standard.

6.2.1 Open systems interconnection (OSI)

Two important ways exist to view any communications system design: architectural and functional. The architectural approach emphasizes the logical divisions of the system and how they fit together. The functional approach emphasizes the actual components, their packaging, and their interconnections.

This subclause presents the architectural view of a WPAN. It emphasizes the traditional large-scale separation of the system into two parts: the IEEE 802.15.1 PHY and the MAC sublayer of the DLL. These layers are intended to correspond closely to the lowest layers of the ITU-T Recommendation X.200 (07/94) or ISO-7498-1:1994.

Figure 2 shows the protocol stacks in the OSI seven-layer model and in the Bluetooth wireless technology and their relation as it pertains to this standard. As shown in Figure 2, the LLC and MAC sublayers together encompass the functions intended for the DLL of the OSI model. The definition of MAC-to-LLC service is specified in ISO/IEC 15802-1:1995.

6.2.2 Overview of the Bluetooth WPAN

The Bluetooth wireless technology uses a short-range radio link that has been optimized for power-conscious, battery-operated, small size, lightweight personal devices. A Bluetooth WPAN supports both synchronous communication channels for telephony-grade voice communication and asynchronous communications channels for data communications. These facilities enable a rich set of devices and applications to participate in the Bluetooth WPAN. For example, a cellular phone may use the circuit-switched channels to carry audio to and from a headset while concurrently using a packet-switched channel to exchange data with a notebook computer.

A Bluetooth WPAN is not created *a priori* and has a limited life span. It is created in an ad hoc manner whenever an application in a device desires to exchange data with matching applications in other devices. The Bluetooth WPAN may cease to exist when the applications involved have completed their tasks and no longer need to continue exchanging data.

The Bluetooth WPAN operates in the unlicensed 2.4 GHz ISM band. A fast frequency-hop (1600 hops/s) transceiver is used to combat interference and fading in this band (i.e., reduce the probability that all transmission is destroyed by interference). A Gaussian-shaped, binary frequency shift keying (FSK) with a

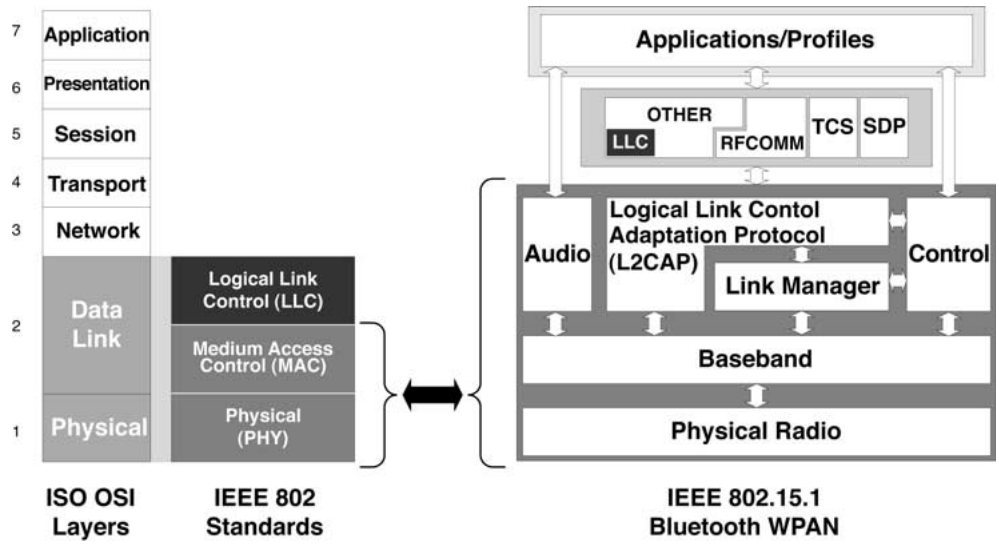


Figure 2—Mapping of ISO OSI to scope of IEEE 802.15.1 WPAN standard

symbol rate of 1 Msymbols/s minimizes transceiver complexity. A slotted channel is used, which has a slot duration of 625 µs. A fast time division duplex (TDD) scheme is used that enables full duplex communications at higher layers. On the channel, information is exchanged through packets. Each packet is transmitted on a different frequency in the hopping sequence. A packet nominally covers a single slot, but can be extended up to either three or five slots. For data traffic, a unidirectional (i.e., asymmetric) maximum of 723.2 kb/s is possible between two devices. A bidirectional 64 kb/s channel supports voice traffic between two devices. The jitter for the voice traffic is kept low by using small transmission slots.

Figure 3 shows the general format of a single-slot, payload-bearing packet transmitted over the air in a Bluetooth WPAN. The packet comprises a fixed-size access code, which is used, among other things, to distinguish one Bluetooth WPAN from another; a fixed-size packet header, which is used for managing the transmission of the packet in a Bluetooth WPAN; and a variable-size payload, which carries upper layer data. Due to the small size of these packets, large upper-layer packets need to be segmented prior to transmission over the air. Detailed information about this packet format can be found in 8.4.

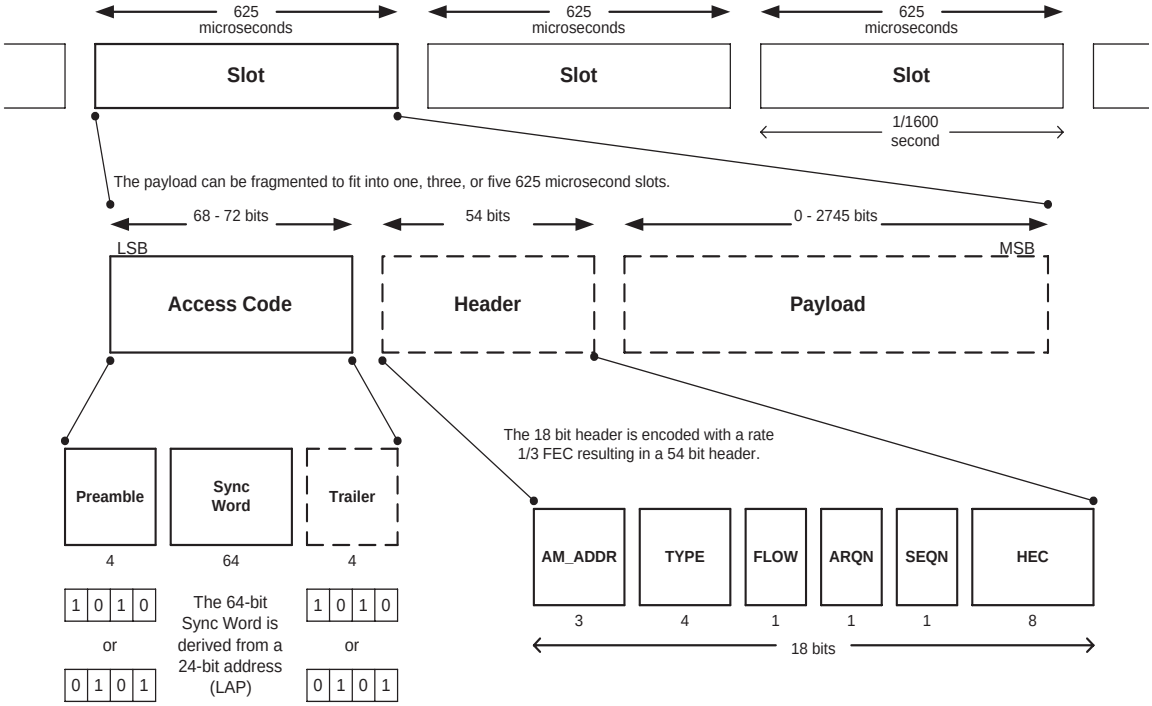


Figure 3—Format of an over-the-air payload bearing Bluetooth WPAN packet

6.2.3 The Bluetooth WPAN connectivity topologies

6.2.3.1 The Bluetooth WPAN piconet

A piconet is a WPAN formed by a Bluetooth device serving as a master in the piconet and one or more Bluetooth devices serving as slaves. A frequency-hopping channel based on the address of the master defines each piconet. All devices participating in communications in a given piconet are synchronized to the frequency-hopping channel for the piconet, using the clock of the master of the piconet. Slaves communicate only with their master in a point-to-point fashion under the control of the master. The master's transmissions may be either point-to-point or point-to-multipoint. Usage scenarios may dictate that certain devices act always as masters or slaves. However, this standard does not distinguish between devices with permanent master and slave designations. A slave device during one communications session could be a master in another and vice versa.

6.2.3.2 The Bluetooth WPAN scatternet

A scatternet is a collection of operational Bluetooth piconets overlapping in time and space. A Bluetooth device may participate in several piconets at the same time, thus allowing for the possibility that information could flow beyond the coverage area of the single piconet. A device in a scatternet could be a slave in several piconets, but master in only one of them. Figure 4 shows the various ways Bluetooth devices interconnect to form communicating systems.

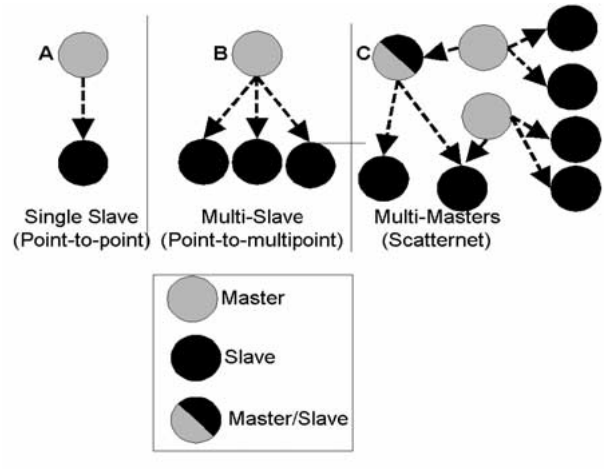


Figure 4—Various piconet formations: (a) single-slave operation; (b) multislave operation; and (c) scatternet operation

6.2.3.3 Integration with LANs

A Bluetooth WPAN may attach to and participate in communications with other LANs in the IEEE 802 family (e.g., IEEE 802.3, IEEE 802.11) through the use of an IEEE 802 LAN attachment gateway (AG) (see Figure 5). An IEEE 802 LAN AG is a logical architectural component that may or may not be implemented directly on a Bluetooth device. Through an IEEE 802 LAN AG, MAC service data units (MSDUs) from or to other LANs can be conditioned for transport over a Bluetooth WPAN.

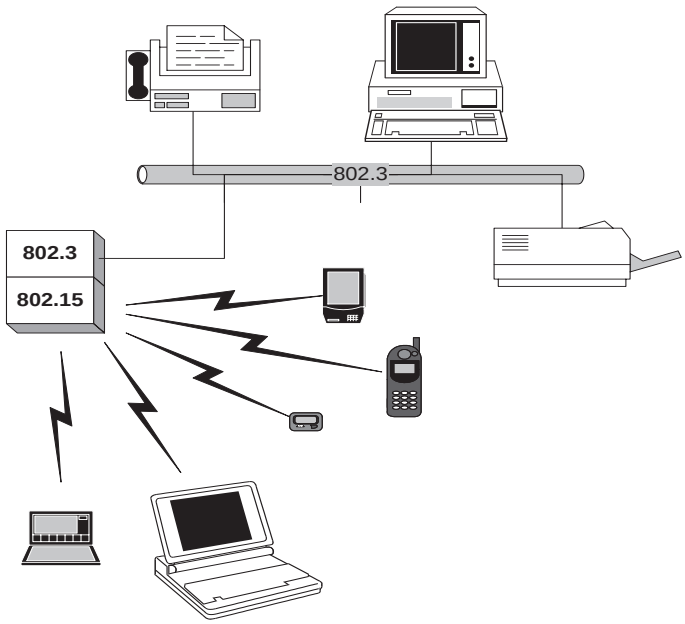


Figure 5—IEEE 802 LAN AG

6.3 Components of the Bluetooth WPAN architecture

The ultimate objective of communication protocols is to enable applications in different devices to interact with each other. To achieve this interactivity, compatible communication stacks need to run on the devices. This implies that not only the communication protocol stacks that run in each device are functionally compatible, but also the applications that run on top of these stacks match.

To enable the creation of interoperable, interactive applications the Bluetooth specifications (see 2.4) comprise both a set of communication protocols, including transport protocols for carrying data between devices over Bluetooth links, and a set of interoperable applications used to instantiate the collection of usage scenarios addressed in the specification. This standard pertains only to a subset of the communication protocols in the Bluetooth specifications related to PHY and MAC protocols as identified in Figure 2.

However, for completeness, the rest of this subclause contains a high-level description of the Bluetooth WPAN PHY and MAC communication protocols and their relation with the rest of the protocols in the Bluetooth specifications.

6.3.1 The Bluetooth protocol stack

Figure 6 shows the Bluetooth protocol stack. It includes both Bluetooth-specific protocols (e.g., LMP, L2CAP) and non-Bluetooth-specific protocols (grouped in the “Other” box). These other protocols include the Object Exchange Protocol (OBEX), the Point-to-Point Protocol (PPP), the Wireless Application Protocol (WAP), and so on. In designing the protocols and the whole protocol stack, the main principle has been to maximize the reuse of existing protocols for different purposes at the higher layers instead of “reinventing the wheel.” Protocol reuse also helps to adapt existing (i.e., legacy) applications to work with the Bluetooth wireless technology and ensure the smooth operation and interoperability of these applications. Thus, many applications already developed by vendors can take immediate advantage of hardware and software systems that are compliant to the Bluetooth specifications (see 2.4). The specifications are publicly available and permit the development of a large number of new applications that take full advantage of the capabilities of the Bluetooth wireless technology.

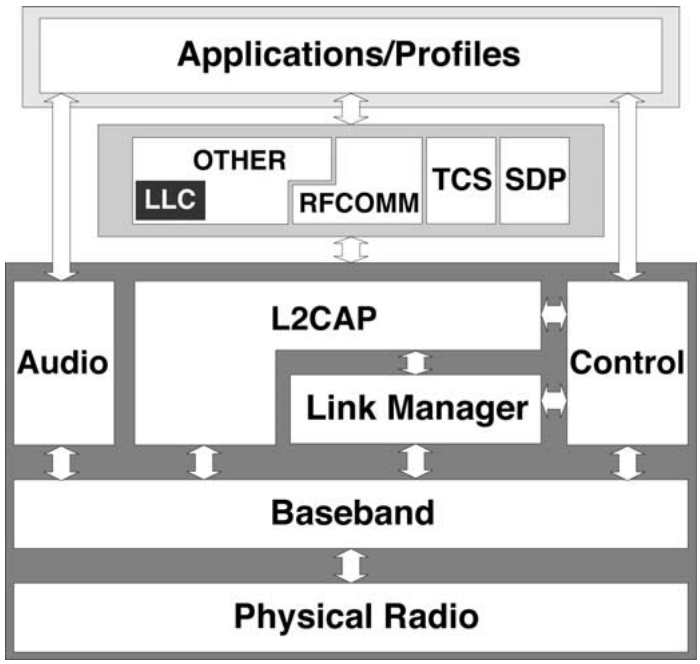


Figure 6—Bluetooth protocol stack

For this standard, an LLC layer has been explicitly added within the “Other” box. The LLC layer is not part of the Bluetooth specifications. It is shown here to demonstrate where it shall be placed in relation to the rest of the Bluetooth protocol layers. LLC traffic may be encapsulated within the BNEP packets prior to entering the lower Bluetooth protocol layers that map to the IEEE 802.15.1 MAC sublayer and PHY (see 2.4.6).

The RFCOMM layer is a serial port emulation layer for enabling legacy applications over Bluetooth links. The TCS is a telephony control and signaling layer for advanced telephony applications. The SDP is a service discovery layer allowing Bluetooth devices to ask other devices for the services that they can provide. Details on these layers can be found in the Bluetooth specifications.

The layers from L2CAP and below are what this standard discusses.

7. Physical layer (PHY)

Figure 7 indicates the relationship of the Bluetooth protocol stack to the PHY. The PHY is the first layer of the seven-layer OSI model and is responsible for transporting bits between adjacent systems over the air. The description of this layer, the radio portion, is limited to the following:

— Receiving a bit stream from the MAC subayer and transmitting the bitstream via radio waves to an associated station

— Receiving radio waves from an associated station and converting them to a bitstream that is passed to the MAC

This reflects the limited scope of the physical radio portion of the IEEE 802.15.1 architecture. Bits and radio waves are transmuted, but this layer does not do any interpretation.

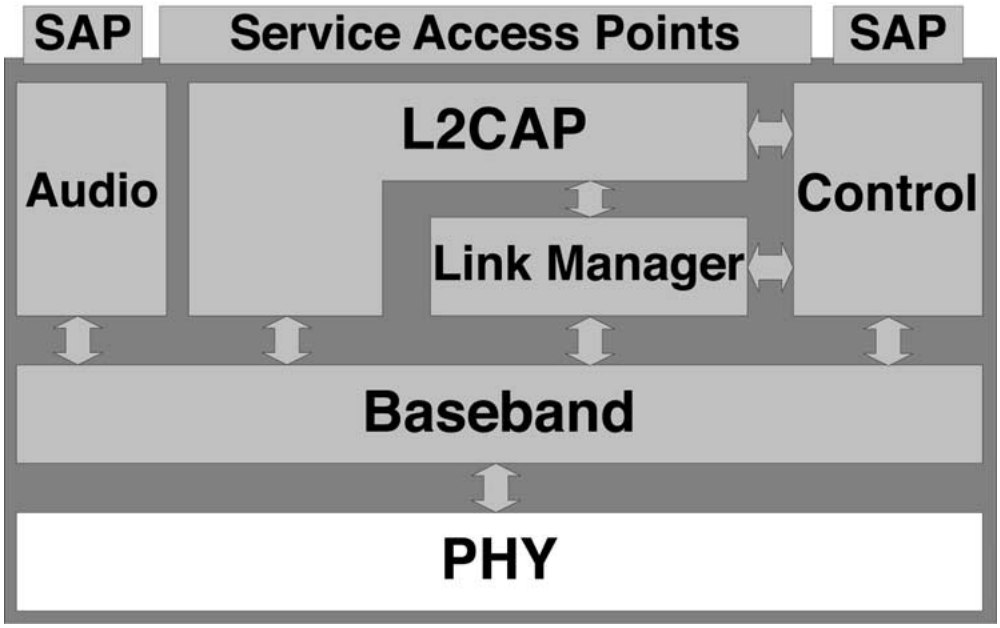


Figure 7—PHY interface relationships

7.1 Regulatory requirements

The Bluetooth transceiver operates in the 2.4 GHz ISM (Industrial, Scientific, Medical) band. This clause defines the requirements for a Bluetooth transceiver operating in this unlicensed band.

Requirements are defined for the following two reasons:

— To provide compatibility between the radios used in the system

— To define the quality of the system

The Bluetooth transceiver shall fulfill the stated requirements under the operating conditions specified in 7.5 and 7.6.

This clause is based on the established regulations for Europe, Japan, and North America. The regulatory documents listed below are for information only and are subject to change or revision at any time.

- **Europe** (except France and Spain)**:**
  - Approval standards: European Telecommunications Standards Institute (ETSI)
  - Documents: ETS 300-328, ETS 300-826 [B2]
  - Approval authority: National type approval authorities
- **France**:
  - Approval standards: La Reglementation en France pour les Equipements fonctionnant dans la bande de frequences 2.4 GHz “RLAN-Radio Local Area Network”
  - Documents: SP/DGPT/ATAS/23, ETS 300-328, ETS 300-826 [B11]
  - Approval authority: Direction Generale des Postes et Telecommunications
- **Spain**:
  - Approval standards: Supplemento Del Numero 164 Del Boletin Oficial Del Estado (Published 10 July 1991, Revised 25 June 1993)
  - Documents: ETS 300-328, ETS 300-826 [B14]
  - Approval authority: Cuadro Nacional de Atribucion de Frecuesias
- **Japan**:
  - Approval standards: Association of Radio Industries and Businesses (ARIB)
  - Documents: RCR STD-33A, ARIB STD-T66 [B1]
  - Approval authority: Ministry of Post and Telecommunications (MPT)
- **North America**:
  - Approval standards: Federal Communications Commission (FCC), United States
  - Documents: CFR 47, Part 15, Sections 15.205, 15.209, 15.247, 15.249 [B3]
  - Approval standards: Industry Canada, Canada
  - Document: GL36 [B5]
  - Approval Authority: FCC (United States), Industry Canada (Canada)

7.2 Frequency bands and channel arrangement

The Bluetooth system operates in the 2.4 GHz ISM band. In a majority of countries around the world, the range of this frequency band is 2400 MHz to 2483.5 MHz. Some countries, however, have national limitations in the frequency range. To comply with these national limitations, special frequency-hopping algorithms have been specified for these countries. It should be noted that products implementing the reduced frequency band *will not* work with products that implement the full band. The products implementing the reduced frequency band shall, therefore, be considered as local versions for a single market (see Table 2).

Channel spacing is 1 MHz. In order to comply with out-of-band regulations in each country, a guard band is used at the lower and upper band edge (see Table 3).

Table 2—Operating frequency bands

Geography	Regulatory range (GHz)	RF channels
United States, Europe, and most other countries	2.400–2.4835	$f = 2402 + k$ MHz, $k = 0, \ldots, 78$
France	2.4465–2.4835	$f = 2454 + k$ MHz, $k = 0, \ldots, 22$

Table 3—Guard bands

Geography	Lower guard band (MHz)	Upper guard band (MHz)
United States, Europe, and most other countries	2	3.5

7.3 Transmitter characteristics

The requirements stated in this clause are given as power levels at the antenna connector of the equipment. If the equipment does not have a connector, a reference antenna with 0 dBi gain is assumed.

As a result of difficulty in measurement accuracy in radiated measurements, it is preferred that systems with an integral antenna provide a temporary antenna connector during type approval.

If transmitting antennas of directional gain greater than 0 dBi are used, the transmission power shall be compensated according to the applicable paragraphs in ETSI ETS 300-328 or the FCC's CFR 47, Part 15.

The equipment is divided into three power classes, as shown in Table 4.

Table 4—Power classes

Power class	Maximum output power (P_{max})	Nominal output power	Minimum output power[a]	Power control
1	100 mW (20 dBm)	N/A	1 mW (0 dBm)	P_{min}< +4 dBm to P_{max} Optional: P_{min}[b] to P_{max}
2	2.5 mW (4 dBm)	1 mW (0 dBm)	0.25 mW (–6 dBm)	Optional: P_{min}[b] to P_{max}
3	1 mW (0 dBm)	N/A	N/A	Optional: P_{min}[b] to P_{max}

[a]Minimum output power at maximum power setting.
[b]The lower power limit P_{min} < –30dBm is suggested, but is not mandatory, and may be chosen according to application needs.

Power control is required for power class 1 equipment. The power control is used for limiting the transmitted power over 0 dBm. Power control capability under 0 dBm is optional and could be used for optimizing the power consumption and overall interference level. The power steps shall form a monotonic sequence with a maximum step size of 8 dB and a minimum step size of 2 dB. Class 1 equipment with a maximum transmit power of +20 dBm shall be able to control its transmit power down to 4 dBm or less.

Equipment with power control capability optimizes the output power in a link with LMP commands (see Clause 9). This is done by measuring receiver signal strength indication (RSSI) and reporting back if the power should be increased or decreased.

Power class 1 shall not be used for sending packets from one device to another if the receiving side of a connection does not support the necessary messaging for power control of the sending side (i.e., RSSI measurements and related messages). In this case, the transmitter should comply with the rules of a class 2 or class 3 transmitter. Additionally, if a class 1 device is paging or inquiring in close proximity to another device, the input power could be larger than the stated requirement (see 7.4.5). This can cause the listening device to fail to respond. A device performing page or inquiry should do so according to power class 2 or class 3.

7.3.1 Modulation characteristics

The modulation is Gaussian frequency shift keying (GFSK) with a bandwidth time (BT) = 0.5. The modulation index shall be between 0.28 and 0.35. A binary 1 is represented by a positive frequency deviation, and a binary 0 is represented by a negative frequency deviation. The symbol timing shall be better than ± 20 ppm (see Figure 8).

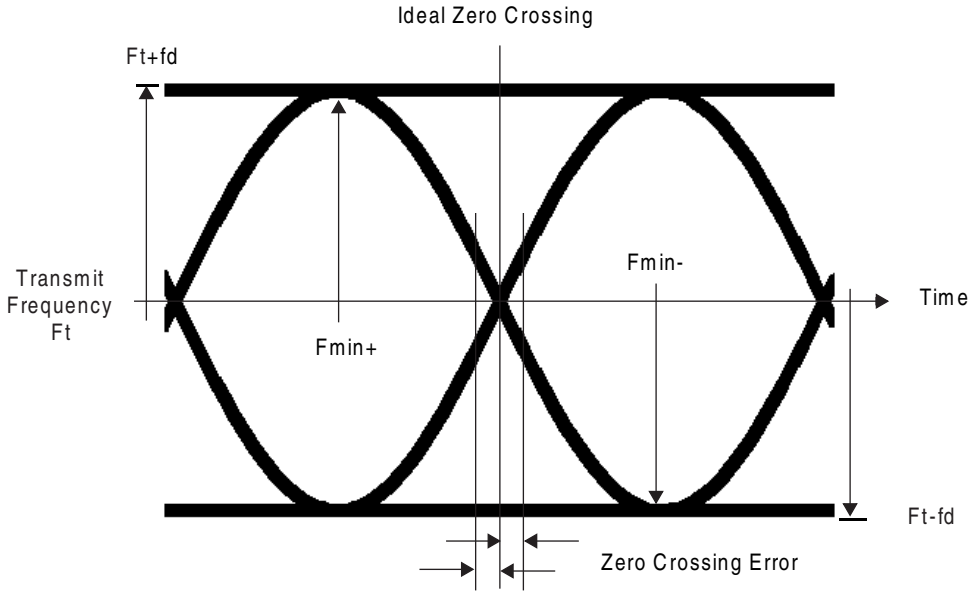


Figure 8—Actual transmit modulation

For each transmit channel, the minimum frequency deviation (F_{min} = the lesser of $\{F_{min+}, F_{min-}\}$) that corresponds to 1010 sequence shall be no smaller than ± 80% of the frequency deviation (*fd*) that corresponds to a 00001111 sequence. Additionally, the minimum deviation shall never be smaller than 115 kHz. Transmitted data have a symbol rate of 1 Msymbol/s.

The zero crossing error is the time difference between the ideal symbol period and the measured crossing time. This shall be less than ± 0.125 of a symbol period. The maximum frequency deviation shall be between 140 kHz and 175 kHz.

7.3.2 Spurious emissions

The spurious emission, in band and out of band, is measured with a frequency-hopping transmitter hopping on a single frequency. In other words, the synthesizer must change frequency between the receive slot and the transmit slot, but always returns to the same transmit frequency.

For the United States, FCC Parts 15.247, 15.249, 15.205, and 15.209 are applicable regulations. For Japan, RCR STD-33 and ARIB STD-T66 apply, and for Europe, ETS 300-328 and ETS 300-826 apply.

7.3.2.1 In-band spurious emission

Within the ISM band, the transmitter shall pass a spectrum mask, given in Table 5. The spectrum shall comply with the FCC's 20 dB bandwidth definition stated below and should be measured accordingly. In addition to the FCC requirement, an adjacent channel power is defined for channels with a difference in channel number of two or greater. This adjacent channel power is defined as the sum of the measured power in a 1 MHz channel. The transmitted power shall be measured in a 100 kHz bandwidth using maximum hold. The transmitter is transmitting on channel M, and the adjacent channel power is measured on channel number N. The transmitter sends a pseudo-random data pattern throughout the test.

Table 5—Transmit spectrum mask

Frequency offset	Transmit power
± 500 kHz	–20 dBc
$\lvert M-N \rvert = 2$	–20 dBm
$\lvert M-N \rvert \geq 3$	–40 dBm
NOTE—If the output power is less than 0 dBm, then, wherever appropriate, the FCC's 20 dB relative requirement overrules the absolute adjacent channel power requirement stated in this table.	

Exceptions are allowed in up to three bands of 1 MHz width centered on a frequency that is an integer multiple of 1 MHz. They shall, however, comply with an absolute value of –20 dBm.

7.3.2.2 Out-of-band spurious emission

The power should be measured in a 100 kHz bandwidth. The out-of-band emission shall conform to the requirements found in Table 6.

Table 6—Out-of-band spurious emission requirement

Frequency band (GHz)	Operation mode (dBm)	Idle mode (dBm)
0.030–1.000	–36	–57
1.000–12.750	–30	–47
1.800–1.900	–47	–47
5.150–5.300	–47	–47

7.3.3 RF tolerance

The transmitter initial center frequency (F_C) accuracy shall be ±75 kHz maximum from F_C. The initial frequency accuracy is defined as the frequency accuracy before any information is transmitted. Note that the frequency drift requirement is not included in the ±75 kHz.

The transmitter center frequency drift in a packet is specified in Table 7. The different packets are defined in 8.4.4.

Table 7—Frequency drift in a packet

Type of packet	Frequency drift
One-slot packet	±25 kHz
Three-slot packet	±40 kHz
Five-slot packet	±40 kHz
Maximum drift rate[a]	400 Hz/µs

[a]The maximum drift rate is allowed anywhere in a packet.

7.4 Receiver characteristics

To measure the bit error rate performance, the equipment shall have a loopback facility. The equipment sends back the decoded information. This facility is specified in the test mode specification (see Annex E).

The reference sensitivity level referred to in this clause equals –70 dBm.

7.4.1 Actual sensitivity level

The actual sensitivity level is defined as the input level for which a raw bit error rate (BER) of 0.1% is met. The requirement for a Bluetooth receiver is an actual sensitivity level of –70 dBm or better. The receiver shall achieve the –70 dBm sensitivity level with any Bluetooth transmitter compliant to the transmitter specification specified in 7.3.

7.4.2 Interference performance

The interference performance on co-channel and adjacent 1 MHz and 2 MHz is measured with the desired signal 10 dB over the reference sensitivity level. On all other frequencies, the desired signal shall be 3 dB over the reference sensitivity level. If the frequency of an interfering signal lies outside the band of 2400 MHz to 2497 MHz, the out-of-band blocking specification (see 7.4.3) shall apply. The interfering signal shall be Bluetooth-modulated (see 7.4.8). The BER shall be ≤ 0.1%. The signal-to-interference ratio shall be as shown in Table 8.

These specifications are to be tested only at nominal temperature conditions with a receiver hopping on one frequency. In other words, the synthesizer must change frequency between the receive slot and the transmit slot, but always return to the same receive frequency.

Frequencies where the requirements are not met are called spurious response frequencies. Five spurious response frequencies are allowed at frequencies with a distance ≥ 2 MHz from the wanted signal. On these spurious response frequencies, a relaxed interference requirement of C/I = –17 dB shall be met.

Table 8—Interference performance

Requirement	Ratio (dB)
Co-channel interference, $C/I_{co\text{-}channel}$	11[a]
Adjacent (1 MHz) interference, C/I_{1MHz}	0[a]
Adjacent (2 MHz) interference, C/I_{2MHz}	–30
Adjacent (≥3 MHz) interference, $C/I_{\geq 3MHz}$	–40
Image frequency interference,[b,c] C/I_{Image}	–9[a]
Adjacent (1 MHz) interference to in-band image frequency, $C/I_{Image\pm 1MHz}$	–20[a]
Note that if two adjacent channel specifications above are applicable to the same channel, the more relaxed specification applies.	

[a]During the period July 1999 to July 2002, Bluetooth devices need to achieve a co-channel interference resistance of +14 dB, an ACI (@1 MHz) resistance of +4 dB, image frequency interference resistance of –6 dB, and an adjacent interference to in-band image frequency resistance of –16 dB to meet Bluetooth qualification requirements.
[b]In-band image frequency.
[c]If the image frequency ≠ n*1 MHz, the image reference frequency is defined as the closest n*1 MHz frequency.

7.4.3 Out-of-band blocking

The out-of-band blocking is measured with the desired signal 3 dB over the reference sensitivity level. The interfering signal shall be a continuous wave signal. The BER shall be ≤ 0.1%. The out-of-band blocking shall fulfill the requirements in Table 9.

Table 9—Out-of-band blocking requirements

Interfering signal frequency (GHz)	Interfering signal power level (dBm)
0.030–2.000	–10
2.000–2.399	–27
2.498–3.000	–27
3.000–12.750	–10

Twenty-four exceptions are permitted that are dependent on the given receive channel frequency and are centered at a frequency that is an integer multiple of 1 MHz. At nineteen of these spurious response frequencies, a relaxed power level –50 dBm of the interferer may be used to achieve a BER of 0.1%. At the remaining five spurious response frequencies, the power level is arbitrary.

7.4.4 Intermodulation characteristics

The reference sensitivity performance of BER = 0.1% shall be met under the following conditions:

— The desired signal at frequency f_0 with a power level 6 dB over the reference sensitivity level

— A static sine wave signal at f_1 with a power level of –39 dBm

— A Bluetooth modulated signal (see 7.4.8) at f_2 with a power level of –39 dBm

such that $f_0 = 2f_1 - f_2$ and $|f_2 - f_1| = n*1$ MHz, where n can be 3, 4, or 5. The system shall fulfill one of the three alternatives.

7.4.5 Maximum usable level

The maximum usable input level the receiver shall operate at shall be better than –20 dBm. The BER shall be less than or equal to 0.1% at –20 dBm input power.

7.4.6 Spurious emissions

The spurious emission for a Bluetooth receiver shall not be more than the value shown in Table 10.

Table 10—Out-of-band spurious emission

Frequency band (GHz)	Requirement (dBm)
0.030–1.000	–57
1.000–12.750	–47

The power should be measured in a 100 kHz bandwidth.

7.4.7 Receiver signal strength indicator (optional)

A transceiver that wishes to take part in a power-controlled link shall be able to measure its own receiver signal strength and determine whether the transmitter on the other side of the link should increase or decrease its output power level. A receiver signal strength indicator (RSSI) makes this possible.

The RSSI measurement compares the received signal power with two threshold levels, which define the ideal receive power range. The lower threshold level corresponds to a received power between –56 dBm and 6 dB above the actual sensitivity of the receiver. The upper threshold level is 20 dB above the lower threshold level to an accuracy of ± 6 dB (see Figure 9).

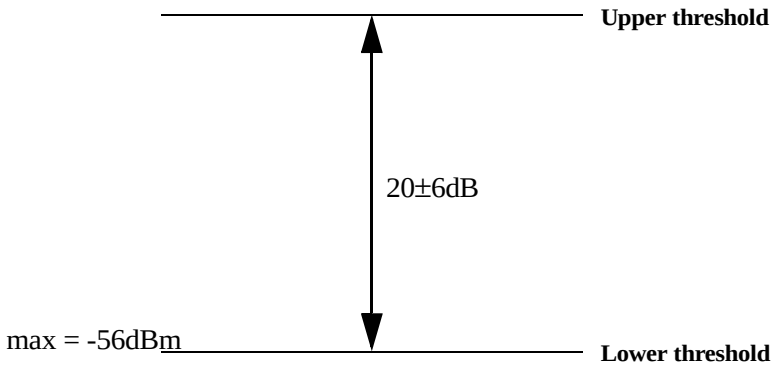


Figure 9—RSSI dynamic range and accuracy

7.4.8 Reference interference-signal definition

A Bluetooth modulated interfering signal is defined as follows:

- Modulation = GFSK
- Modulation index = 0.32 ± 1%
- Bandwidth time (BT) = 0.5 ± 1%
- Bit rate = 1 Mb/s ± 1 ppm
- Modulating data = PRBS-9 (see Annex E.2.1.2)
- Modulating data for interfering signal = PRBS-15 [see ITU-T Recommendation O.150 (05/96)]
- Frequency accuracy better than ± 1 ppm

7.5 Test conditions

7.5.1 Nominal test conditions (NTC)

7.5.1.1 Nominal temperature

The nominal temperature conditions for tests shall be +15 °C to +35 °C. When it is impractical to carry out the test under this condition, a note to this effect, stating the ambient temperature, shall be recorded.

Note that the actual value during the test shall be recorded in the test report.

7.5.1.2 Nominal power source

7.5.1.2.1 Mains voltage

The nominal test voltage for equipment to be connected to the mains shall be the nominal mains voltage. The nominal voltage shall be the declared voltage or any of the declared voltages for which the equipment was designed. The frequency of the test power source corresponding to the ac mains shall be within 2% of the nominal frequency.

7.5.1.2.2 Lead-acid battery power sources used in vehicles

When radio equipment is intended for operation from the alternator-fed lead-acid battery power sources that are standard in vehicles, the nominal test voltage shall be 1.1 times the nominal voltage of the battery (e.g., 6 V, 12 V, etc.).

7.5.1.2.3 Other power sources

For operation from other power sources or types of battery (primary or secondary), the nominal test voltage shall be as declared by the equipment manufacturer.

7.5.2 Extreme test conditions (ETC)

7.5.2.1 Extreme temperatures

The extreme temperature range is defined as the largest temperature range given by the combination of

— The minimum temperature range 0 °C to +35 °C
— The product operating temperature range declared by the manufacturer

7.5.2.2 Extreme power source voltages

Tests at the extreme power source voltages specified below (7.5.2.2.1 through 7.5.2.2.4) are not required when the equipment under test is designed for operation as part of and is powered by another system or piece of equipment. In such cases, the limit values of the host system or host equipment shall apply.

7.5.2.2.1 Mains voltage

The extreme test voltage for equipment to be connected to an ac mains source shall be the nominal mains voltage ± 10%.

7.5.2.2.2 Lead-acid battery power source used on vehicles

When radio equipment is intended for operation from the alternator-fed lead-acid battery power sources that are standard in vehicles, the extreme test voltage shall be 1.3 and 0.9 times the nominal voltage of the battery (e.g., 6 V, 12 V, etc.).

7.5.2.2.3 Power sources using other types of batteries

The lower extreme test voltage for equipment with power sources using the following types of batteries shall be:

— For Leclanché, alkaline, or lithium type batteries: 0.85 times the nominal voltage of the battery
— For mercury or nickel-cadmium type batteries: 0.9 times the nominal voltage of the battery

In both cases, the upper extreme test voltage shall be 1.15 times the nominal voltage of the battery.

7.5.2.2.4 Other power sources

For equipment using other power sources or capable of being operated from a variety of power sources (primary or secondary), the extreme test voltages shall be those declared by the manufacturer.

7.6 Radio parameters

The radio parameters shall be tested as shown in Table 11:

Table 11—Radio parameter test conditions

Parameter	Temperature	Power source
Output power	ETC[a]	ETC
Power control	NTC[b]	NTC
Modulation index	ETC	ETC
Initial carrier frequency accuracy	ETC	ETC
Carrier frequency drift	ETC	ETC
In-band spurious emissions	ETC	ETC
Out-of-band spurious emissions	ETC	ETC
Sensitivity	ETC	ETC
Interference performance	NTC	NTC
Intermodulation characteristics	NTC	NTC
Out-of-band blocking	NTC	NTC
Maximum usable level	NTC	NTC
Receiver signal strength indicator	NTC	NTC

[a]Extreme test conditions
[b]Nominal test conditions

8. Baseband specification

Figure 10 indicates the relationship of the Bluetooth protocol stack to this clause. This clause describes the specifications of the Bluetooth link controller which carries out the baseband protocols and other low-level link routines.

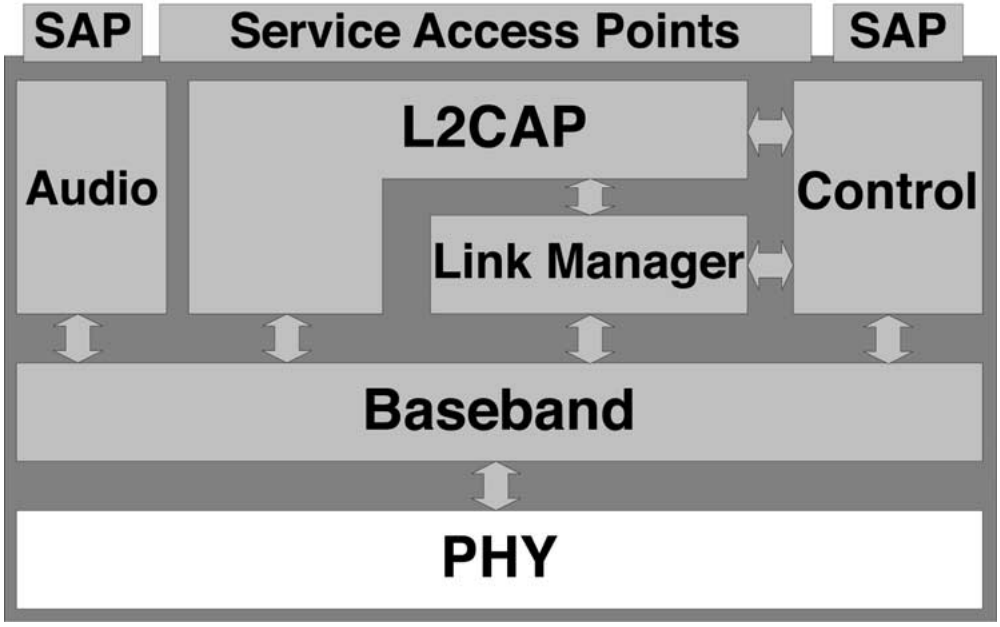


Figure 10—BB interface relationships

8.1 General description

Bluetooth is a short-range radio link intended to replace the cable(s) connecting portable and/or fixed electronic devices. Key features are robustness, low complexity, low power, and low cost.

Bluetooth operates in the unlicensed ISM band at 2.4 GHz. A frequency hop transceiver is applied to combat interference and fading. A shaped, binary FM modulation is applied to minimize transceiver complexity. The symbol rate is 1 Msymbol/s. A slotted channel is applied with a nominal slot length of 625 μs. To emulate full duplex transmission, a Time-Division Duplex (TDD) scheme is used. On the channel, information is exchanged through packets. Each packet is transmitted on a different hop frequency. A packet nominally covers a single slot, but can be extended to cover up to five slots.

The Bluetooth protocol uses a combination of circuit and packet switching. Slots can be reserved for synchronous packets. Bluetooth can support an asynchronous data channel, up to three simultaneous synchronous voice channels, or a channel which simultaneously supports asynchronous data and synchronous voice. Each voice channel supports a 64 kb/s synchronous (voice) channel in each direction. The asynchronous channel can support maximal 723.2 kb/s asymmetric (and still up to 57.6 kb/s in the return direction), or 433.9 kb/s symmetric.

The Bluetooth system consists of a radio unit (see Clause 7), a link control unit, and a support unit for link management and host terminal interface functions (see Figure 11). This clause describes the specifications of the Bluetooth link controller, which carries out the baseband protocols and other low-level link routines. Link layer messages for link set-up and control are defined in Clause 9.

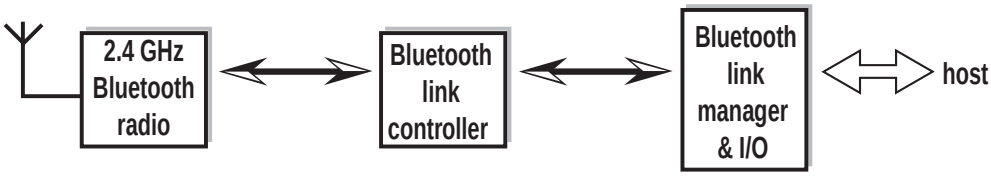


Figure 11—Different functional blocks in the Bluetooth system

The Bluetooth system provides a point-to-point connection (only two Bluetooth units involved), or a point-to-multipoint connection (see Figure 12). In the point-to-multipoint connection, the channel is shared among several Bluetooth units. Two or more units sharing the same channel form a *piconet*. One Bluetooth unit acts as the master of the piconet, whereas the other unit(s) acts as slave(s). Up to seven slaves can be active in the piconet. In addition, many more slaves can remain locked to the master in a so-called parked state. These parked slaves cannot be active on the channel, but remain synchronized to the master. Both for active and parked slaves, the channel access is controlled by the master.

Multiple piconets with overlapping coverage areas form a *scatternet*. Each piconet can only have a single master. However, slaves can participate in different piconets on a time-division multiplex basis. In addition, a master in one piconet can be a slave in another piconet. The piconets shall not be frequency-synchronized. Each piconet has its own hopping channel.

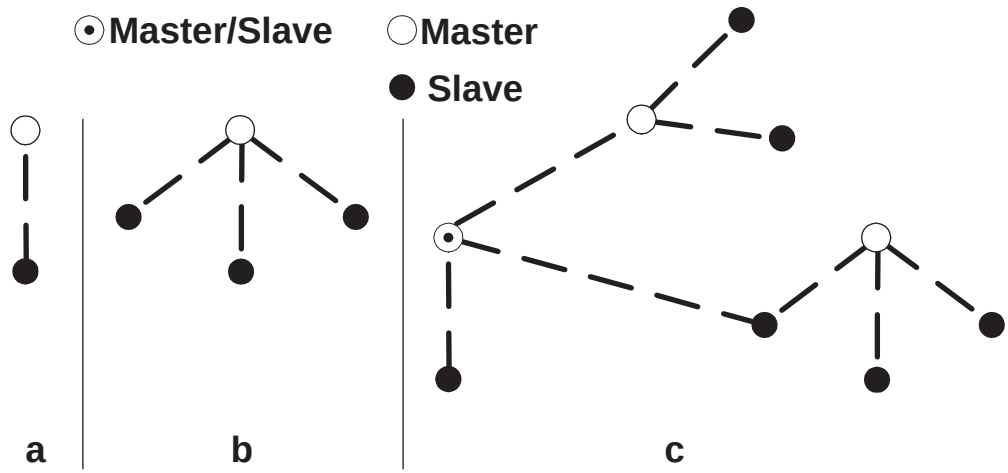


Figure 12—Various piconet formations: (a) single slave operation; (b) multislave operation; and (c) scatternet operation (Master with dot is Master/Slave)

8.2 Physical channel

8.2.1 Channel definition

The channel is represented by a pseudo-random hopping sequence hopping through the 79 or 23 RF channels. The hopping sequence is unique for the piconet and is determined by the Bluetooth device address of the master. The phase in the hopping sequence is determined by the Bluetooth clock of the master. The channel is divided into time slots where each slot corresponds to an RF hop frequency. Consecutive hops

correspond to different RF hop frequencies. The nominal hop rate is 1600 hops/s. All Bluetooth units participating in the piconet are time- and hop-synchronized to the channel.

8.2.2 Time slots

The channel is divided into time slots, each 625 µs in length. The time slots are numbered according to the Bluetooth clock of the piconet master. The slot numbering ranges from 0 to 2^{27}-1 and is cyclic with a cycle length of 2^{27}.

A TDD scheme is used where master and slave alternatively transmit (see Figure 13). The master shall start its transmission in even-numbered time slots only, and the slave shall start its transmission in odd-numbered time slots only. The packet start shall be aligned with the slot start. Packets transmitted by the master or the slave may extend over up to five time slots.

The RF hop frequency shall remain fixed for the duration of the packet. For a single packet, the RF hop frequency to be used is derived from the current Bluetooth clock value. For a multislot packet, the RF hop frequency to be used for the entire packet is derived from the Bluetooth clock value in the first slot of the packet. The RF hop frequency in the first slot after a multislot packet shall use the frequency as determined by the current Bluetooth clock value. Figure 14 illustrates the hop definition on single-slot and multislot packets. If a packet occupies more than one time slot, the hop frequency applied shall be the hop frequency as applied in the time slot where the packet transmission was started.

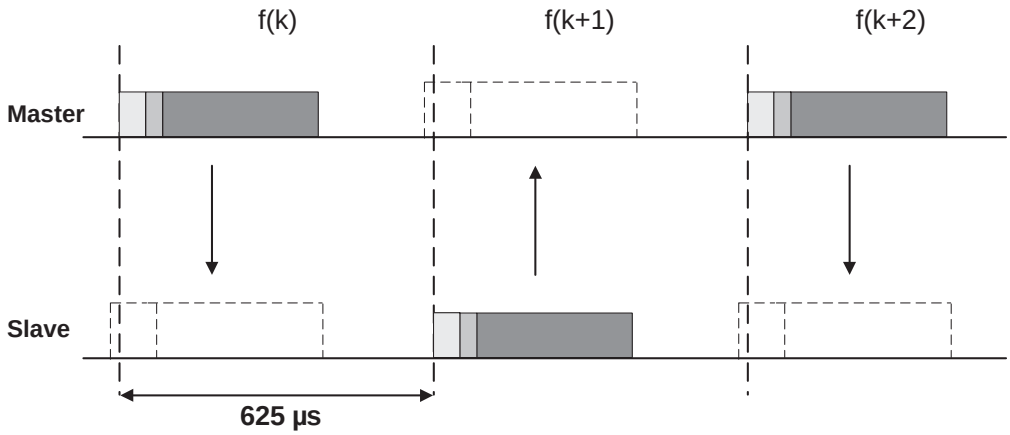


Figure 13—TDD and timing

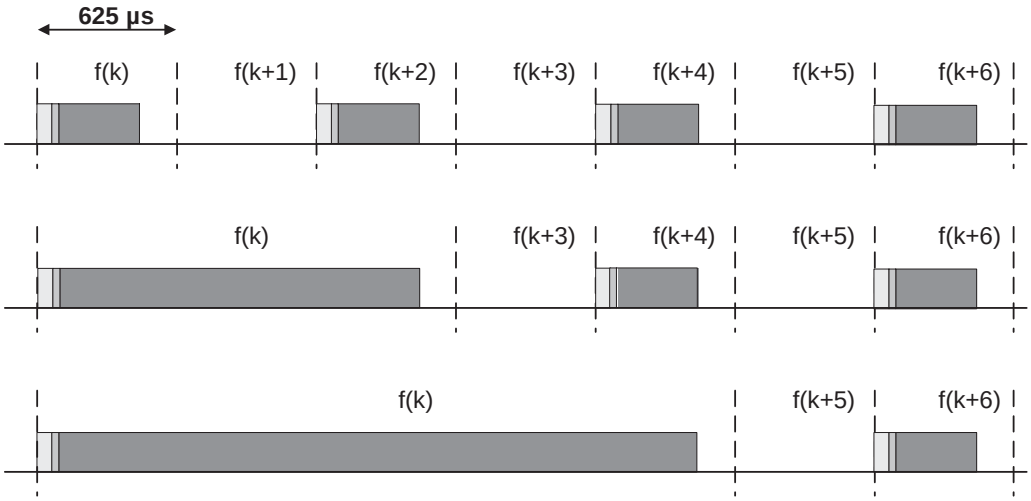


Figure 14—Multislot packets

8.3 Physical links

8.3.1 General

Between master and slave(s), different types of links can be established. Two link types have been defined:

— Synchronous Connection-Oriented (SCO) link

— Asynchronous Connection-Less (ACL) link

The SCO link is a point-to-point link between a master and a single slave in the piconet. The master maintains the SCO link by using reserved slots at regular intervals. The ACL link is a point-to-multipoint link between the master and all the slaves participating on the piconet. In the slots not reserved for the SCO link(s), the master can establish an ACL link on a per-slot basis to any slave, including the slave(s) already engaged in an SCO link.

8.3.2 SCO link

The SCO link is a symmetric, point-to-point link between the master and a specific slave. The SCO link reserves slots and can therefore be considered as a circuit-switched connection between the master and the slave. The SCO link typically supports time-bounded information like voice. The master can support up to three SCO links to the same slave or to different slaves. A slave can support up to three SCO links from the same master, or two SCO links if the links originate from different masters. SCO packets are never retransmitted.

The master will send SCO packets at regular intervals, the so-called SCO interval T_{SCO} (counted in slots) to the slave in the reserved master-to-slave slots. The SCO slave is always allowed to respond with an SCO packet in the following slave-to-master slot unless a different slave was addressed in the previous master-to-slave slot. If the SCO slave fails to decode the slave address in the packet header, it is still allowed to return an SCO packet in the reserved SCO slot.

The SCO link is established by the master sending an SCO setup message via the LM protocol. This message will contain timing parameters such as the SCO interval T_{SCO} and the offset D_{SCO} to specify the reserved slots.

In order to prevent clock wrap-around problems, an initialization flag in the LMP setup message indicates whether initialization procedure 1 or 2 is being used. The slave shall apply the initialization method as indicated by the initialization flag. The master uses initialization 1 when the MSB of the current master clock (CLK_{27}) is 0; it uses initialization 2 when the MSB of the current master clock (CLK_{27}) is 1. The master-to-slave SCO slots reserved by the master and the slave shall be initialized on the slots for which the clock satisfies the following equation:

$CLK_{27-1} \bmod T_{SCO} = D_{SCO}$ for initialization 1

$(\overline{CLK}_{27}, CLK_{26-1}) \bmod T_{SCO} = D_{SCO}$ for initialization 2

The slave-to-master SCO slots shall directly follow the reserved master-to-slave SCO slots. After initialization, the clock value CLK(k + 1) for the next master-to-slave SCO slot is found by adding the fixed interval T_{SCO} to the clock value of the current master-to-slave SCO slot:

$CLK(k + 1) = CLK(k) + T_{SCO}$

8.3.3 ACL link

In the slots not reserved for SCO links, the master can exchange packets with any slave on a per-slot basis. The ACL link provides a packet-switched connection between the master and all active slaves participating in the piconet. Both asynchronous and isochronous services are supported. Between a master and a slave only a single ACL link can exist. For most ACL packets, packet retransmission is applied to assure data integrity.

A slave is permitted to return an ACL packet in the slave-to-master slot if and only if it has been addressed in the preceding master-to-slave slot. If the slave fails to decode the slave address in the packet header, it is not allowed to transmit.

ACL packets not addressed to a specific slave are considered as broadcast packets and are read by every slave. If there is no data to be sent on the ACL link and no polling is required, no transmission shall take place.

8.4 Packets

8.4.1 General format

The bit ordering when defining packets and messages in this clause, follows the Little Endian format, i.e., the following rules apply:

— The least significant bit (LSB) corresponds to b_0.

— The LSB is the first bit sent over the air.

— In illustrations, the LSB is shown on the left side.

The link controller interprets the first bit arriving from a higher software layer as b_0; i.e. this is the first bit to be sent over the air. Furthermore, data fields generated internally at baseband level, such as the packet header fields and payload header length, are transmitted with the LSB first. For instance, a 3-bit parameter X = 3 is sent as $b_0 b_1 b_2 = 110$ over the air where 1 is sent first and 0 is sent last.

The data on the piconet channel is conveyed in packets. The general packet format is shown in Figure 15. Each packet consists of three entities: the access code, the header, and the payload. In the figure, the number of bits per entity is indicated.

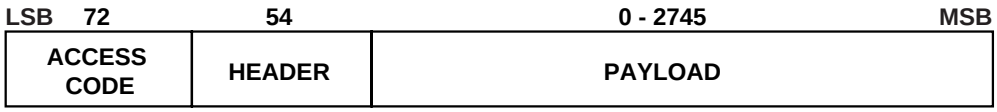


Figure 15—Standard packet format

The access code and header are of fixed size: 72 bits and 54 bits, respectively. The payload can range from zero to a maximum of 2745 bits. Different packet types have been defined. Packets may consist of the (shortened) access code only (see 8.4.4.1.1), of the access code - header, or of the access code - header - payload.

8.4.2 Access code

Each packet starts with an access code. If a packet header follows, the access code is 72 bits long, otherwise the access code is 68 bits long. This access code is used for synchronization, DC offset compensation and identification. The access code identifies all packets exchanged on the channel of the piconet: all packets sent in the same piconet are preceded by the same channel access code. In the receiver of the Bluetooth unit, a sliding correlator correlates against the access code and triggers when a threshold is exceeded. This trigger signal is used to determine the receive timing.

The access code is also used in paging and inquiry procedures. In this case, the access code itself is used as a signalling message and neither a header nor a payload is present.

The access code consists of a preamble, a sync word, and possibly a trailer; see Figure 16. For details see 8.4.2.1.

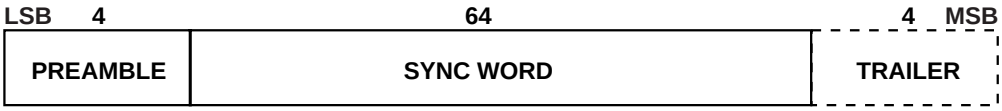


Figure 16—Access code format

8.4.2.1 Access code types

There are three different types of access codes defined:

— Channel access code (CAC)

— Device access code (DAC)

— Inquiry access code (IAC)

The respective access code types are used for a Bluetooth unit in different operating modes. The channel access code identifies a piconet. This code is included in all packets exchanged on the piconet channel. The device access code is used for special signalling procedures, e.g., paging and response to paging. For the inquiry access code there are two variations. A general inquiry access code (GIAC) is common to all devices. The GIAC can be used to discover which other Bluetooth units are in range. The dedicated inquiry

access code (DIAC) is common for a dedicated group of Bluetooth units that share a common characteristic. The DIAC can be used to discover only these dedicated Bluetooth units in range.

The CAC consists of a preamble, sync word, and trailer and its total length is 72 bits. When used as self-contained messages without a header, the DAC and IAC do not include the trailer bits and are of length 68 bits.

The different access code types use different Lower Address Parts (LAPs) to construct the sync word. The LAP field of the Bluetooth device address is explained in 8.13.1. A summary of the different access code types can be found in Table 12.

Table 12—Summary of access code types

Code type	LAP	Code length	Comments
CAC	Master	72	See also 8.13.2
DAC	Paged unit	68/72[a]	
GIAC	Reserved	68/72*	
DIAC	Dedicated	68/72*	

[a]length 72 is only used in combination with FHS packets

8.4.2.2 Preamble

The preamble is a fixed zero-one pattern of four symbols used to facilitate dc compensation. The sequence is either 1010 or 0101, depending whether the LSB of the following sync word is 1 or 0, respectively. The preamble is shown in Figure 17.

Figure 17—Preamble

8.4.2.3 Sync Word

The sync word is a 64-bit code word derived from a 24 bit address (LAP); for the CAC the master's LAP is used; for the GIAC and the DIAC, reserved, dedicated LAPs are used; for the DAC, the slave unit LAP is used. The construction guarantees large Hamming distance between sync words based on different LAPs. In addition, the good auto correlation properties of the sync word improve on the timing synchronization process. The derivation of the sync word is described in 8.13.2.

8.4.2.4 Trailer

The trailer is appended to the sync word as soon as the packet header follows the access code. This is typically the case with the CAC, but the trailer is also used in the DAC and IAC when these codes are used in FHS packets exchanged during page response and inquiry response procedures.

The trailer is a fixed zero-one pattern of four symbols. The trailer together with the three MSBs of the syncword form a 7-bit pattern of alternating ones and zeroes which may be used for extended dc compensation. The trailer sequence is either 1010 or 0101 depending on whether the MSB of the sync word is 0 or 1, respectively. The choice of trailer is illustrated in Figure 18.

Figure 18—Trailer in CAC when MSB of sync word is 0 (a), and when MSB of sync word is 1 (b)

8.4.3 Packet header

The header contains link control (LC) information and consists of six fields:

— AM_ADDR: 3-bit active member address
— TYPE: 4-bit type code
— FLOW: 1-bit flow control
— ARQN: 1-bit acknowledge indication
— SEQN: 1-bit sequence number
— HEC: 8-bit header error check

The total header, including the HEC, consists of 18 bits, see Figure 19, and is encoded with a rate 1/3 FEC (not shown but described in 8.5.1) resulting in a 54-bit header. Note that the AM_ADDR and TYPE fields are sent with their LSB first. The function of the different fields will be explained in 8.4.3.1 through 8.4.3.6.

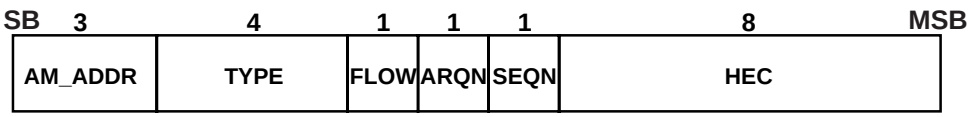


Figure 19—Header format

8.4.3.1 AM_ADDR

The AM_ADDR represents a member address and is used to distinguish between the active members participating on the piconet. In a piconet, one or more slaves are connected to a single master. To identify each slave separately, each slave is assigned a temporary 3-bit address to be used when it is active. Packets exchanged between the master and the slave all carry the AM_ADDR of this slave; that is, the AM_ADDR

of the slave is used in both master-to-slave packets and in the slave-to-master packets. The all-zero address is reserved for broadcasting packets from the master to the slaves. An exception is the FHS packet which may use the all-zero member address but is *not* a broadcast message (see 8.4.4.1.4). Slaves that are disconnected or parked give up their AM_ADDR. A new AM_ADDR has to be assigned when they re-enter the piconet.

8.4.3.2 TYPE

Sixteen different types of packets can be distinguished. The 4-bit TYPE code specifies which packet type is used. Important to note is that the interpretation of the TYPE code depends on the physical link type associated with the packet. First, it shall be determined whether the packet is sent on an SCO link or an ACL link. Then it can be determined which type of SCO packet or ACL packet has been received. The TYPE code also reveals how many slots the current packet will occupy. This allows the non-addressed receivers to refrain from listening to the channel for the duration of the remaining slots. In 8.4.4, each packet type will be described in more detail.

8.4.3.3 FLOW

This bit is used for flow control of packets over the ACL link. When the RX buffer for the ACL link in the recipient is full and is not emptied, a STOP indication (FLOW = 0) is returned to stop the transmission of data temporarily. Note, that the STOP signal only concerns ACL packets. Packets including only link control information (ID, POLL and NULL packets) or SCO packets can still be received. When the RX buffer is empty, a GO indication (FLOW = 1) is returned. When no packet is received, or the received header is in error, a GO is assumed implicitly. In this case, the slave can receive a new packet with CRC although its RX buffer is still not emptied. The slave shall then return a negative acknowledgment (NAK) in response to this packet even if the packet passed the CRC check.

8.4.3.4 ARQN

The 1-bit acknowledgment indication ARQN is used to inform the source of a successful transfer of payload data with CRC, and can be positive acknowledge ACK or negative acknowledge NAK. If the reception was successful, an ACK (ARQN = 1) is returned, otherwise a NAK (ARQN = 0) is returned. When no return message regarding acknowledge is received, a NAK is assumed implicitly. NAK is also the default return information.

The ARQN is piggy-backed in the header of the return packet. The success of the reception is checked by means of a cyclic redundancy check (CRC) code. An unnumbered ARQ scheme, which means that the ARQN relates to the latest received packet from the same source, is used. See 8.5.3 for initialization and usage of this bit.

8.4.3.5 SEQN

The SEQN bit provides a sequential numbering scheme to order the data packet stream. For each new transmitted packet that contains data with CRC, the SEQN bit is inverted. This is required to filter out retransmissions at the destination; if a retransmission occurs due to a failing ACK, the destination receives the same packet twice. By comparing the SEQN of consecutive packets, correctly received retransmissions can be discarded. See 8.5.3.2 for initialization and usage of the SEQN bit. For broadcast packets, a modified sequencing method is used, see 8.5.3.5.

8.4.3.6 HEC

Each header has a header-error-check (HEC) to check the header integrity. The HEC consists of an 8-bit word generated by the polynomial `647` (octal representation). Before generating the HEC, the HEC generator is initialized with an 8-bit value. For FHS packets sent in master page response state, the slave

upper address part (UAP) is used. For FHS packets sent in inquiry response, the default check initialization (DCI, see 8.5.4) is used. In all other cases, the UAP of the master device is used. For the definition of Bluetooth device addresses, see 8.13.1.

After the initialization, a HEC is calculated for the 10 header bits. Before checking the HEC, the receiver shall initialize the HEC check circuitry with the proper 8-bit UAP (or DCI). If the HEC does not check, the entire packet is disregarded. More information can be found in 8.5.4.

8.4.4 Packet types

The packets used on the piconet are related to the physical links they are used in. This standard has defined two physical links: the SCO link and the ACL link. For each of these links, 12 different packet types can be defined. Four control packets will be common to all link types: their TYPE code is unique irrespective of the link type.

To indicate the different packets on a link, the 4-bit TYPE code is used. The packet types have been divided into four segments. The first segment is reserved for the four control packets common to all physical link types; all four packet types have been defined. The second segment is reserved for packets occupying a single time slot; six packet types have been defined. The third segment is reserved for packets occupying three time slots; two packet types have been defined. The fourth segment is reserved for packets occupying five time slots; two packet types have been defined. The slot occupancy is reflected in the segmentation and can directly be derived from the type code. Table 13 summarizes the packets defined for the SCO and ACL link types.

Table 13—Packets defined for SCO and ACL link types

Segment	TYPE code $b_3b_2b_1b_0$	Slot occupancy	SCO link	ACL link
1	0000	1	NULL	NULL
	0001	1	POLL	POLL
	0010	1	FHS	FHS
	0011	1	DM1	DM1
2	0100	1	undefined	DH1
	0101	1	HV1	undefined
	0110	1	HV2	undefined
	0111	1	HV3	undefined
	1000	1	DV	undefined
	1001	1	undefined	AUX1
3	1010	3	undefined	DM3
	1011	3	undefined	DH3
	1100	3	undefined	undefined
	1101	3	undefined	undefined
4	1110	5	undefined	DM5
	1111	5	undefined	DH5

8.4.4.1 Common packet types

There are five packet types defined, which are independent of the underlying physical link type. Additionally, the ID packet type is defined which is used in the paging and inquiry procedures, described in 8.10.6 and 8.10.7, respectively. These packet types are defined as follows.

8.4.4.1.1 ID packet

The identity or ID packet consists of the device access code (DAC) or inquiry access code (IAC). It has a fixed length of 68 bits. It is a very robust packet since the receiver uses a bit correlator to match the received packet to the known bit sequence of the ID packet. The packet is used, for example, in paging, inquiry, and response routines.

8.4.4.1.2 NULL packet

The NULL packet has no payload and therefore consists of the channel access code and packet header only. Its total (fixed) length is 126 bits. The NULL packet is used to return link information to the source regarding the success of the previous transmission (ARQN), or the status of the RX buffer (FLOW). The NULL packet itself does not have to be acknowledged.

8.4.4.1.3 POLL packet

The POLL packet is very similar to the NULL packet. It does not have a payload either. In contrast to the NULL packet, it requires a confirmation from the recipient. It is not a part of the ARQ scheme. The POLL packet does not affect the ARQN and SEQN fields. Upon reception of a POLL packet the slave must respond with a packet. This return packet is an implicit acknowledgement of the POLL packet. This packet can be used by the master in a piconet to poll the slaves, which must then respond even if they do not have information to send.

8.4.4.1.4 FHS packet

The FHS packet is a special control packet revealing, among other things, the Bluetooth device address and the clock of the sender. The payload contains 144 information bits plus a 16-bit CRC code. The payload is coded with a rate 2/3 FEC, which brings the gross payload length to 240 bits. The FHS packet covers a single time slot.

Figure 20 illustrates the format and contents of the FHS payload. The payload consists of eleven fields (see Table 14). The FHS packet is used in page master response, inquiry response and in master slave switch. In page master response or master slave switch, it is retransmitted until its reception is acknowledged or a timeout has exceeded. In inquiry response, the FHS packet is not acknowledged. The FHS packet contains real-time clock information. Subsequent transmissions of the FHS packet contain updated clock information. Each transmission always contains the most recent native clock data from the sender. The FHS packet is used for frequency hop synchronization before the piconet channel has been established, or when an existing piconet changes to a new piconet. In the former case, the recipient has not been assigned an active member address yet, in which case the AM_ADDR field in the FHS packet header is set to all-zeroes; however, the FHS packet should not be considered as a broadcast packet. In the latter case the slave already has an AM_ADDR in the existing piconet, which is then used in the FHS packet header.

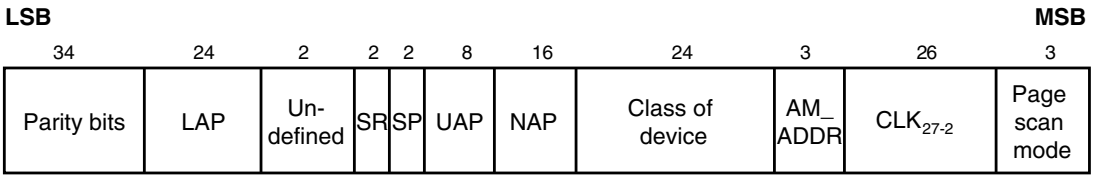


Figure 20—Format of the FHS payload

Each field is described in more detail below:

Table 14—Description of the FHS payload

Parity bits	This 34-bit field contains the parity bits that form the first part of the sync word of the access code of the unit that sends the FHS packet. These bits are derived from the LAP as described in 8.13.2.
LAP	This 24-bit field contains the lower address part of the unit that sends the FHS packet.
Reserved	This 2-bit field is reserved for future use and shall be set to zero.
SR	This 2-bit field is the scan repetition field and indicates the interval between two consecutive page scan windows, see also Table 15 and Table 23.
SP	This 2-bit field is the scan period field and indicates the period in which the mandatory page scan mode is applied after transmission of an inquiry response message, see also Table 16 and Table 28.
UAP	This 8-bit field contains the upper address part of the unit that sends the FHS packet.
NAP	This 16-bit field contains the non-significant address part of the unit that sends the FHS packet (see also 8.13.1 for LAP, UAP, and NAP).
Class of device	This 24-bit field contains the class of device of the unit that sends the FHS packet. The field is defined in 2.4.3.
AM_ADDR	This 3-bit field contains the member address the recipient shall use if the FHS packet is used at call setup or master-slave switch. A slave responding to a master or a unit responding to an inquiry request message shall include an all-zero AM_ADDR field if it sends the FHS packet.
CLK_{27-2}	This 26-bit field contains the value of the native system clock of the unit that sends the FHS packet, sampled at the beginning of the transmission of the access code of this FHS packet. This clock value has a resolution of 1.25ms (two-slot interval). For each new transmission, this field is updated so that it accurately reflects the real-time clock value.
Page scan mode	This 3-bit field indicates which scan mode is used by default by the sender of the FHS packet. The interpretation of the page scan mode is illustrated in Table 17. Currently, the standard supports one mandatory scan mode and up to three optional scan modes (see also Annex D).

Table 15—Contents of SR field

SR bit format b_1b_0	SR mode
00	R0
01	R1
10	R2
11	reserved

Table 16—Contents of SP field

SP bit format b_1b_0	SP mode
00	P0
01	P1
10	P2
11	reserved

Table 17—Contents of page scan mode field

Bit format $b_2b_1b_0$	Page scan mode
000	Mandatory scan mode
001	Optional scan mode I
010	Optional scan mode II
011	Optional scan mode III
100	Reserved for future use
101	Reserved for future use
110	Reserved for future use
111	Reserved for future use

The LAP, UAP, and NAP together form the 48-bit IEEE address of the unit that sends the FHS packet. Using the parity bits and the LAP, the recipient can directly construct the channel access code of the sender of the FHS packet.

8.4.4.1.5 DM1 packet

DM1 serves as part of segment 1 in order to support control messages in any link type. However, it can also carry regular user data. Since the DM1 packet is recognized on the SCO link, it can interrupt the synchronous information to send control information. Since the DM1 packet can be regarded as an ACL packet, it will be discussed in 8.4.4.3.

8.4.4.2 SCO packets

SCO packets are used on the synchronous SCO link. The packets do not include a CRC and are never retransmitted. SCO packets are routed to the synchronous I/O (voice) port. This standard has defined three pure SCO packets. In addition, an SCO packet is defined which carries an asynchronous data field in addition to a synchronous (voice) field. The SCO packets defined so far are typically used for 64 kb/s speech transmission.

8.4.4.2.1 HV1 packet

The HV1 packet carries 10 information bytes. The bytes are protected with a rate 1/3 FEC. No CRC is present. The payload length is fixed at 240 bits. There is no payload header present.

HV packets are used for transmission of voice and transparent synchronous data (see also 9.3.21.1). HV stands for High-quality Voice. The voice packets are never retransmitted and need no CRC. An HV1 packet carries 1.25 ms of speech at a 64 kb/s rate. An HV1 packet has to be sent every two time slots ($T_{SCO} = 2$).

8.4.4.2.2 HV2 packet

The HV2 packet carries 20 information bytes. The bytes are protected with a rate 2/3 FEC. No CRC is present. The payload length is fixed at 240 bits. There is no payload header present.

An HV2 packet carries 2.5 ms of speech at a 64 kb/s rate. An HV2 packet has to be sent every four time slots ($T_{SCO} = 4$).

8.4.4.2.3 HV3 packet

The HV3 packet carries 30 information bytes. The bytes are not protected by FEC. No CRC is present. The payload length is fixed at 240 bits. There is no payload header present.

An HV3 packet carries 3.75ms of speech at a 64 kb/s rate. An HV3 packet has to be sent every six time slots ($T_{SCO} = 6$).

8.4.4.2.4 DV packet

The DV packet is a combined data - voice packet. The payload is divided into a voice field of 80 bits and a data field containing up to 150 bits; see Figure 21. The voice field is not protected by FEC. The data field contains up to 10 information bytes (including the 1-byte payload header) and includes a 16-bit CRC. The data field is encoded with a rate 2/3 FEC. If necessary, extra zeroes are appended to assure that the total number of payload bits is a multiple of 10 prior to FEC encoding. Since the DV packet has to be sent at regular intervals due to its synchronous (voice) contents, it is listed under the SCO packet types. The voice and data fields are treated separately. The voice field is handled like normal SCO data and is never retransmitted; that is, the voice field is always new. The data field is checked for errors and is retransmitted if necessary.

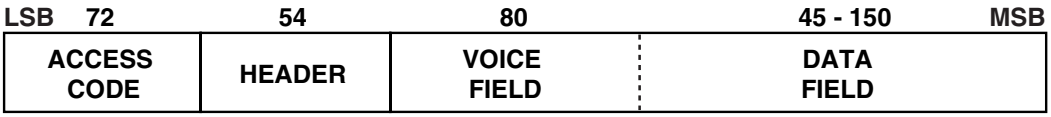


Figure 21—DV packet format

8.4.4.3 ACL packets

ACL packets are used on the asynchronous links. The information carried can be user data or control data. Including the DM1 packet, seven ACL packets have been defined. Six of the ACL packets contain a CRC code and retransmission is applied if no acknowledgement of proper reception is received (except in case a flush operation is carried out, see 8.5.3.3). The 7th ACL packet, the AUX1 packet, has no CRC and is not retransmitted.

8.4.4.3.1 DM1 packet

The DM1 packet is a packet that carries data information only. DM stands for Data – Medium rate. The payload contains up to 18 information bytes (including the 1-byte payload header) plus a 16-bit CRC code. The DM1 packet may cover up to a single time slot. The information plus CRC bits are coded with a rate 2/3 FEC which adds 5 parity bits to every 10-bit segment. If necessary, extra zeros are appended after the CRC bits to get the total number of bits (information bits, CRC bits, and tail bits) equal a multiple of 10. The payload header in the DM1 packet is only 1 byte long; see Figure 22. The length indicator in the payload header specifies the number of user bytes (excluding payload header and the CRC code).

8.4.4.3.2 DH1 packet

This packet is similar to the DM1 packet, except that the information in the payload is not FEC encoded. As a result, the DH1 frame can carry up to 28 information bytes (including the 1 byte payload header) plus a 16-bit CRC code. DH stands for Data – High rate. The DH1 packet may cover up to a single time slot.

8.4.4.3.3 DM3 packet

The DM3 packet is a DM1 packet with an extended payload. The DM3 packet may cover up to three time slots. The payload contains up to 123 information bytes (including the 2-byte payload header) plus a 16-bit CRC code. The payload header in the DM3 packet is 2 bytes long, see Figure 23. The length indicator in the payload header specifies the number of user bytes (excluding payload header and the CRC code). When a DM3 packet is sent or received, the radio frequency (RF) hop frequency shall not change for a duration of three time slots (the first time slot being the slot where the channel access code was transmitted).

8.4.4.3.4 DH3 packet

This packet is similar to the DM3 packet, except that the information in the payload is not FEC encoded. As a result, the DH3 packet can carry up to 185 information bytes (including the 2-byte payload header) plus a 16-bit CRC code. The DH3 packet may cover three time slots. When a DH3 packet is sent or received, the hop frequency shall not change for a duration of three time slots (the first time slot being the slot where the channel access code was transmitted).

8.4.4.3.5 DM5 packet

The DM5 packet is a DM1 packet with an extended payload. The DM5 packet may cover up to five time slots. The payload contains up to 226 information bytes (including the 2-byte payload header) plus a 16-bit

CRC code. The payload header in the DM5 packet is 2 bytes long. The length indicator in the payload header specifies the number of user bytes (excluding payload header and the CRC code). When a DM5 packet is sent or received, the hop frequency shall not change for a duration of five time slots (the first time slot being the slot where the channel access code was transmitted).

8.4.4.3.6 DH5 packet

This packet is similar to the DM5 packet, except that the information in the payload is not FEC encoded. As a result, the DH5 packet can carry up to 341 information bytes (including the two bytes payload header) plus a 16-bit CRC code. The DH5 packet may cover five time slots. When a DH5 packet is sent or received, the hop frequency shall not change for a duration of five time slots (the first time slot being the slot where the channel access code was transmitted).

8.4.4.3.7 AUX1 packet

This packet resembles a DH1 packet but has no CRC code. The AUX1 packet can carry up to 30 information bytes (including the 1-byte payload header). The AUX1 packet may cover up to a single time slot.

8.4.5 Payload Format

In the previous packet overview, several payload formats were considered. In the payload, two fields are distinguished: the (synchronous) voice field and the (asynchronous) data field. The ACL packets only have the data field and the SCO packets only have the voice field – with the exception of the DV packets, which have both.

8.4.5.1 Voice field

The voice field has a fixed length. For the HV packets, the voice field length is 240 bits; for the DV packet the voice field length is 80 bits. No payload header is present.

8.4.5.2 Data field

The data field consists of three segments: a payload header, a payload body, and possibly a CRC code (only the AUX1 packet does not carry a CRC code).

1) **Payload header**:

   Only data fields have a payload header. The payload header is 1 or 2 bytes long. Packets in segments one and two have a 1-byte payload header; packets in segments three and four have a 2-bytes payload header. The payload header specifies the logical channel (2-bit L_CH indication), controls the flow on the logical channels (1-bit FLOW indication), and has a payload length indicator (5 bits and 9 bits for 1-byte and 2-bytes payload header, respectively). In the case of a 2-bytes payload header, the length indicator is extended by four bits into the next byte. The remaining four bits of the second byte are reserved for future use and shall be set to zero. The formats of the 1-byte and 2-bytes payload headers are shown in Figure 22 and Figure 23.

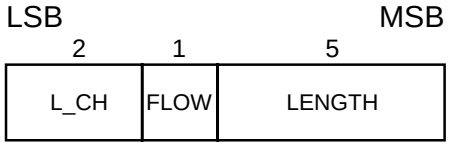


Figure 22—Payload header format for single-slot packets

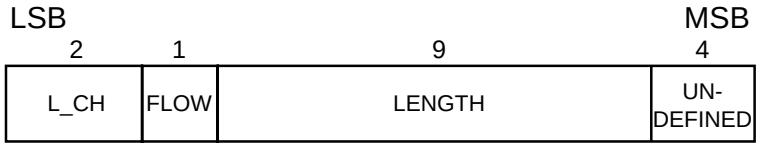


Figure 23—Payload header format for multislot packets

The L_CH field is transmitted first, the length field last. In Table 18, more details about the contents of the L_CH field are listed.

Table 18— Logical channel L_CH field contents

L_CH code b_1b_0	Logical Channel	Information
00	NA	undefined
01	UA/UI	Continuation fragment of an L2CAP message
10	UA/UI	Start of an L2CAP message or no fragmentation
11	LM	LMP message

An L2CAP message can be fragmented into several packets. Code 10 is used for an L2CAP packet carrying the first fragment of such a message; code 01 is used for continuing fragments. If there is no fragmentation, code 10 is used for every packet. Code 11 is used for LMP messages. Code 00 is reserved for future use.

The flow indicator in the payload is used to control the flow at the L2CAP level. It is used to control the flow per logical channel (when applicable). FLOW = 1 means flow-on (“OK to send") and FLOW = 0 means flow-off ("stop"). After a new connection has been established the flow indicator should be set to FLOW = 1. When a Bluetooth unit receives a payload header with the flow bit set to "stop" (FLOW = 0), it shall stop the transmission of ACL packets before an additional amount of payload data is sent. This amount can be defined as the flow control lag, expressed with a number of bytes. The shorter the flow control lag, the less buffering the other Bluetooth device may dedicate to this function. The flow control lag shall not exceed 1792 bytes (7*256 bytes), but in order to allow units to optimize the selection of packet length and buffer space, the flow control lag of a given implementation is provided in the LMP_features_res message (9.3.11).

If a packet containing the payload flow bit of "stop" is received with a valid packet header but bad payload, the payload flow control bit will not be recognized. The packet level ACK contained in the packet header will be received and a further ACL packet can be transmitted. Each occurrence of this situation allows a further ACL packet to be sent in spite of the flow control request being sent via the payload header flow control bit. It is recommended that Bluetooth units that use the payload header flow bit should ensure that no further ACL packets are sent until the payload flow bit has been correctly received. This can be accomplished by simultaneously turning on the flow bit in the packet header and keeping it on until an ACK is received back (ARQN = 1). This will typically be only one round trip time. Since they lack a payload CRC, AUX1 packets should not be used with a payload flow bit of "stop".

The link manager is responsible for setting and processing the flow bit in the payload header. Real-time flow control is carried out at the packet level by the link controller via the flow bit in the packet header (see 8.4.3.3). With the payload flow bit, traffic from the remote end can be controlled. It is allowed to generate and send an ACL packet with payload length zero irrespective of flow status. L2CAP start- and continue-fragment indications (L_CH = 10 and L_CH = 01) also retain their meaning when the payload length is equal to zero (i.e. an empty start-fragment should not be sent in the middle of an on-going L2CAP packet transmission). It is always safe to send an ACL packet with payload length = 0 and L_CH = 01. The payload flow bit has its own meaning for each logical channel (UA/I or LM); see Table 19. On the LM channel, no flow control is applied and the payload flow bit is always set at one.

Table 19—Use of payload header flow bit on the logical channels

L_CH code b_1b_0	Usage and semantics of the ACL payload header FLOW bit
00	Not defined, reserved for future use.
01 or 10	Flow control of the UA/I channels (which are used to send L2CAP messages)
11	Always set FLOW = 1 on transmission and ignore the bit on reception

The length indicator indicates the number of bytes (i.e. 8-bit words) in the payload excluding the payload header and the CRC code; i.e. the payload body only. With reference to Figure 22 and Figure 23, the MSB of the length field in a 1-byte header is the last (right-most) bit in the payload header; the MSB of the length field in a 2-byte header is the fourth bit (from left) of the second byte in the payload header.

2) **Payload body**:

The payload body includes the user host information and determines the effective user throughput. The length of the payload body is indicated in the length field of the payload header.

3) **CRC code generation**:

The 16-bit cyclic redundancy check code in the payload is generated by the CRC-CCITT polynomial `210041` (octal representation). It is generated in a way similar to the HEC. Before determining the CRC code, an 8-bit value is used to initialize the CRC generator. For the CRC code in the FHS packets sent in master page response state, the UAP of the slave is used. For the FHS packet sent in inquiry response state, the DCI is used (see 8.5.4). For all other packets, the UAP of the master is used.

The 8 bits are loaded into the 8 least significant (left-most) positions of the LFSR circuit (see Figure 33). The other 8 bits are at the same time reset to zero. Subsequently, the CRC code is calculated over the information. Then the CRC code is appended to the information; the UAP (or DCI) is disregarded. At the receive side, the CRC circuitry is in the same way initialized with the 8-bit UAP (DCI) before the received information is checked. More information can be found in 8.5.4.

8.4.6 Packet summary

A summary of the packets and their characteristics is shown in Table 20, Table 21, and Table 22. The user payload represents the packet payload excluding FEC, CRC, and payload header.

Table 20—Link control packets

Type	User payload (bytes)	FEC	CRC	Symmetric max. rate	Asymmetric max. rate
ID	na	na	na	na	na
NULL	na	na	na	na	na
POLL	na	na	na	na	na
FHS	18	2/3	yes	na	na

Table 21—ACL packets

Type	Payload header (bytes)	User payload (bytes)	FEC	CRC	Symmetric max. rate (kb/s)	Asymmetric max. rate (kb/s)	
						Forward	Reverse
DM1	1	0–17	2/3	yes	108.8	108.8	108.8
DH1	1	0–27	no	yes	172.8	172.8	172.8
DM3	2	0–121	2/3	yes	258.1	387.2	54.4
DH3	2	0–183	no	yes	390.4	585.6	86.4
DM5	2	0–224	2/3	yes	286.7	477.8	36.3
DH5	2	0–339	no	yes	433.9	723.2	57.6
AUX1	1	0–29	no	no	185.6	185.6	185.6

Table 22—SCO packets

Type	Payload Header (bytes)	User Payload (bytes)	FEC	CRC	Symmetric Max. Rate (kb/s)
HV1	na	10	1/3	no	64.0
HV2	na	20	2/3	no	64.0
HV3	na	30	no	no	64.0
DV[a]	1 D	10+(0–9) D	2/3 D	yes D	64.0+57.6 D

[a]Items followed by ‘D’ relate to data field only.

8.5 Error Correction

There are three error correction schemes defined for Bluetooth:

— 1/3 rate FEC

— 2/3 rate FEC

— ARQ scheme for the data

The purpose of the FEC scheme on the data payload is to reduce the number of retransmissions. However, in a reasonably error-free environment, FEC gives unnecessary overhead that reduces the throughput. Therefore, the packet definitions given in 8.4 have been kept flexible to use FEC in the payload or not, resulting in the DM and DH packets for the ACL link and the HV packets for the SCO link. The packet header is always protected by a 1/3 rate FEC; it contains valuable link information and should be able to sustain more bit errors.

Correction measures to mask errors in the voice decoder are not included in this clause. This matter is discussed in 8.12.3.

8.5.1 FEC Code: Rate 1/3

A simple 3-times repetition FEC code is used for the header. The repetition code is implemented by repeating the bit three times, see the illustration in Figure 24. The 3-bit repetition code is used for the entire header, and also for the voice field in the HV1 packet.

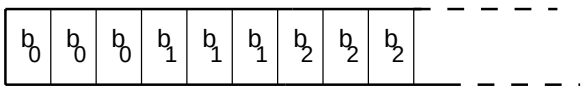


Figure 24—Bit-repetition encoding scheme

8.5.2 FEC Code: Rate 2/3

The other FEC scheme is a (15,10) shortened Hamming code. The generator polynomial is $g(D) = (D + 1)(D^4 + D + 1)$. This corresponds to 65 in octal notation. The LFSR generating this code is depicted in Figure 25. Initially all register elements are set to zero. The 10 information bits are sequentially fed into the LFSR with the switches S1 and S2 set in position 1. Then, after the final input bit, the switches S1 and S2 are set in position 2, and the five parity bits are shifted out. The parity bits are appended to the information bits. Consequently, each block of 10 information bits is encoded into a 15 bit codeword. This code can correct all single errors and detect all double errors in each codeword. This 2/3 rate FEC is used in the DM packets, in the data field of the DV packet, in the FHS packet, and in the HV2 packet. Since the encoder operates with information segments of length 10, tail bits with value zero may have to be appended after the CRC bits. The total number of bits to encode, i.e., payload header, user data, CRC, and tail bits, shall be a multiple of 10. Thus, the number of tail bits to append is the least possible that achieves this (i.e., in the interval 0...9). These tail bits are not included in the payload length indicator.

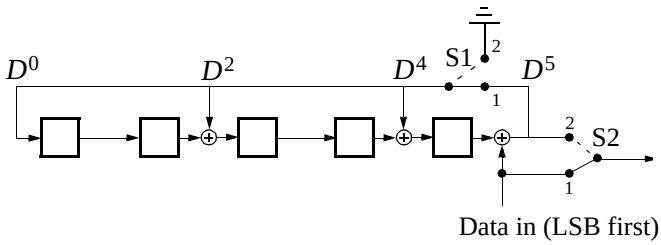


Figure 25—LFSR generating the (15,10) shortened Hamming code

8.5.3 ARQ Scheme

With an automatic repeat request scheme, DM, DH and the data field of DV packets are transmitted and retransmitted until acknowledgement of a successful reception is returned by the destination (or timeout is exceeded). The acknowledgement information is included in the header of the return packet, so-called piggy-backing. To determine whether the payload is correct or not, a cyclic redundancy check (CRC) code is added to the packet. The ARQ scheme only works on the payload in the packet (only that payload which has a CRC). The packet header and the voice payload are not protected by the ARQ scheme.

8.5.3.1 Unnumbered ARQ

Bluetooth uses a fast, unnumbered acknowledgment scheme: an ACK (ARQN = 1) or a NAK (ARQN = 0) is returned in response to the receipt of previously received packet. The slave will respond in the slave-to-master slot directly following the master-to-slave slot; the master will respond in its next transmission to the same slave. In a multislave piconet operation, between two successive transmissions from the master to the same slave, the master may transmit to other slaves as well. Hence, an ACK to a slave's transmission may not always be included in the master transmission that immediately follows the particular slave's transmission. For a packet reception to be successful, at least the HEC must check. In addition, the CRC must check if present.

At the start of a new connection which may be the result of a page, page scan, master-slave switch or unpark, the master sends a POLL packet to verify the connection. In this packet the master initializes the ARQN bit to NAK. The response packet sent by the slave also has the ARQN bit set to NAK. The subsequent packets use the following rules.

The ARQN bit is affected by data packets containing CRC and empty slots only. As shown in Figure 26, upon successful reception of a CRC packet, the ARQN bit is set to ACK. If, in any receive slot in the slave, or, in a receive slot in the master following transmission of a packet, one of these events applies:

1) no access code is detected
2) the HEC fails
3) the CRC fails

then the ARQN bit is set to NAK.

Packets that have correct HEC but that are addressed to other slaves, or packets other than DH, DM, or DV packets, do not affect the ARQN bit. In these cases the ARQN bit is left as it was prior to reception of the packet. If a CRC packet with a correct header has the same SEQN as the previously received CRC packet, the ARQN bit is set to ACK and the payload is disregarded without checking the CRC.

The ARQN bit in the FHS packet is not meaningful. Contents of the ARQN bit in the FHS packet should not be checked.

Broadcast packets are checked on errors using the CRC, but no ARQ scheme is applied. Broadcast packets are never acknowledged.

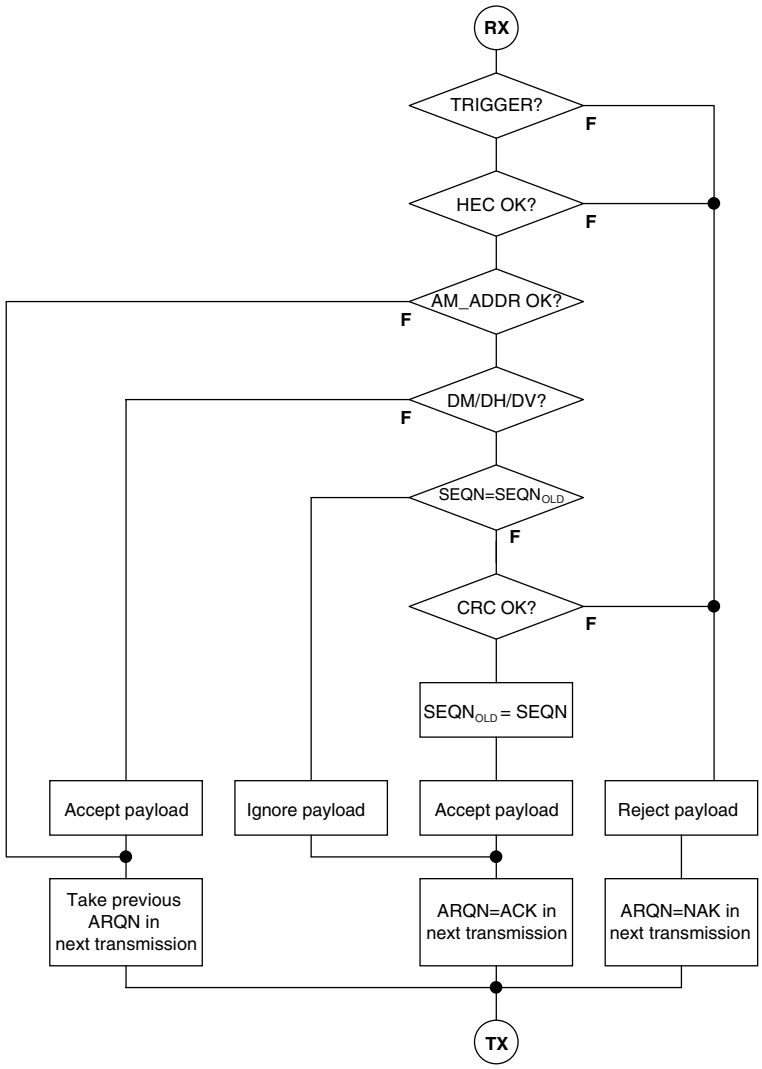


Figure 26—Receive protocol for determining the ARQN bit

8.5.3.2 Retransmit filtering

The data payload is retransmitted until a positive acknowledgment is received (or a timeout is exceeded). A retransmission is carried out either because the packet transmission itself failed, or because the piggy-backed acknowledgment transmitted in the return packet failed (note that the latter has a lower failure probability since the header is more heavily coded). In the latter case, the destination keeps receiving the same payload over and over again. In order to filter out the retransmissions in the destination, the SEQN bit is added in the header. Normally, this bit is alternated for every new CRC data payload transmission. In case of a retransmission, this bit is not changed. So the destination can compare the SEQN bit with the previous SEQN value. If different, a new data payload has arrived; otherwise it is the same data payload and can be discarded. Only new data payloads are transferred to the link manager. Note that CRC data payloads can be carried only by DM, DH, or DV packets.

At the start of a new connection which may be the result of a page, page scan, master slave switch or unpark, the master sends a POLL packet to verify the connection. The slave responds with a packet. The SEQN bit of the first CRC data packet, on both the master and the slave sides, is set to 1. The subsequent packets use the rules given below.

The SEQN bit is affected only by the CRC data packets as shown in Figure 27. It is inverted every time a new CRC data packet is sent. The CRC data packet is retransmitted with the same SEQN number until an ACK is received or the packet is flushed. When an ACK is received, the SEQN bit is inverted and a new payload is sent. When the packet is flushed (see 8.5.3.3), a new payload is sent. The SEQN bit is not necessarily inverted. However, if an ACK is received before the new packet is sent, the SEQN bit is inverted. This procedure prevents loss of the first packet of a message (after the flush command has been given) due to the retransmit filtering.

If a device decides to flush, and it has not received an acknowledgement for the current packet, it replaces the current packet with an ACL L2CAP continuation packet with length zero. It transmits this packet with the same sequence number as the packet it is trying to flush until it does get an ACK. Only then can it move on to the next packet. If a flush is needed for the next packet in the transmit queue before the zero-length packet has been transmitted, that next packet can be removed from the queue directly without it also being replaced by a zero-length packet. The described flushing procedure is considered optional, although strongly recommended.

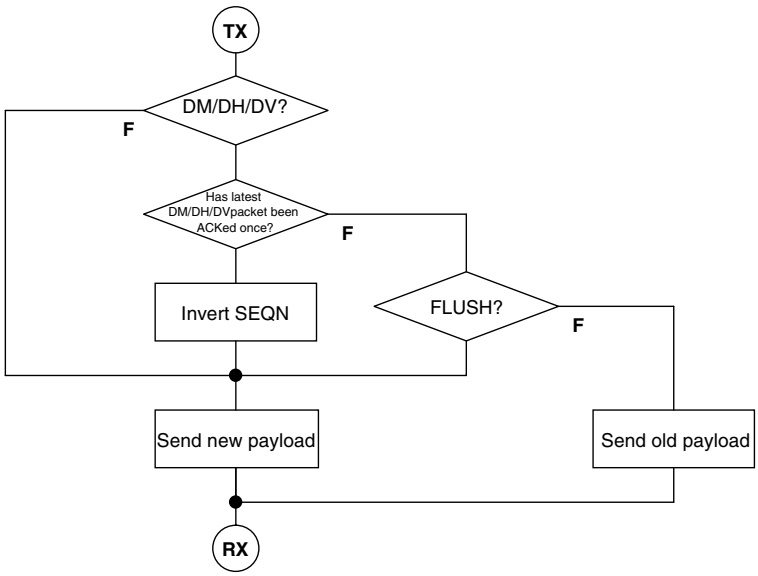


Figure 27—Retransmit filtering for packets with CRC

The SEQN bit in the FHS packet is not meaningful. This bit can be set to any value. Contents of the SEQN bit in the FHS packet should not be checked. During transmission of all other packets the SEQN bit remains the same as it was in the previous packet.

8.5.3.3 Flushing payloads

The ARQ scheme can cause variable delay in the traffic flow since retransmissions are inserted to assure error-free data transfer. For certain communication links, only a limited amount of delay is allowed: retransmissions are allowed up to a certain limit at which the current payload must be disregarded and the next payload must be considered. This data transfer is indicated as isochronous traffic. This means that the retransmit process shall be overruled in order to continue with the next data payload. Aborting the retransmit

scheme is accomplished by *flushing* the old data and forcing the Bluetooth controller to take the next data instead.

Flushing results in loss of remaining portions of an L2CAP message. Therefore, the packet following the flush will have a start packet indication of L_CH = 10 in the payload header for the next L2CAP message. This informs the destination of the flush (see 8.4.5). Flushing will not necessarily result in a change in the SEQN bit value (see 8.5.3.2).

8.5.3.4 Multislave considerations

In case of a piconet with multiple slaves, the master carries out the ARQ protocol independently to each slave.

8.5.3.5 Broadcast packets

Broadcast packets are packets transmitted by the master to all the slaves simultaneously. A broadcast packet is indicated by the all-zero AM_ADDR. (Note that the FHS packet is the only packet which may have an all-zero address but is not a broadcast packet.) Broadcast packets are not acknowledged (at least not at the LC level).

Since broadcast messages are not acknowledged, each broadcast packet is repeated for a fixed number of times. A broadcast packet is repeated N_{BC} times before the next broadcast packet of the same broadcast message is repeated (see Figure 28). However, time-critical broadcast information may abort the ongoing broadcast repetition train. For instance, unpark messages sent at beacon instances may do this (see 8.10.8.4.5).

Broadcast packets with a CRC have their own sequence number. The SEQN of the first broadcast packet with a CRC is set to SEQN = 1 by the master and it is inverted for each new broadcast packet with CRC thereafter. Broadcast packets without a CRC have no influence on the sequence number. The slave accepts the SEQN of the first broadcast packet it receives in a connection and checks for change in SEQN for consequent broadcast packets. Since there is no acknowledgement of broadcast messages and there is no end packet indication, it is important to receive the start packets correctly. To ensure this, repetitions of the broadcast packets that are L2CAP start packets and LMP packets will not be filtered out. These packets are indicated by L_CH=1X in the payload header as explained in 8.4.5. Only repetitions of the L2CAP continuation packets will be filtered out.

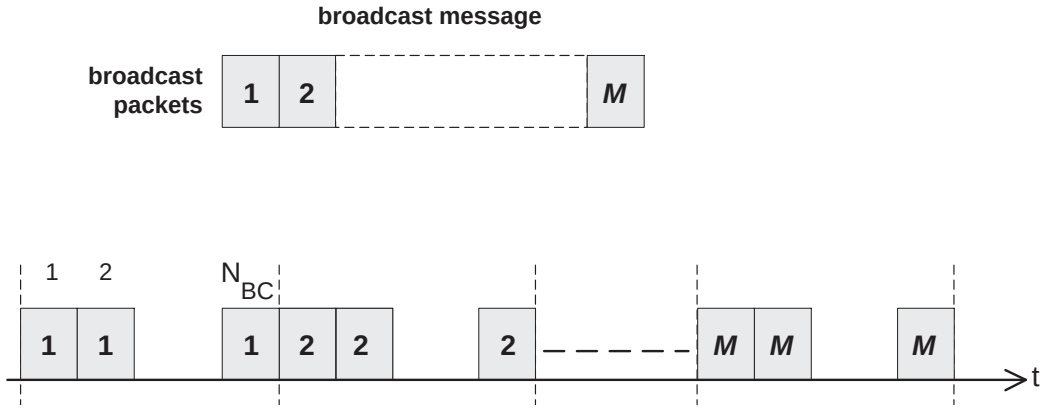


Figure 28—Broadcast repetition scheme

8.5.4 Error checking

Errors in packets or wrong delivery can be checked using the channel access code, the HEC in the header of a packet, and the CRC in payload of a packet. At packet reception, first the access code is checked. Since the 64-bit sync word in the channel access code is derived from the 24-bit master LAP, this checks if the LAP is correct, and prevents the receiver from accepting a packet of another piconet.

The HEC and CRC are used to check both on errors and on a wrong address: to increase the address space with 8 bits, the UAP is normally included in the HEC and CRC checks. Then, even when a packet with the same access code (i.e., an access code of a device owning the same LAP, but different UAP) passes the access code test, it will be discarded after the HEC and CRC tests when the UAP bits do not match. However, there is an exception where no common UAP is available in the transmitter and receiver. This is the case when the HEC and CRC are generated for the FHS packet in inquiry response state. In this case the default check initialization (DCI) value is used. The DCI is defined to be `0x00` (hexadecimal).

The generation and check of the HEC and CRC are summarized in Figure 31 and Figure 34. Before calculating the HEC or CRC, the shift registers in the HEC/CRC generators are initialized with the 8-bit UAP (or DCI) value. Then the header and payload information is shifted into the HEC and CRC generators, respectively (with the LSB first).

The HEC generating LFSR is depicted in Figure 79. The generator polynomial is $g(D) = (D + 1)(D^7 + D^4 + D^3 + D^2 + 1) = D^8 + D^7 + D^5 + D^2 + D + 1$. Initially this circuit is preloaded with the 8-bit UAP such that the LSB of the UAP (denoted UAP_0) goes to the left-most shift register element, and, UAP_7 goes to the right-most element. The initial state of the HEC LFSR is depicted in Figure 30. Then the data is shifted in with the switch S set in position 1. When the last data bit has been clocked into the LFSR, the switch S is set in position 2, and, the HEC can be read out from the register. The LFSR bits are read out from right to left (i.e., the bit in position 7 is the first to be transmitted, followed by the bit in position 6, etc.).

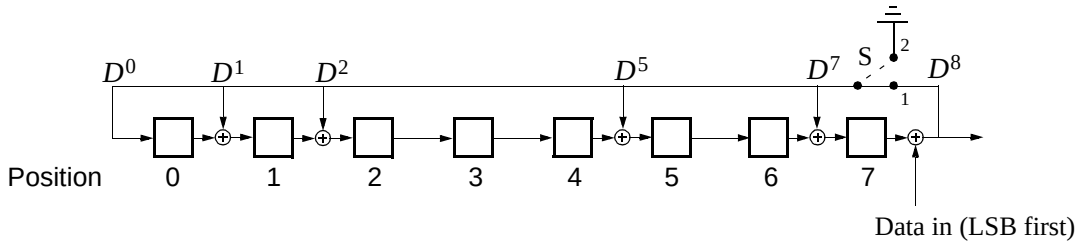


Figure 29—LFSR circuit generating the HEC

Position:	0	1	2	3	4	5	6	7
LFSR:	UAP_0	UAP_1	UAP_2	UAP_3	UAP_4	UAP_5	UAP_6	UAP_7

Figure 30—Initial state of the HEC generating circuit

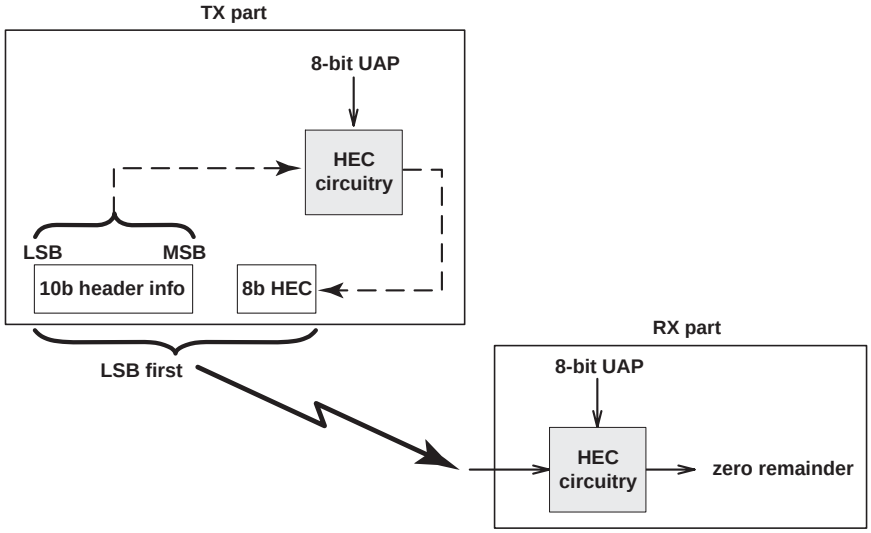


Figure 31—HEC generation and checking

The 16-bit LFSR for the CRC is constructed similarly using the CRC-CCITT generator polynomial $g(D) = D^{16} + D^{12} + D^5 + 1$ (see Figure 32). For this case, the 8 left-most bits are initially loaded with the 8-bit UAP (UAP_0 to the left and UAP_7 to the right) while the 8 right-most bits are reset to zero. The initial state of the 16 bit LFSR is depicted in Figure 33. The switch S is set in position 1 while the data is shifted in. After the last bit has entered the LFSR, the switch is set in position 2, and, the register's contents are transmitted, from right to left (i.e., starting with position 15, then position 14, etc.).

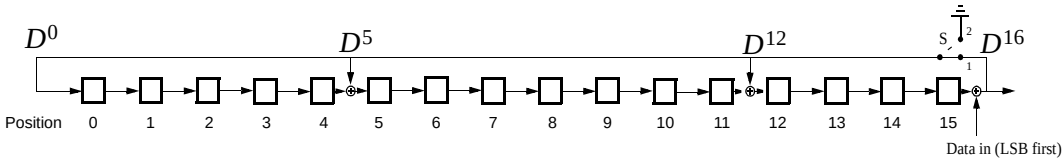


Figure 32—LFSR circuit generating the CRC

Position:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LFSR:	UAP_0	UAP_1	UAP_2	UAP_3	UAP_4	UAP_5	UAP_6	UAP_7	0	0	0	0	0	0	0	0

Figure 33—Initial state of the CRC generating circuit

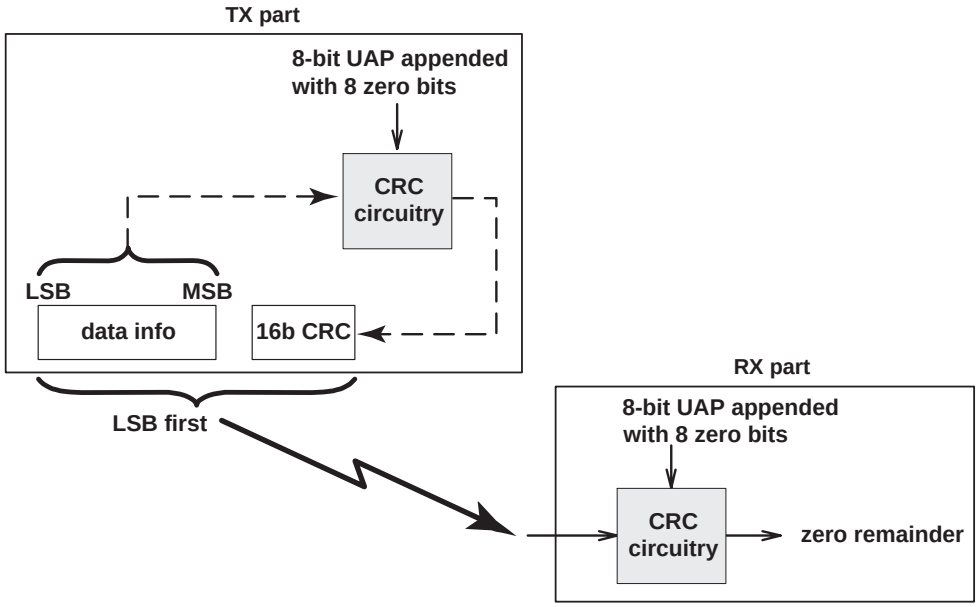


Figure 34—CRC generation and checking

8.6 Logical channels

In the Bluetooth system, five logical channels are defined:

— LC control channel
— LM control channel
— UA user channel
— UI user channel
— US user channel

The control channels LC and LM are used at the link control level and link manager level, respectively. The user channels UA, UI, and US are used to carry asynchronous, isochronous, and synchronous user information, respectively. The LC channel is carried in the packet header; all other channels are carried in the packet payload. The LM, UA, and UI channels are indicated in the L_CH field in the payload header. The US channel is carried by the SCO link only. The UA and UI channels are normally carried by the ACL link; however, they can also be carried by the data in the DV packet on the SCO link. The LM channel can be carried either by the SCO or the ACL link.

8.6.1 LC channel (Link control)

The LC channel is mapped onto the packet header. This channel carries low-level link control information like ARQ, flow control, and payload characterization. The LC channel is carried in every packet except in the **ID** packet which has no packet header.

8.6.2 LM channel (Link manager)

The LM control channel carries control information exchanged between the link managers of the master and the slave(s). Typically, the LM channel uses protected DM packets. The LM channel is indicated by the L_CH code 11 in the payload header.

8.6.3 UA/UI channel (User asynchronous/Isochronous data)

The UA channel carries L2CAP transparent asynchronous user data. This data may be transmitted in one or more baseband packets. For fragmented messages, the start packet uses an L_CH code of 10 in the payload header. Remaining continuation packets use L_CH code 01. If there is no fragmentation, all packets use the L2CAP start code 10.

Isochronous data channel is supported by timing start packets properly at higher levels. At the baseband level, the L_CH code usage is the same as the UA channel.

8.6.4 US Channel (User Synchronous data)

The US channel carries transparent synchronous user data. This channel is carried over the SCO link.

8.6.5 Channel mapping

The LC channel is mapped onto the packet header. All other channels are mapped onto the payload. The US channel can only be mapped onto the SCO packets. All other channels are mapped on the ACL packets, or possibly the SCO DV packet. The LM, UA, and UI channels may interrupt the US channel if it concerns information of higher priority.

8.7 Data whitening

Before transmission, both the header and the payload are scrambled with a data whitening word in order to randomize the data from highly redundant patterns and to reduce DC bias in the packet. The scrambling is performed prior to the FEC encoding.

At the receiver, the received data is descrambled using the same whitening word generated in the recipient. The descrambling is performed after FEC decoding.

The whitening word is generated with the polynomial $g(D) = D^7 + D^4 + 1$ (i.e., `221` in octal representation) and is subsequently EXORed with the header and the payload. The whitening word is generated with the linear feedback shift register shown in Figure 35. Before each transmission, the shift register is initialized with a portion of the master Bluetooth clock, $CLK_{6\text{-}1}$, extended with an MSB of value one. This initialization is carried out with CLK_1 written to position 0, CLK_2 written to position 1, etc. An exception exists when forming the FHS packet sent during frequency hop acquisition. In this case, initialization of the whitening register is carried out differently. Instead of the master clock, the X-input used in the inquiry or page response (depending on current state) routine is used; see Table 31 and Table 32 for the 79-hop and 23-hop systems, respectively. In case of a 79-hop system, the 5-bit values is extended with two MSBs of value one. In case of a 23-hop system, the 4-bit value is extended with three bits; the two MSBs are set to one and the third most significant bit is set to zero. During register initialization, the LSB of X (i.e., X_0) is written to position 0, X_1 is written to position 1, etc.

After initialization, the packet header and the payload (including the CRC) are scrambled. The payload whitening continues from the state the whitening LFSR had at the end of HEC. There is no re-initialization of the shift register between packet header and payload. The first bit of the “Data In” sequence is the LSB of the packet header.

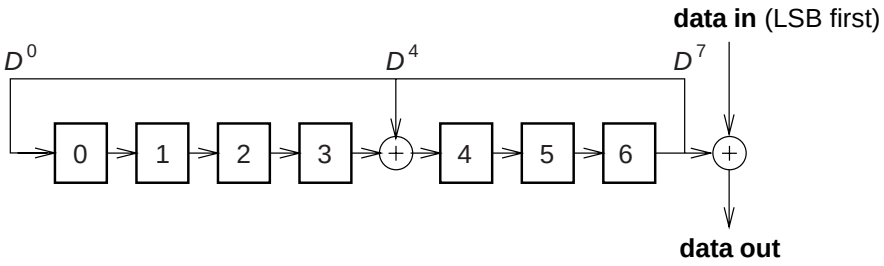


Figure 35—Data whitening LFSR

8.8 Transmit/Receive routines

This subclause describes the way to use the packets as defined in 8.4 in order to support the traffic on the ACL and SCO links. Both single-slave and multi-slave configurations are considered. In addition, the use of buffers for the TX and RX routines are described.

Note that the TX and RX routines described in 8.8.1 and 8.8.2 are of an informative character only. The final implementation may be carried out differently.

8.8.1 TX routine

The TX routine is carried out separately for each ACL link and each SCO link. Figure 36 shows the ACL and SCO buffers as used in the TX routine. In this figure, only a single TX ACL buffer and a single TX SCO buffer are shown. In the master, there is a separate TX ACL buffer for each slave. In addition there may be one or more TX SCO buffers for each SCO slave (different SCO links may either reuse the same TX SCO buffer, or each have their own TX SCO buffer). Each TX buffer consists of two FIFO registers: one current register which can be accessed and read by the Bluetooth controller in order to compose the packets, and one next register that can be accessed by the Bluetooth Link Manager to load new information. The positions of the switches S1 and S2 determine which register is current and which register is next; the switches are controlled by the Bluetooth Link Controller. The switches at the input and the output of the FIFO registers can never be connected to the same register simultaneously.

Of the packets common on the ACL and SCO links (ID, NULL, POLL, FHS, DM1) only the DM1 packet carries a payload that is exchanged between the Link Controller and the Link Manager; this common packet makes use of the ACL buffer. All ACL packets make use of the ACL buffer. All SCO packets make use of the SCO buffer except for the DV packet where the voice part is handled by the SCO buffer and the data part is handled by the ACL buffer. In 8.8.1.1 through 8.8.1.3, the operation for ACL traffic, SCO traffic, and combined data-voice traffic on the SCO link will be considered.

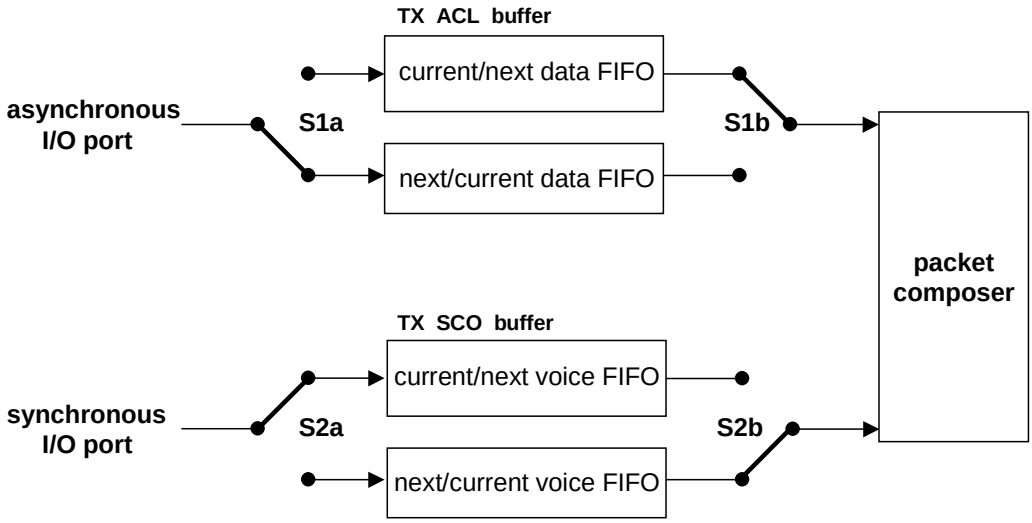


Figure 36—Functional diagram of TX buffering

8.8.1.1 ACL traffic

In the case of pure (asynchronous) data, only the TX ACL buffer in Figure 36 has to be considered. In this case, only packet types DM or DH are used, and these can have different lengths. The length is indicated in the payload header. The selection of high-rate data or medium-rate data shall depend on the quality of the link. When the quality is good, the FEC in the data payload can be omitted, resulting in a DH packet. Otherwise, DM packets must be used.

The default TYPE in pure data traffic is NULL. This means that, if there is no data to be sent (the data traffic is asynchronous, and therefore pauses occur in which no data is available) or no slaves need to be polled, NULL packets are sent instead – in order to send link control information to the other Bluetooth unit (e.g. ACK/STOP information for received data). When no link control information is available either (no need to acknowledge and/or no need to stop the RX flow) no packet is sent at all.

The TX routine works as follows. The Bluetooth Link Manager loads new data information in the register to which the switch S1a points. Next, it gives a flush command to the Bluetooth Link Controller, which forces the switch S1 to change (both S1a and S1b switch in synchrony). When the payload needs to be sent, the packet composer reads the current register and, depending on the packet TYPE, builds a payload which is appended to the channel access code and the header and is subsequently transmitted. In the response packet (which arrives in the following RX slot if it concerned a master transmission, or may be postponed until some later RX slot if it concerned a slave transmission), the result of the transmission is reported back. In case of an ACK, the switch S1 changes position; if a NAK (explicit or implicit) is received instead, the switch S1 will not change position. In that case, the same payload is retransmitted at the next TX occasion.

As long as the Link Manager keeps loading the registers with new information, the Bluetooth Link Controller will automatically transmit the payload; in addition, retransmissions are performed automatically in case of errors. The Link Controller will send NULL or nothing when no new data is loaded. If no new data has been loaded in the next register, during the last transmission, the packet composer will be pointing to an empty register after the last transmission has been acknowledged and the next register becomes the current register. If new data is loaded in the next register, a flush command is required to switch the S1 switch to the proper register. As long as the Link Manager keeps loading the data and type registers before each TX slot, the data is automatically processed by the Link Controller since the S1 switch is controlled by the ACK information received in response. However, if the traffic from the Link Manager is interrupted once and a default packet is sent instead, a flush command is required to continue the flow in the Link Controller.

The flush command can also be used in case of time-bounded (isochronous) data. In case of a bad link, many retransmission are necessary. In certain applications, the data is time-bounded: if a payload is retransmitted all the time because of link errors, it may become outdated, and the system might decide to continue with more recent data instead and skip the payload that does not come through. This is accomplished by the flush command as well. With the flush, the switch S1 is forced to change and the Link Controller is forced to consider the next data payload and overrules the ACK control.

8.8.1.2 SCO traffic

In case of an SCO link, we only use HV packet types. The synchronous port continuously loads the next register in the SCO buffer. The S2 switches are changed according to the T_{sco} interval. This T_{sco} interval is negotiated between the master and the slave at the time the SCO link is established.

For each new SCO slot, the packet composer reads the current register after which the S2 switch is changed. If the SCO slot has to be used to send control information with high priority concerning a control packet between the master and the considered SCO slave, or a control packet between the master and any other slave, the packet composer will discard the SCO information and use the control information instead. This control information shall be sent in a DM1 packet. Data or link control information can also be exchanged between the master and the SCO slave by using the DV or DM1 packets. Any ACL type of packet can be used to sent data or link control information to any other ACL slave. This is discussed in 8.8.1.3.

8.8.1.3 Mixed data/voice traffic

In 8.4.4.2, a DV packet has been defined that can support both data and voice simultaneously on a single SCO link. When the TYPE is DV, the Link Controller reads the data register to fill the data field and the voice register to fill the voice field. Thereafter, the switch S2 is changed. However, the position of S1 depends on the result of the transmission like on the ACL link: only if an ACK has been received will the S1 switch change its position. In each DV packet, the voice information is new, but the data information might be retransmitted if the previous transmission failed. If there is no data to be sent, the SCO link will automatically change from DV packet type to the current HV packet type used before the mixed data/voice transmission. Note that a flush command is required when the data stream has been interrupted and new data has arrived.

Combined data-voice transmission can also be accomplished by using separate ACL links in addition to the SCO link(s) if channel capacity permits this.

8.8.1.4 Default packet types

On the ACL links, the default type is always NULL both for the master and the slave. This means that if no user information needs to be sent, either a NULL packet is sent if there is ACK or STOP information, or no packet is sent at all. The NULL packet can be used by the master to allocate the next slave-to-master slot to a certain slave (namely the one addressed). However, the slave is not forced to respond to the NULL packet from the master. If the master requires a response, it has to send a POLL packet.

The SCO packet type is negotiated at the LM level when the SCO link is established. The agreed packet type is also the default packet type for the SCO slots.

8.8.2 RX routine

The RX routine is carried out separately for the ACL link and the SCO link. However, in contrast to the master TX ACL buffer, a single RX buffer is shared among all slaves. For the SCO buffer, it depends how the different SCO links are distinguished whether extra SCO buffers are required or not. Figure 37 shows the ACL and SCO buffers as used in the RX routine. The RX ACL buffer consists of two FIFO registers: one register that can be accessed and loaded by the Bluetooth Link Controller with the payload of the latest RX

packet, and one register that can be accessed by the Bluetooth Link Manager to read the previous payload. The RX SCO buffer also consists of two FIFO registers: one register that is filled with newly arrived voice information, and one register that can be read by the voice processing unit.

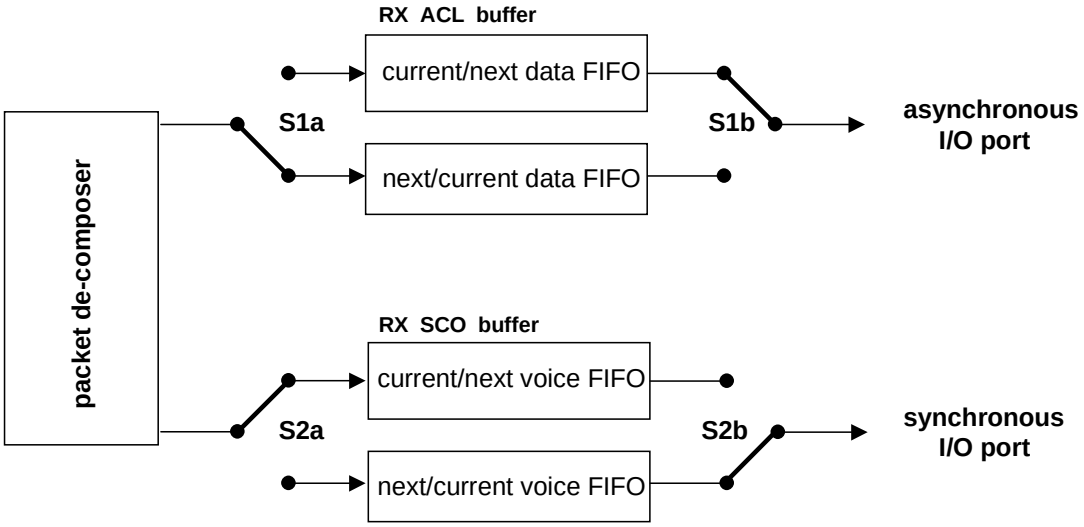


Figure 37—Functional diagram of RX buffering

Since the TYPE indication in the header of the received packet indicates whether the payload contains data and/or voice, the packet decomposer can automatically direct the traffic to the proper buffers. The switch S1 changes every time the Link Manager has read the old register. If the next payload arrives before the RX register is emptied, a STOP indication must be included in the packet header of the next TX packet that is returned. The STOP indication is removed again as soon as the RX register is emptied. The SEQN field is checked before a new ACL payload is stored into the ACL register (flush indication in L_CH and broadcast messages influence the interpretation of the SEQN field see 8.5.3).

The S2 switch is changed every T_{SCO}. If – due to errors in the header – no new voice payload arrives, the switch still changes. The voice processing unit then has to process the voice signal to account for the missing speech parts.

8.8.3 Flow control

Since the RX ACL buffer can be full while a new payload arrives, flow control is required. As was mentioned in 8.4.3.3, the header field FLOW in the return TX packet can use STOP or GO in order to control the transmission of new data.

8.8.3.1 Destination control

As long as data cannot be received, a STOP indication is transmitted which is automatically insertbed by the Link Controller into the header of the return packet. STOP is returned as long as the RX ACL buffer is not emptied by the Link Manager. When new data can be accepted again, the GO indication is returned. GO is the default value. Note that all packet types not including data can still be received. Voice communication, for example, is not affected by the flow control. Also note that although a Bluetooth unit cannot receive new information, it can still continue to transmit information: the flow control is separate for each direction.

8.8.3.2 Source control

On the reception of a STOP signal, the Link Controller will automatically switch to the default packet type. The ACL packet transmitted just before the reception of the STOP indication is kept until a GO signal is received. It is retransmitted as soon as a GO indication is received. Default packets are sent as long as the STOP indication is received. When no packet is received, GO is assumed implicitly. Note that the default packets contain link control information (in the header) for the receive direction (which may still be open) and may contain voice (HV packets). When a GO indication is received, the Link Controller resumes to transmit the data as is present in the TX ACL buffers.

In a multi-slave configuration, only the transmission to the slave that issued the STOP signal is stalled. This means that the previously described routine implemented in the master only concerns the TX ACL buffer that corresponds to the slave that cannot accept data momentarily.

8.8.4 Bitstream processes

Before the user information is sent over the air interface, several bit manipulations are performed in the transmitter to increase reliability and security. To the packet header, an HEC is added, the header bits are scrambled with a whitening word, and FEC coding is applied. In the receiver, the inverse processes are carried out. Figure 38 shows the processes carried out for the packet header both at the transmit and the receive side. All header bit processes are mandatory.

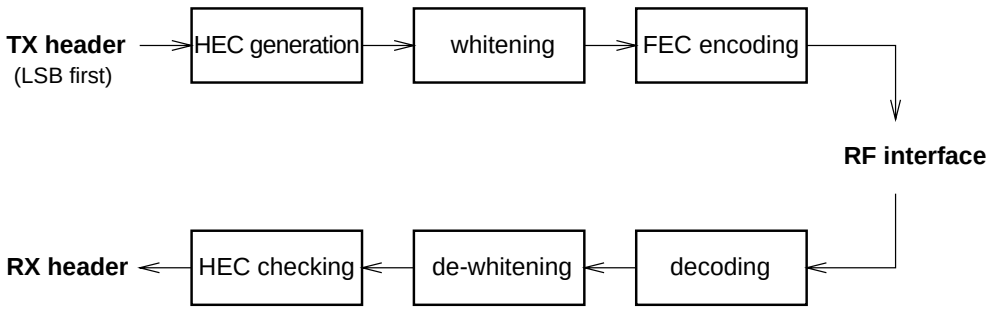


Figure 38—Header bit processes

For the payload, similar processes are performed. It depends on the packet type, which processes are carried out. Figure 39 shows the processes that may be carried out on the payload. In addition to the processes defined for the packet header, encryption can be applied on the payload. Only whitening and de-whitening, as explained in 8.7, are mandatory for every payload; all other processes are optional and depend on the packet type and the mode enabled. In Figure 39, optional processes are indicated by dashed blocks.

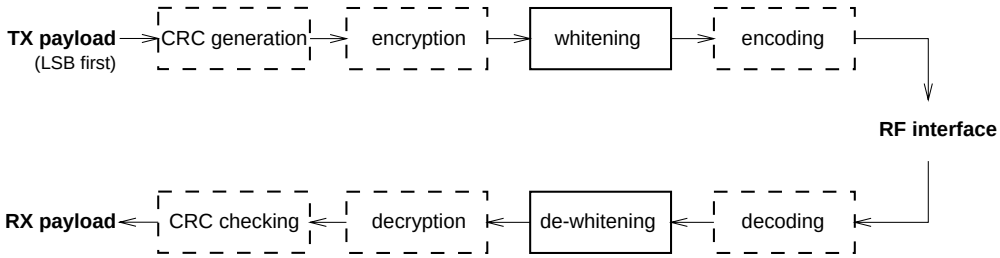


Figure 39—Payload bit processes

8.9 Transmit/receive timing

The Bluetooth transceiver applies a time-division duplex (TDD) scheme. This means that it alternately transmits and receives in a synchronous manner. It depends on the mode of the Bluetooth unit what the exact timing of the TDD scheme is. In the normal connection mode, *the master transmission shall always start at even numbered time slots (master CLK1 = 0) and the slave transmission shall always start at odd numbered time slots (master CLK1 = 1).* Due to packet types that cover more than a single slot, master transmission may continue in odd numbered slots and slave transmission may continue in even numbered slots.

All timing diagrams shown in this subclause are based on the signals as present at the antenna. The term “exact” when used to describe timing refers to an ideal transmission or reception and neglects timing jitter and clock frequency imperfections.

The average timing of master packet transmission shall not drift faster than 20 ppm relative to the ideal slot timing of 625 µs. The instantaneous timing shall not deviate more than 1 µs from the average timing. Thus, the absolute packet transmission timing t_k of slot boundary k shall fulfill the equation:

$$t_k = \left(\sum_{i=1}^{k} (1 + d_i) T_N \right) + j_k + \text{offset}, \quad (1)$$

where T_N is the nominal slot length (625 µs), j_k denotes jitter ($|j_k| \leq 1$ µs) at slot boundary k, and, d_k, denotes the drift ($|d_k| \leq 20$ ppm) within slot k. The jitter and drift may vary arbitrarily within the given limits for every slot, while “offset” is an arbitrary but fixed constant. For hold, park, and sniff mode the drift and jitter parameters as described in 9.3.9 apply.

8.9.1 Master/slave timing synchronization

The piconet is synchronized by the system clock of the master. The master never adjusts its system clock during the existence of the piconet. The slaves adapt their native clocks with a timing offset in order to match the master clock. This offset is updated each time a packet is received from the master: by comparing the exact RX timing of the received packet with the estimated RX timing, the slaves correct the offset for any timing misalignments. Note that the slave RX timing can be corrected with any packet sent in the master-to-slave slot, since only the channel access code is required to synchronize the slave.

The slave TX timing shall be based on the most recent slave RX timing. The RX timing is based on the latest successful trigger during a master-to-slave slot. For ACL links, this trigger must have occurred in the master-to-slave slot directly preceding the current slave transmission; for SCO links, the trigger may have

occurred several master-to-slave slots before since a slave is allowed to send an SCO packet even if no packet was received in the preceding master-to-slave slot. The slave shall be able to receive the packets and adjust the RX timing as long as the timing mismatch remains within the ±10 µs uncertainty window.

The master TX timing is strictly related to the master clock. The master shall keep an exact interval of $M \times 1250$ µs (where M is a positive integer larger than 0) between the start of successive transmissions; the RX timing is based on this TX timing with a shift of exactly $N \times 625$ µs (where N is an odd, positive integer larger than 0). During the master RX cycle, the master will also use the ±10 µs uncertainty window to allow for slave misalignments. The master will adjust the RX processing of the considered packet accordingly, but will not adjust its RX/TX timing for the following TX and RX cycles. During periods when an active slave is not able to receive any valid channel access codes from the master, the slave may increase its receive uncertainty window and/or use predicted timing drift to increase the probability of receiving the master's bursts when reception resumes.

Timing behavior may differ slightly depending on the current state of the unit. The different states are described in 8.9.2 through 8.9.5.

8.9.2 Connection state

In the connection mode, the Bluetooth transceiver transmits and receives alternately (see Figure 40 and Figure 41). In the figures, only single-slot packets are shown as an example. Depending on the type and the payload length, the packet size can be up to 366 µs. Each RX and TX transmission is at a different hop frequency. For multislot packets, several slots are covered by the same packet, and the hop frequency used in the first slot will be used throughout the transmission.

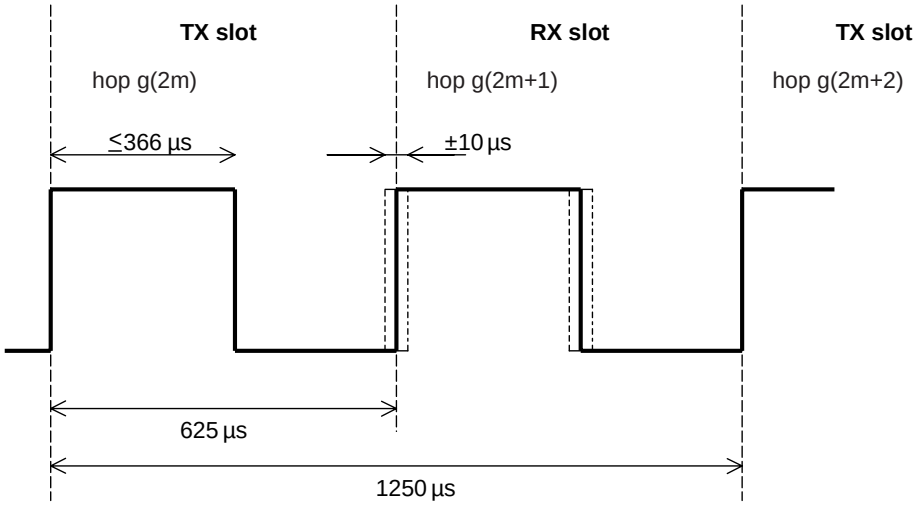


Figure 40—RX/TX cycle of Bluetooth master transceiver in normal mode for single-slot packets

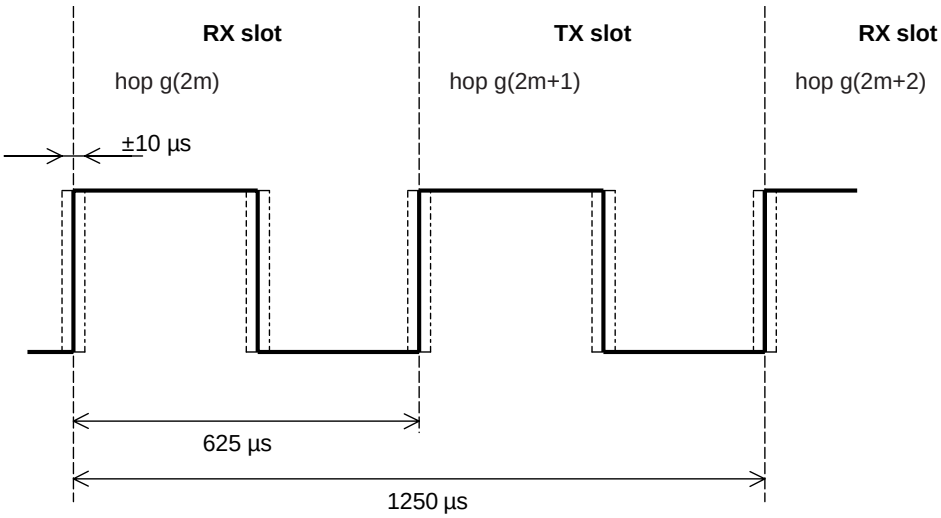


Figure 41—RX/TX cycle of Bluetooth slave transceiver in normal mode for single-slot packets

The master TX/RX timing is shown in Figure 40. In Figure 40 through Figure 45, $f'(k)$ is used for the frequencies of the page hopping sequence and $f'(k)$ denotes the corresponding page response sequence frequencies. The channel hopping frequencies are indicated by *g(m)*. After transmission, a return packet is expected $N \times 625$ µs after the start of the TX burst where N is an odd, positive integer. N depends on the type of the transmitted packet. To allow for some time slipping, an uncertainty window is defined around the exact receive timing. During normal operation, the window length is 20 µs, which allows the RX burst to arrive up to 10 µs too early or 10 µs too late. During the beginning of the RX cycle, the access correlator searches for the correct channel access code over the uncertainty window. If no trigger event occurs, the receiver goes to sleep until the next RX event. If in the course of the search, it becomes apparent that the correlation output will never exceed the final threshold, the receiver may go to sleep earlier. If a trigger event does occur, the receiver remains open to receive the rest of the packet.

The current master transmission is based on the previous master transmission: it is scheduled $M \times 1250$ µs after the start of the previous master TX burst where M depends on the transmitted and received packet type. Note that the master TX timing is not affected by time drifts in the slave(s). If no transmission takes place during a number of consecutive slots, the master will take the TX timing of the latest TX burst as reference.

The slave's transmission is scheduled $N \times 625$ µs after the start of the slave's RX burst. If the slave's RX timing drifts, so will its TX timing. If no reception takes place during a number of consecutive slots, the slave will take the RX timing of the latest RX burst as reference.

8.9.3 Return from hold mode

In the connection state, the Bluetooth unit can be placed in a hold mode (see 8.10.8). In the hold mode, a Bluetooth transceiver neither transmits nor receives information. When returning to the normal operation after a hold mode in a slave Bluetooth unit, the slave must listen for the master before it may send information. In that case, the length of the search window in the slave unit may be increased from 20 µs to a larger value X µs as illustrated in Figure 42. Note that only RX hop frequencies are used: the hop frequency used in the master-to-slave (RX) slot is also used in the uncertainty window extended into the preceding time interval normally used for the slave-to-master (TX) slot.

If the length of search window (X) exceeds 1250 µs, consecutive windows shall not be centered at the start of RX hops $g(2m)$, $g(2m + 2)$, ... $g(2m + 2i)$ (where "i" is an integer) to avoid overlapping search windows.

Consecutive windows should instead be centred at $g(2m)$, $g(2m + 4)$, ... g($2m + 4i$), which gives a maximum value X = 2500 µs, or even at $g(2m)$, $g(2m + 6)$, ... $g(2m + 6i)$ which gives a maximum value X = 3750 µs. The RX hop frequencies used shall correspond to the RX slot numbers.

It is recommended that single slot packets are used upon return from hold to minimize the synchronization time, especially after long hold periods that require search windows exceeding 625 µs.

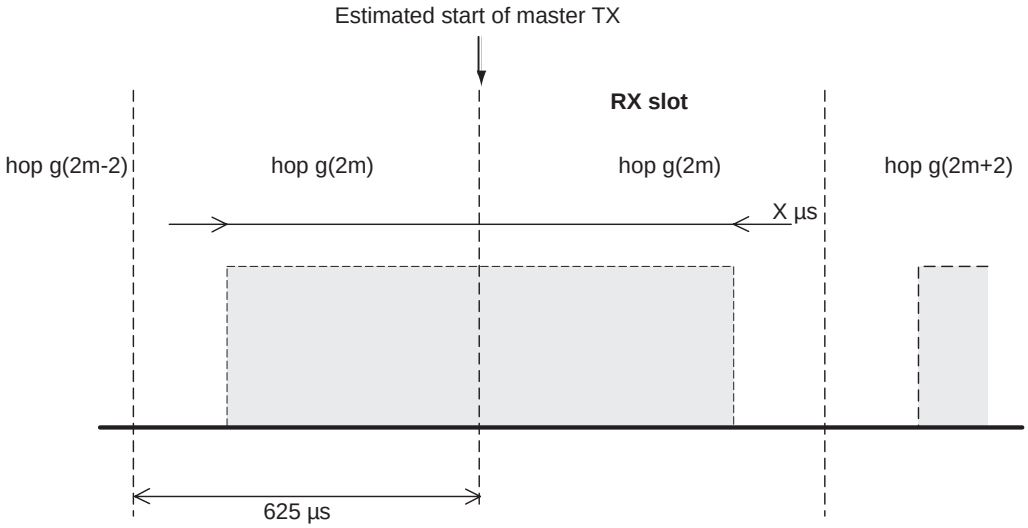


Figure 42—RX timing of slave returning from hold state

8.9.4 Park and sniff modes wake-up

The park and sniff modes are similar to the hold mode. A slave in park or sniff mode periodically wakes up to listen to transmissions from the master and to re-synchronize its clock offset. As in the return from hold mode, a slave in park or sniff mode when waking up may increase the length of the search window from 20 µs to a larger value X µs as illustrated in Figure 42.

8.9.5 Page state

In the page state, the master transmits the device access code (ID packet) corresponding to the slave to be connected, rapidly on a large number of different hop frequencies. Since the ID packet is a very short packet, the hop rate is increased from 1600 hops/s to 3200 hops/s. In a single TX slot interval, the paging master transmits on two different hop frequencies. In a single RX slot interval, the paging transceiver listens on two different hop frequencies; see Figure 43. During the TX slot, the paging unit sends an ID packet at the TX hop frequencies $f(k)$ and $f(k + 1)$. In the RX slot, it listens for a response on the corresponding RX hop frequencies $f'(k)$ and $f'(k + 1)$. The listening periods are exactly timed 625 µs after the corresponding paging packets, and include a ±10 µs uncertainty window.

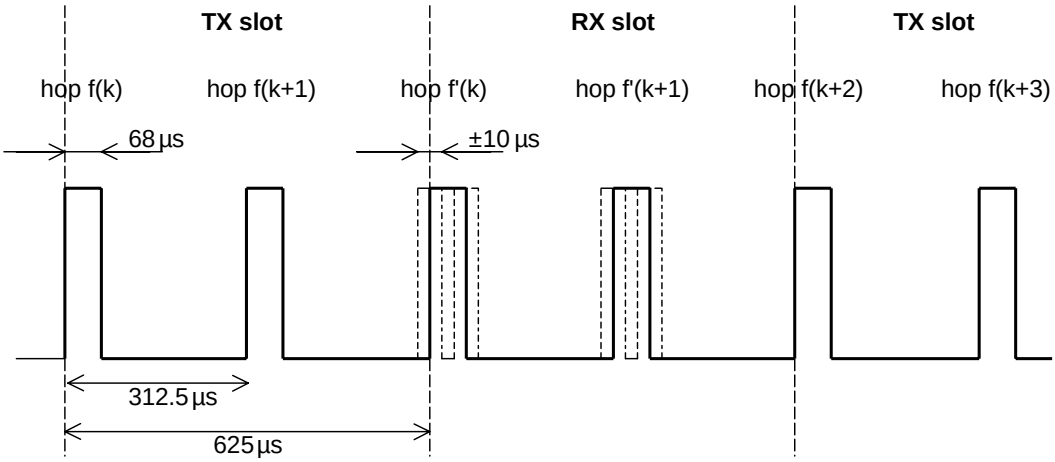


Figure 43—RX/TX cycle of Bluetooth transceiver in PAGE mode

8.9.6 FHS packet

At connection setup and during a master-slave switch, an FHS packet is transferred from the master to the slave. This packet will establish the timing and frequency synchronization (see also 8.4.4.1.4). After the slave unit has received the page message, it will return a response message which again consists of the ID packet and follows exactly 625 µs after the receipt of the page message. The master will send the FHS packet in the TX slot following the RX slot in which it received the slave response, according to the RX/TX timing of the master. The time difference between the response and FHS message will depend on the timing of the page message the slave received. In Figure 44, the slave receives the paging message sent first in the master-to-slave slot. It will then respond with an ID packet in the first half of the slave-to-master slot. The timing of the FHS packet is based on the timing of the page message sent first in the preceding master-to-slave slot: there is an exact 1250 µs delay between the first page message and the FHS packet. The packet is sent at the hop frequency *f(k + 1)* which is the hop frequency following the hop frequency *f(k)* the page message was received in. In Figure 45, the slave receives the paging message sent on the second frequency of the preceding master-to-slave slot. It will then respond with an ID packet in the second half of the slave-to-master slot exactly 625 µs after the receipt of the page message. The timing of the FHS packet is still based on the timing of the page message sent first in the preceding master-to-slave slot: there is an exact 1250 µs delay between the first page message and the FHS packet. The packet is sent at the hop frequency *f(k + 2)* which is the hop frequency following the hop frequency *f(k + 1)* the page message was received in.

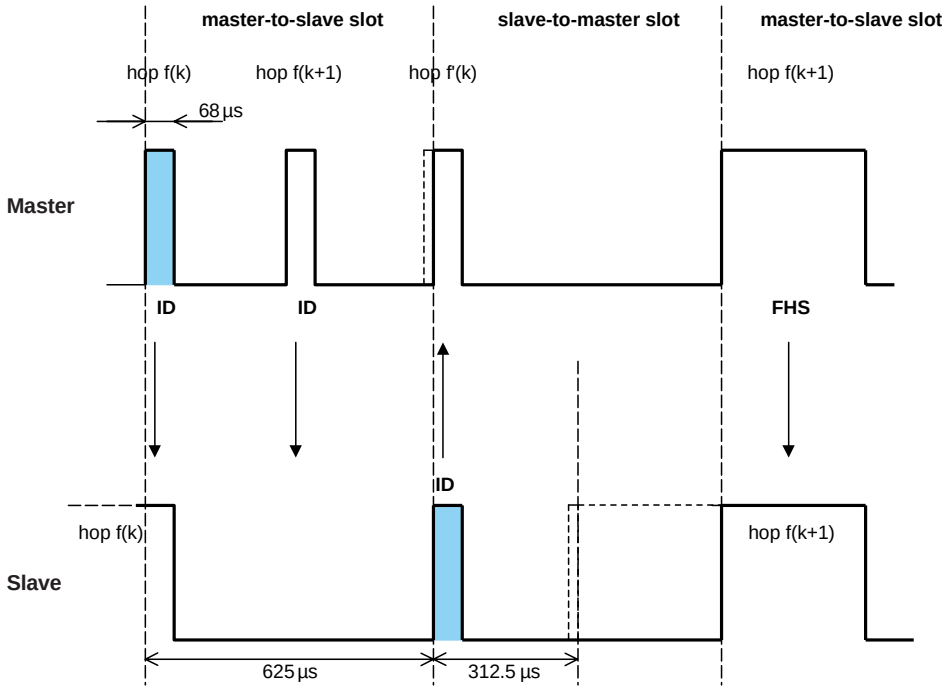


Figure 44—Timing of FHS packet on successful page in first half slot

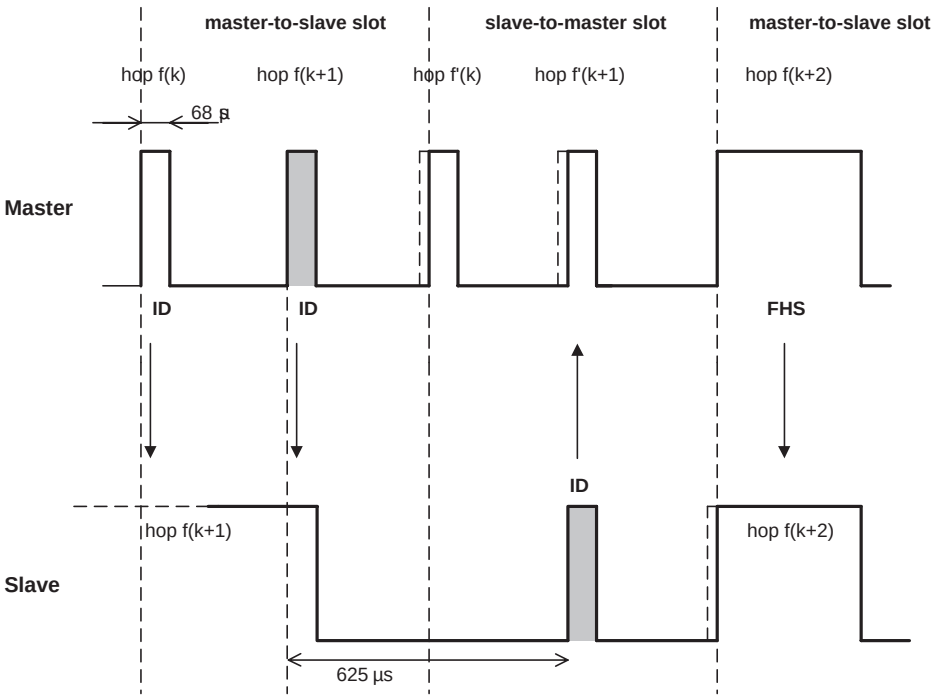


Figure 45—Timing of FHS packet on successful page in second half slot

The slave will adjust its RX/TX timing according to the reception of the FHS packet (and not according to the reception of the page message). The response sent 625 µs after the start of the FHS packet acknowledges receipt of the FHS packet.

8.9.7 Multislave operation

As was mentioned in the beginning of this clause, the master always starts the transmission in the even-numbered slots whereas the slaves start their transmission in the odd-numbered slots. This means that the timing of the master and the slave(s) is shifted by one slot (625 µs); see Figure 46.

Only the slave that is addressed by its AM_ADDR can return a packet in the next slave-to-master slot. If no valid AM_ADDR is received, the slave may only respond if it concerns its reserved SCO slave-to-master slot. In case of a broadcast message, no slave is allowed to return a packet (an exception is the access window for access requests in the park mode; see 8.10.8.4).

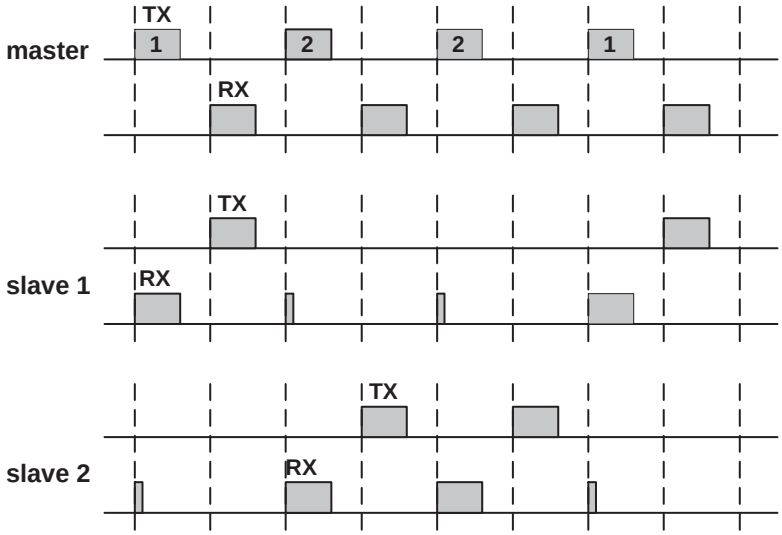


Figure 46—RX/TX timing in multislave configuration

8.10 Channel control

8.10.1 Scope

This section describes how the channel of a piconet is established and how units can be added to and released from the piconet. Several states of operation of the Bluetooth units are defined to support these functions. In addition, the operation of several piconets sharing the same area, the so-called scatternet, is discussed. A special section is attributed to the Bluetooth clock which plays a major role in the FH synchronization.

8.10.2 Master-slave definition

The channel in the piconet is characterized entirely by the master of the piconet. The Bluetooth device address (BD_ADDR) of the master determines the FH hopping sequence and the channel access code; the system clock of the master determines the phase in the hopping sequence and sets the timing. In addition, the master controls the traffic on the channel by a polling scheme.

By definition, the master is represented by the Bluetooth unit that initiates the connection (to one or more slave units). Note that the names “master” and “slave” only refer to the protocol on the channel: the Bluetooth units themselves are identical; that is, any unit can become a master of a piconet. Once a piconet has been established, master-slave roles can be exchanged. This is described in more detail in 8.10.9.3.

8.10.3 Bluetooth clock

Every Bluetooth unit has an internal system clock which determines the timing and hopping of the transceiver. The Bluetooth clock is derived from a free running native clock which is never adjusted and is never turned off. For synchronization with other units, only offsets are used that, added to the native clock, provide temporary Bluetooth clocks which are mutually synchronized. It should be noted that the Bluetooth clock has no relation to the time of day; it can therefore be initialized at any value. The Bluetooth clock provides the heart beat of the Bluetooth transceiver. Its resolution is at least half the TX or RX slot length, or 312.5 μs. The clock has a cycle of about a day. If the clock is implemented with a counter, a 28-bit counter is required that wraps around at 2^{28}–1. The LSB ticks in units of 312.5 μs, giving a clock rate of 3.2 kHz.

The timing and the frequency hopping on the channel of a piconet is determined by the Bluetooth clock of the master. When the piconet is established, the master clock is communicated to the slaves. Each slave adds an offset to its native clock to be synchronized to the master clock. Since the clocks are free-running, the offsets have to be updated regularly.

The clock determines critical periods and triggers the events in the Bluetooth receiver. Four periods are important in the Bluetooth system: 312.5 μs, 625 μs, 1.25 ms, and 1.28 s; these periods correspond to the timer bits CLK_0, CLK_1, CLK_2, and CLK_{12}, respectively (see Figure 47). Master-to-slave transmission starts at the even-numbered slots when CLK_0 and CLK_1 are both zero.

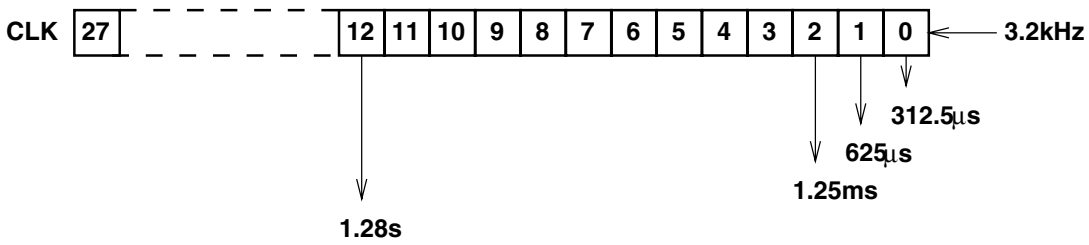


Figure 47—Bluetooth clock

In the different modes and states a Bluetooth unit can reside in, the clock has different appearances:

— CLKN native clock
— CLKE estimated clock
— CLK master clock

CLKN is the free-running native clock and is the reference to all other clock appearances. In states with high activity, the native clock is driven by the reference (e.g., crystal) oscillator with worst case accuracy of ±20 ppm. In the low power states, like STANDBY, HOLD, PARK and SNIFF, the native clock may be driven by a low power oscillator (LPO) with relaxed accuracy (±250 ppm).

CLKE and CLK are derived from the reference CLKN by adding an offset. CLKE is a clock estimate a paging unit makes of the native clock of the recipient; i.e. an offset is added to the CLKN of the pager to approximate the CLKN of the recipient (see Figure 48). By using the CLKN of the recipient, the pager speeds up the connection establishment.

CLK is the master clock of the piconet. It is used for all timing and scheduling activities in the piconet. All Bluetooth devices use the CLK to schedule their transmission and reception. The CLK is derived from the native clock CLKN by adding an offset (see Figure 49). The offset is zero for the master since CLK is identical to its own native clock CLKN. Each slave adds an appropriate offset to its CLKN such that the CLK corresponds to the CLKN of the master. Although all CLKNs in the Bluetooth devices run at the same nominal rate, mutual drift causes inaccuracies in CLK. Therefore, the offsets in the slaves are regularly updated such that CLK is approximately CLKN of the master.

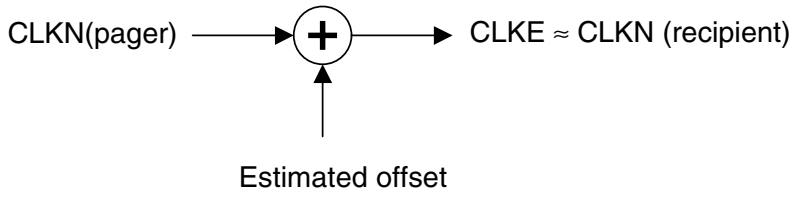


Figure 48—Derivation of CLKE

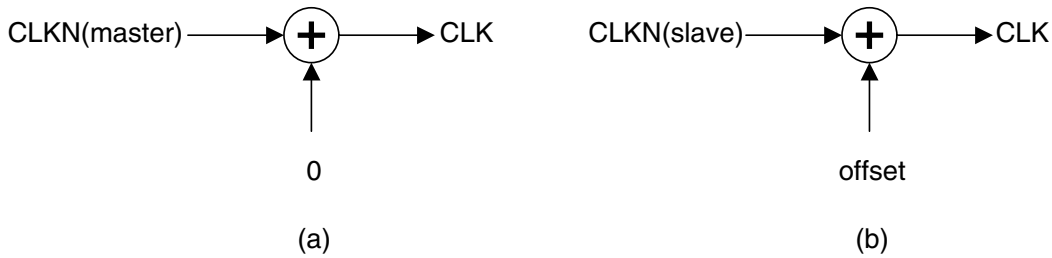


Figure 49—Derivation of CLK in master (a) and in slave (b)

8.10.4 Overview of states

Figure 50 shows a state diagram illustrating the different states used in the Bluetooth link controller. There are two major states: STANDBY and CONNECTION; in addition, there are seven substates, page, page scan, inquiry, inquiry scan, master response, slave response, and inquiry response. The substates are interim states that are used to add new slaves to a piconet. To move from one state to the other, either commands from the Bluetooth link manager are used, or internal signals in the link controller are used (such as the trigger signal from the correlator and the timeout signals).

.

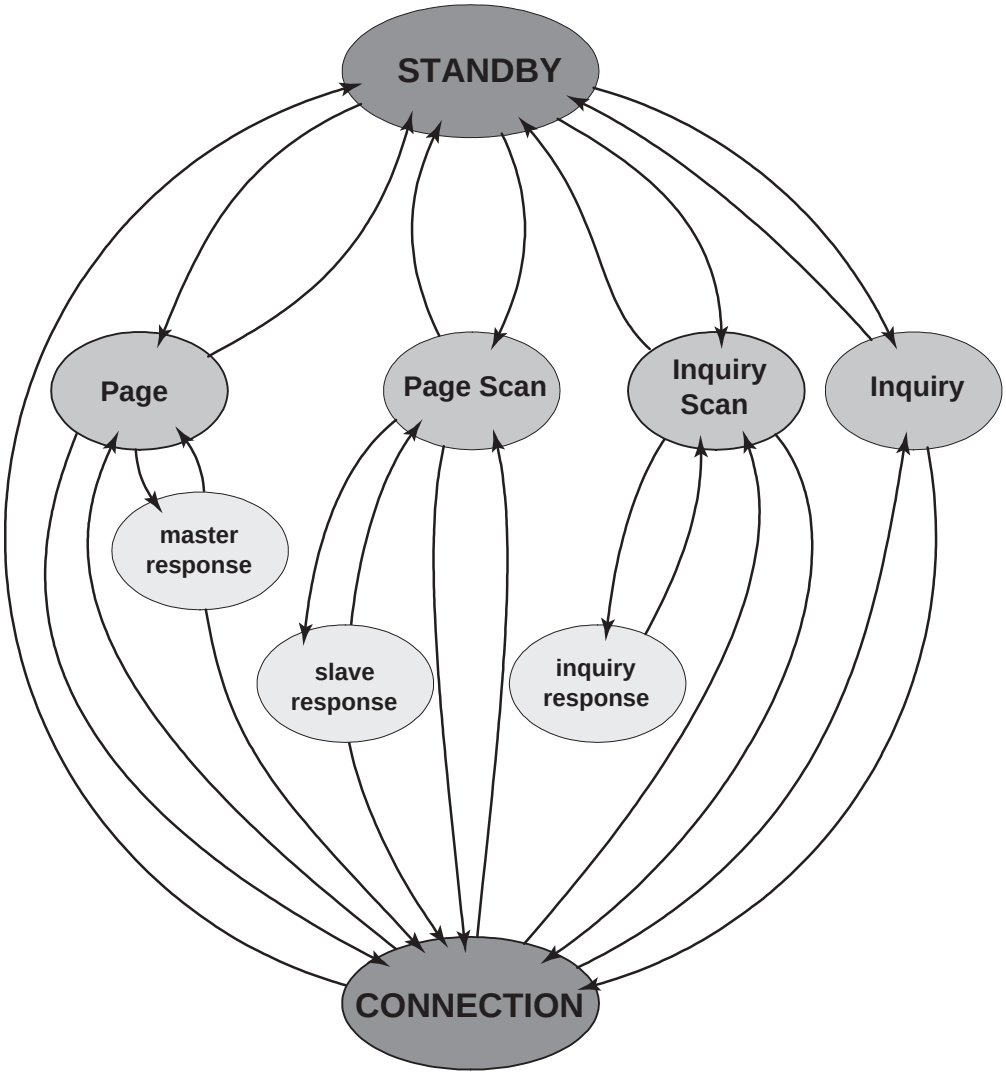


Figure 50—State diagram of Bluetooth link controller

8.10.5 Standby state

The STANDBY state is the default state in the Bluetooth unit. In this state, the Bluetooth unit is in a low-power mode. Only the native clock is running at the accuracy of the LPO (or better).

The controller may leave the STANDBY state to scan for page or inquiry messages, or to page or inquiry itself. When responding to a page message, the unit will not return to the STANDBY state but enter the CONNECTION state as a slave. When carrying out a successful page attempt, the unit will enter the CONNECTION state as a master. The intervals with which scan activities can be carried out are discussed in 8.10.6.2 and 8.10.7.2.

8.10.6 Access procedures

8.10.6.1 General

In order to establish new connections the procedures inquiry and paging are used. The inquiry procedure enables a unit to discover which units are in range, and what their device addresses and clocks are. With the paging procedure, an actual connection can be established. Only the Bluetooth device address is required to set up a connection. Knowledge about the clock will accelerate the setup procedure. A unit that establishes a connection will carry out a page procedure and will automatically be the master of the connection.

In the paging and inquiry procedures, the device access code (DAC) and the inquiry access code (IAC) are used, respectively. A unit in the page scan or inquiry scan substate correlates against these respective access codes with a matching correlator.

For the paging process, several paging schemes can be applied. There is one mandatory paging scheme which has to be supported by each Bluetooth device. This mandatory scheme is used when units meet for the first time, and in case the paging process directly follows the inquiry process. Two units, once connected using a mandatory paging/scanning scheme, may agree on an optional paging/scanning scheme. Optional paging schemes are discussed in Annex D. In the current chapter, only the mandatory paging scheme is considered.

8.10.6.2 Page scan

In the page scan substate, a unit listens for its own device access code for the duration of the scan window $T_{w\ page\ scan}$. (11.2.7.20) During the scan window, the unit listens at a single hop frequency, its correlator matched to its device access code. The scan window shall be long enough to completely scan 16 page frequencies.

When a unit enters the page scan substate, it selects the scan frequency according to the page hopping sequence corresponding to this unit (see 8.11.3.1). This is a 32-hop sequence (or a 16-hop sequence in case of a reduced-hop system) in which each hop frequency is unique. The page hopping sequence is determined by the unit's Bluetooth device address (BD_ADDR). The phase in the sequence is determined by $CLKN_{16\text{-}12}$ of the unit's native clock ($CLKN_{15\text{-}12}$ in case of a reduced-hop system); that is, every 1.28s a different frequency is selected.

If the correlator exceeds the trigger threshold during the page scan, the unit will enter the slave response substate, which is described in 8.10.6.4.1.

The page scan substate can be entered from the STANDBY state or the CONNECTION state. In the STANDBY state, no connection has been established and the unit can use all the capacity to carry out the page scan. Before entering the page scan substate from the CONNECTION state, the unit preferably reserves as much capacity for scanning. If desired, the unit may place ACL connections in the HOLD mode or even use the PARK mode, see 8.10.8.3 and 8.10.8.4. SCO connections are preferably not interrupted by the page scan. In this case, the page scan may be interrupted by the reserved SCO slots which have higher priority than the page scan. SCO packets should be used requiring the least amount of capacity (HV3 packets). The scan window shall be increased to minimize the setup delay. If one SCO link is present using HV3 packets and T_{SCO}=6 slots, a total scan window $T_{w\ page\ scan}$ of at least 36 slots (22.5ms) is recommended; if two SCO links are present using HV3 packets and T_{SCO}=6 slots, a total scan window of at least 54 slots (33.75ms) is recommended.

The scan interval $T_{page\ scan}$ is defined as the interval between the beginnings of two consecutive page scans. A distinction is made between the case where the scan interval is equal to the scan window $T_{w\ page\ scan}$ (continuous scan), the scan interval is maximal 1.28s, or the scan interval is maximal 2.56s. These three cases determine the behavior of the paging unit; that is, whether the paging unit shall use R0, R1 or R2

(see also 8.10.6.3). Table 23 illustrates the relationship between $T_{page\ scan}$ and modes R0, R1 and R2. Although scanning in the R0 mode is continuous, the scanning may be interrupted by, for example, reserved SCO slots. The scan interval information is included in the SR field in the FHS packet.

During page scan the Bluetooth unit may choose to use an optional scanning scheme. (An exception is the page scan after returning an inquiry response message. See 8.10.7.4 for details.)

Table 23—Relationship between scan interval, train repetition, and paging modes R0, R1, and R2

SR mode	$T_{page\ scan}$	N_{page}
R0	continuous	≥1
R1	≤ 1.28s	≥ 128
R2	≤ 2.56s	≥ 256
Reserved	—	—

8.10.6.3 Page

The page substate is used by the master (source) to activate and connect to a slave (destination) which periodically wakes up in the page scan substate. The master tries to capture the slave by repeatedly transmitting the slave's device access code (DAC) in different hop channels. Since the Bluetooth clocks of the master and the slave are not synchronized, the master does not know exactly when the slave wakes up and on which hop frequency. Therefore, it transmits a train of identical DACs at different hop frequencies, and listens in between the transmit intervals until it receives a response from the slave.

The page procedure in the master consists of a number of steps. First, the slave's device address is used to determine the page hopping sequence (see 8.11.3.2). This is the sequence the master will use to reach the slave. For the phase in the sequence, the master uses an estimate of the slave's clock. This estimate can for example be derived from timing information that was exchanged during the last encounter with this particular device (which could have acted as a master at that time), or from an inquiry procedure. With this estimate CLKE of the slave's Bluetooth clock, the master can predict on which hop channel the slave will start page scan.

The estimate of the Bluetooth clock in the slave can be completely wrong. Although the master and the slave use the same hopping sequence, they use different phases in the sequence and will never meet each other. To compensate for the clock drifts, the master will send its page message during a short time interval on a number of wake-up frequencies. It will also transmit on hop frequencies just before and after the current, predicted hop frequency. During each TX slot, the master sequentially transmits on two different hop frequencies. (Since the page message is the ID packet which is only 68 bits in length, there is ample time (224.5 μs minimal) to switch the synthesizer.) In the following RX slot, the receiver will listen sequentially to two corresponding RX hops for ID packet. The RX hops are selected according to the page_response hopping sequence. The page_response hopping sequence is strictly related to the page hopping sequence; that is: for each page hop there is a corresponding page_response hop. The RX/TX timing in the page substate has been described in 8.9, see also Figure 43. In the next TX slot, it will transmit on two hop frequencies different from the former ones. The synthesizer hop rate is increased to 3200 hops/s.

A distinction must be made between the 79-hop systems and the 23-hop systems. First the 79-hop systems are considered. With the increased hopping rate as described above, the transmitter can cover 16 different hop frequencies in 16 slots or 10 ms. The page hopping sequence is divided over two paging trains A and B

of 16 frequencies. Train A includes the 16 hop frequencies surrounding the current, predicted hop frequency $f(k)$, where k is determined by the clock estimate $CLKE_{16\text{-}12}$. So the first train consists of hops $f(k - 8)$, $f(k - 7)$, ..., $f(k)$, ..., $f(k + 7)$.

When the difference between the Bluetooth clocks of the master and the slave is between -8×1.28 s and $+7 \times 1.28$ s, one of the frequencies used by the master will be the hop frequency the slave will listen to. However, since the master does not know when the slave will enter the page scan substate, it has to repeat this train **A** N_{page} times or until a response is obtained. If the slave scan interval corresponds to R1, the repetition number is at least 128; if the slave scan interval corresponds to R2, the repetition number is at least 256.

Note that $CLKE_{16\text{-}12}$ changes every 1.28 s; therefore, every 1.28 s, the trains will include different frequencies of the page hopping set.

When the difference between the Bluetooth clocks of the master and the slave is less than -8×1.28 s or larger than $+7 \times 1.28$ s, more distant hops should be probed. Since in total, there are only 32 dedicated wake-up hops, the more distant hops are the remaining hops not being probed yet. The remaining 16 hops are used to form the new 10 ms train B. The second train consists of hops

$f(k - 16)$, $f(k - 15)$, ..., $f(k - 9)$, $f(k + 8)$, ..., $f(k + 15)$

Train B is repeated for N_{page} times. If still no response is obtained, the first train A is tried again N_{page} times. Alternate use of train A and train B is continued until a response is received or the timeout *pageTO* is exceeded. If during one of the listening occasions, a response is returned by the slave, the master unit enters the master response substate.

The description for paging and page scan procedures given here has been tailored towards the 79-hop systems used in the US and Europe. For the 23-hop systems as used in France, the procedure is slightly different. In the 23-hop case, the length of the page hopping sequence is reduced to 16. As a consequence, there is only a single train (train A) including all the page hopping frequencies. The phase to the page hopping sequence is not $CLKE_{16\text{-}12}$ but $CLKE_{15\text{-}12}$. An estimate of the slave's clock does not have to be made.

The page substate can be entered from the STANDBY state or the CONNECTION state. In the STANDBY state, no connection has been established and the unit can use all the capacity to carry out the page. Before entering the page substate from the CONNECTION state, the unit shall free as much capacity as possible for scanning. To ensure this, it is recommended that the ACL connections are put on hold or park. However, the SCO connections shall not be disturbed by the page. This means that the page will be interrupted by the reserved SCO slots which have higher priority than the page. In order to obtain as much capacity for paging, it is recommended to use the SCO packets which use the least amount of capacity (HV3 packets). If SCO links are present, the repetition number N_{page} of a single train shall be increased, see Table 24. Here it has been assumed that the HV3 packets are used with an interval $T_{SCO} = 6$ slots, which would correspond to a 64 kb/s voice link.

Table 24—Relationship between train repetition and paging modes R0, R1, and R2 when SCO links are present

SR mode	No SCO link	One SCO link (HV3)	Two SCO links (HV3)
R0	$N_{page} \geq 1$	$N_{page} \geq 2$	$N_{page} \geq 3$
R1	$N_{page} \geq 128$	$N_{page} \geq 256$	$N_{page} \geq 384$
R2	$N_{page} \geq 256$	$N_{page} \geq 512$	$N_{page} \geq 768$

The construction of the page train is independent of the presence of SCO links; that is, SCO packets are sent on the reserved slots but do not affect the hop frequencies used in the unreserved slots (see Figure 51).

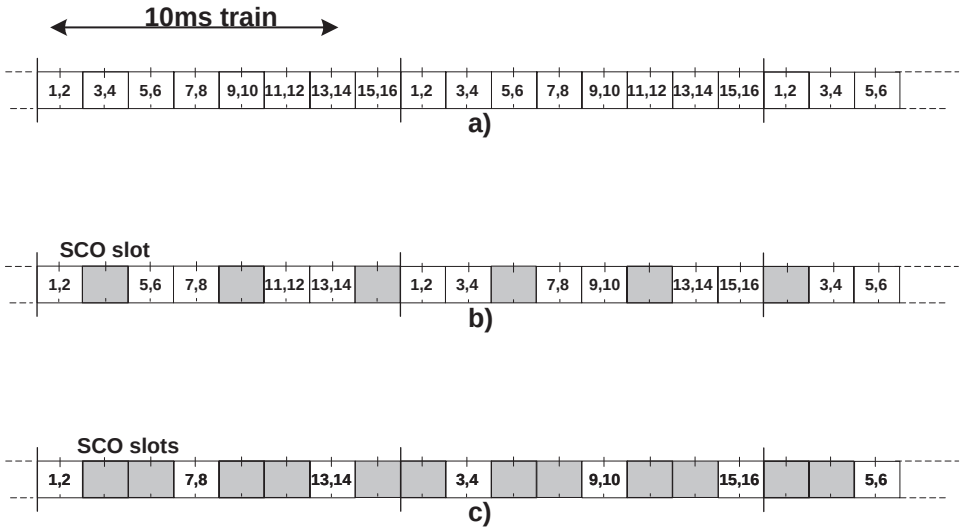


Figure 51—Conventional page (a), page while one SCO link present (b), page while two SCO links present (c)

For the descriptions of optional paging schemes see Annex D.

8.10.6.4 Page response procedures

When a page message is successfully received by the slave, there is a coarse FH synchronization between the master and the slave. Both the master and the slave enter a response routine to exchange vital information to continue the connection setup. Important for the piconet connection is that both Bluetooth units use the same channel access code, use the same channel hopping sequence, and that their clocks are synchronized. These parameters are derived from the master unit. The unit that initializes the connection (starts paging) is defined as the master unit (which is thus only valid during the time the piconet exists). The channel access code and channel hopping sequence are derived from the Bluetooth device address (BD_ADDR) of the master. The timing is determined by the master clock. An offset is added to the slave's native clock to temporarily synchronize the slave clock to the master clock. At start-up, the master parameters have to be transmitted from the master to the slave. The messaging between the master and the slave at start-up will be considered in this section.

The initial messaging between master and slave is shown in Table 25 and in Figure 52 and Figure 53. In those two figures frequencies $f(k)$, $f(k + 1)$, etc. are the frequencies of the page hopping sequence determined by the slave's BD_ADDR. The frequencies $f'(k)$, $f'(k + 1)$, etc. are the corresponding page_response frequencies (slave-to-master). The frequencies $g(m)$ belong to the channel hopping sequence.

In step 1 (see Table 25), the master unit is in page substate and the slave unit in the page scan substate. Assume in this step that the page message (in this case the slave's device access code) sent by the master is received by the slave. On recognizing its device access code, the slave enters the slave response in step 2. The master waits for a reply from the slave and when this arrives in step 2, it will enter the master response in step 3. Note that during the initial message exchange, all parameters are derived from the slave's BD_ADDR, and that only the page hopping and page_response hopping sequences are used (which are also derived from the slave's BD_ADDR). Note that when the master and slave enter the response states, their clock input to the page and page_response hop selection is frozen as is described in 8.11.3.3.

Table 25—Initial messaging during startup

Step	Message	Direction	Hopping sequence	Access code and clock
1	slave ID	master to slave	page	slave
2	slave ID	slave to master	page response	slave
3	FHS	master to slave	page	slave
4	slave ID	slave to master	page response	slave
5	1st packet master	master to slave	channel	master
6	1st packet slave	slave to master	channel	master

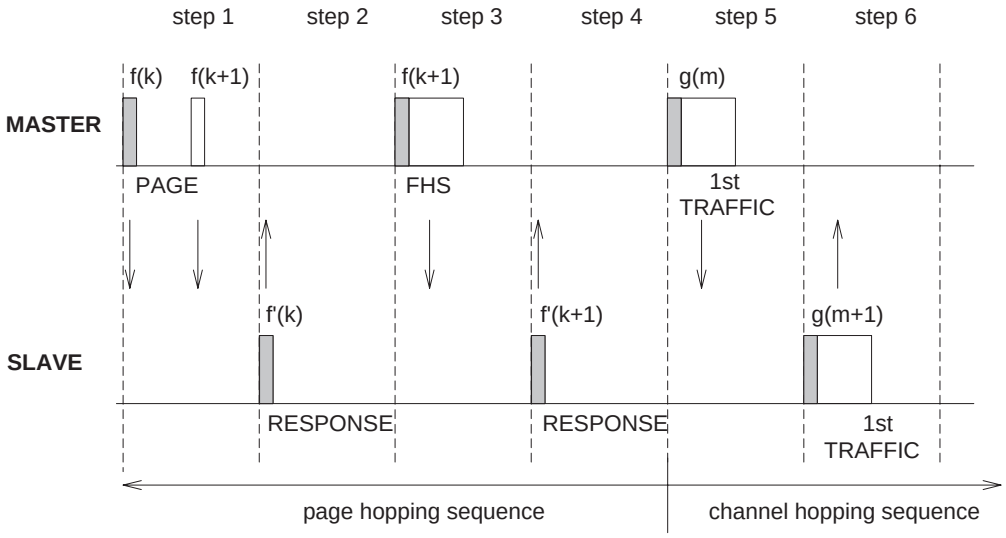


Figure 52—Messaging at initial connection when slave responds to first page message

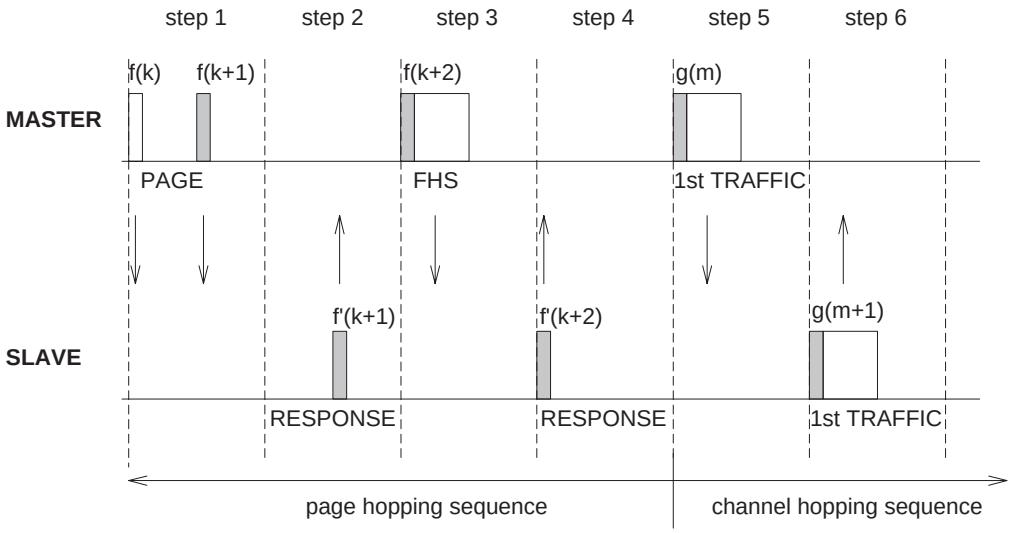


Figure 53—Messaging at initial connection when slave responds to second page message

8.10.6.4.1 Slave response

After having received its own device access code in step 1, the slave unit transmits a response message in step 2. This response message again only consists of the slave's device access code. The slave will transmit this response 625 µs after the beginning of the received page message (slave ID packet) and at the response hop frequency that corresponds to the hop frequency in which the page message was received. The slave transmission is therefore time aligned to the master transmission. During initial messaging, the slave still uses the page response hopping sequence to return information to the master. The clock input $CLKN_{16\text{-}12}$ is frozen at the value it had at the time the page message was received.

After having sent the response message, the slave's receiver is activated (312.5 µs after the start of the response message) and awaits the arrival of an FHS packet. Note that an FHS packet can arrive 312.5 µs after the arrival of the page message as shown in Figure 53, and not after 625 µs as is usually the case in the RX/TX timing. More details about the timing can be found in 8.9.6.

If the setup fails before the CONNECTION state has been reached, the following procedure is carried out. The slave will keep listening as long as no FHS packet is received until *pagerespTO* is exceeded. Every 1.25 ms, however, it will select the next master-to-slave hop frequency according to the page hop sequence. If nothing is received after *pagerespTO*, the slave returns back to the page scan substate for one scan period. Length of the scan period depends on the SCO slots present. If no page message is received during this additional scan period, the slave will resume scanning at its regular scan interval and return to the state it was in prior to the first page scan state.

If an FHS packet is received by the slave in the slave response substate, the slave returns a response (slave's device access code only) in step 4 to acknowledge the reception of the FHS packet (still using the page response hopping sequence). The transmission of this response packet is based on the reception of the FHS packet. Then the slave changes to the channel (master's) access code and clock as received from the FHS packet. Only the 26 MSBs of the master clock are transferred: the timing is assumed such that CLK_1 and CLK_0 are both zero at the time the FHS packet was received as the master transmits in even slots only. From the master clock in the FHS packet, the offset between the master's clock and the slave's clock is determined and reported to the slave's link manager.

Finally, the slave enters the CONNECTION state in step 5. From then on, the slave will use the master's clock and the master BD_ADDR to determine the channel hopping sequence and the channel access code. The connection mode starts with a POLL packet transmitted by the master. The slave responds with any type of packet. If the POLL packet is not received by the slave, or the response packet is not received by the master, within *newconnectionTO* number of slots after FHS packet acknowledgement, the master and the slave will return to page and page scan substates, respectively (see 8.10.8).

8.10.6.4.2 Master response

When the master has received a response message from the slave in step 2, it will enter the master response routine. It freezes the current clock input to the page hop selection scheme. Then the master will transmit an FHS packet in step 3 containing the master's real-time Bluetooth clock, the master's 48-bit BD_ADDR address, the BCH parity bits, and the class of device. The FHS packet contains all information to construct the channel access code without requiring a mathematical derivation from the master device address. The FHS packet is transmitted at the beginning of the master-to-slave slot following the slot in which the slave has responded. So the TX timing of the FHS is not based on the reception of the response packet from the slave. The FHS packet may therefore be sent 312.5 μs after the reception of the response packet like shown in Figure 53 and not 625 μs after the received packet as is usual in the RX/TX timing (see also 8.9.6).

After the master has sent its FHS packet, it waits for a second response from the slave in step 4 which acknowledges the reception of the FHS packet. Again this is only the slave's device access code. If no response is received, the master retransmits the FHS packet, but with an updated clock and still using the slave's parameters. It will retransmit (the clock is updated every retransmission) until a second slave response is received, or the timeout of *pagerespTO* is exceeded. In the latter case, the master turns back to the page substate and sends an error message to the link manager. During the retransmissions of the FHS packet, the master keeps using the page hopping sequence.

If the slave's response is indeed received, the master changes to the master parameters for the channel access code and the master clock. The lower clock bits CLK_0 and CLK_1 are zero at the start of the FHS packet transmission and are not included in the FHS packet. Finally, the master enters the CONNECTION state in step 5. The master BD_ADDR is used to change to a new hopping sequence, the *channel hopping sequence*. The channel hopping sequence uses all 79 hop channels in a (pseudo) random fashion (see also 8.11.3.6).

The master can now send its first traffic packet in a hop determined with the new (master) parameters. This first packet will be a POLL packet (see 8.10.8). This packet will be sent within *newconnectionTO* number of slots after reception of the FHS packet acknowledgement. The slave will respond with any type of packet. If the POLL packet is not received by the slave or the POLL packet response is not received by the master within *newconnectionTO* number of slots, the master and the slave will return to page and page scan substates, respectively.

8.10.7 Inquiry procedures

8.10.7.1 General

In the Bluetooth system, an inquiry procedure is defined which is used in applications where the destination's device address is unknown to the source. One can think of public facilities like printers or facsimile machines, or access points to a LAN. Alternatively, the inquiry procedure can be used to discover which other Bluetooth units are within range. During an inquiry substate, the discovering unit collects the Bluetooth device addresses and clocks of all units that respond to the inquiry message. It can then, if desired, make a connection to any one of them by means of the previously described page procedure.

The inquiry message broadcast by the source does not contain any information about the source. However, it may indicate which class of devices should respond. There is one general inquiry access code (GIAC) to inquire for any Bluetooth device, and a number of dedicated inquiry access codes (DIAC) that only inquire

for a certain type of devices. The inquiry access codes are derived from reserved Bluetooth device addresses and are further described in 8.4.2.1.

A unit that wants to discover other Bluetooth units enters an inquiry substate. In this substate, it continuously transmits the inquiry message (which is the ID packet, see 8.4.4.1.1) at different hop frequencies. The inquiry hop sequence is always derived from the LAP of the GIAC. Thus, even when DIACs are used, the applied hopping sequence is generated from the GIAC LAP. A unit that allows itself to be discovered, regularly enters the inquiry scan substate to respond to inquiry messages. The following subclauses (8.10.7.2 through 8.10.7.4) describe the message exchange and contention resolution during inquiry response. The inquiry response is optional: a unit is not forced to respond to an inquiry message.

8.10.7.2 Inquiry scan

The inquiry scan substate is very similar to the page scan substate. However, instead of scanning for the unit's device access code, the receiver scans for the inquiry access code long enough to completely scan for 16 inquiry frequencies. The length of this scan period is denoted $T_{w\_inquiry\_scan}$. The scan is performed at a single hop frequency. As in the page procedure, the inquiry procedure uses 32 dedicated inquiry hop frequencies according to the *inquiry hopping sequence*. These frequencies are determined by the general inquiry address. The phase is determined by the native clock of the unit carrying out the inquiry scan; the phase changes every 1.28 s.

Instead or in addition to the general inquiry access code, the unit may scan for one or more dedicated inquiry access codes. However, the scanning will follow the inquiry scan hopping sequence which is determined by the general inquiry address. If an inquiry message is recognized during an inquiry wake-up period, the Bluetooth unit either performs a backoff in CONNECTION or STANDBY state before reentering the inquiry scan substate or enters the inquiry response substate if a random backoff was performed before entering the inquiry scan substate.

The inquiry scan substate can be entered from the STANDBY state or the CONNECTION state. In the STANDBY state, no connection has been established and the unit can use all the capacity to carry out the inquiry scan. Before entering the inquiry scan substate from the CONNECTION state, the unit preferably reserves as much capacity as possible for scanning. If desired, the unit may place ACL connections in the HOLD mode or even use the PARK mode (see 8.10.8.3). SCO connections are preferably not interrupted by the inquiry scan. In this case, the inquiry scan may be interrupted by the reserved SCO slots which have higher priority than the inquiry scan. SCO packets should be used requiring the least amount of capacity (HV3 packets). The scan window, $T_{w\ inquiry\ scan}$, shall be increased to increase the probability to respond to an inquiry message. If one SCO link is present using HV3 packets and $T_{SCO} = 6$ slots, a total scan window of at least 36 slots (22.5 ms) is recommended; if two SCO links are present using HV3 packets and $T_{SCO} = 6$ slots, a total scan window of at least 54 slots (33.75 ms) is recommended.

The scan interval $T_{inquiry\ scan}$ is defined as the interval between two consecutive inquiry scans. The inquiry scan interval shall be at most 2.56 s.

8.10.7.3 Inquiry

The inquiry substate is used by the unit that wants to discover new devices. This substate is very similar to the page substate, the same TX/RX timing is used as used for paging (see 8.9.6 and Figure 43). The TX and RX frequencies follow the inquiry hopping sequence and the inquiry response hopping sequence, and are determined by the general inquiry access code and the native clock of the discovering device. In between inquiry transmissions, the Bluetooth receiver scans for inquiry response messages. When found, the entire response packet (which is in fact a FHS packet) is read, after which the unit continues with the inquiry transmissions. So the Bluetooth unit in an inquiry substate does not acknowledge the inquiry response messages. It keeps probing at different hop channels and in between listens for response packets. Like in the page substate, two 10 ms trains A and B are defined, splitting the 32 frequencies of the inquiry hopping

sequence into two 16-hop parts. A single train shall be repeated for at least $N_{inquiry}$ = 256 times before a new train is used. In order to collect all responses in an error-free environment, at least three train switches should have taken place. As a result, the inquiry substate may have to last for 10.24 s unless the inquirer collects enough responses and determines to abort the inquiry substate earlier. If desired, the inquirer can also prolong the inquiry substate to increase the probability of receiving all responses in an error-prone environment. If an inquiry procedure is automatically initiated periodically (say a 10 s period every minute), then the interval between two inquiry instances must be determined randomly. This is done to avoid two Bluetooth units to synchronize their inquiry procedures.

The inquiry substate is continued until stopped by the Bluetooth link manager (when it decides that it has sufficient number of responses), or when a timeout has been reached (*inquiryTO*).

The inquiry substate can be entered from the STANDBY state or the CONNECTION state. In the STANDBY state, no connection has been established and the unit can use all the capacity to carry out the inquiry. Before entering the inquiry substate from the CONNECTION state, the unit shall free as much capacity as possible for scanning. To ensure this, it is recommended that the ACL connections are put on hold or park. However, the SCO connections shall not be disturbed by the inquiry. This means that the inquiry will be interrupted by the reserved SCO slots which have higher priority than the inquiry. In order to obtain as much capacity for inquiry, it is recommended to use the SCO packets which use the least amount of capacity (HV3 packets). If SCO links are present, the repetition number $N_{inquiry}$ shall be increased, see Table 26. Here it has been assumed that the HV3 packet is used with an interval T_{SCO}=6 slots, which would correspond to a 64 kb/s voice link.

Table 26—Increase of train repetition when SCO links are present

	No SCO link	One SCO link (HV3)	Two SCO links (HV3)
$N_{inquiry}$	≥ 256	≥ 512	≥ 768

8.10.7.4 Inquiry response

For the inquiry operation, there is only a slave response, no master response. The master listens between inquiry messages for responses, but after reading a response, it continues to transmit inquiry messages. The slave response routine for inquiries differs completely from the slave response routine applied for pages. When the inquiry message is received in the inquiry scan substate, a response message containing the recipient's address shall be returned. This response message is a conventional FHS packet carrying the unit's parameters. However, a contention problem may arise when several Bluetooth units are in close proximity to the inquiring unit and all respond to an inquiry message at the same time. First of all, every Bluetooth unit has a free running clock; therefore, it is highly unlikely that they all use the same phase of the inquiry hopping sequence. However, in order to avoid collisions between units that do wake up in the same inquiry hop channel simultaneously, the following protocol in the slave's inquiry response is used. If the slave receives an inquiry message, it generates a random number RAND between 0 and 1023. The slave then returns to the CONNECTION or STANDBY state for the duration of RAND time slots. Before returning to the CONNECTION or STANDBY state, the unit may go through the page scan substate; this page scan shall use the mandatory page scan scheme. After at least RAND slots, the unit will return to the inquiry scan substate. On the first inquiry message received in this substate the slave goes into the inquiry response substate and returns an FHS response packet to the master 625 μs after the inquiry message was received. If during the scan no trigger occurs within a timeout period of *inqrespTO*, the slave returns to the STANDBY or CONNECTION state. If the unit does receive an inquiry message and returns an FHS packet, it adds an offset of 1 to the phase in the inquiry hop sequence (the phase has a 1.28 s resolution) and enters the inquiry scan substate again. If the slave is triggered again, it repeats the procedure using a new RAND. The offset to

the clock accumulates each time a FHS packet is returned. During a 1.28 s probing window, a slave on average responds four times, but on different frequencies and at different times. Possible SCO slots should have priority over response packets; that is, if a response packet overlaps with an SCO slot, it is not sent but the next inquiry message is awaited.

The messaging during the inquiry routines is summarized in Table 27. In step 1, the master transmits an inquiry message using the inquiry access code and its own clock. The slave responds with the FHS packet which contains the slave's device address, native clock, and other slave information. This FHS packet is returned at a semi-random time. The FHS packet is not acknowledged in the inquiry routine, but it is retransmitted at other times and frequencies as long as the master is probing with inquiry messages.

Table 27—Messaging during inquiry routines

Step	Message	Direction	Hopping sequence	Access code
1	ID	master to slave	inquiry	inquiry
2	FHS	slave to master	inquiry response	inquiry

If the scanning unit uses an optional scanning scheme, after responding to an inquiry with an FHS packet, it will perform page scan using the mandatory page scan scheme for $T_{mandatory\ pscan}$ period. Every time an inquiry response is sent the unit will start a timer with a timeout of $T_{mandatory\ pscan}$. The timer will be reset at each new inquiry response. Until the timer times out, when the unit performs page scan, it will use the mandatory page scanning scheme in the SR mode it uses for all its page scan intervals. Using the mandatory page scan scheme after the inquiry procedure enables all units to connect even if they do not support an optional paging scheme (yet). In addition to using the mandatory page scan scheme, an optional page scan scheme can be used in parallel for the $T_{mandatory\ pscan}$ period.

The $T_{mandatory\ pscan}$ period is included in the SP field of the FHS packet returned in the inquiry response routine (see 8.4.4.1.4). The value of the period is indicated in the Table 28.

Table 28—Mandatory scan periods for P0, P1, and P2 scan period modes

SP mode	$T_{mandatory\ pscan}$
P0	≥20s
P1	≥ 40s
P2	≥ 60s
Reserved	—

8.10.8 Connection state

In the CONNECTION state, the connection has been established and packets can be sent back and forth. In both units, the channel (master) access code and the master Bluetooth clock are used. The hopping scheme uses the *channel hopping sequence*. The master starts its transmission in even slots (CLK_{1-0} = 00), the slave starts its transmission in odd slots (CLK_{1-0} = 10).

The CONNECTION state starts with a POLL packet sent by the master to verify the switch to the master's timing and channel frequency hopping. The slave can respond with any type of packet. If the slave does not receive the POLL packet or the master does not receive the response packet for *newconnectionTO* number of slots, both devices will return to page/page scan substates.

The first information packets in the CONNECTION state contain control messages that characterize the link and give more details regarding the Bluetooth units. These messages are exchanged between the link managers of the units. For example, it defines the SCO links and the sniff parameters. Then the transfer of user information can start by alternately transmitting and receiving packets.

The CONNECTION state is left through a detach or reset command. The detach command is used if the link has been disconnected in the normal way. All configuration data in the Bluetooth link controller is still valid. The reset command is a hard reset of all controller processes. After a reset, the controller has to be reconfigured.

The Bluetooth units can be in several modes of operation during the CONNECTION state: active mode, sniff mode, hold mode, and park mode. These modes are now described in more detail.

8.10.8.1 Active mode

In the active mode, the Bluetooth unit actively participates on the channel. The master schedules the transmission based on traffic demands to and from the different slaves. In addition, it supports regular transmissions to keep slaves synchronized to the channel. Active slaves listen in the master-to-slave slots for packets. If an active slave is not addressed, it may sleep until the next new master transmission. From the type indication in the packet, the number of slots the master has reserved for its transmission can be derived; during this time, the nonaddressed slaves do not have to listen on the master-to-slave slots. A periodic master transmission is required to keep the slaves synchronized to the channel. Since the slaves only need the channel access code to synchronize with, any packet type can be used for this purpose.

8.10.8.2 Sniff mode

In the sniff mode, the duty cycle of the slave's listen activity can be reduced. If a slave participates on an ACL link, it has to listen in every ACL slot to the master traffic. With the sniff mode, the time slots where the master can start transmission to a specific slave is reduced; that is, the master can only start transmission in specified time slots. These so-called sniff slots are spaced regularly with an interval of $T_{\text{sniff.}}$

The slave starts listening at the sniff slots for $N_{\text{sniff attempt}}$ consecutive receive slots unless a packet with matching AM_ADDR is received. After every reception of a packet with matching AM_ADDR, the slave continues listening at the subsequent $N_{\text{sniff timeout}}$ or remaining of the receive slots, whichever is greater. So, for $N_{\text{sniff timeout}} > 0$, the slave continues listening as long as it receives packets with matching AM_ADDR.

Note that Receive slots here are every odd-numbered slots, in which the master may start sending a packet.

Note that $N_{\text{sniff attempt}} = 1$ and $N_{\text{sniff timeout}} = 0$ cause the slave to listen only at the first sniff slot, irrespective of packets received from the master.

Note that $N_{\text{sniff attempt}} = 0$ is not allowed.

To enter the sniff mode, the master or slave shall issue a sniff command via the LM protocol. This message will contain the sniff interval T_{sniff} and an offset $D_{\text{sniff.}}$ The timing of the sniff mode is then determined similar as for the SCO links. In addition, an initialization flag indicates whether initialization procedure 1 or 2 is being used. The device uses initialization 1 when the MSB of the current master clock (CLK_{27}) is 0; it uses initialization 2 when the MSB of the current master clock (CLK_{27}) is 1. The slave shall apply the initialization method as indicated by the initialization flag irrespective of its clock bit value CLK_{27}. The

master-to-slave sniff slots determined by the master and the slave shall be initialized on the slots for which the clock satisfies the following equation

$$CLK_{27-1} \bmod T_{sniff} = D_{sniff} \text{ for initialization 1}$$

$$(\overline{CLK}_{27}, CLK_{26-1}) \bmod T_{sniff} = D_{sniff} \text{ for initialization 2}$$

The slave-to-master sniff slot determined by the master and the slave shall be initialized on the slots after the master-to-slave sniff slot defined above. After initialization, the clock value CLK(k+1) for the next master-to-slave SNIFF slot is found by adding the fixed interval T_{sniff} to the clock value of the current master-to-slave sniff slot:

$$CLK(k+1) = CLK(k) + T_{sniff}$$

8.10.8.3 Hold mode

During the CONNECTION state, the ACL link to a slave can be put in a hold mode. This means that the slave temporarily does not support ACL packets on the channel any more (note: possible SCO links will still be supported). With the hold mode, capacity can be made free to do other things like scanning, paging, inquiring, or attending another piconet. The unit in hold mode can also enter a low-power sleep mode. During the hold mode, the slave unit keeps its active member address (AM_ADDR).

Prior to entering the hold mode, master and slave agree on the time duration the slave remains in the hold mode. A timer is initialized with the *holdTO* value. When the timer is expired, the slave will wake up, synchronize to the traffic on the channel and will wait for further master instructions.

8.10.8.4 Park mode

When a slave does not need to participate on the piconet channel, but still wants to remain synchronized to the channel, it can enter the park mode which is a low-power mode with very little activity in the slave. In the park mode, the slave gives up its active member address AM_ADDR. Instead, it receives two new addresses to be used in the park mode

— PM_ADDR: 8-bit Parked Member Address

— AR_ADDR: 8-bit Access Request Address

The PM_ADDR distinguishes a parked slave from the other parked slaves. This address is used in the master-initiated unpark procedure. In addition to the PM_ADDR, a parked slave can also be unparked by its 48-bit BD_ADDR. The all-zero PM_ADDR is a reserved address: if a parked unit has the all-zero PM_ADDR it can only be unparked by the BD_ADDR. In that case, the PM_ADDR has no meaning. The AR_ADDR is used by the slave in the slave-initiated unpark procedure. All messages sent to the parked slaves have to be carried by broadcast packets (the all-zero AM_ADDR) because of the missing AM_ADDR.

The parked slave wakes up at regular intervals to listen to the channel in order to re-synchronize and to check for broadcast messages. To support the synchronization and channel access of the parked slaves, the master supports a beacon channel described in the next section. The beacon structure is communicated to the slave when it is being parked. When the beacon structure changes, the parked slaves are updated through broadcast messages.

In addition for using it for low power consumption, the park mode is used to connect more than seven slaves to a single master. At any one time, only seven slaves can be active. However, by swapping active and parked slaves out respectively in the piconet, the number of slave virtually connected can be much larger

(255 if the PM_ADDR is used, and even a larger number if the BD_ADDR is used). There is no limitation to the number of slaves that can be parked.

8.10.8.4.1 Beacon channel

To support parked slaves, the master establishes a beacon channel when one or more slaves are parked. The beacon channel consists of one beacon slot or a train of equidistant beacon slots which is transmitted periodically with a constant time interval. The beacon channel is illustrated in Figure 54. A train of N_B ($N_B \geq 1$) beacon slots is defined with an interval of T_B slots. The beacon slots in the train are separated by Δ_B. The start of the first beacon slot is referred to as the beacon instant and serves as the beacon timing reference. The beacon parameters N_B and T_B are chosen such that there are sufficient beacon slots for a parked slave to synchronize to during a certain time window in an error-prone environment.

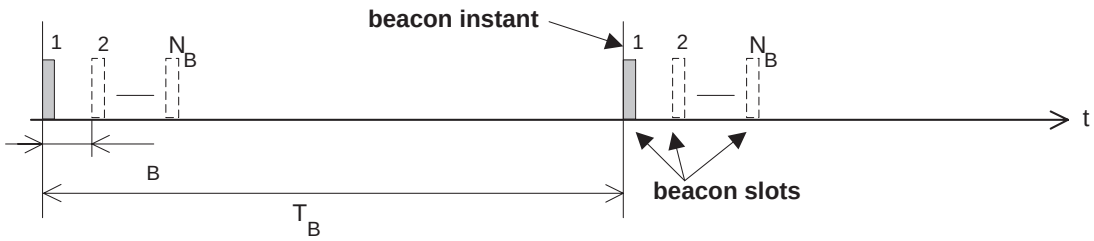


Figure 54—General beacon channel format

When parked, the slave will receive the beacon parameters through an LMP command. In addition, the timing of the beacon instant is indicated through the offset D_B. Like for the SCO link (see 8.3.2), two initialization procedures 1 or 2 are used. The master uses initialization 1 when the MSB of the current master clock (CLK_{27}) is 0; it uses initialization 2 when the MSB of the current master clock (CLK_{27}) is 1. The chosen initialization procedure is also carried by an initialization flag in the LMP command. The slave shall apply the initiations method as indicated by the initialization flag irrespective of its clock bit CLK_{27}. The master-to-slave slot positioned at the beacon instant shall be initialized on the slots for which the clock satisfies the following equation

$$CLK_{27\text{-}1} \bmod T_B = D_B \text{ for initialization 1}$$

$$(\overline{CLK}_{27}, CLK_{26\text{-}1}) \bmod T_B = D_B \text{ for initialization 2}$$

After initialization, the clock value CLK(k+1) for the next beacon instant is found by adding the fixed interval T_B to the clock value of the current beacon instant:

$$CLK(k+1) = CLK(k) + T_B$$

The beacon channel serves four purposes:

1) Transmission of master-to-slave packets which the parked slaves can use for re-synchronization
2) Carrying messages to the parked slaves to change the beacon parameters
3) Carrying general broadcast messages to the parked slaves
4) Unparking of one or more parked slaves

Since a slave can synchronize to any packet which is preceded by the proper channel access code, the packets carried on the beacon slots do not have to contain specific broadcast packets for parked slaves to be

able to synchronize; any packet can be used. The only requirement placed on the beacon slots is that there is master-to-slave transmission present. If there is no information to be sent, NULL packets can be transmitted by the master. If there is indeed broadcast information to be sent to the parked slaves, the first packet of the broadcast message shall be repeated in every beacon slot of the beacon train. However, synchronous traffic like on the SCO link, may interrupt the beacon transmission.

The master communicates with parked slaves using broadcast messages. Since these messages can be time critical, an ongoing repetition train of broadcast message can be prematurely aborted by broadcast information destined to parked slaves in beacon slots and in access windows (see 8.10.8.4.2).

8.10.8.4.2 Beacon access window

In addition to the beacon slots, an access window is defined where the parked slaves can send requests to be unparked. To increase reliability, the access window can be repeated M_{access} times ($M_{access} \geq 1$), see Figure 55. The access window starts a fixed delay D_{access} after the beacon instant. The width of the access window is T_{access}.

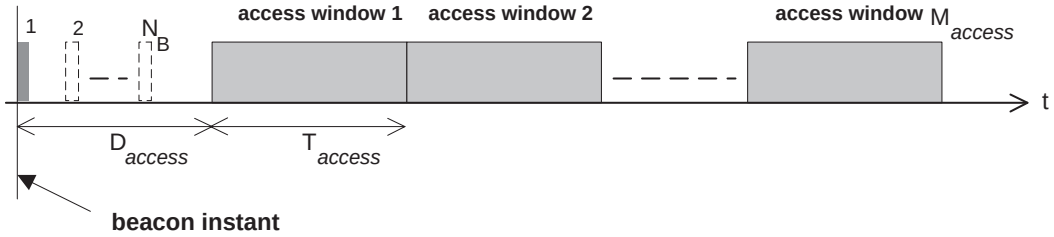


Figure 55—Definition of access window

The access window may support different slave access techniques, like polling, random access, or other forms of access. At this stage, only the polling technique has been defined. The format of the polling technique is shown in Figure 56. The same TDD structure is used as on the piconet channel, i.e. master-to-slave transmission is alternated by slave-to-master transmission. The slave-to-master slot is divided into two half slots of 312.5 µs each. The half slot a parked slave is allowed to respond in corresponds to its access request address (AR_ADDR), see also 8.10.8.4.6. For counting the half slots to determine the access request slot, the start of the access window is used (see Figure 56). The slave is only allowed to send an access request in the proper slave-to-master half slot if in the preceding master-to-slave slot a broadcast packet has been received. In this way, the master polls the parked slaves.

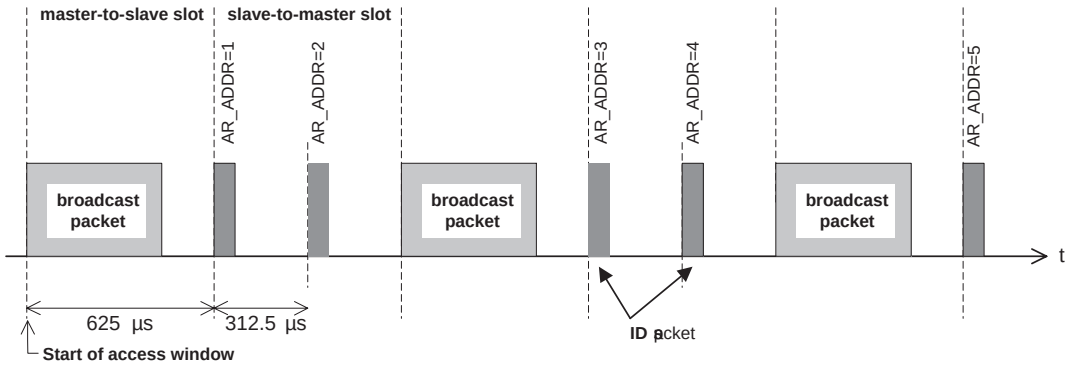


Figure 56—Access procedure applying the polling technique

However, the slots of the access window can also be used for traffic on the piconet if required. For example, if an SCO connection has to be supported, the slots reserved for the SCO link may carry SCO information instead of being used for access requests, i.e. if the master-to-slave slot in the access window contains a packet different from a broadcast packet, the following slave-to-master slot cannot be used for slave access requests. Slots in the access window not affected by traffic can still be used according to the defined access structure; an example is shown in Figure 57. The access procedure is continued as if no interruption had taken place.

When the slave is parked, it is indicated what type of access scheme will be used. For the polling scheme, the number of slave-to-master access slots N_{acc_slot} is indicated.

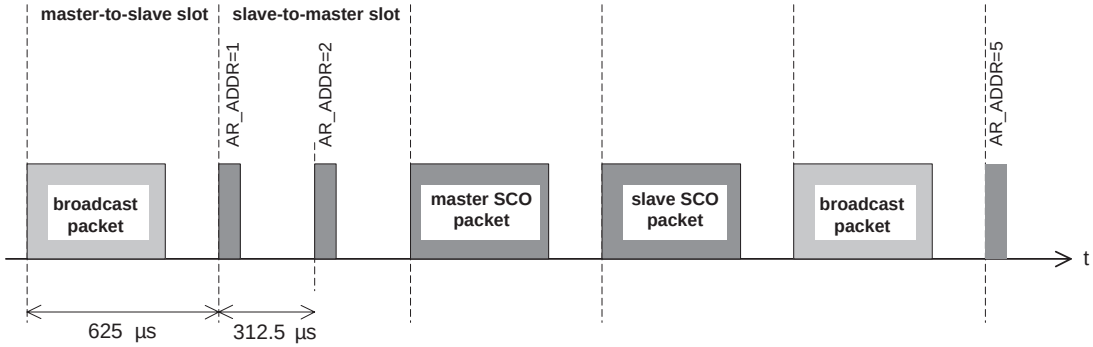


Figure 57—Disturbance of access window by SCO traffic

By default, the access window is always present. However, its activation depends on the master sending broadcast messages to the slave at the appropriate slots in the access window. A flag in a broadcast LMP message within the beacon slots may indicate that the access window(s) belonging to this instant will not be activated. This prevents unnecessary scanning of parked slaves that want to request access.

8.10.8.4.3 Parked slave synchronization

Parked slaves sleep most of the time. However, periodically they wake up to re-synchronize to the channel. Any packet exchanged on the channel can be used for synchronization. Since master transmission is mandatory on the beacon slots, parked slaves will exploit the beacon channel to re-synchronize. A parked slave will wake-up at the beacon instant to read the packet sent on the first beacon slot. If this fails, it will retry on the next beacon slot in the beacon train; in total, there are N_B opportunities per beacon instant to re-synchronize. During the search, the slave may increase its search window (see also 8.9.4). The separation between the beacon slots in the beacon train Δ_B is chosen such that consecutive search windows will not overlap.

The parked slave does not have to wake up at every beacon instant. Instead, a sleep interval can be applied which is longer than the beacon interval T_B (see Figure 58). The slave sleep window shall be a multiple N_{B_sleep} of T_B. The precise beacon instant the slave shall wake up on is indicated by the master with D_{B_sleep} which indicates the offset (in multiples of T_B) with respect to the beacon instant ($0 < D_{B\_sleep} < N_{B\_sleep} - 1$). To initialize the wake-up period, the following equations are used:

$$CLK_{27\text{-}1} \bmod (N_{B\_sleep} \bullet T_B) = D_B + D_{B\_sleep} \bullet T_B \text{ for initialization 1}$$

$$(\overline{CLK}_{27}, CLK_{26\text{-}1}) \bmod (N_{B\_sleep} \bullet T_B) = D_B + D_{B\_sleep} \bullet T_B \text{ for initialization 2}$$

where initialization 1 is chosen by the master if the MSB in the current master clock is 0 and initialization 2 is chosen if the MSB in the current master clock is 1.

When the master wants to send broadcast messages to the parked slaves, it may use the beacon slots for these broadcast messages. However, if $N_B < N_{BC}$, the slots following the last beacon slot in the beacon train shall be used for the remaining $N_{BC} - N_B$ broadcast packets. If $N_B > N_{BC}$, the broadcast message is repeated on all N_B beacon slots.

A parked slave shall at least read the broadcast messages sent in the beacon slot(s) it wakes up in; the minimum wake-up activity is to read the channel access code for re-synchronization and the packet header to check for broadcast messages.

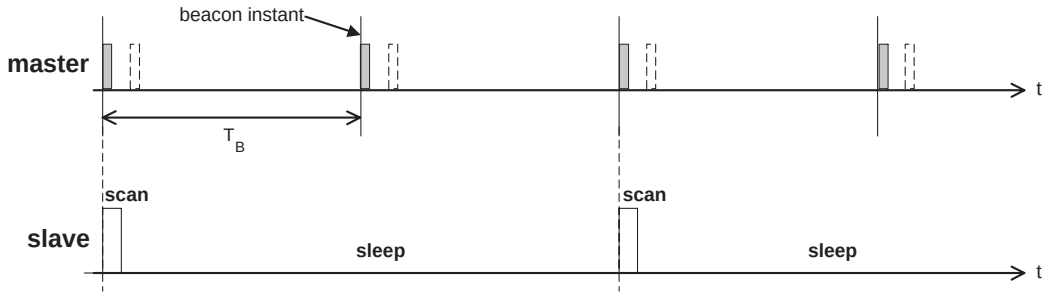


Figure 58—Extended sleep interval of parked slaves

8.10.8.4.4 Parking

A master can park an active slave through the exchange of one or a few LMP commands. Before put into the park mode, the slave is assigned a PM_ADDR and an AR_ADDR. Every parked slave has a unique PM_ADDR; however, the AR_ADDR is not necessarily unique. Also, the beacon parameters are given by the master when the slave is parked. The slave then gives up its AM_ADDR and enters the park mode. A master can park only a single slave at a time. The park message is carried with a normal data packet and addresses the slave through its AM_ADDR.

8.10.8.4.5 Master-activated unparking

The master can unpark a parked slave by sending a dedicated LMP unpark command including the parked slave's address. This message is sent in a broadcast packet on the beacon slots. Either the slave's PM_ADDR is used, or its full BD_ADDR is used. The message also includes the active member address AM_ADDR the slave will use after it has re-entered the piconet. The unpark message can include a number of slave addresses so that multiple slaves can be unparked simultaneously. For each slave, a different AM_ADDR is assigned.

After having received the unpark message, the parked slave matching the PM_ADDR or BD_ADDR will leave the park mode and enter the active mode. It will keep listening to the master until it is addressed by the master through its AM_ADDR. The first packet sent by the master shall be a POLL packet. The return packet in response to the POLL packet confirms that the slave has been unparked. If no response packets from the slave is received for *newconnectionTO* number of slots after the end of beacon repetition period, the master will unpark the slave again. If the slave does not receive the POLL packet for *newconnectionTO* number of slots after the end of beacon repetition period, it will return to park, with the same beacon parameters. After confirming that the slave is active, the master decides in which mode the slave will continue.

8.10.8.4.6 Slave-activated unparking

A slave can request access to the channel through the access window defined in 8.10.8.4.2. As shown in Figure 56, the access window includes several slave-to-master half slots where the slave can send an access request message. The specific half slot the slave is allowed to respond in, corresponds to its access request address (AR_ADDR) which it has received when it was parked. The order of the half slots (in Figure 56 the AR_ADDR numbers linearly increase from 1 to 5) is not fixed: an LMP command sent in the beacon slots may reconfigure the access window. When a slave desires access to the channel, it sends an access request message in the proper slave-to-master half slot.The access request message of the slave is the ID packet containing the device access code (DAC) of the master (which is in this case the channel access code without the trailer). The parked slave is only allowed to transmit an access request message in the half slot when in the preceding master-to-slave slot, a broadcast packet has been received. This broadcast message can contain any kind of broadcast information not necessarily related to the parked slave(s). If no broadcast information is available, a broadcast NULL or broadcast POLL packet shall be sent.

After having sent an access request, the parked slave will listen for an unpark message from the master. As long as no unpark message is received, the slave will repeat the access requests in the subsequent access windows. After the last access window (there are M_{access} windows in total, see 8.10.8.4.2), the parked slave shall listen for an additional N_{poll} time slots for an unpark message. If no unpark message is received within N_{poll} slots after the end of the last access window, the slave may return to sleep and retry an access attempt after the next beacon instant.

After having received the unpark message, the parked slave matching the PM_ADDR or BD_ADDR will leave the park mode and enter the active mode. It will keep listening to the master until it is addressed by the master through its AM_ADDR. The first packet sent by the master shall be a POLL packet. The return packet in response to the POLL packet confirms that the slave has been unparked. If no response packet from the slave is received for *newconnectionTO* number of slots after N_{poll} slots after the end of the last access window, the master will send the unpark message to the slave again. If the slave does not receive the POLL packet for *newconnectionTO* number of slots after N_{poll} slots after the end of the last access window, it will return to park, with the same beacon parameters. After confirming that the slave is active, the master decides in which mode the slave will continue.

8.10.8.4.7 Broadcast scan window

In the beacon train, the master can support broadcast messages to the parked slaves. However, it may extend its broadcast capacity by indicating to the parked slaves that more broadcast information is following after the beacon train. This is achieved by a special LMP command ordering the parked slaves (as well as the active slaves) to listen to the channel for broadcast messages during a limited time window. This time window starts at the beacon instant and continues for the period as indicated in the LMP command sent in the beacon train.

8.10.8.5 Polling schemes

8.10.8.5.1 Polling in active mode

The master always has full control over the piconet. Due to the stringent TDD scheme, slaves can only communicate with the master and not to other slaves. In order to avoid collisions on the ACL link, a slave is only allowed to transmit in the slave-to-master slot when addressed by the AM_ADDR in the packet header in the preceding master-to-slave slot. If the AM_ADDR in the preceding slot does not match, or an AM_ADDR cannot be derived from the preceding slot, the slave is not allowed to transmit.

On the SCO links, the polling rule is slightly modified. The slave is allowed to transmit in the slot reserved for its SCO link unless the (valid) AM_ADDR in the preceding slot indicates a different slave. If no valid

AM_ADDR can be derived in the preceding slot, the slave is still allowed to transmit in the reserved SCO slot.

8.10.8.5.2 Polling in park mode

In the park mode, parked slaves are allowed to send access requests in the access window provided a broadcast packet is received in the preceding master-to-slave slot. Slaves in active mode will not send in the slave-to-master slots following the broadcast packet since they are only allowed to send if addressed specifically.

8.10.8.6 Slot reservation scheme

The SCO link is established by negotiations between the link managers which involves the exchange of important SCO timing parameters like T_{SCO} and D_{SCO} through LMP messages.

8.10.8.7 Broadcast scheme

The master of the piconet can broadcast messages which will reach all slaves. A broadcast packet is characterized by the all-zero AM_ADDR. Each new broadcast message (which may be carried by a number of packets) shall start with the flush indication (L_CH=10).

A broadcast packet is never acknowledged. In an error-prone environment, the master may carry out a number of retransmissions to increase the probability for error-free delivery (see also 8.5.3.5).

In order to support the park mode (as described in 8.10.8.4), a master transmission shall take place at fixed intervals. This master transmission will act as a beacon to which slaves can synchronize. If no traffic takes place at the beacon event, broadcast packets shall be sent. More information is given in 8.10.8.4.

8.10.9 Scatternet

8.10.9.1 General

Multiple piconets may cover the same area. Since each piconet has a different master, the piconets hop independently, each with their own channel hopping sequence and phase as determined by the respective master. In addition, the packets carried on the channels are preceded by different channel access codes as determined by the master device addresses. As more piconets are added, the probability of collisions increases; a graceful degradation of performance results as is common in frequency-hopping spread spectrum systems.

If multiple piconets cover the same area, a unit can participate in two or more overlaying piconets by applying time multiplexing. To participate on the proper channel, it should use the associated master device address and proper clock offset to obtain the correct phase. A Bluetooth unit can act as a slave in several piconets, but only as a master in a single piconet: since two piconets with the same master are synchronized and use the same hopping sequence, they are one and the same piconet. A group of piconets in which connections consists between different piconets is called a scatternet.

A master or slave can become a slave in another piconet by being paged by the master of this other piconet. On the other hand, a unit participating in one piconet can page the master or slave of another piconet. Since the paging unit always starts out as master, a master-slave role exchange is required if a slave role is desired. This is described in the 8.10.9.3.

8.10.9.2 Inter-piconet communications

Time multiplexing shall be used to switch between piconets. In case of ACL links only, a unit can request to enter the hold or park mode in the current piconet during which time it may join another piconet by just changing the channel parameters. Units in the sniff mode may have sufficient time to visit another piconet in between the sniff slots. If SCO links are established, other piconets can only be visited in the nonreserved slots in between. This is only possible if there is a single SCO link using HV3 packets. In the four slots in between, one other piconet can be visited. Since the multiple piconets are not synchronized, guard time shall be left to account for misalignment. This means that only 2 slots can effectively be used to visit another piconet in between the HV3 packets.

Since the clocks of two masters of different piconets are not synchronized, a slave unit participating in two piconets has to take care of two offsets that, added to its own native clock, create one or the other master clock. Since the two master clocks drift independently, regular updates of the offsets are required in order for the slave unit to keep synchronization to both masters.

8.10.9.3 Master-slave switch

There are several occasions when a master-slave (MS) switch is desirable. Firstly, a MS switch is necessary when a unit paging the master of an existing piconet wants to join this piconet, since, by definition, the paging unit initially is master of a “small” piconet only involving the pager (master) and the paged (slave) unit. Secondly, when a slave in an existing piconet wants to set up a new piconet, involving itself as master and the current piconet master as slave. The latter case implies a double role of the original piconet master; it becomes a slave in the new piconet while still maintaining the original piconet as master. Thirdly, a much more complicated example is when a slave wants to fully take over an existing piconet, i.e., the switch also involves transfer of other slaves of the existing piconet to the new piconet. Clearly, this can be achieved by letting the new master setup a completely new piconet through the conventional paging scheme. However, that would require individual paging of the old slaves, and, thus, take an unnecessarily long time. Instead, letting the new master utilize timing knowledge of the old master is more efficient. As a consequence of the MS switch, the slaves in the piconet have to be transferred to the new piconet, changing their timing and their hopping scheme.

The MS switch is described below. Prior to the MS switch, encryption if present, shall be stopped in the old piconet. For the third example involving the transfer, new piconet parameters have to be communicated to each slave. The process of this is described below. Unfortunately, even though all the hooks are defined for an efficient transfer at baseband level, there are still many issues that lack sufficient support in the higher layers of the Bluetooth specification (such as how to handle security and transfer all kind of slave information from old to new master). Until all levels of the specification fully supports this kind of transfer, this functionality will have to be taken care of at application layer. These transfer procedures are outside the scope of the baseband specification.

The MS switch procedure will now be described in more detail. For the master and slave involved in the role switch, the MS switch results in a reversal of their TX and RX timing: a TDD switch. Moreover, since the piconet parameters are derived from the device address and clock of the master, an MS switch inherently involves a redefinition of the piconet as well: a piconet switch. The new piconet's parameters are derived from the former slave's device address and clock.

Assume unit A wants to become master; unit B was the former master. Then there are basically two alternative scenarios: either the slave takes the MS switch initiative or the master takes the MS switch initiative. These scenarios are described in 9.3.12.

Both slave A and master B do the TDD switch but keep the former hopping scheme (still using the device address and clock of unit B), so there is no piconet switch yet. The slot offset information sent by slave A is

not used yet, but is used later in the process. Unit A now becomes the master, unit B the slave. The AM_ADDR formerly used by unit A in its slave role, is now used by slave B.

At the moment of the TDD switch, both units A and B will start a timer with a time out of newconnectionTO. The timer is stopped in slave B as soon as it receives an FHS packet from master A on the TDD-switched channel, the timer is stopped in master A as soon as it receives an ID packet from slave B. If the newconnectionTO expires, the master and slave will return to the old piconet timing and take their old role of master and slave. The FHS packet is sent by master A using the “old” piconet parameters. The AM_ADDR in the FHS packet header is the former AM_ADDR used by unit A. The AM_ADDR carried in the FHS payload is the new AM_ADDR intended for unit B when operating on the new piconet. After the FHS acknowledgment, which consists of the ID packet and is sent by the slave on the old hopping sequence, both master A and slave B turn to the new channel parameters of the new piconet as indicated by the FHS.

Since the old and new masters' clocks are synchronous, the clock information sent in the FHS payload should indicate the new master's clock at the begining of the FHS packet transmission. Furthermore, the 1.25 ms resolution of the clock information given in the FHS packet is not sufficient for aligning the slot boundaries of the two piconets. The slot-offset information in the LMP message previously sent by unit A is used to provide more accurate timing information. The slot offset indicates the delay between the start of the master-to-slave slots of the old and new piconet channels. This timing information ranges from 0 to 1249 µs with a resolution of 1µs. It is used together with the clock information in the FHS packet to accurately position the correlation window when switching to the new master's timing after acknowledgment of the FHS packet. After reception of the FHS packet acknowledgment, the new master A switches to its own timing and sends a POLL packet to verify the switch. Both the master and the slave will start a new timer with a time out of newconnectionTO on FHS packet acknowledgment. The start of this timer shall be aligned with the beginning of the first master TX slot boundary of the new piconet, following the FHS packet acknowledgment. The slave stops the timer when the POLL packet is received; the master stops the timer when the POLL packet is acknowledged. The slave uses a NULL packet to acknowledge the POLL. If no response is received, the master re-sends the POLL packet until newconnectionTO is reached. Should this timer expire, both the slave and the master return to the old piconet timing with the old master and slave roles. The procedure may then start again at the beginning. Aligning the timer with TX boundaries of the new piconet ensures that no unit returns to the old piconet timing in the middle of a master RX slot.

If the new master wishes to take over slaves from the old piconet (which were slaves to the old master B), a piconet switch is enforced on each slave separately. Since the existing slaves already have the correct TDD timing, a TDD switch is not required. Master A sends a slot-offset LMP message to indicate the difference in slot timing of the old and new piconet channel. Thereafter, master A sends an FHS packets and waits for an acknowledgment in the form of an ID packet. When sending the FHS packet, the master starts a timer with a time out of newconnectionTO. The timer is stopped in master A as soon as it receives an ID packet from the slave. Transmission of the FHS packet and the acknowledgment is carried out with the "old" piconet parameters of unit B (compare this to the page hopping scheme used during connection establishment, see 8.10.6.4). Should the timer in master A expire, it may restart the transfer operation. After FHS acknowledgment by the slave, the communication to this slave continues with the new device address and clock of unit A. The FHS packet sent to each slave has the old AM_ADDR in the FHS packet header and their new AM_ADDR in the FHS packet payload (the new AM_ADDR may be identical to the old AM_ADDR).

After reception of the FHS packet acknowledgment, the new master A switches to its own timing and hopping sequence and sends a POLL packet to verify the switch. Both the master and the slave will start a timer with a time out of newconnectionTO on FHS packet acknowledgment. The start of this timer shall be aligned with the beginning of the first master TX slot boundary of the new piconet, following the FHS packet acknowledgment. The slave stops the timer when the POLL packet is received; the master stops the timer when the POLL packet is acknowledged. The slave uses a NULL packet to acknowledge the POLL. If no response is received, the master re sends the POLL packet until newconnectionTO is reached. If the timer expires, both the slave and the master return to the old piconet parameters. The procedure may then be

repeated. If an existing slave is out of the range of the new master, master A cannot switch the slave to the new piconet. In that case, the slave loses the connection with the existing piconet after the TDD switch when master B is replaced with master A. As a result, the first message sent by master A being the slot-offset LMP message is never acknowledged by this slave. Due to the link supervision mechanism, the slave will eventually be detached from the old piconet.

Summarized, the MS-switch takes place in two steps: first a TDD switch of the considered master and slave, followed by a piconet switch of the both participants. Then, if so desired, other slaves of the old piconet can be transferred to the new piconet. When a unit have acknowledged the reception of the FHS packet, this unit uses the new piconet parameters defined by the new master and the piconet switch is completed. Note that the SEQN of the first data packet containing a CRC on the new piconet channel is set to 1 (see 8.5.3.2).

A parked slave must be unparked before it can participate in a MS switch. Parked slaves that are transferred to a new piconet shall be activated using the old park parameters, changed to the new piconet parameters, and then returned to the park mode using the new park parameters.

8.10.10 Power management

Features are included into Bluetooth to ensure a low-power operation. These features are both at the microscopic level when handling the packets, and at the macroscopic level using certain operation modes.

8.10.10.1 Packet handling

In order to minimize power consumption, packet handling is minimized both at TX and RX sides. At the TX side, power is minimized by only sending useful data. This means that if only link control information needs to be exchanged, NULL packets will be used. No transmission is carried out at all if there is no link control information or involves a NAK only (NAK is implicit on no reply). If there is data to be sent, the payload length is adapted in order to send only the valid data bytes. At the RX side, packet processing takes place in different steps. If no valid access code is found in the search window, the transceiver returns to sleep. If an access code is found, the receiver unit is woken up and starts to process the packet header. If the HEC fails, the unit will return to sleep after the packet header. A valid header will indicate if a payload will follow and how many slots are involved.

8.10.10.2 Slot occupancy

As was described in 8.4.4, the packet type indicates how many slots a packet may occupy. A slave not addressed in the first slot can go to sleep for the remaining slots the packet may occupy. This can be read from the TYPE code.

8.10.10.3 Low-power modes

In 8.10.8, three modes were described during the CONNECTION state which reduce power consumption. If we list the modes in increasing order of power efficiency then the sniff mode has the higher duty cycle, followed by the hold mode with a lower duty cycle, and finishing with the park mode with the lowest duty cycle.

8.10.11 Link supervision

A connection may break down due to various reasons such as a device moving out of range or a power failure condition. Since this may happen without any prior warning, it is important to monitor the link on both the master and the slave side to avoid possible collisions when the AM_ADDR is reassigned to another slave.

To be able to supervise link loss, both the master and the slave use link supervision timers, $T_{supervision}$. Upon reception of a packet that passes the HEC check and has the correct AM_ADDR, the timer is reset. If at any time in connection state, the timer reaches the *supervisionTO* value, the connection is reset. The same timeout value is used for both SCO and ACL connections.

The timeout period, *supervisionTO*, is negotiated at the LM level. Its value is chosen so that the supervision timeout will be longer than hold and sniff periods. Link supervision of a parked slave will be done by unparking and re-parking the slave.

8.11 Hop selection

In total, ten types of hopping sequences are defined: five for the 79-hop and five for the 23-hop system, respectively. Using the notation of parentheses for figures related to the 23-hop system, these sequences are as follows:

— A page hopping sequence with 32 (16) unique wake-up frequencies distributed equally over the 79 (23) MHz, with a period length of 32 (16);

— A page response sequence covering 32 (16) unique response frequencies that all are in an one-to-one correspondence to the current page hopping sequence. The master and slave use different rules to obtain the same sequence;

— An inquiry sequence with 32 (16) unique wake-up frequencies distributed equally over the 79 (23) MHz, with a period length of 32 (16);

— A inquiry response sequence covering 32 (16) unique response frequencies that all are in an one-to-one correspondence to the current inquiry hopping sequence.

— A channel hopping sequence which has a very long period length, which does not show repetitive patterns over a short time interval, but which distributes the hop frequencies equally over the 79 (23) MHz during a short time interval.

For the page hopping sequence, it is important that we can easily shift the phase forward or backward, so we need a 1-1 mapping from a counter to the hop frequencies. For each case, both a hop sequence from master to slave and from slave to master are required.

The inquiry and inquiry response sequences always utilizes the GIAC LAP as lower address part and the DCI (see 8.5.4) as upper address part in deriving the hopping sequence, even if it concerns a DIAC inquiry.

8.11.1 General selection scheme

The selection scheme consists of two parts:

— Selecting a sequence

— Mapping this sequence on the hop frequencies

The general block diagram of the hop selection scheme is shown in Figure 59. The mapping from the input to a particular hop frequency is performed in the selection box. Basically, the input is the native clock and the current address. In CONNECTION state, the native clock (CLKN) is modified by an offset to equal the master clock (CLK). Only the 27 MSBs of the clock are used. In the page and inquiry substates, all 28 bits of the clock are used. However, in page substate the native clock will be modified to the master's estimate of the paged unit.

The address input consists of 28 bits, i.e., the entire LAP and the four LSBs of the UAP. In CONNECTION state, the address of the master is used. In page substate the address of the paged unit is used. When in inquiry substate, the UAP/LAP corresponding to the GIAC is used. The output constitutes a pseudo-random sequence, either covering 79 hop or 23 hops, depending on the state.

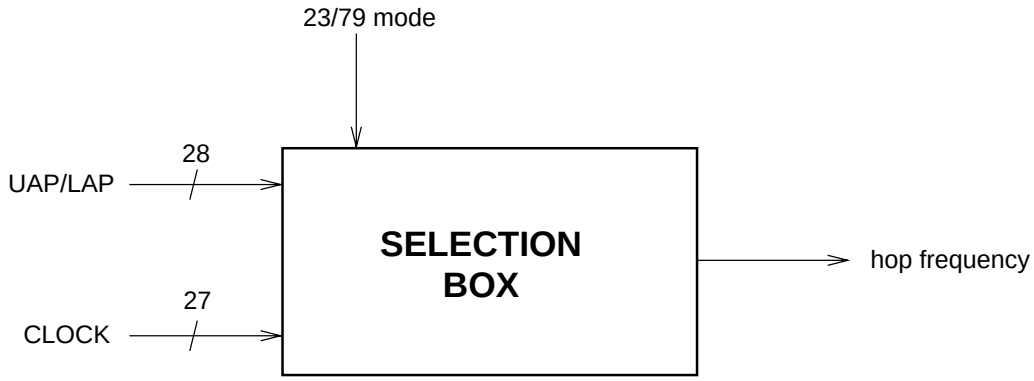


Figure 59—General block diagram of hop selection scheme

For the 79-hop system, the selection scheme chooses a segment of 32 hop frequencies spanning about 64 MHz and visits these hops once in a random order. Next, a different 32-hop segment is chosen, etc. In case of the page, page scan, or page response substates, the same 32-hop segment is used all the time (the segment is selected by the address; different units will have different paging segments). In connection state, the output constitutes a pseudo-random sequence that slides through the 79 hops or 23 hops, depending on the selected hop system. For the 23-hop systems, the segment size is 16. The principle is depicted in Figure 60.

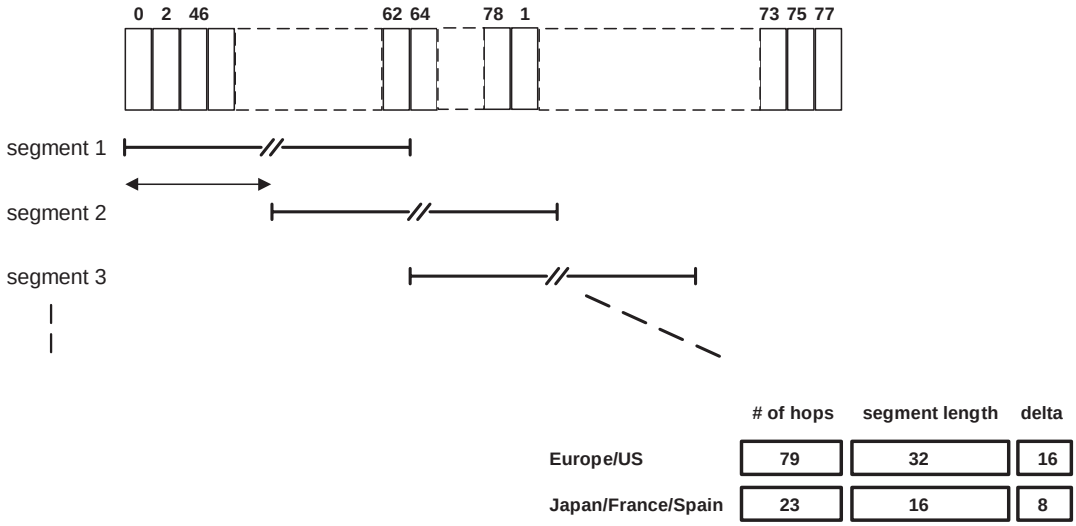


Figure 60—Hop selection scheme in CONNECTION state

8.11.2 Selection kernel

The hop selection kernels for the 79-hop system and the 23-hop system are shown in Figure 61 and Figure 62, respectively. The *X* input determines the phase in the 32-hop segment, whereas *Y*1 and *Y*2 selects between master-to-slave and slave-to-master transmission. The inputs *A* to *D* determine the ordering within the segment, the inputs *E* and *F* determine the mapping onto the hop frequencies. The kernel addresses a register containing the hop frequencies. This list should be created such that first all even hop frequencies are listed and then all odd hop frequencies. In this way, a 32-hop segment spans about 64 MHz, whereas a 16-hop segment spans the entire 23 MHz.

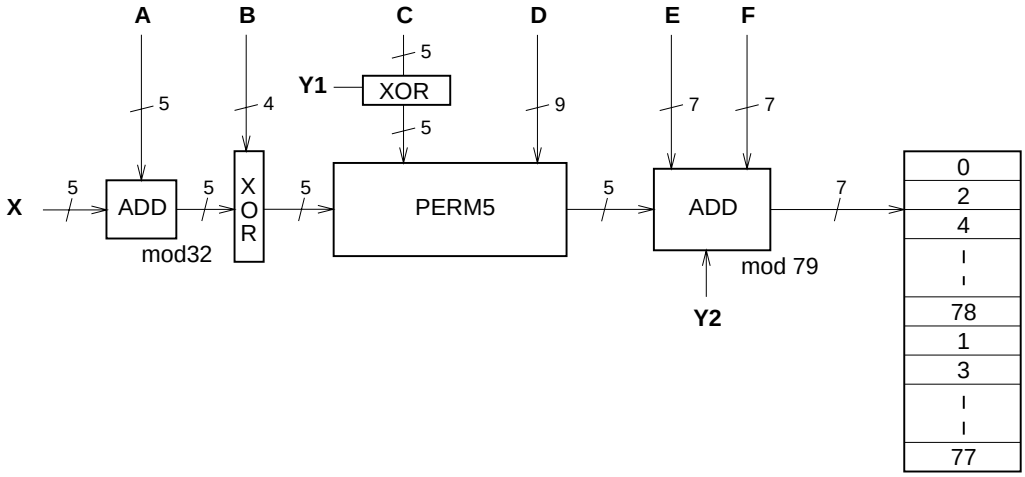


Figure 61—Block diagram of hop selection kernel for the 79-hop system

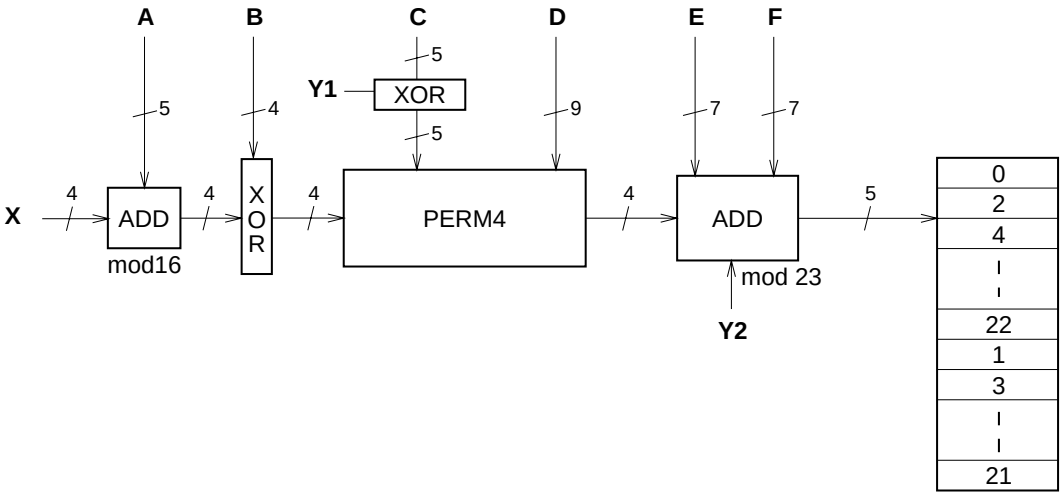


Figure 62—Block diagram of hop selection kernel for the 23-hop system

The selection procedure consists of an addition, an XOR operation, a permutation operation, an addition, and finally a register selection. In the remainder of this chapter, the notation A_i is used for bit i of the BD_ADDR.

8.11.2.1 First addition operation

The first addition operation only adds a constant to the phase and applies a modulo 32 or a modulo 16 operation. For the page hopping sequence, the first addition is redundant since it only changes the phase within the segment. However, when different segments are concatenated (as in the channel hopping sequence), the first addition operation will have an impact on the resulting sequence.

8.11.2.2 XOR operation

Let *Z*' denote the output of the first addition. In the XOR operation, the four LSBs of *Z*' are modulo-2 added to the address bits $A_{22–19}$. The operation is illustrated in Figure 63.

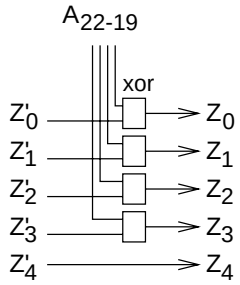


Figure 63—XOR operation for the 79-hop system.
(The 23-hop system is the same except for the Z'_4/Z_4 wire that does not exist.)

8.11.2.3 Permutation operation

The permutation operation involves the switching from 5 inputs to 5 outputs for the 79 hop system and from 4 inputs to 4 outputs for 23 hop system, in a manner controlled by the control word. In Figure 64 and Figure 65 the permutation or switching box is shown. It consists of 7 stages of butterfly operations. Table 29 and Table 30 shows the control of the butterflies by the control signals *P*. Note that $P_{0–8}$ corresponds to $D_{0–8}$, and, P_{i+9} corresponds to $C_i \oplus Y1$ for $i = 0 \ldots 4$ in Figure 61 and Figure 62.

The *Z* input is the output of the XOR operation as described in 8.11.2.2. The butterfly operation can be implemented with multiplexers as depicted in Figure 66.

Table 29—Control of the butterflies for the 79-hop system

Control signal	Butterfly		Control signal	Butterfly
P_0	$\{Z_0,Z_1\}$		P_8	$\{Z_1,Z_4\}$
P_1	$\{Z_2,Z_3\}$		P_9	$\{Z_0,Z_3\}$
P_2	$\{Z_1,Z_2\}$		P_{10}	$\{Z_2,Z_4\}$
P_3	$\{Z_3,Z_4\}$		P_{11}	$\{Z_1,Z_3\}$
P_4	$\{Z_0,Z_4\}$		P_{12}	$\{Z_0,Z_3\}$
P_5	$\{Z_1,Z_3\}$		P_{13}	$\{Z_1,Z_2\}$
P_6	$\{Z_0,Z_2\}$			
P_7	$\{Z_3,Z_4\}$		—	—

Table 30—Control of the butterflies for the 23-hop system

Control signal	Butterfly		Control signal	Butterfly
P_0	$\{Z_0,Z_1\}$		P_8	$\{Z_0,Z_2\}$
P_1	$\{Z_2,Z_3\}$		P_9	$\{Z_1,Z_3\}$
P_2	$\{Z_0,Z_3\}$		P_{10}	$\{Z_0,Z_3\}$
P_3	$\{Z_1,Z_2\}$		P_{11}	$\{Z_1,Z_2\}$
P_4	$\{Z_0,Z_2\}$		P_{12}	$\{Z_0,Z_1\}$
P_5	$\{Z_1,Z_3\}$		P_{13}	$\{Z_2,Z_3\}$
P_6	$\{Z_0,Z_1\}$		—	—
P_7	$\{Z_2,Z_3\}$		—	—

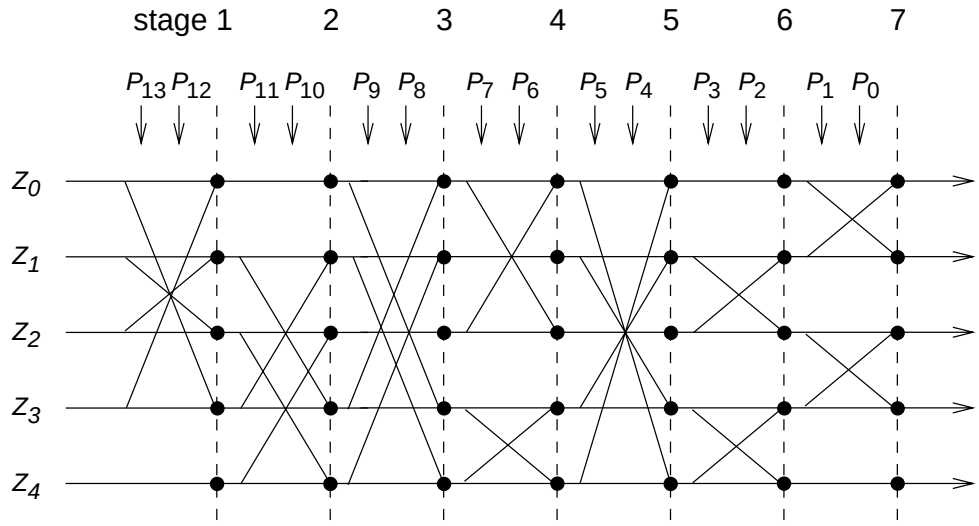


Figure 64—Permutation operation for the 79-hop system

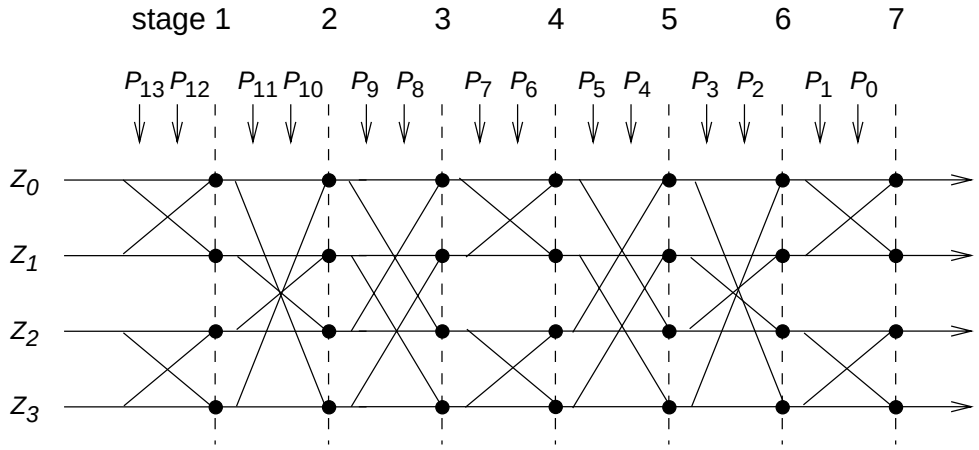


Figure 65—Permutation operation for the 23-hop system

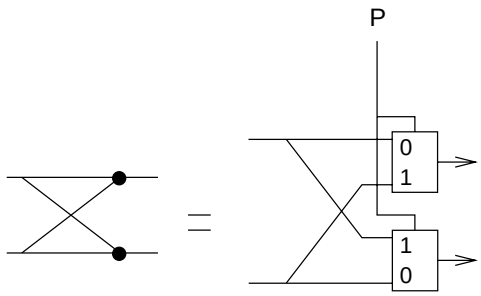


Figure 66—Butterfly implementation

8.11.2.4 Second addition operation

The addition operation only adds a constant to the output of the permutation operation. As a result, the 16-hop or 32-hop segment is mapped differently on the hop frequencies. The addition is applied modulo 79 or modulo 23 depending on the system type.

8.11.2.5 Register bank

The output of the adder addresses a bank of 79 or 23 registers. The registers are loaded with the synthesizer code words corresponding to the hop frequencies 0 to 78 or 0 to 22. Note that the upper half of the bank contains the even hop frequencies, whereas the lower half of the bank contains the odd hop frequencies.

8.11.3 Control word

In the following section X_{j-i}, $i<j$, will denote bits i, i+1,...,j of the bit vector X. By convention, X_0 is the least significant bit of the vector X.

The control word P of the kernel is controlled by the overall control signals X, $Y1$, $Y2$, and A to F as illustrated in Figure 61 and Figure 62. During paging and inquiry, the inputs A to E use the address values as given in the corresponding columns of Table 31 and Table 32. In addition, the inputs X, $Y1$ and $Y2$ are used. The F input is unused. In the 79-hop system, the clock bits CLK_{6-2} (i.e., input X) specifies the phase within the length 32 sequence, while for the 23-hop system, CLK_{5-2} specifies the phase within the length 16 sequence. For both systems, CLK_1 (i.e., inputs $Y1$ and $Y2$) is used to select between TX and RX. The address inputs determine the sequence order within segments. The final mapping onto the hop frequencies is determined by the register contents.

In the following we will distinguish between three types of clocks: the piconet's master clock, the Bluetooth unit's native clock, and the clock estimate of a paged Bluetooth unit. These types are marked in the following way:

1) CLK_{27-0}: Master clock of the current piconet.
2) $CLKN_{27-0}$: Native clock of the unit.
3) $CLKE_{27-0}$: The paging unit's estimate of the paged unit's native clock.

During the CONNECTION state, the inputs A, C, and D result from the address bits being bit-wise XORed with the clock bits as shown in the "Connection state" column of Table 31 and Table 32 (the two MSBs are XORed together, the two second MSBs are XORed together, etc.). Consequently, after every 32 (16) time slots, a new length 32 (16) segment is selected in the 79-hop (23-hop) system. The sequence order within a specific segment will not be repeated for a very long period. Thus, the overall hopping sequence consists of concatenated segments of 32-hops each. Since each 32-hop sequence spans more than 80% of the 79 MHz band, the desired frequency spreading over a short time interval is obtained.

Table 31—Control for 79-hop system

	Page scan/ inquiry scan	Page/inquiry	Page response (master/slave) and Inquiry response	Connection state
X	CLKN_{16-12} / $Xir^{(79)}_{4-0}$	$Xp^{(79)}_{4-0}/Xi^{(79)}_{4-0}$	$Xprm^{(79)}_{4-0}/Xprs^{(79)}_{4-0}/Xir^{(79)}_{4-0}$	CLK_{6-2}
*Y*1	0	$\mathrm{CLKE}_1/\mathrm{CLKN}_1$	$\mathrm{CLKE}_1/\mathrm{CLKN}_1/1$	CLK_1
*Y*2	0	$32\times\mathrm{CLKE}_1/$ $32\times\mathrm{CLKN}_1$	$32\times\mathrm{CLKE}_1$ / $32\times\mathrm{CLKE}_1$ 32×1	$32\times\mathrm{CLK}_1$
A	A_{27-23}	A_{27-23}	A_{27-23}	$A_{27-23}\oplus\mathrm{CLK}_{25-21}$
B	A_{22-19}	A_{22-19}	A_{22-19}	A_{22-19}
C	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}\oplus\mathrm{CLK}_{20-16}$
D	A_{18-10}	A_{18-10}	A_{18-10}	$A_{18-10}\oplus\mathrm{CLK}_{15-7}$
E	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$
F	0	0	0	$16\times\mathrm{CLK}_{27-7}$ mod 79

Table 32—Control for 23-hop system

	Page scan/ inquiry scan	Page/inquiry	Page response (master/slave) and inquiry response	Connection state
X	CLKN_{15-12} / $Xir^{(23)}_{3-0}$	$Xp^{(23)}_{3-0}/Xi^{(23)}_{3-0}$	$Xprm^{(23)}_{3-0}/Xprs^{(23)}_{3-0}/Xir^{(23)}_{3-0}$	CLK_{5-2}
*Y*1	0	$\mathrm{CLKE}_1/\mathrm{CLKN}_1$	$\mathrm{CLKE}_1/\mathrm{CLKN}_1/1$	CLK_1
*Y*2	0	$16\times\mathrm{CLKE}_1/$ $16\times\mathrm{CLKN}_1$	$16\times\mathrm{CLKE}_1$ / $16\times\mathrm{CLKE}_1$ 16×1	$16\times\mathrm{CLK}_1$
A	A_{27-23}	A_{27-23}	A_{27-23}	$A_{27-23}\oplus\mathrm{CLK}_{25-21}$
B	A_{22-19}	A_{22-19}	A_{22-19}	A_{22-19}
C	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}\oplus\mathrm{CLK}_{20-16}$
D	A_{18-10}	A_{18-10}	A_{18-10}	$A_{18-10}\oplus\mathrm{CLK}_{15-7}$
E	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$
F	0	0	0	$6\times\mathrm{CLK}_{27-6}$ mod 23

8.11.3.1 Page scan and Inquiry scan substates

In page scan, the Bluetooth device address of the scanning unit is used as address input. In inquiry scan, the GIAC LAP and the four LSBs of the DCI (as A_{27-24}), are used as address input for the hopping sequence. Naturally, for the transmitted access code and in the receiver correlator, the appropriate GIAC or DIAC is used. The application decides which inquiry access code to use depending on the purpose of the inquiry.

The five *X* input bits vary depending on the current state of the unit. In the page scan and inquiry scan substates, the native clock (CLKN) is used. In CONNECTION state the master clock (CLK) is used as input. The situation is somewhat more complicated for the other states.

8.11.3.2 Page substate

In the page substate of the 79-hop system, the paging unit shall start using the **A**-train, i.e., $\{f(k-8), \ldots, f(k), \ldots, f(k+7)\}$, where $f(k)$ is the source's estimate of the current receiver frequency in the paged unit. Clearly, the index k is a function of all the inputs in Figure 61. There are 32 possible paging frequencies within each 1.28 second interval. Half of these frequencies belongs to the **A**-train, the rest (i.e., $\{f(k+8), \ldots, f(k+15), f(k-16), \ldots, f(k-9)\}$) belongs to the **B**-train. In order to achieve the -8 offset of the **A**-train, a constant of 24 can be added to the clock bits (which is equivalent to -8 due to the modulo 32 operation). Clearly, the **B**-train may be accomplished by setting the offset to 8. A cyclic shift of the order within the trains is also necessary in order to avoid a possible repetitive mismatch between the paging and scanning units. Thus,

$$Xp^{(79)} = [\mathrm{CLKE}_{16-12} + k_{offset} + (\mathrm{CLKE}_{4-2,0} - \mathrm{CLKE}_{16-12}) \bmod 16] \bmod 32, \quad (2)$$

where

$$k_{offset} = \begin{cases} 24 & \text{A-train,} \\ 8 & \text{B-train.} \end{cases} \quad (3)$$

Alternatively, each switch between the **A**- and **B**-trains may be accomplished by adding 16 to the current value of k_{offset} (originally initialized with 24).

In the page substate of the 23-hop system, the paging unit makes use of the **A**-train only. A constant offset of 8 is used in order to start with $f(k-8)$. Moreover, only four bits are needed since the additions are modulo 16. Consequently,

$$Xp^{(23)} = [\mathrm{CLKE}_{15-12} + 8 + \mathrm{CLKE}_{4-2,0}] \bmod 16, \quad (4)$$

8.11.3.3 Page response

8.11.3.3.1 Slave response

A unit in the page scan substate recognizing its own access code enters the slave response substate. In order to eliminate the possibility of loosing the link due to discrepancies of the native clock CLKN and the master's clock estimate CLKE, the four bits CLKN_{16-12} shall be frozen at their current value. The value is frozen to the content it has in the slot where the recipient's access code is detected. Note that the actual native clock is *not* stopped; it is merely the values of the bits used for creating the X-input that are kept fixed for a while. In the sequel, a frozen value is marked by an asterisk (*).

For each response slot the paged unit will use an *X*-input value one larger (modulo 32 or 16) than in the preceding response slot. However, the first response is made with the *X*-input kept at the same value as it was when the access code was recognized. Let N be a counter starting at zero. Then, the *X*-input in the $(N+1)$-th response slot (the first response slot being the one immediately following the page slot now responding to) of the slave response substate becomes

$$Xprs^{(79)} = [\mathrm{CLKN}^*_{16-12} + N] \bmod 32, \quad (5)$$

and

$$Xprs^{(23)} = [\mathrm{CLKN}^*_{15-12} + N] \bmod 16, \tag{6}$$

for the 79-hop and 23-hop systems, respectively. The counter N is set to zero in the slot where the slave acknowledges the page (see Figure 52 and Figure 53). Then, the value of N is increased by one each time CLKN_1 is set to zero, which corresponds to the start of a master TX slot. The *X*-input is constructed this way until the first accepted FHS packet is received *and* the immediately following response packet has been transmitted. After this the slave enters the CONNECTION state using the parameters received in the FHS packet.

8.11.3.3.2 Master response

The paging unit enters master response substate upon receiving a slave response. Clearly, also the master must freeze its estimated slave clock to the value that triggered a response from the paged unit. It is equivalent to using the values of the clock estimate when receiving the slave response (since only CLKE_1 will differ from the corresponding page transmission). Thus, the values are frozen when the slave ID packet is received. In addition to the used clock bits, also the current value of k_{offset} shall be frozen. The master will adjust its *X*-input in the same way the paged unit does, i.e., by incrementing this value by one for each time CLKE_1 is set to zero. The first increment shall be done before sending the FHS packet to the paged unit. Let N be a counter starting at one. The rules for forming the *X*-inputs become

$$\begin{aligned} Xprm^{(79)} = [&\mathrm{CLKE}^*_{16-12} + k_{offset}{}^* + \\ &(\mathrm{CLKE}^*_{4-2,0} - \mathrm{CLKE}^*_{16-12}) \bmod 16 + N] \bmod 32, \end{aligned} \tag{7}$$

and

$$Xprm^{(23)} = [\mathrm{CLKE}^*_{15-12} + 8 + \mathrm{CLKE}^*_{4-2,0} + N] \bmod 16, \tag{8}$$

for the 79-hop and 23-hop systems, respectively. The value of N is increased each time CLKE_1 is set to zero, which corresponds to the start of a master TX slot.

8.11.3.4 Inquiry substate

The *X*-input of the inquiry substate is quite similar to what is used in the page substate. Since no particular unit is addressed, the native clock CLKN of the inquirer is used. Moreover, which of the two train offsets to start with is of no real concern in this state. Consequently,

$$Xi^{(79)} = [\mathrm{CLKN}_{16-12} + k_{offset} + (\mathrm{CLKN}_{4-2,0} - \mathrm{CLKN}_{16-12}) \bmod 16] \bmod 32, \tag{9}$$

where k_{offset} is defined by Equation (3). The initial choice of the offset is arbitrary. For the 23-hop system,

$$Xi^{(23)} = [\mathrm{CLKN}_{15-12} + 8 + \mathrm{CLKN}_{4-2,0}] \bmod 16, \tag{10}$$

The GIAC LAP and the four LSBs of the DCI (as A_{27-24}) are used as address input for the hopping sequence generator.

8.11.3.5 Inquiry response

The inquiry response substate is similar to the slave response substate with respect to the X-input. However, there is no need to freeze the clock input, thus

$$Xir^{(79)} = [\mathrm{CLKN}_{16-12} + N] \mod 32, \quad (11)$$

and

$$Xir^{(23)} = [\mathrm{CLKN}_{15-12} + N] \mod 16, \quad (12)$$

for the 79-hop and 23-hop systems, respectively. Furthermore, the counter N is increased not on CLKN_1 basis, but rather after each FHS packet has been transmitted in response to the inquiry. There is no restriction on the initial value of N as it is independent of the corresponding value in the inquiring unit.

The GIAC LAP and the four LSBs of the DCI (as A_{27-24}) are used as address input for the hopping sequence generator. The other input bits to the generator are the same as in the case of page response.

8.11.3.6 Connection state

In CONNECTION state, the clock bits to use in the channel hopping sequence generation are always according to the master clock, CLK. The address bits are taken from the Bluetooth device address of the master.

8.12 Bluetooth audio

On the Bluetooth air-interface, either a 64 kb/s log PCM format (A-law or μ-law) is used, or a 64 kb/s CVSD (Continuous Variable Slope Delta Modulation) is used. The latter format applies an adaptive delta modulation algorithm with syllabic companding.

The voice coding on the line interface should have a quality equal to or better than the quality of 64 kb/s log PCM.

Table 33 summarizes the voice coding schemes supported on the air interface. The appropriate voice coding scheme is selected after negotiations between the Link Managers.

Table 33—Voice coding schemes supported on the air interface

Voice Codecs	
linear	CVSD
8-bit logarithmic	A-law
	μ-law

8.12.1 LOG PCM CODEC

Since the voice channels on the air-interface can support a 64 kb/s information stream, a 64 kb/s log PCM traffic can be used for transmission. Either A-law or μ-law compression can be applied. In the event that the

line interface uses A-law and the air interface uses μ-law or vice versa, a conversion from A-law to μ-law is performed. The compression method follows ITU-T Recommendation G.711 (11/88).

8.12.2 CVSD CODEC

A more robust format for voice over the air interface is a delta modulation. This modulation scheme follows the waveform where the output bits indicate whether the prediction value is smaller or larger then the input waveform. To reduce slope overload effects, syllabic companding is applied: the step size is adapted according to the average signal slope. The input to the CVSD encoder is 64 ksamples/s linear PCM. Block diagrams of the CVSD encoder and CVSD decoder are shown in Figure 67, Figure 68 and Figure 69. The system is clocked at 64 kHz.

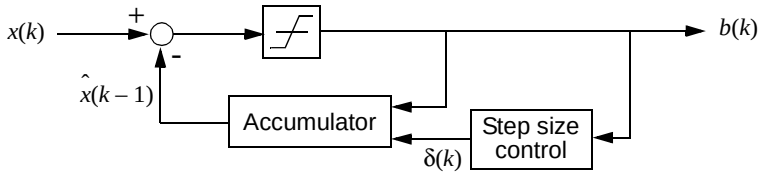


Figure 67—Block diagram of CVSD encoder with syllabic companding

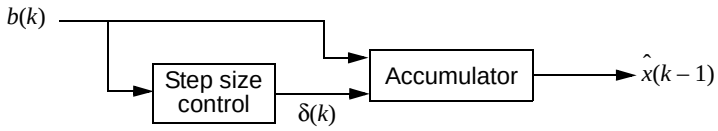


Figure 68—Block diagram of CVSD decoder with syllabic companding

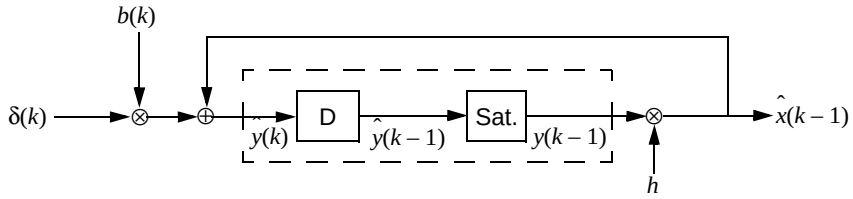


Figure 69—Accumulator procedure

Let $\mathrm{sgn}(x) = 1$ for $x \geq 0$, otherwise, $\mathrm{sgn}(x) = -1$. On air these numbers are represented by the sign bit; i.e. negative numbers are mapped on "1" and positive numbers are mapped on "0". Denote the CVSD encoder output bit $b(k)$, the accumulator contents $y(k)$, and the step size $\delta(k)$. Furthermore, let h be the decay factor for the accumulator, let β denote the decay factor for the step size, and, let α be the syllabic companding parameter. The latter parameter monitors the slope by considering the K most recent output bits.

Let

$$\hat{x}(k) = hy(k). \tag{13}$$

Then, the CVSD encoder internal state is updated according to the following set of equations:

$$b(k) = \text{sgn}\{x(k) - \hat{x}(k-1)\}, \tag{14}$$

$$\alpha = \begin{cases} 1, & \text{if } J \text{ bits in the last } K \text{ output bits are equal,} \\ 0, & \text{otherwise,} \end{cases} \tag{15}$$

$$\delta(k) = \begin{cases} \min\{\delta(k-1) + \delta_{min}, \delta_{max}\}, & \alpha = 1, \\ \max\{\beta\delta(k-1), \delta_{min}\}, & \alpha = 0, \end{cases} \tag{16}$$

$$y(k) = \begin{cases} \min\{\hat{y}(k), y_{max}\}, & \hat{y}(k) \geq 0. \\ \max\{\hat{y}(k), y_{min}\}, & \hat{y}(k) < 0. \end{cases} \tag{17}$$

where

$$\hat{y}(k) = \hat{x}(k-1) + b(k)\delta(k). \tag{18}$$

In these equations, δ_{min} and δ_{max} are the minimum and maximum step sizes, and y_{min} and y_{max} are the accumulator's negative and positive saturation values, respectively. Over air, the bits are sent in the same order they are generated by the CVSD encoder.

For a 64 kb/s CVSD, the parameters as shown in Table 34 shall be used. The numbers are based on a 16 bit signed number output from the accumulator. These values result in a time constant of 0.5 ms for the accumulator decay, and a time constant of 16 ms for the step size decay.

8.12.3 Error handling

In the DV and HV3 packet, the voice is not protected by FEC. The quality of the voice in an error-prone environment then depends on the robustness of the voice coding scheme. CVSD, in particular, is rather insensitive to random bit errors, which are experienced as white background noise. However, when a packet is rejected because either the channel access code or the HEC test was unsuccessful, measures have to be taken to "fill" in the lost speech segment.

The voice payload in the HV2 packet is protected by a 2/3 rate FEC. For errors that are detected but cannot be corrected, the receiver should try to minimize the audible effects. For instance, from the 15-bit FEC segment with uncorrected errors, the 10-bit information part as found before the FEC decoder should be used. The HV1 packet is protected by a 3-bit repetition FEC. For this code, the decoding scheme will always assume zero or one-bit errors. Thus, there exist no detectable but uncorrectable error events for HV1 packets.

8.12.4 General audio requirements

The following specifications are considered to be default values and shall be supported as a minimum.

Table 34—CVSD parameter values[a]

Parameter	Value
h	$1 - \frac{1}{32}$
b	$1 - \frac{1}{1024}$
J	4
K	4
δ_{min}	10
δ_{max}	1280
y_{min}	-2^{15} or $-2^{15} + 1$
y_{max}	$2^{15} - 1$

[a]The values are based on a 16-bit signed number output from the accumulator.

8.12.4.1 Signal levels

For A-law and μ-law log-PCM encoded signals the requirements on signal levels follows ITU-T Recommendation G.711 (11/88).

Full swing at the 16 bit linear PCM interface to the CVSD encoder is defined to be 3 dBm0. A digital CVSD encoded test signal is provided in a Test Signal file available via 2.3. This signal is generated by a software implementation of a reference CVSD encoder. The digital encoder input signal (1020 Hz, sine-wave) generating the test signal has a nominal power of –15 dBm0. When the CVSD encoded test signal is fed through the CVSD receiver chain, the nominal output power should be –15 ± 1.0 dBm0.

8.12.4.2 CVSD audio quality

For Bluetooth audio quality the requirements are put on the transmitter side. The 64 ksamples/s linear PCM input signal must have negligible spectral power density above 4 kHz. A set of reference input signals are encoded by the transmitter and sent through a reference decoder (available via 2.3). The power spectral density in the 4–32 kHz band of the decoded signal at the 64 ksample/s linear PCM output, should be more than 20 dB below the maximum in the 0–4 kHz range.

8.13 Bluetooth addressing

8.13.1 Bluetooth device address (BD_ADDR)

Each Bluetooth transceiver is allocated a unique 48-bit Bluetooth device address (BD_ADDR). This address is derived from IEEE Std 802-2001. This 48-bit address is divided into three fields:

— LAP field: lower address part consisting of 24 bits

— UAP field: upper address part consisting of 8 bits

— NAP field: non-significant address part consisting of 16 bits

The LAP and UAP form the significant part of the BD_ADDR (see Figure 70). The total address space obtained is 2^{32}.

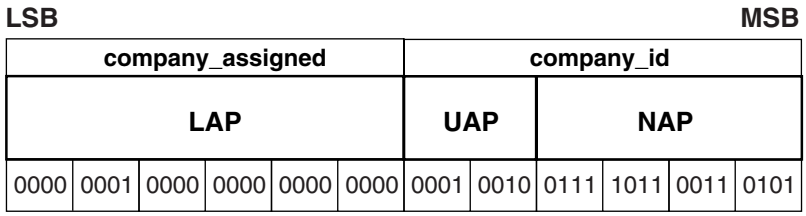


Figure 70—Format of BD_ADDR
(company_id should be organizationally unique identifier)

8.13.2 Access codes

In the Bluetooth system, 72-bit and 68-bit access codes are used for signalling purposes. Three different access codes are defined (see also 8.4.2.1):

— device access code (DAC)

— channel access code (CAC)

— inquiry access code (IAC)

There is one general IAC (GIAC) for general inquiry operations and there are 63 dedicated IACs (DIACs) for dedicated inquiry operations. All codes are derived from a LAP of the BD_ADDR. The device access code is used during page, page scan and page response substates. It is a code derived from the unit's BD_ADDR. The channel access code characterizes the channel of the piconet and forms the preamble of all packets exchanged on the channel. The channel access code is derived from the LAP of the master BD_ADDR. Finally, the inquiry access code is used in inquiry operations. A general inquiry access code is common to all Bluetooth units; a set of dedicated inquiry access codes is used to inquire for classes of devices.

The access code is also used to indicate to the receiver the arrival of a packet. It is used for timing synchronization and offset compensation. The receiver correlates against the entire sync word in the access code, providing a very robust signalling. During channel setup, the code itself is used as an ID packet to support the acquisition process. In addition, it is used during random access procedures in the PARK state.

The access code consists of preamble, sync word and a trailer (see 8.4.2). The next two subclauses describe the generation of the sync word.

8.13.2.1 Synchronization word definition

The sync words are based on a (64,30) expurgated block code with an overlay (bit-wise XOR) of an 64 bit full length PN-sequence. The expurgated code guarantees large Hamming distance ($d_{min} = 14$) between sync words based on different addresses. The PN sequence improves the auto correlation properties of the access code. The following steps describe how to generate the sync word:

1) Generate information sequence;
2) XOR this with the "information covering" part of the PN overlay sequence;
3) Generate the codeword;
4) XOR the codeword with all 64 bits of the PN overlay sequence;

The information sequence is generated by appending 6 bits to the 24 bit LAP (step 1). The appended bits are 001101 if the MSB of the LAP equals 0. If the MSB of the LAP is 1 the appended bits are 110010. The LAP MSB together with the appended bits constitute a length-seven Barker sequence. The purpose of including a Barker sequence is to further improve the auto correlation properties. In step 2 the information is pre-scrambled by XORing it with the bits $p_{34} \ldots p_{63}$ of the *pseudo-random noise* (PN) sequence (defined in 8.13.2.2). After generating the codeword (step 3), the complete PN sequence is XORed to the codeword (step 4). This step de-scrambles the information part of the codeword. At the same time the parity bits of the codeword are scrambled. Consequently, the original LAP and Barker sequence are ensured a role as a part of the access code sync word, and the cyclic properties of the underlying code is removed. The principle is depicted in Figure 71.

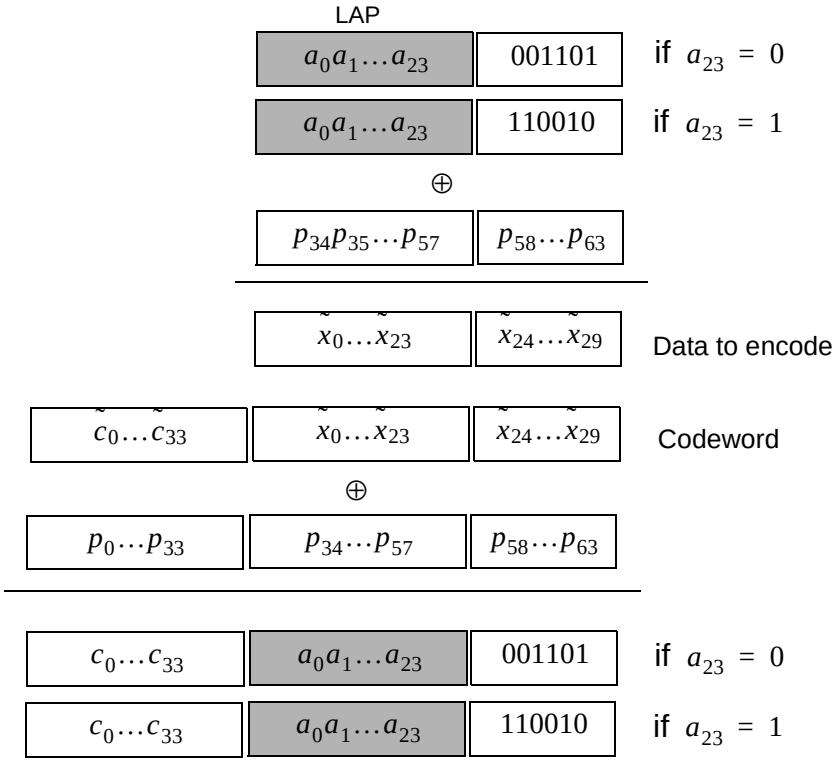


Figure 71—Construction of the sync word

In the sequel, binary sequences will be denoted by their corresponding D-transform (in which D^i represents a delay of i time units). Let $p'(D) = p'_0 + p'_1 D + \ldots + p'_{62} D^{62}$ be the 63 bit pseudo-random sequence, where p'_0 is the first bit (LSB) leaving the PRNG (see Figure 72), and p'_{62} is the last bit (MSB). To obtain 64 bits, an extra zero is appended at the *end* of this sequence (thus, $p'(D)$ is unchanged). For notational convenience, the reciprocal of this extended polynomial, $p(D) = D^{63} p'(1/D)$, will be used in the sequel. This is the sequence $p'(D)$ in reverse order. We denote the 24 bit lower address part (LAP) of the Bluetooth address by $a(D) = a_0 + a_1 D + \ldots + a_{23} D^{23}$ (a_0 is the LSB of the Bluetooth address).

The (64,30) block code generator polynomial is denoted $g(D) = (1 + D) g'(D)$, where $g'(D)$ is the generator polynomial `157464165547` (octal notation) of a primitive binary (63,30) BCH code. Thus, in octal notation we have

$$g(D) = \texttt{260534236651}, \tag{19}$$

the left-most bit corresponds to the high-order (g_{34}) coefficient.The dc-free four bit sequences `0101` and `1010` can be written

$$\begin{cases} F_0(D) = D + D^3, \\ F_1(D) = 1 + D^2, \end{cases} \tag{20}$$

respectively. Furthermore, we define

$$\begin{cases} B_0(D) = D^2 + D^3 + D^5, \\ B_1(D) = 1 + D + D^4, \end{cases} \tag{21}$$

which are used to create the length seven Barker sequences. Then, the access code is generated by the following procedure:

1) Format the 30 information bits to encode:

$$x(D) = a(D) + D^{24} B_{a_{23}}(D).$$

2) Add the information covering part of the PN overlay sequence:

$$\tilde{x}(D) = x(D) + p_{34} + p_{35} D + \ldots + p_{63} D^{29}.$$

3) Generate parity bits of the (64,30) expurgated block code:[10]

$$\tilde{c}(D) = D^{34} \tilde{x}(D) \mod g(D).$$

4) Create the codeword:

$$\tilde{s}(D) = D^{34} \tilde{x}(D) + \tilde{c}(D).$$

[10] $x(D)$ mod $y(D)$ denotes the rest when $x(D)$ is divided by $y(D)$.

5) Add the PN sequence:

$$s(D) = \tilde{s}(D) + p(D).$$

6) Append the (DC-free) preamble and trailer:

$$y(D) = F_{c_0}(D) + D^4 s(D) + D^{68} F_{a_{23}}(D).$$

8.13.2.2 Pseudo-random noise sequence generation

To generate the pseudo-random noise sequence we use the primitive polynomial $h(D) = 1 + D + D^3 + D^4 + D^6$. The LFSR and its starting state are shown in Figure 72. The PN sequence generated (including the extra terminating zero) becomes (hexadecimal notation) 83848D96BBCC54FC. The LFSR output starts with the left-most bit of this PN sequence. This corresponds to $p'(D)$ of the previous section. Thus, using the reciprocal $p(D)$ as overlay gives the 64 bit sequence

$$\boldsymbol{p} = 3F2A33DD69B121C1, \tag{22}$$

where the left-most bit is $p_0 = 0$ (there are two initial zeros in the binary representation of the hexadecimal digit 3), and $p_{63} = 1$ is the right-most bit.

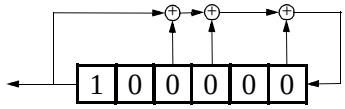


Figure 72—LFSR and the starting state to generate $p'(D)$

8.13.2.3 Reserved addresses for GIAC and DIAC

There is a block of 64 contiguous LAPs reserved for Bluetooth inquiry operations; one LAP common to all Bluetooth devices is reserved for general inquiry, the remaining 63 LAPs are reserved for dedicated inquiry of specific classes of Bluetooth devices. The same 64-block is used regardless of the contents of UAP and NAP. Consequently, none of these LAPs can be part of a user BD_ADDR.

When initializing HEC and CRC for the FHS packet of inquiry response, the UAP is replaced by DCI. Likewise, whenever one of the reserved BD_ADDRs is used for generating a frequency hop sequence, the UAP will be replaced by the DCI.

The reserved LAP addresses are tentatively chosen as 0x9E8B00-0x9E8B3F. The general inquiry LAP is tentatively chosen to 0x9E8B33. All addresses have the LSB at the rightmost position, hexadecimal notation.

8.13.3 Active member address (AM_ADDR)

Each slave active in a piconet is assigned a 3-bit active member address (AM_ADDR). The all-zero AM_ADDR is reserved for broadcast messages. The master does not have an AM_ADDR. Its timing relative to the slaves distinguishes it from the slaves. A slave only accepts a packet with a matching AM_ADDR and broadcast packets. The AM_ADDR is carried in the packet header. The AM_ADDR is only valid as long as a slave is active on the channel. As soon as it is disconnected or parked, it loses the AM_ADDR.

The AM_ADDR is assigned by the master to the slave when the slave is activated. This is either at connection establishment or when the slave is unparked. At connection establishment, the AM_ADDR is carried in the FHS payload (the FHS header itself carries the all-zero AM_ADDR). When unparking, the AM_ADDR is carried in the unpark message.

8.13.4 Parked Member Address (PM_ADDR)

A slave in park mode can be identified by its BD_ADDR or by a dedicated parked member address (PM_ADDR). This latter address is a 8-bit member address that separates the parked slaves. The PM_ADDR is only valid as long as the slave is parked. When the slave is activated it is assigned an AM_ADDR and loses the PM_ADDR. The PM_ADDR is assigned to the slave the moment it is parked.

The all-zero PM_ADDR is reserved for parked slaves that only use their BD_ADDR to be unparked.

8.13.5 Access request address (AR_ADDR)

The access request address is used by the parked slave to determine the slave-to-master half slot in the access window it is allowed to send access request messages in (see also 8.10.8.4.6). The AR_ADDR is assigned to the slave when it enters the park mode and is only valid as long as the slave is parked. The AR_ADDR is not necessarily unique; i.e., different parked slaves may have the same AR_ADDR.

8.14 Bluetooth security

The Bluetooth technology provides peer-to-peer communications over short distances. In order to provide usage protection and information confidentiality, the system has to provide security measures both at the application layer and the link layer. These measures shall be appropriate for a peer environment. This means that in each Bluetooth unit, the authentication and encryption routines are implemented in the same way. Four different entities are used for maintaining security at the link layer: a public address which is unique for each user[11], two secret keys, and a random number which is different for each new transaction. The four entities and their sizes as used in Bluetooth are summarized in Table 35.

Table 35—Entities used in authentication and encryption procedures

Entity	Size
BD_ADDR	48 bits
Private user key, authentication	128 bits
Private user key, encryption configurable length (byte-wise)	8-128 bits
RAND	128 bits

The Bluetooth device address (BD_ADDR) is the 48-bit IEEE address which is unique for each Bluetooth unit. The Bluetooth addresses are publicly known, and can be obtained via MMI interactions, or, automatically, via an inquiry routine by a Bluetooth unit.

The secret keys are derived during initialization and are further never disclosed. Normally, the encryption key is derived from the authentication key during the authentication process. For the authentication algorithm, the size of the key used is always 128 bits. For the encryption algorithm, the key size may vary

[11]The BD_ADDR is not a secured identity.

between 1 and 16 octets (8–128 bits). The size of the encryption key shall be configurable for two reasons. The first has to do with the many different requirements imposed on cryptographic algorithms in different countries, both with respect to export regulations and official attitudes towards privacy in general. The second reason is to facilitate a future upgrade path for the security without the need of a costly redesign of the algorithms and encryption hardware; increasing the effective key size is the simplest way to combat increased computing power at the opponent side.

The encryption key is entirely different from the authentication key (even though the latter is used when creating the former, as is described in 8.14.5.4). Each time encryption is activated, a new encryption key shall be generated. Thus, the lifetime of the encryption key does not necessarily correspond to the lifetime of the authentication key.

It is anticipated that the authentication key will be more static in its nature than the encryption key, once established, the particular application running on the Bluetooth device decides when, or if, to change it. To underline the fundamental importance of the authentication key to a specific Bluetooth link, it will often be referred to as the link key.

The RAND is a random number which can be derived from a random or pseudo-random process in the Bluetooth unit. This is not a static parameter, it will change frequently.

In the remainder of this chapter, the terms user and application will be used interchangeably to designate the entity that is at the originating or receiving side.

8.14.1 Random number generation

Each Bluetooth unit has a random number generator. Random numbers are used for many purposes within the security functions, for instance, for the challenge-response scheme, for generating authentication and encryption keys, etc. Ideally, a true random generator based on some physical process with inherent randomness is used. Examples of such processes are thermal noise from a semiconductor or resistor and the frequency instability of a free running oscillator. For practical reasons, a software based solution with a pseudo-random generator is probably preferable. In general, it is quite difficult to classify the randomness of a pseudo-random sequence. Within Bluetooth, the requirements placed on the random numbers used are that they be nonrepeating and randomly generated.

The expression “nonrepeating” means that it shall be highly unlikely that the value should repeat itself within the lifetime of the authentication key. For example, a nonrepeating value could be the output of a counter that is unlikely to repeat during the lifetime of the authentication key, or a date/time stamp.

The expression “randomly generated” means that it shall not be possible to predict its value with a chance that is significantly larger than 0 (e.g., greater than $1/2^L$ for a key length of L bits).

Clearly, the LM can use such a generator for various purposes; i.e. whenever a random number is needed (such as the RANDs, the unit keys, K_{init}, K_{master}, and random back-off or waiting intervals).

8.14.2 Key management

It is important that the encryption key size within a specific unit cannot be set by the user; this should be a factory preset entity. In order to prevent the user from over-riding the permitted key size, the Bluetooth baseband processing does not accept an encryption key given from higher software layers. Whenever a new encryption key is required, it shall be created as defined in 8.14.5.4.

Changing a link key should also be done through the defined baseband procedures. Depending on what kind of link key it is, different approaches are required. The details are found in 8.14.2.2.7.

8.14.2.1 Key types

The link key is a 128-bit random number, which is shared between two or more parties and is the base for all security transactions between these parties. The link key itself is used in the authentication routine. Moreover, the link key is used as one of the parameters when the encryption key is derived.

In the following, a session is defined as the time interval for which the unit is a member of a particular piconet. Thus, the session terminates when the unit disconnects from the piconet.

The link keys are either semipermanent or temporary. A semipermanent link key is stored in nonvolatile memory and may be used after the current session is terminated. Consequently, once a semipermanent link key is defined, it may be used in the authentication of several subsequent connections between the Bluetooth units sharing it. The designation semipermanent is justified by the possibility to change it. How to do this is described in 8.14.2.2.7.

The lifetime of a temporary link key is limited by the lifetime of the current session. It cannot be reused in a later session. Typically, in a point-to-multipoint configuration where the same information is to be distributed securely to several recipients, a common encryption key is useful. To achieve this, a special link key (denoted master key) can temporarily replace the current link keys. The details of this procedure are found in 8.14.2.2.6.

In the sequel we sometimes refer to what is denoted as the current link key. This is simply the link key in use at the current moment. It can be semipermanent or temporary. Thus, the current link key is used for all authentications and all generation of encryption keys in the on-going connection (session).

In order to accommodate for different types of applications, four types of link keys have been defined:

— Combination key K_{AB}

— Unit key K_A

— Temporary key K_{master}

— Initialization key K_{init}

In addition to these keys there is an encryption key, denoted K_c. This key is derived from the current link key. Whenever the encryption is activated by a LM command, the encryption key shall be changed automatically. The purpose of separating the authentication key and encryption key is to facilitate the use of a shorter encryption key without weakening the strength of the authentication procedure. There are no governmental restrictions on the strength of authentication algorithms. However, in some countries, such restrictions exist on the strength of encryption algorithms.

For a Bluetooth unit, the combination key K_{AB} and the unit key K_A are functionally indistinguishable; the difference is in the way they are generated. The unit key K_A is generated in, and therefore dependent on, a single unit A. The unit key is generated once at installation of the Bluetooth unit; thereafter, it is very rarely changed. The combination key is derived from information in both units A and B, and is therefore always dependent on two units. The combination key is derived for each new combination of two Bluetooth units.

It depends on the application or the device whether a unit key or a combination key is used. Bluetooth units which have little memory to store keys, or, when installed in equipment that are accessible to a large group of users, will preferably use their own unit key. In that case, they only have to store a single key. Applications that require a higher security level preferably use the combination keys. These applications will require more memory since a combination key for each link to a different Bluetooth unit has to be stored.

The master key, K_{master}, is a link key only used during the current session. It will replace the original link key only temporarily. For example, this may be utilized when a master wants to reach more than two Bluetooth units simultaneously using the same encryption key (see 8.14.2.2.6).

The initialization key, K_{init}, is used as link key during the initialization process when no combination or unit keys have been defined and exchanged yet or when a link key has been lost. The initialization key protects the transfer of initialization parameters. The key is derived from a random number, an L-octet PIN code, and a BD_ADDR. This key is only to be used during initialization.

The PIN can be a fixed number provided with the Bluetooth unit (for example when there is no MMI as in a PSTN plug). Alternatively, the PIN can be selected arbitrarily by the user, and then entered in both units that have to be matched. The latter procedure is used when both units have an MMI, for example a phone and a laptop. Entering a PIN in both units is more secure than using a fixed PIN in one of the units, and should be used whenever possible. Even if a fixed PIN is used, it shall be possible to change the PIN; this in order to prevent re-initialization by users who once got hold of the PIN. If no PIN is available, a default value of zero is to be used.

For many applications the PIN code will be a relatively short string of numbers. Typically, it may consist of only four decimal digits. Even though this gives sufficient security in many cases, there exist countless other, more sensitive, situations where this is not reliable enough. Therefore, the PIN code can be chosen to be any length from 1 to 16 octets. For the longer lengths, we envision the units exchanging PIN codes not through mechanical (i.e. human) interaction, but rather through means supported by software at the application layer. For example, this can be a Diffie-Hellman key agreement, where the exchanged key is passed on to the K_{init} generation process in both units, just as in the case of a shorter PIN code.

8.14.2.2 Key generation and initialization

The link keys have to be generated and distributed among the Bluetooth units in order to be used in the authentication procedure. Since the link keys must be secret, they cannot be obtained through an inquiry routine in the same way as the Bluetooth addresses. The exchange of the keys takes place during an initialization phase which has to be carried out separately for each two units that want to implement authentication and encryption. All initialization procedures consist of the following five parts:

— Generation of an initialization key

— Generation of link key

— Link key exchange

— Authentication

— Generating of encryption key in each unit (optional)

After the initialization procedure, the units can proceed to communicate, or the link can be disconnected. If encryption is implemented, the E_0 algorithm is used with the proper encryption key derived from the current link key. For any new connection established between units A and B, they will use the common link key for authentication, instead of once more deriving K_{init} from the PIN. A new encryption key derived from that particular link key will be created next time encryption is activated.

If no link key is available, the LM shall automatically start an initialization procedure.

8.14.2.2.1 Generation of initialization key, K_{init}

A link key used temporarily during initialization is derived – the initialization key K_{init}. This key is derived by the E_{22} algorithm from a BD_ADDR, a PIN code, the length of the PIN (in octets), and a random number IN_RAND. The principle is depicted in Figure 87. The 128-bit output from E_{22} will be used for

key exchange during the generation of a link key. When the units have performed the link key exchange, the initialization key shall be discarded.

When the initialization key is generated, the PIN is augmented with the BD_ADDR. If one unit has a fixed PIN the BD_ADDR of the other unit is used. If both units have a variable PIN the BD_ADDR of the device that received IN_RAND is used. If both units have a fixed PIN they cannot be paired. Since the maximum length of the PIN used in the algorithm cannot exceed 16 octets, it is possible that not all octets of BD_ADDR will be used. This procedure ensures that K_{init} depends on the identity of the unit with a variable PIN. A fraudulent Bluetooth unit may try to test a large number of PINs by each time claiming another BD_ADDR. It is the application's responsibility to take countermeasures against this threat. If the device address is kept fixed, the waiting interval until next try is permitted is increased exponentially (see 8.14.4.1).

The details of the E_{22} algorithm can be found in 8.14.5.3.

8.14.2.2.2 Authentication

The authentication procedure is carried out as described in 8.14.4. Note that during each authentication, a new AU_RAND_A is issued.

Mutual authentication is achieved by first performing the authentication procedure in one direction and immediately followed by performing the authentication procedure in the opposite direction.

As a side effect of a successful authentication procedure an auxiliary parameter, the Authenticated Ciphering Offset (ACO), will be computed. The ACO is used for ciphering key generation as described in 8.14.2.2.5.

The claimant/verifier status is determined by the LM.

8.14.2.2.3 Generation of a unit key

A unit key K_A is generated when the Bluetooth unit is for the first time in operation; i.e. not during each initialization! The unit key is generated by the E_{21} algorithm as described in 8.14.5.3. Once created, the unit key is stored in non-volatile memory and (almost) never changed. If after initialization the unit key is changed, the previously initialized units will possess a wrong link key. At initialization, the application has to determine which of the two parties will provide the unit key as link key. Typically, this will be the unit with restricted memory capabilities, since this unit only has to remember its own unit key. The unit key is transferred to the other party and then stored as link key for that particular party. So, for example in Figure 73, the unit key of unit A, K_A, is being used as link key for the connection A-B; unit A sends the unit key K_A to unit B; unit B will store K_A as the link key K_{BA}. When the unit key has been exchanged, the initialization key shall be discarded in both units. For another initialization, for example with unit C, unit A will reuse its unit key K_A, whereas unit C stores it as K_{CA}.

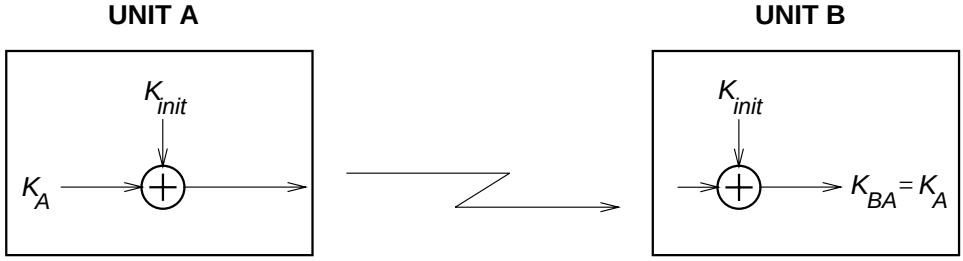


Figure 73—Generation of unit key

8.14.2.2.4 Generation of a combination key

If it is desired to use a combination key, this key is first generated during the initialization procedure. The combination key is the combination of two numbers generated in unit A and B, respectively. First, each unit generates a random number, say LK_RAND_A and LK_RAND_B. Then, utilizing E_{21} with the random number and the own BD_ADDR, the two random numbers

$$LK\_K_A = E_{21}(LK\_RAND_A, BD\_ADDR_A), \tag{23}$$

and

$$LK\_K_B = E_{21}(LK\_RAND_B, BD\_ADDR_B), \tag{24}$$

are created in unit A and unit B, respectively. These numbers constitute the units' contribution to the combination key that is to be created. Then, the two random numbers LK_RAND_A and LK_RAND_B are exchanged securely by XORing with the current link key, say K. Thus, unit A sends $K \oplus LK\_RAND_A$ to unit B, while unit B sends $K \oplus LK\_RAND_B$ to unit A. Clearly, if this is done during the initialization phase the link key $K = K_{init}$.

When the random numbers LK_RAND_A and LK_RAND_B have been mutually exchanged, each unit recalculates the other units contribution to the combination key. This is possible since each unit knows the Bluetooth device address of the other unit. Thus, unit A calculates Equation (24) and unit B calculates Equation (23). After this, both units combine the two numbers to generate the 128-bit link key. The combining operation is a simple bitwise modulo-2 addition (i.e. XOR). The result is stored in unit A as the link key K_{AB} and in unit B as the link key K_{BA}. When both units have derived the new combination key, a mutual authentication procedure shall be initiated to confirm the success of the transaction. The old link key shall be discarded after a successful exchange of a new combination key. The message flow between master and slave and the principle for creating the combination key is depicted in Figure 74. The old link key (*K*) shall be discarded after the exchange of a new combination key has succeeded.

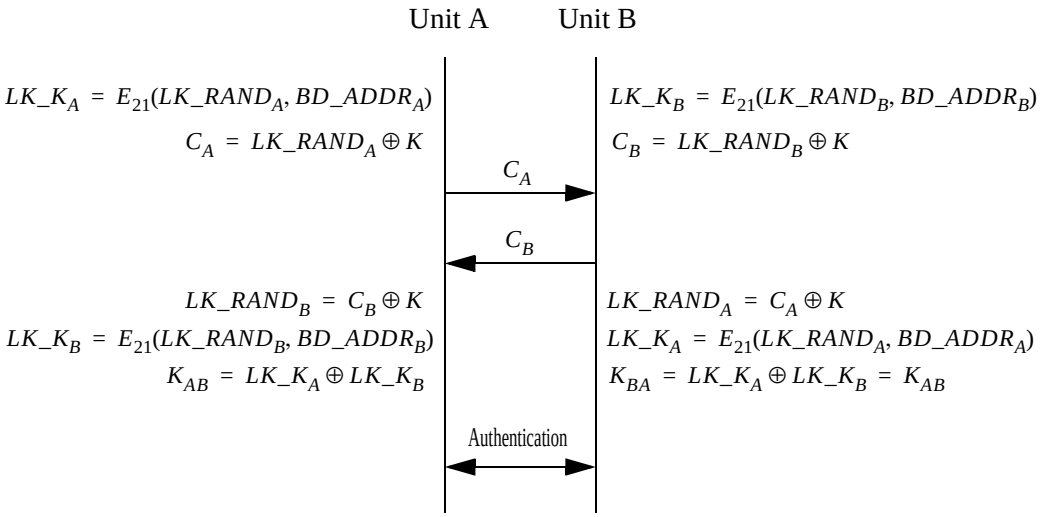


Figure 74—Generating a combination key

8.14.2.2.5 Generating the encryption key

The encryption key, K_C, is derived by algorithm E_3 from the current link key, a 96-bit Ciphering OFfset number (COF), and a 128-bit random number. The COF is determined in one of two ways. If the current link key is a master key, then COF is derived from the master BD_ADDR. Otherwise the value of COF is set to the value of ACO as computed during the authentication procedure. More precisely, we have[12]

$$\text{COF} = \begin{cases} \text{BD\_ADDR} \cup \text{BD\_ADDR}, & \text{if link key is a master key} \\ \text{ACO}, & \text{otherwise.} \end{cases} \quad (25)$$

There is an explicit call of E_3 when the LM activates encryption. Consequently, the encryption key is automatically changed each time the unit enters the encryption mode. The details of the key generating function E_3 can be found in 8.14.5.4.

8.14.2.2.6 Point-to-multipoint configuration

It is quite possible for the master to use separate encryption keys for each slave in a point-to-multipoint configuration with ciphering activated. Then, if the application requires more than one slave to listen to the same payload, each slave must be addressed individually. This may cause unwanted capacity loss for the piconet. Moreover, a Bluetooth unit (slave) is not capable of switching between two or more encryption keys in real time (e.g., after looking at the AM_ADDR in the header). Thus, the master cannot use different encryption keys for broadcast messages and individually addressed traffic. Alternatively, the master may tell several slave units to use a common link key (and, hence, indirectly also to use a common encryption key) and broadcast the information encrypted. For many applications, this key is only of temporary interest. In the sequel, this key is denoted K_{master}.

The transfer of necessary parameters is protected by the routine described in 8.14.2.2.8. After the confirmation of successful reception in each slave, the master shall issue a command to the slaves to replace their respective current link key by the new (temporary) master key. Before encryption can be activated, the master also has to generate and distribute a common EN_RAND to all participating slaves. Using this random number and the newly derived master key, each slave generates a new encryption key.

Note that the master must negotiate what encryption key length to use individually with each slave who wants to use the master key. In case the master already has negotiated with some of these slaves, it has knowledge of what sizes can be accepted. Clearly, there might be situations where the permitted key lengths of some units are incompatible. In that case, the master must have the limiting unit excluded from the group.

When all slaves have received the necessary data, the master can communicate information on the piconet securely using the encryption key derived from the new temporary link key. Clearly, each slave in possession of the master key can eavesdrop on all encrypted traffic, not only the traffic intended for itself. If so desired, the master can tell all participants to fall back to their old link keys simultaneously.

8.14.2.2.7 Modifying the link keys

In certain circumstances, it is desirable to be able to modify the link keys. A link key based on a unit key can be changed, but not very easily. The unit key is created once during the first use. Changing the unit key is a less desirable alternative, as several units may share the same unit key as link key (e.g., a printer whose unit key is distributed among all users using the printer's unit key as link key). Changing the unit key will require re-initialization of all units trying to connect. In certain cases, this might be desirable; for example, to deny access to previously allowed units.

[12] $x \cup y$ denotes the concatenation of the octet strings x and y.

If the key change concerns combination keys, then the procedure is rather straightforward. The change procedure is identical to the procedure illustrated in Figure 74, using the current value of the combination key as link key. This procedure can be carried out at any time after the authentication and encryption start. In fact, since the combination key corresponds to a single link, it can be modified each time this link is established. This will improve the security of the system since then old keys lose their validity after each session.

Of course, starting up an entirely new initialization procedure is also a possibility. In that case, user interaction is necessary since a PIN is required in the authentication and encryption procedures.

8.14.2.2.8 Generating a master key

The key-change routines described in 8.14.2.2.1 through 8.14.2.2.7 are semipermanent. To create the master link key, which can replace the current link key during an initiated session (see 8.14.2.2.6), other means are needed. First, the master creates a new link key from two 128-bit random numbers, RAND1 and RAND2. This is done by

$$K_{\text{master}} = E_{22}(\text{RAND1}, \text{RAND2}, 16). \quad (26)$$

Clearly, this key is a 128-bit random number. The reason to use the output of E_{22} and not directly chose a random number as the key, is to avoid possible problems with degraded randomness due to a poor implementation of the random number generator within the Bluetooth unit.

Then, a third random number, e.g., RAND, is transmitted to the slave. Using E_{22} with the current link key and RAND as inputs, both the master and slave computes a 128-bit overlay. The master sends the bitwise XOR of the overlay and the new link key to the slave. The slave, who knows the overlay, recalculates K_{master}. To confirm the success of this transaction, the units shall perform a mutual authentication procedure using the new link key. This procedure is then repeated for each slave who shall receive the new link key. The ACO values from the involved authentications should not replace the current existing ACO as this ACO is needed to (re)compute a ciphering key when the master wants to fall back to the previous link (non-temporary) key.

When so required, and potentially long after the actual distribution of the master key, the master activates encryption by an LM command. Before doing that, the master shall ensure that all slaves receive the same random number, say EN_RAND, since the encryption key is derived through the means of E_3 individually in all participating units. Then, each slave computes a new encryption key,

$$K_C = E_3(K_{\text{master}}, \text{EN\_RAND}, \text{COF}), \quad (27)$$

where the value of COF is derived from the master's BD_ADDR as specified by Equation (25). The details on the encryption key generating function can be found in 8.14.5.4. The principle of the message flow between the master and slave when generating the master key is depicted in Figure 75. Note that in this case the ACO produced during the authentication is not used when computing the ciphering key.

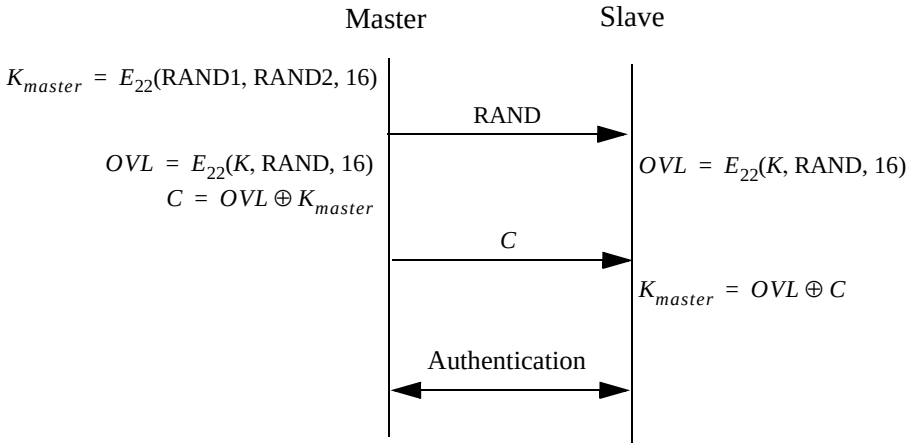


Figure 75—Master link key distribution and computation of the corresponding encryption key

8.14.3 Encryption

User information can be protected by encryption of the packet payload; the access code and the packet header are never encrypted. The encryption of the payloads is carried out with a stream cipher called E_0 that is re-synchronized for every payload. The overall principle is shown in Figure 76.

The stream cipher system E_0 consists of three parts. One part performs the initialization (generation of the payload key), the second part generates the key stream bits, and the third part performs the encryption and decryption. The payload key generator is very simple, it merely combines the input bits in an appropriate order and shift them into the four LFSRs used in the key stream generator. The main part of the cipher system is the second, as it also will be used for the initialization. The key stream bits are generated by a method derived from the summation stream cipher generator attributable to Massey and Rueppel [B12]. The method has been thoroughly investigated, and there exist good estimates of its strength with respect to presently known methods for cryptanalysis. Although the summation generator has weaknesses that can be used in so-called correlation attacks, the high re-synchronization frequency will disrupt such attacks.

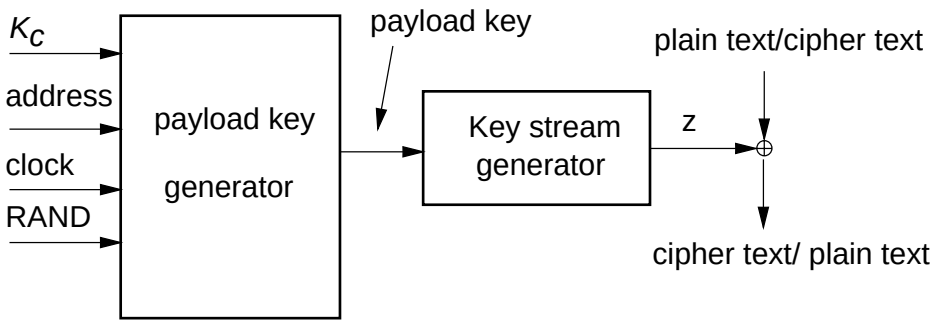


Figure 76—Stream ciphering for Bluetooth with E_0

8.14.3.1 Encryption key size negotiation

Each Bluetooth device implementing the baseband specification needs a parameter defining the maximal allowed key length, L_{max}, $1 \leq L_{max} \leq 16$ (number of octets in the key). For each application, a number L_{min} is defined indicating the smallest acceptable key size for that particular application. Before generating the encryption key, the involved units shall negotiate to decide what key size to actually use.

The master sends a suggested value, $L_{sug}^{(M)}$, to the slave. Initially, the suggested value is set to $L_{max}^{(M)}$. If $L_{min}^{(S)} \leq L_{sug}^{(M)}$, and, the slave supports the suggested length, the slave acknowledges, and this value will be the length of the encryption key for this link. However, if both conditions are not fulfilled, the slave sends a new proposal, $L_{sug}^{(S)} < L_{sug}^{(M)}$, to the master. This value should be the largest among all supported lengths less than the previous master suggestion. Then, the master performs the corresponding test on the slave suggestion. This procedure is repeated until a key length agreement is reached, or, one unit aborts the negotiation. An abortion may be caused by lack of support for L_{sug} and all smaller key lengths, or if $L_{sug} < L_{min}$ in one of the units. In case of abortion, Bluetooth link encryption can not be employed.

The possibility of a failure in setting up a secure link is an unavoidable consequence of letting the application decide whether to accept or reject a suggested key size. However, this is a necessary precaution. Otherwise, a fraudulent unit could enforce a weak protection on a link by claiming a small maximum key size.

8.14.3.2 Encryption modes

If a slave has a semipermanent link key (i.e., a combination key or a unit key), it can only accept encryption on slots individually addressed to itself (and, of course, in the reverse direction to the master). In particular, it will assume that broadcast messages are not encrypted. The possible traffic modes are described in Table 36. When an entry in the table refers to a link key, it means that the encryption/decryption engine uses the encryption key derived from that link key.

Table 36—Possible traffic modes for a slave using a semipermanent link key

Broadcast traffic	Individually addressed traffic
No encryption	No encryption
No encryption	Encryption, Semi-permanent link key

If the slave has received a master key, there are three possible combinations, as defined in Table 37. In this case, all units in the piconet uses a common link key, K_{master}. Since the master uses encryption keys derived from this link key for all secure traffic on the piconet, it is possible to avoid ambiguity in the participating slaves on which encryption key to use. Also in this case, the default mode is that broadcast messages are not encrypted. A specific LM-command is required to activate encryption, both for broadcast and for individually addressed traffic.

The master can issue an LM-command to the slaves telling them to fall back to their previous semipermanent link key. Then, regardless of the previous mode they were in, they will end up in the first row of Table 36; i.e., no encryption.

Table 37—Possible encryption modes for a slave in possession of a master key

Broadcast traffic	Individually addressed traffic
No encryption	No encryption
No encryption	Encryption, K_{master}
Encryption, K_{master}	Encryption, K_{master}

8.14.3.3 Encryption concept

For the encryption routine, a stream cipher algorithm will be used in which ciphering bits are bit-wise modulo-2 added to the data stream to be sent over the air interface. The payload is ciphered after the CRC bits are appended, but, prior to the FEC encoding.

Each packet payload is ciphered separately. The cipher algorithm E_0 uses the master Bluetooth address, 26 bits of the master real-time clock ($CLK_{26\text{-}1}$) and the encryption key K_C as input, see Figure 77 (where it is assumed that unit A is the master).

The encryption key K_C is derived from the current link key, COF, and a random number, EN_RAND_A (see 8.14.5.4). The random number is issued by the master before entering encryption mode. Note that EN_RAND_A is publicly known since it is transmitted as plain text over the air.

Within the E_0 algorithm, the encryption key K_C is modified into another key denoted K'_C. The maximum effective size of this key is factory preset and may be set to any multiple of eight between one and sixteen (8–128 bits). The procedure for deriving the key is described in 8.14.3.5.

The real-time clock is incremented for each slot. The E_0 algorithm is re-initialized at the start of each new packet (i.e., for master-to-slave as well as for slave-to-master transmission). By using $CLK_{26\text{-}1}$ at least one bit is changed between two transmissions. Thus, a new keystream is generated after each re-initialization. For packets covering more than a single slot, the Bluetooth clock as found in the first slot is being used for the entire packet.

The encryption algorithm E_0 generates a binary keystream, K_{cipher}, which is modulo-2 added to the data to be encrypted. The cipher is symmetric; decryption is performed in exactly the same way using the same key as used for encryption.

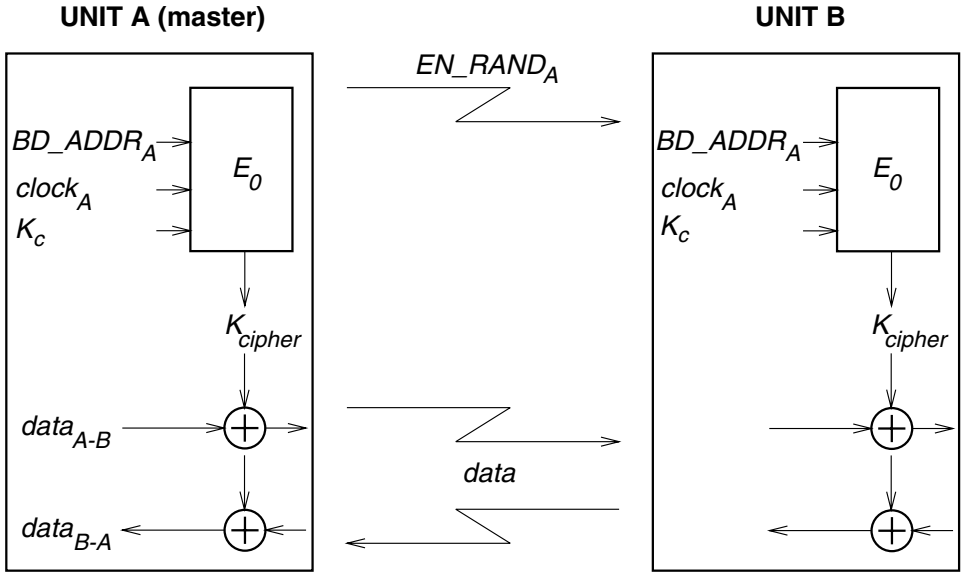


Figure 77—Functional description of the encryption procedure

8.14.3.4 Encryption algorithm

The system uses linear feedback shift registers (LFSRs) whose output is combined by a simple finite state machine (called the summation combiner) with 16 states. The output of this state machine is the key stream sequence, or, during initialization phase, the randomized initial start value. The algorithm is presented with an encryption key K_C, an 48-bit Bluetooth address, the master clock bits $CLK_{26\text{-}1}$, and a 128-bit RAND value. Figure 78 shows the setup.

There are four LFSRs ($LFSR_1$,...,$LFSR_4$) of lengths $L_1 = 25$, $L_2 = 31$, $L_3 = 33$, and, $L_4 = 39$, with feedback polynomials as specified in Table 38. The total length of the registers is 128. These polynomials are all primitive. The Hamming weight of all the feedback polynomials is chosen to be five, a reasonable trade-off between reducing the number of required XOR gates in the hardware realization and obtaining good statistical properties of the generated sequences.

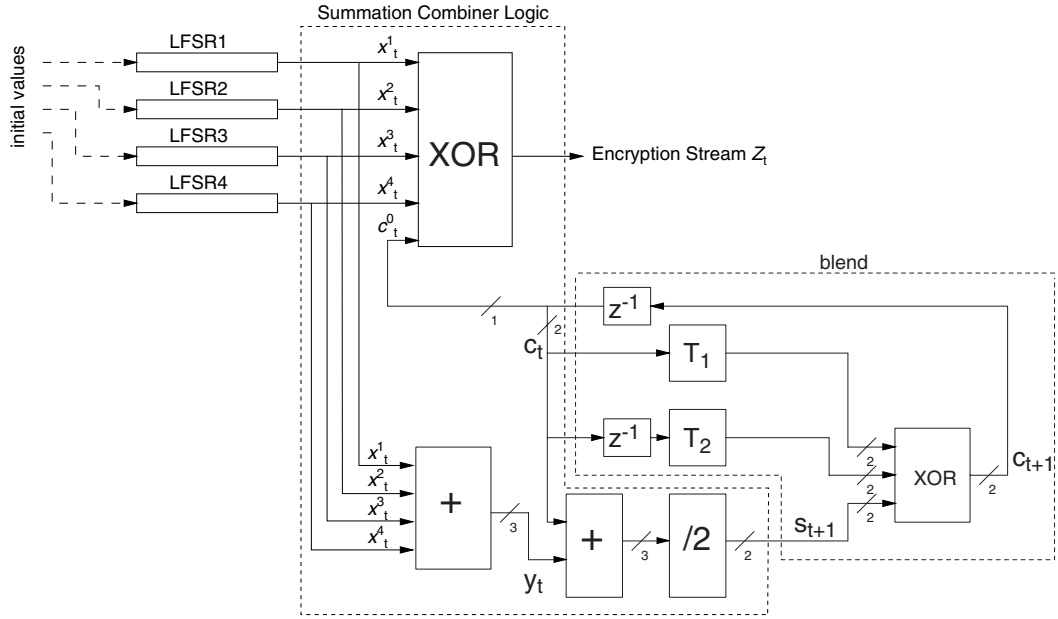


Figure 78—Concept of the encryption engine

Table 38—The four primitive feedback polynomials

i	L_i	**feedback** $f_i(t)$	**weight**
1	25	$t^{25} + t^{20} + t^{12} + t^8 + 1$	5
2	31	$t^{31} + t^{24} + t^{16} + t^{12} + 1$	5
3	33	$t^{33} + t^{28} + t^{24} + t^4 + 1$	5
4	39	$t^{39} + t^{36} + t^{28} + t^4 + 1$	5

Let x_t^i denote the t^{th} symbol of LSFR_i. From the four-tuple $x_t^1, \ldots, x_t^4$ we derive the value y_t as follows:

$$y_t = \sum_{i=1}^{4} x_t^i, \tag{28}$$

where the sum is over the integers. Thus y_t can take the values 0, 1, 2, 3, or 4. The output of the summation generator is now given by the following equations:

$$z_t = x_t^1 \oplus x_t^2 \oplus x_t^3 \oplus x_t^4 \oplus c_t^0 \in \{0, 1\}, \tag{29}$$

$$s_{t+1} = (s^1_{t+1}, s^0_{t+1}) = \left\lfloor \frac{y_t + c_t}{2} \right\rfloor \in \{0, 1, 2, 3\}, \quad (30)$$

$$c_{t+1} = (c^1_{t+1}, c^0_{t+1}) = s_{t+1} \oplus T_1[c_t] \oplus T_2[c_{t-1}], \quad (31)$$

where $T_1[.]$ and $T_2[.]$ are two different linear bijections over GF(4). Suppose GF(4) is generated by the irreducible polynomial $x^2 + x + 1$, and let α be a zero of this polynomial in GF(4). The mappings T_1 and T_2 are now defined as follows:

$$T_1: \text{GF(4)} \rightarrow \text{GF(4)}$$

$$x \mapsto x$$

$$T_2: \text{GF(4)} \rightarrow \text{GF(4)}$$

$$x \mapsto (\alpha + 1)x.$$

Table 39 summarizes the elements of GF(4) written as binary vectors.

Table 39—Mappings T_1 and T_2

x	$T_1[x]$	$T_2[x]$
00	00	00
01	01	11
10	10	01
11	11	10

Since the mappings are linear, the can be realized using XOR gates; i.e.,

$$T_1: \quad (x_1, x_0) \mapsto (x_1, x_0),$$

$$T_2: \quad (x_1, x_0) \mapsto (x_0, x_1 \oplus x_0).$$

8.14.3.4.1 The operation of the cipher

Figure 79 gives an overview of the operation in time. The encryption algorithm shall run through the initialization phase before the start of transmitting or receiving a new packet. Thus, for multislot packets the cipher is initialized using the clock value of the first slot in the multislot sequence.

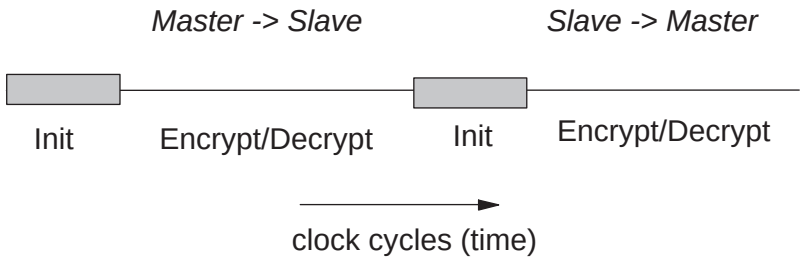


Figure 79—Overview of the operation of the encryption engine. Between each start of a packet (TX or RX), the LFSRs are re-initialized

8.14.3.5 LFSR initialization

The key stream generator needs to be loaded with an initial value for the four LFSRs (in total 128 bits) and the 4 bits that specify the values of c_0 and c_{-1}. The 132-bit initial value is derived from four inputs by using the key stream generator itself. The input parameters are the key K_C, a 128-bit random number RAND, a 48-bit Bluetooth address, and the 26 master clock bits $CLK_{26\text{-}1}$.

The effective length of the encryption key can vary between 8 and 128 bits. Note that the actual key length as obtained from E_3 is 128 bits. Then, within E_0, the key length is reduced by a modulo operation between K_C and a polynomial of desired degree. After reduction, the result is encoded with a block code in order to distribute the starting states more uniformly. The operation is defined in Equation (32).

When the encryption key has been created, the LFSRs are loaded with their initial values. Then, 200 stream cipher bits are created by operating the generator. Of these bits, the last 128 are fed back into the key stream generator as an initial value of the four LFSRs. The values of c_t and c_{t-1} are kept. From this point on, when clocked, the generator produces the encryption (decryption) sequence, which is bitwise XORed to the transmitted (received) payload data.

In the following, octet *i* of a binary sequence X is denoted by the notation $X[i]$. We define bit 0 of X to be the LSB. Then, the LSB of $X[i]$ corresponds to bit $8i$ of the sequence X, the MSB of $X[i]$ is bit $8i + 7$ of X. For instance, bit 24 of the Bluetooth address is the LSB of ADR[3].

The details of the initialization are as follows:

1) Create the encryption key to use from the 128-bit secret key K_C and the 128-bit publicly known EN_RAND. Let L, $1 \le L \le 16$, be the effective key length in number of octets. The resulting encryption key will be denoted K'_C:

$$K'_C(x) = g_2^{(L)}(x)(K_C(x) \bmod g_1^{(L)}(x)), \tag{32}$$

where $\deg(g_1^{(L)}(x)) = 8L$ and $\deg(g_2^{(L)}(x)) \le 128 - 8L$. The polynomials are defined in Table 40.

2) Shift in the 3 inputs K'_C, the Bluetooth address, the clock, and the six-bit constant 111001 into the LFSRs. In total 208 bits are shifted in.

   i) Open all switches shown in Figure 80.

   ii) Arrange inputs bits as shown in Figure 80; Set the content of all shift register elements to zero. Set $t = 0$.

iii) Start shifting bits into the LFSRs. The rightmost bit at each level of Figure 80 is the first bit to enter the corresponding LFSR.

iv) When the first input bit at level i reaches the rightmost position of $LFSR_i$, close the switch of this LFSR.

v) At $t = 39$ (when the switch of $LFSR_4$ is closed), reset both blend registers $c_{39} = c_{39-1} = 0$; Up to this point, the content of c_t and c_{t-1} has been of no concern. However, from this moment forward their content will be used in computing the output sequence.

vi) From now on output symbols are generated. The remaining input bits are continuously shifted into their corresponding shift register. When the last bit has been shifted in, the shift register is clocked with input = 0.

Note that when finished, $LFSR_1$ has effectively clocked 30 times with feedback closed, $LFSR_2$ has clocked 24 times, $LFSR_3$ has clocked 22 times, and $LFSR_4$ has effectively clocked 16 times with feedback closed.

7) To mix initial data, continue to clock until 200 symbols have been produced with all switches closed ($t = 239$).

8) Keep blend registers c_t and c_{t-1}, make a parallel load of the last 128 generated bits into the LFSRs according to Figure 81 at $t = 240$.

After the parallel load in item 4, the blend register contents will be updated for each subsequent clock.

Table 40—Polynomials used when creating K'_C [a]

L	deg	$g_1^{(L)}$	deg	$g_2^{(L)}$
1	[8]	00000000 00000000 00000000 0000011d	[119]	00e275a0 abd218d4 cf928b9b bf6cb08f
2	[16]	00000000 00000000 00000000 0001003f	[112]	0001e3f6 3d7659b3 7f18c258 cff6efef
3	[24]	00000000 00000000 00000000 010000db	[104]	000001be f66c6c3a b1030a5a 1919808b
4	[32]	00000000 00000000 00000001 000000af	[96]	00000001 6ab89969 de17467f d3736ad9
5	[40]	00000000 00000000 00000100 00000039	[88]	00000000 01630632 91da50ec 55715247
6	[48]	00000000 00000000 00010000 00000291	[77]	00000000 00002c93 52aa6cc0 54468311
7	[56]	00000000 00000000 01000000 00000095	[71]	00000000 000000b3 f7fffce2 79f3a073
8	[64]	00000000 00000001 00000000 0000001b	[63]	00000000 00000000 a1ab815b c7ec8025
9	[72]	00000000 00000100 00000000 00000609	[49]	00000000 00000000 0002c980 11d8b04d
10	[80]	00000000 00010000 00000000 00000215	[42]	00000000 00000000 0000058e 24f9a4bb
11	[88]	00000000 01000000 00000000 0000013b	[35]	00000000 00000000 0000000c a76024d7
12	[96]	00000001 00000000 00000000 000000dd	[28]	00000000 00000000 00000000 1c9c26b9
13	[104]	00000100 00000000 00000000 0000049d	[21]	00000000 00000000 00000000 0026d9e3
14	[112]	00010000 00000000 00000000 0000014f	[14]	00000000 00000000 00000000 00004377
15	[120]	01000000 00000000 00000000 000000e7	[7]	00000000 00000000 00000000 00000089
16	[128]	1 00000000 00000000 00000000 00000000	[0]	00000000 00000000 00000000 00000001

[a]All polynomials are in hexadecimal notation. The LSB is in the rightmost position.

In Figure 80, all bits are shifted into the LFSRs, starting with the least significant bit (LSB). For instance, from the third octet of the address, ADR[2], first ADR_{16} is entered, followed by ADR_{17}, etc. Furthermore, CL_0 corresponds to CLK_1, ..., CL_{25} corresponds to CLK_{26}.

Note that the output symbols x_t^i, $i = 1, \ldots, 4$ are taken from the positions 24, 24, 32, and 32 for $LFSR_1$, $LFSR_2$, $LFSR_3$, and $LFSR_4$, respectively (counting the leftmost position as number 1).

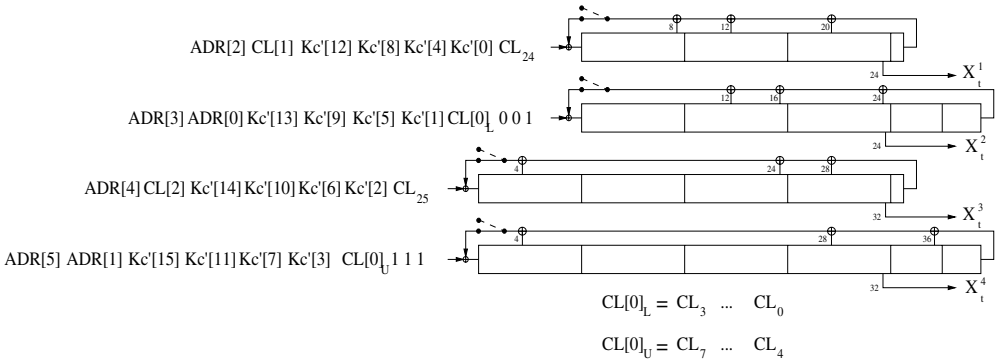


Figure 80—Arranging the input to the LFSRs

In Figure 81, the 128 binary output symbols Z_0, ..., Z_{127} are arranged in octets denoted Z[0], ..., Z[15]. The LSB of Z[0] corresponds to the first of these symbols, the MSB of Z[15] is the latest output from the generator. These bits shall be loaded into the LFSRs according to the figure. It is a parallel load and no update of the blend registers is done. The first output symbol is generated at the same time. The octets are written into the registers with the LSB in the leftmost position (i.e., the opposite of before). For example, Z_{24} is loaded into position 1 of $LFSR_4$.

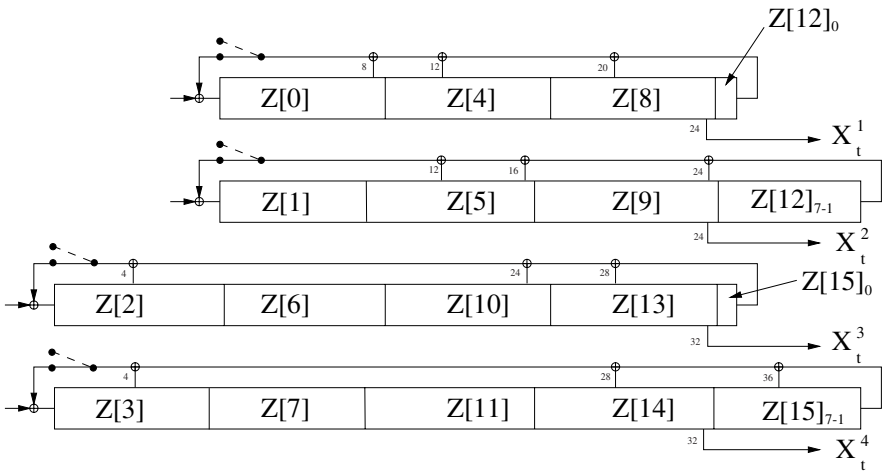


Figure 81—Distribution of the 128 last generated output symbols within the LFSRs

8.14.3.6 Key stream sequence

When the initialization is finished, the output from the summation combiner is used for encryption/decryption. The first bit to use is the one produced at the parallel load, i.e., at $t = 240$. The circuit is run for the entire length of the current payload. Then, before the reverse direction is started, the entire initialization process is repeated with updated values on the input parameters.

Sample data of the encryption output sequence can be found in the 2.4.1, Encryption Sample Data. A necessary, but not sufficient, condition for all Bluetooth-compliant implementations is to produce these encryption streams for identical initialization values.

8.14.4 Authentication

Authentication in a Bluetooth piconet uses a challenge-response scheme in which a claimant's knowledge of a secret key is checked through a two-move protocol using symmetric secret keys. The latter implies that a correct claimant/verifier pair share the same secret key, for example K. In the challenge-response scheme the verifier challenges the claimant to authenticate a random input (the challenge), denoted by AU_RAND_A, with an authentication code, denoted by E_1, and return the result SRES to the verifier (see Figure 82). This figure shows also that in Bluetooth the input to E_1 consists of the tuple AU_RAND_A and the Bluetooth device address (BD_ADDR) of the claimant. The use of this address prevents a simple reflection attack.[13] The secret K shared by units A and B is the current link key.

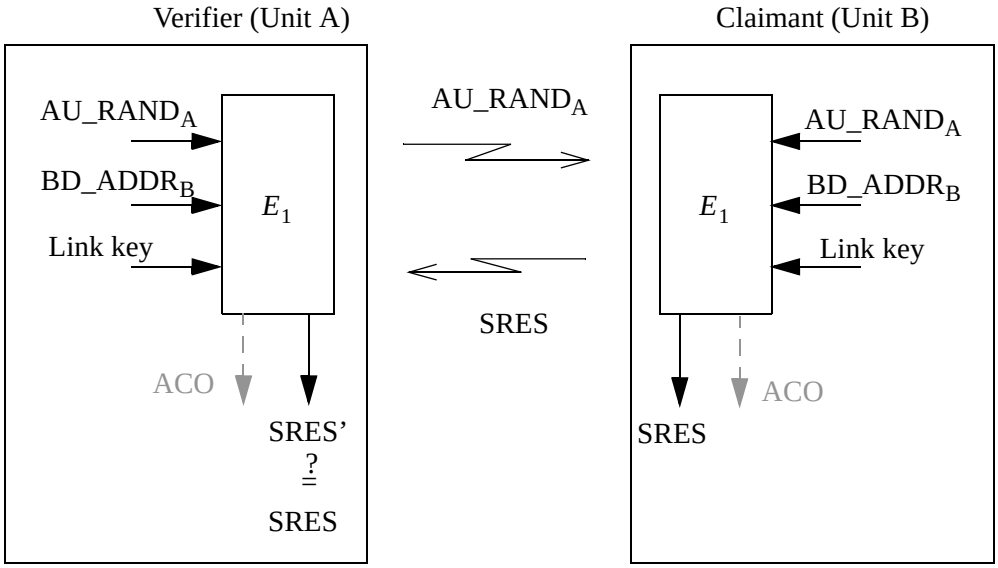


Figure 82—Challenge-response for the Bluetooth

[13]The reflection attack actually forms no threat in Bluetooth because all service requests are dealt with on a FIFO bases. When preemption is introduced, this attack is potentially dangerous.

The challenge-response scheme for symmetric keys used in the Bluetooth is depicted in Figure 83.

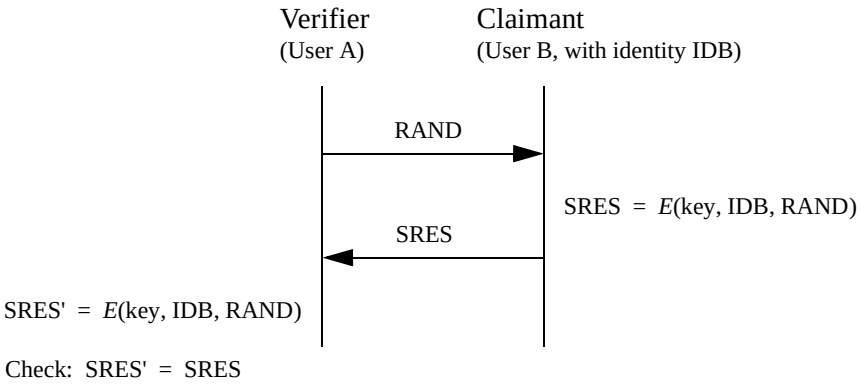


Figure 83—Challenge-response for symmetric key systems

In Bluetooth authentication, the verifier is not necessarily the master. The application indicates who has to be authenticated by whom. Certain applications only require a one-way authentication. However, in some peer-to-peer communications, one might prefer a mutual authentication in which each unit is subsequently the challenger (verifier) in two authentication procedures. The LM coordinates the indicated authentication preferences by the application to determine in which direction(s) the authentication(s) has to take place. For mutual authentication with the units of Figure 82, after unit A has successfully authenticated unit B, unit B could authenticate unit A by sending a AU_RAND_B (different from the AU_RAND_A that unit A issued) to unit A, and deriving the SRES and SRES' from the new AU_RAND_B, the address of unit A, and the link key K_{AB}.

If an authentication is successful, the value of ACO as produced by E_1 should be retained.

8.14.4.1 Repeated attempts

When the authentication attempt fails, a certain waiting interval shall pass before the verifier will initiate a new authentication attempt to the same claimant, or before it will respond to an authentication attempt initiated by a unit claiming the same identity as the suspicious unit. For each subsequent authentication failure with the same Bluetooth address, the waiting interval shall be increased exponentially. That is, after each failure, the waiting interval before a new attempt can be made, for example, twice as long as the waiting interval prior to the previous attempt[14]. The waiting interval shall be limited to a maximum. The maximum waiting interval depends on the implementation. The waiting time shall exponentially decrease to a minimum when no new failed attempts are being made during a certain time period. This procedure prevents an intruder from repeating the authentication procedure with a large number of different keys.

To make the system somewhat less vulnerable to denial-of-service attacks, the Bluetooth units should keep a list of individual waiting intervals for each unit it has established contact with. Clearly, the size of this list must be restricted only to contain the *N* units with which the most recent contact has been made. The number *N* can vary for different units depending on available memory size and user environment.

[14]Another appropriate value larger than 1 may be used.

8.14.5 The authentication and key-generating functions

This subclause describes the algorithmic means for supporting the Bluetooth security requirements on authentication and key generation.

8.14.5.1 The authentication function E_1

The authentication function E_1 proposed for the Bluetooth is a computationally secure authentication code, or often called a MAC. E_1 uses the encryption function called SAFER+. The algorithm is an enhanced version of an existing 64-bit block cipher SAFER-SK128, and it is freely available. In the sequel, the block cipher will be denoted as the function A_r which maps under a 128-bit key, a 128-bit input to a 128-bit output, i.e.,

$$\begin{aligned} A_r\colon\ \{0,1\}^{128}\times\{0,1\}^{128} &\rightarrow \{0,1\}^{128} \\ (k\times x) &\mapsto t. \end{aligned} \tag{33}$$

The details of A_r are given in 8.14.5.2. The function E_1 is constructed using A_r as follows:

$$\begin{aligned} E_1\colon\ \{0,1\}^{128}\times\{0,1\}^{128}\times\{0,1\}^{48} &\rightarrow \{0,1\}^{32}\times\{0,1\}^{96} \\ (K, \text{RAND}, \text{address}) &\mapsto (\text{SRES}, \text{ACO}), \end{aligned} \tag{34}$$

where $\text{SRES} = Hash(K, \text{RAND}, \text{address}, 6)[0, \ldots, 3]$, where $Hash$ is a keyed hash function defined as[15]

$$\begin{aligned} &Hash\colon\{0,1\}^{128}\times\{0,1\}^{128}\times\{0,1\}^{8\times L}\times\{6,12\}\rightarrow\{0,1\}^{128} \\ &(K, I_1, I_2, L)\ \mapsto A'_r(\tilde{[K]}, [E(I_2, L) +_{16} (A_r(K, I_1)\oplus_{16} I_1)]), \end{aligned} \tag{35}$$

and where

$$\begin{aligned} &E\colon\{0,1\}^{8\times L}\times\{6,12\}\rightarrow\{0,1\}^{8\times 16} \\ &(X[0, \ldots, L-1], L)\ \mapsto (X[i(\text{mod L})]\ \text{ for } i = 0...15), \end{aligned} \tag{36}$$

is an expansion of the L octet word X into a 128-bit word. Thus we see that we have to evaluate the function A_r twice for each evaluation of E_1. The key K for the second use of A_r (actually A'_r) is offseted from K as follows:[16]

[15]The operator $+_{16}$ denotes bytewise addition mod 256 of the 16 octets, and the operator $\oplus_{16}$ denotes bytewise X-raying of the 16 octets.

[16]The constants are the first largest primes below 257 for which 10 is a primitive root.

$$
\begin{aligned}
\tilde{K}[0] &= (\tilde{K}[0] + 233) \mod 256, & \tilde{K}[1] &= \tilde{K}[1] \oplus 229,\\
\tilde{K}[2] &= (\tilde{K}[2] + 223) \mod 256, & \tilde{K}[3] &= \tilde{K}[3] \oplus 193,\\
\tilde{K}[4] &= (\tilde{K}[4] + 179) \mod 256, & \tilde{K}[5] &= \tilde{K}[5] \oplus 167,\\
\tilde{K}[6] &= (\tilde{K}[6] + 149) \mod 256, & \tilde{K}[7] &= \tilde{K}[7] \oplus 131,\\
\tilde{K}[8] &= \tilde{K}[8] \oplus 233, & \tilde{K}[9] &= (\tilde{K}[9] + 229) \mod 256,\\
\tilde{K}[10] &= \tilde{K}[10] \oplus 223, & \tilde{K}[11] &= (\tilde{K}[11] + 193) \mod 256,\\
\tilde{K}[12] &= \tilde{K}[12] \oplus 179, & \tilde{K}[13] &= (\tilde{K}[13] + 167) \mod 256,\\
\tilde{K}[14] &= \tilde{K}[14] \oplus 149, & \tilde{K}[15] &= (\tilde{K}[15] + 131) \mod 256.
\end{aligned}
\tag{37}
$$

A data flowchart of the computation of E_1 is depicted in Figure 84. E_1 is also used to deliver the parameter ACO (Authenticated Ciphering Offset) that is used in the generation of the ciphering key by E_3 [see Equation (25) and Equation (45)]. The value of ACO is formed by the octets 4 through 15 of the output of the hash function defined in Equation (35), i.e.,

$$\text{ACO} = Hash(K, \text{RAND}, \text{address}, 6)[4, \ldots, 15]. \tag{38}$$

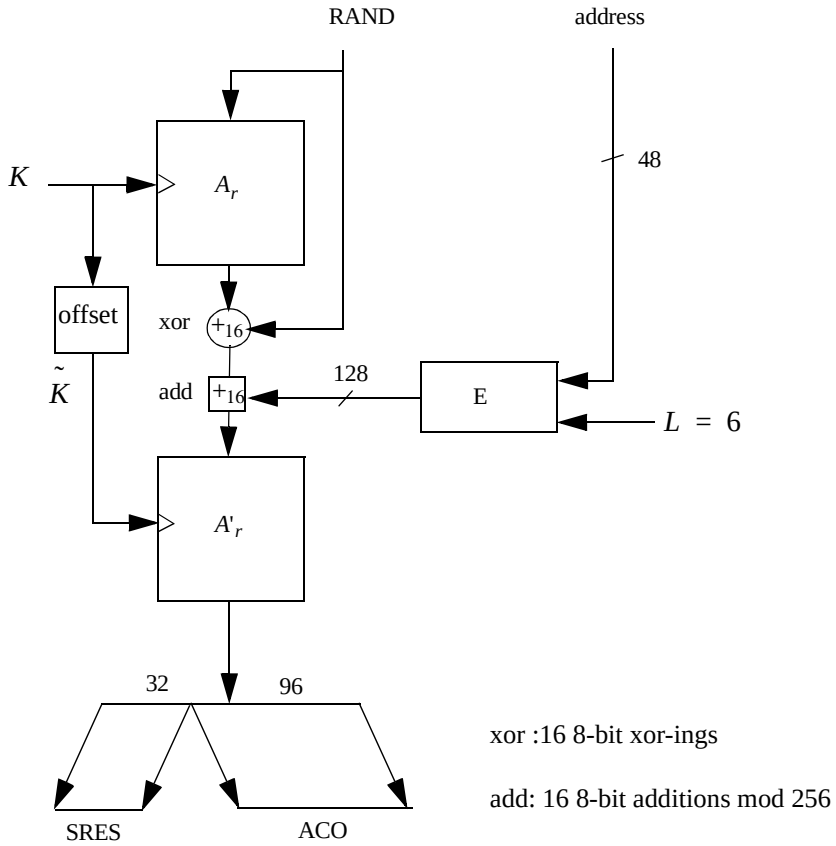


Figure 84—Flow of data for the computation of E_1

8.14.5.2 The functions A_r and A'_r

The function A_r is identical to SAFER+. It consists of a set of eight layers, (each layer is called a round) and a parallel mechanism for generating the sub keys $K_p[j]$, $p = 1, 2, \ldots, 17$, the so-called round keys to be used in each round. The function will produce a 128-bit result from a 128-bit "random" input string and a 128-bit "key". Besides the function A_r, a slightly modified version referred to as A'_r is used in which the input of round 1 is added to the input of the 3rd round. This is done to make the modified version non-invertible and prevents the use of A'_r (especially in E_{2x}) as an encryption function. See Figure 85 for details.

8.14.5.2.1 The round computations

The computations in each round are a composition of encryption with a round key, substitution, encryption with the next round key, and, finally, a Pseudo Hadamard Transform (PHT). The computations in a round are shown in Figure 85. The permutation boxes show how input byte indices are mapped onto output byte indices. Thus, position 0 (leftmost) is mapped on position 8, position 1 is mapped on position 11, etc. The sub keys for round $r, r = 1, 2, \ldots, 8$ are denoted $K_{2r-1}[j]$, $K_{2r}[j]$, $= 0, 1, \ldots, 15$. After the last round $k_{17}[j]$ is applied in a similar fashion as all previous odd numbered keys.

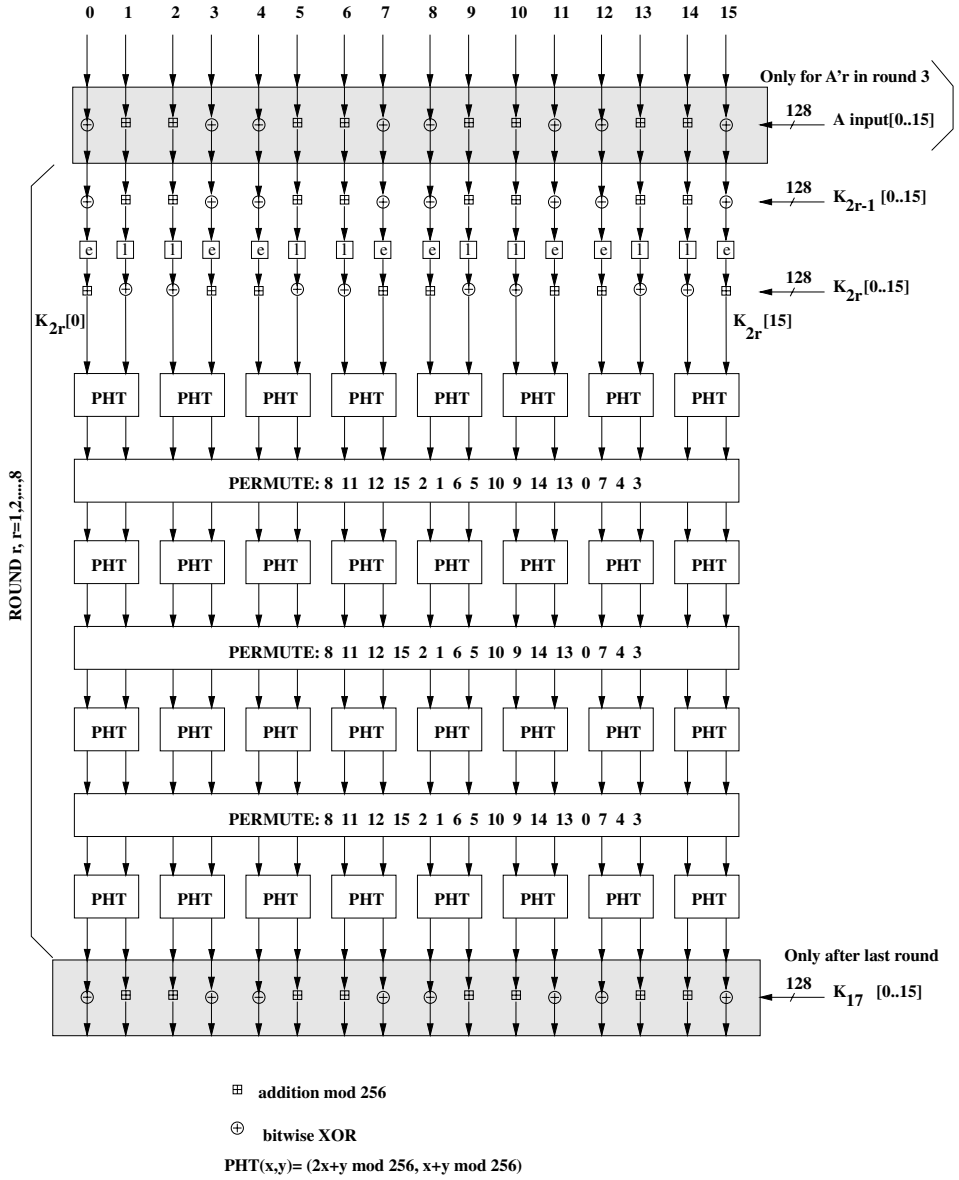


Figure 85—One round in A_r and A'_r

8.14.5.2.2 The substitution boxes “e” and “l”

In Figure 85 two boxes occur, marked “e” and “l”. These boxes implement the same substitutions as used in SAFER+; i.e., they implement the following:

$$
\begin{aligned}
e, l \quad &: \quad \{0, \ldots, 255\} \rightarrow \{0, \ldots, 255\}, \\
e \quad &: \quad i \mapsto (45^i \pmod{257}) \pmod{256}, \\
l \quad &: \quad i \mapsto j \text{ s.t. } i = e(j).
\end{aligned}
$$

Their role, as in the SAFER+ algorithm, is to introduce non-linearity.

8.14.5.2.3 Key scheduling

In each round, two batches of 16 octet-wide keys are needed. These so-called round keys are derived as specified by the key scheduling in SAFER+. Figure 86 gives an overview of how the round keys $K_p[j]$ are determined. The bias vectors B_2, B_3, ..., B_{17} are computed according to following equation:

$$B_p[i] = ((45^{(45^{17p+i+1} \bmod 257)} \bmod 257) \bmod 256), \text{ for } i = 0, \ldots, 15. \quad (39)$$

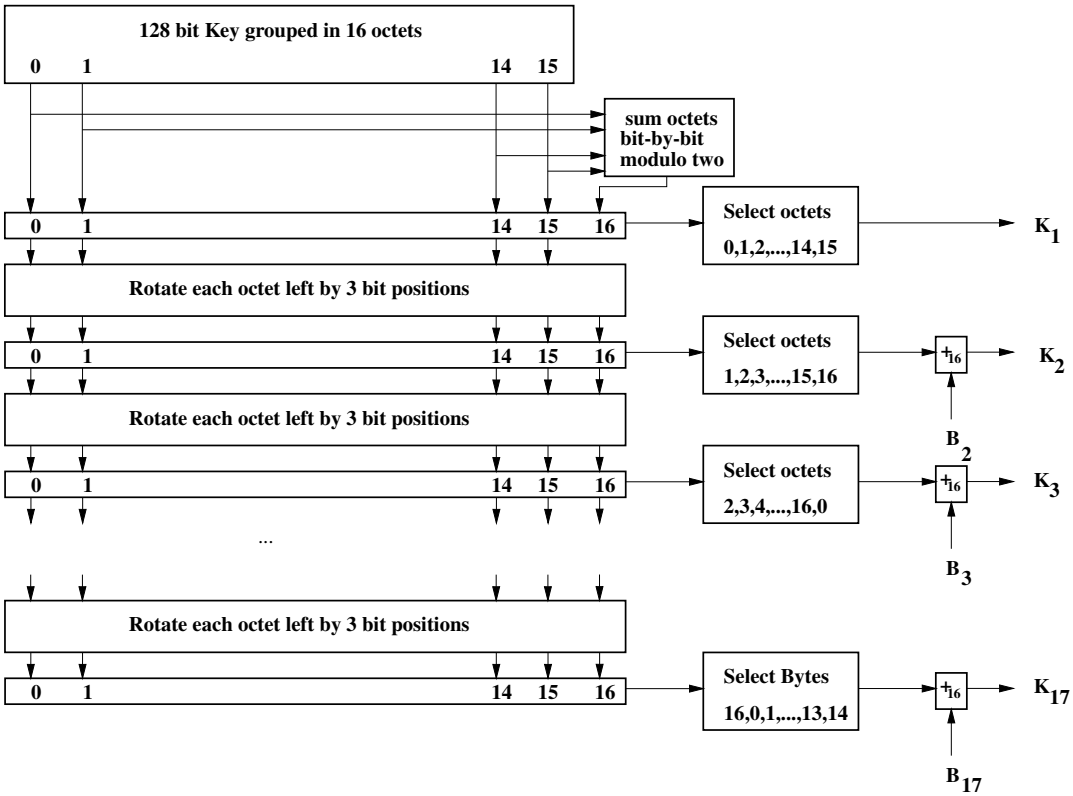


Figure 86—Key scheduling in A_r

8.14.5.3 E_2-key generation function for authentication

The key used for authentication is derived through a procedure that is shown in Figure 87. The figure shows two different modes of operation for the algorithm. In the first mode, the function E_2 should produce on input of a 128-bit RAND value and a 48-bit address, a 128-bit link key K. This mode is utilized when creating unit keys and combination keys. In the second mode the function E_2 should produce, on input of a 128-bit RAND value and an L octet user PIN, a 128-bit link key K. The second mode is used to create the initialization key, and also whenever a master key is to be generated.

NOTE—Mode 1 is used for unit and combination keys, while mode 2 is used for K_{init} and K_{master}.

When the initialization key is generated, the PIN is augmented with the BD_ADDR (see 8.14.2.2.1 for which address to use). The augmentation always starts with the least significant octet of the address immediately following the most significant octet of the PIN. Since the maximum length of the PIN used in the algorithm cannot exceed 16 octets, it is possible that not all octets of BD_ADDR will be used.

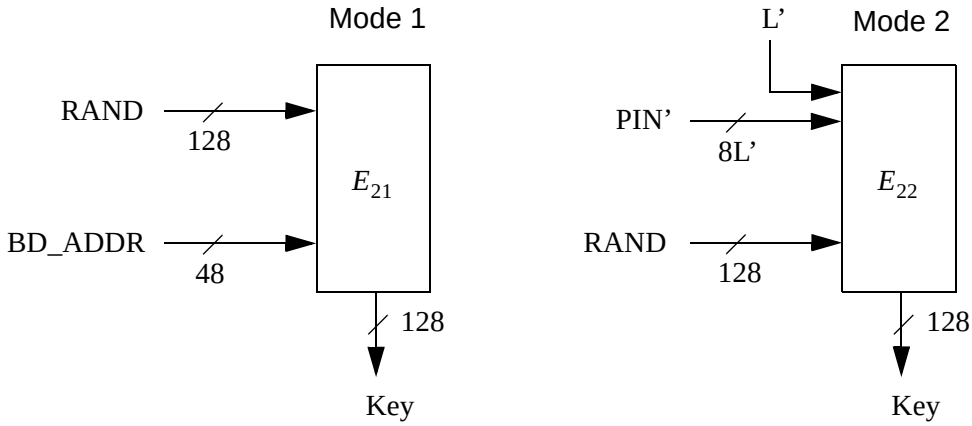


Figure 87—Key generating algorithm E_2 and its two modes

This key-generating algorithm again exploits the cryptographic function. Formally E_2 can be expressed for mode 1 (denoted E_{21}) as

$$E_{21}\colon \{0,1\}^{128} \times \{0,1\}^{48} \rightarrow \{0,1\}^{128}$$
$$(\text{RAND, address}) \mapsto A'_r(X, Y) \tag{40}$$

where (for mode 1)

$$\begin{cases} X = \text{RAND}[0 \ldots 14] \cup (\text{RAND}[15] \oplus 6) \\ Y = \bigcup_{i=0}^{15} \text{address}[i \pmod 6] \end{cases} \tag{41}$$

Let L be the number of octets in the user PIN. The augmenting is defined by

$$\text{PIN}' = \begin{cases} \text{PIN}[0 \ldots L-1] \cup \text{BD\_ADDR}[0 \ldots \min\{5, 15-L\}], & L < 16, \\ \text{PIN}[0 \ldots L-1], & L = 16, \end{cases} \tag{42}$$

Then, in mode 2, E_2 (denoted E_{22}) can be expressed as

$$E_{22}\colon \{0,1\}^{8L'} \times \{0,1\}^{128} \times \{1, 2, \ldots, 16\} \rightarrow \{0,1\}^{128}$$
$$(\text{PIN}', \text{RAND}, \text{L}') \mapsto A'_r(X, Y) \tag{43}$$

where

$$\begin{cases} X = \bigcup_{i=0}^{15} \text{PIN}'[i \pmod{L'}], \\ Y = \text{RAND}[0 \ldots 14] \cup (\text{RAND}[15] \oplus L'), \end{cases} \tag{44}$$

and $L' = \min\{16, L + 6\}$ is the number of octets in PIN'.

8.14.5.4 E_3-Key generation function for encryption

The ciphering key K_C used by E_0 is generated by E_3. The function E_3 is constructed using A'_r as follows:

$$\begin{aligned} E_3\colon \{0,1\}^{128} \times \{0,1\}^{128} \times \{0,1\}^{96} &\rightarrow \{0,1\}^{128} \\ (K, \text{RAND}, \text{COF}) &\mapsto Hash(K, \text{RAND}, \text{COF}, 12) \end{aligned} \tag{45}$$

where $Hash$ is the hash function as defined by Equation (35). Note that the produced key length is 128 bits. However, before use within E_0, the encryption key K_C will be shortened to the correct encryption key length, as described in 8.14.3.5. A block scheme of E_3 is depicted in Figure 88.

The value of COF is determined as specified by equation Equation (25).

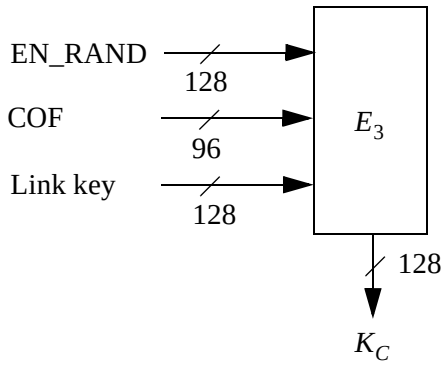


Figure 88—Generation of the encryption key

9. Link Manager Protocol

Figure 89 indicates the relationship of the Bluetooth protocol stack to this clause. This clause describes the Link Manager Protocol (LMP) which is used for link setup and control. The signals are interpreted and filtered out by the Link Manager on the receiving side and are not propagated to higher layers.

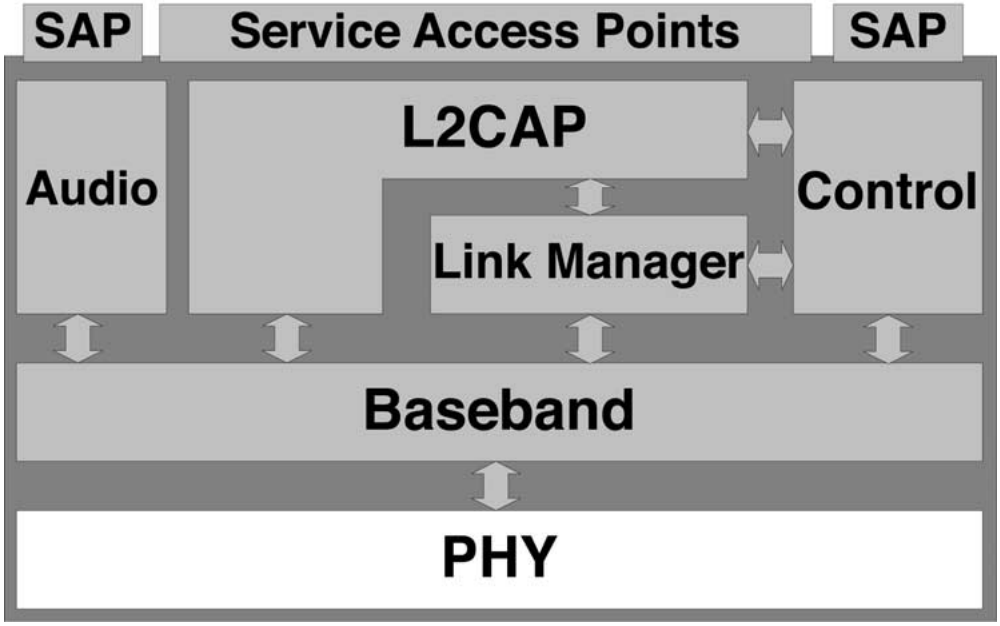


Figure 89—LM interface relationships

9.1 General

LMP messages are used for link setup, security, and control. They are transferred in the payload instead of L2CAP and are distinguished by a reserved value in the L_CH field of the payload header. The messages are filtered out and interpreted by LM on the receiving side and are not propagated to higher layers (see Figure 90).

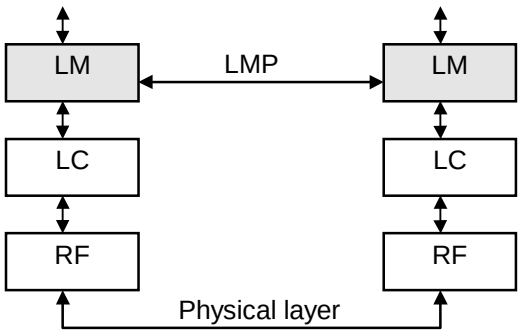


Figure 90—Link Manager's place on the global scene

Link Manager messages have higher priority than user data. This means that if the Link Manager needs to send a message, it shall not be delayed by the L2CAP traffic, although it can be delayed by many retransmissions of individual baseband packets.

We do not need to explicitly acknowledge the messages in LMP since LC (see 8.5) provides us with a reliable link.

LC does not guarantee either the time taken to deliver a message to the remote device or the delay between the delivery of the message to the remote device and the reception of the corresponding ACK by the sender. This means that we must be aware of the underlying LC mechanism's limitations to synchronize state changes between master and slave. The criteria for determining when the master can reuse an AM_ADDR following the detach or park of a slave is based on the reception of the Baseband-level acknowledgement. Synchronization of a master-slave switch or the starting of hold mode utilizes the Bluetooth master clock, which the LM reads from the LC.

LC only guarantees that it will attempt to communicate with each slave once per T_{poll} slots.

T_{poll} is the poll interval as described in 9.3.20.

The time between receiving a baseband packet carrying an LMP PDU and sending a baseband packet carrying a valid response PDU, according to the procedure rules in 9.3, shall be less than the LMP Response Timeout. The value of this timeout is 30 s. Note that the LMP Response Timeout is applied not only to sequences described in 9.3, but also to the series of the sequences defined as the transactions in 9.3. It is also applied to the series of the transactions, as long as no L2CAP PDUs are allowed, for example, any transactions until the PDUs LMP_setup_complete are exchanged.

9.2 Format of LMP

LM PDUs are always sent as single-slot packets, and the payload header is, therefore, 1 byte. The two least significant bits in the payload header determine the logical channel. For LM PDUs these bits are set. See Table 41.

Table 41—Logical channel L_CH field contents

L_CH code	Logical channel	Information
00	NA	Undefined
01	UA/I	Continuing L2CAP message
10	UA/I	Start L2CAP message
11	LM	LMP message

The FLOW bit in the payload header is always one and is ignored on the receiving side. Each PDU is assigned a 7-bit opcode used to uniquely identify different types of PDUs (see Table 67). The opcode and a 1-bit transaction ID are positioned in the first byte of the payload body (see Figure 91). The transaction ID is positioned in the LSB. It is 0 if the PDU belongs to a transaction initiated by the master and 1 if the PDU belongs to a transaction initiated by the slave. If the PDU contains one or more parameters these are placed in the payload starting at the second byte of the payload body. The number of bytes used depends on the length of the parameters. If an SCO link is present using HV1 packets and length of *content* is less than

9 bytes, the PDUs can be transmitted in DV packets. Otherwise DM1 packets must be used. All parameters have little endian format, i.e. the least significant byte is transmitted first.

The source/destination of the PDUs is determined by the AM_ADDR in the packet header.

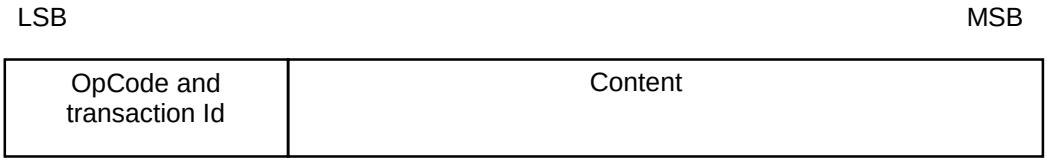


Figure 91—Payload body when LM PDUs are sent

Each PDU is either mandatory or optional. The M/O field in the tables of 9.3 indicates this. The LM does not need to be able to transmit a PDU that is optional. The LM shall recognize all optional PDUs that it receives and, if a response is required, send a valid response according to the procedure rules in 9.3. The reason that should be used in this case is *unsupported LMP feature*. If the optional PDU that is received does not require a response, no response is sent. Which of the optional PDUs a device supports can be requested (see 9.3.11).

Each sequence described in 9.3 is normally defined as a transaction. For pairing (see 9.3.3) or encryption (see 9.3.6), all sequences belonging to each section are counted as one transaction and shall use the same transaction ID. For connection establishment (see 9.4), LMP_host_connection_req and the response with LMP_accepted or LMP_not_accepted form one transaction and have the transaction ID of 0. LMP_setup_complete is a stand-alone PDU, which forms a transaction by itself. For error handling (see 9.7), the PDU to be rejected and LMP_not_accepted form a single transaction. Therefore the LMP_not_accepted shall have the same transaction ID as the PDU which is being rejected.

9.3 The Procedure rules and PDUs

Each procedure is described and depicted with a sequence diagram (see also Figure 92). The following symbols are used in the sequence diagrams:

PDU1 is a PDU sent from A to B. PDU2 is a PDU sent from B to A. PDU3 is a PDU that is optionally sent from A to B. PDU4 is a PDU that is optionally sent from B to A. PDU5 is a PDU sent from either A or B. A vertical line indicates that more PDUs can optionally be sent.

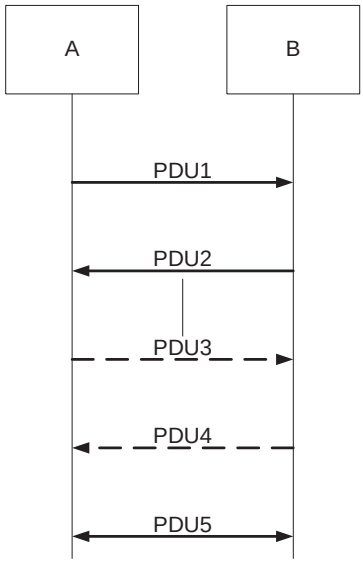


Figure 92—Symbols used in sequence diagrams

9.3.1 General response messages

The PDUs LMP_accepted and LMP_not_accepted are used as response messages to other PDUs in a number of different procedures. The PDU LMP_accepted includes the opcode of the message that is accepted. The PDU LMP_not_accepted includes the opcode of the message that is not accepted and the reason why it is not accepted. See Table 42.

Table 42—General response messages

M/O	PDU	Contents
M	LMP_accepted	op code
M	LMP_not_accepted	op code reason

9.3.2 Authentication

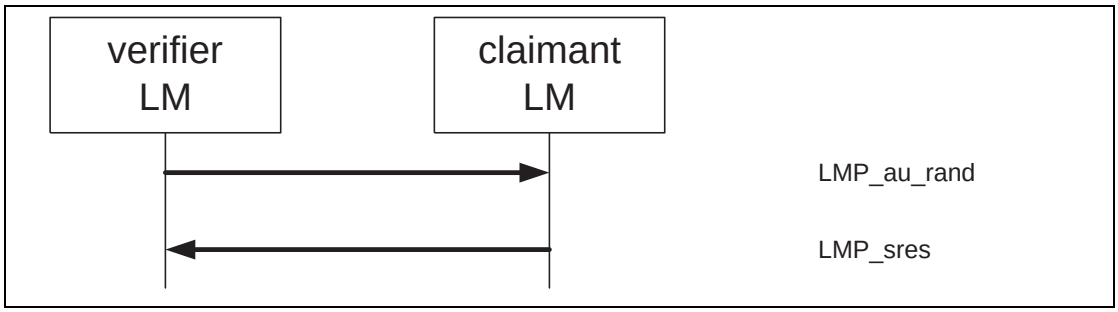
The authentication procedure is based on a challenge-response scheme as described in 8.14.4. The verifier sends an LMP_au_rand PDU which contains a random number (the challenge) to the claimant. The claimant calculates a response, which is a function of the challenge, the claimant's BD_ADDR and a secret key. The response is sent back to the verifier, which checks if the response was correct or not. How the response should be calculated is described in 8.14.5.1. A successful calculation of the authentication response requires that two devices share a secret key. How this key is created is described in 9.3.3. Both the master and the slave can be verifiers. The following PDUs, summarized in Table 43, are used in the authentication procedure:

Table 43—PDUs used for authentication

M/O	PDU	Contents
M	LMP_au_rand	random number
M	LMP_sres	authentication response

9.3.2.1 Claimant has link key

If the claimant has a link key associated with the verifier, it calculates the response and sends it to the verifier with LMP_sres. The verifier checks the response. If the response is not correct, the verifier can end the connection by sending LMP_detach with the reason code *authentication failure (*see 9.3.14).

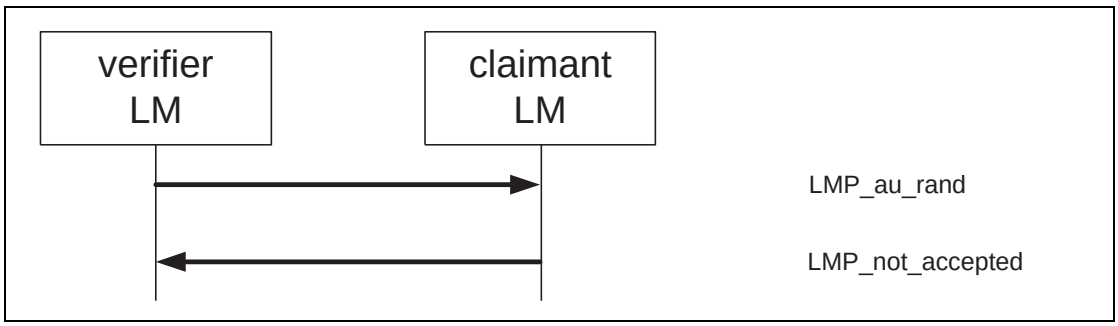


Sequence 1: Authentication. Claimant has link key.

If an LM receives LMP_au_rand and also wants to initiate an authentication it shall first reply with LMP_sres before starting its own challenge. There can, however, be concurrent requests caused by master and slave simultaneously initiating an authentication. To avoid that this results in different ACOs in the units, this situation is resolved by the rule outlined in 9.7: If the master sends LMP_au_rand and receives another LMP_au_rand before receiving LMP_sres it shall respond with LMP_not_accepted with the reason code *LMP Error Transaction Collision;* in that case the slave LM shall complete the master's challenge by sending LMP_sres and may then initiate its authentication again.

9.3.2.2 Claimant has no link key

If the claimant does not have a link key associated with the verifier it sends LMP_not_accepted with the reason code *key missing* after receiving LMP_au_rand.



Sequence 2: Authentication fails. Claimant has no link key.

9.3.2.3 Repeated attempts

The scheme described in 8.14.4.1 shall be applied when an authentication fails. This will prevent an intruder from trying a large number of keys in a relatively short time.

9.3.3 Pairing

When two devices do not have a common link key an initialization key (K_{init}) is created based on a PIN and a random number and a BD address. How the K_{init} is calculated is described in 8.14.5.3. When both devices have calculated K_{init} the link key is created, and finally a mutual authentication is made. The pairing procedure starts with a device sending LMP_in_rand; this device is referred to as "initiating LM" or "initiator" in 9.3.3.1 through 9.3.3.5. The other device is referred to as "responding LM" or "responder". The PDUs used in the pairing procedure are summarized in Table 44.

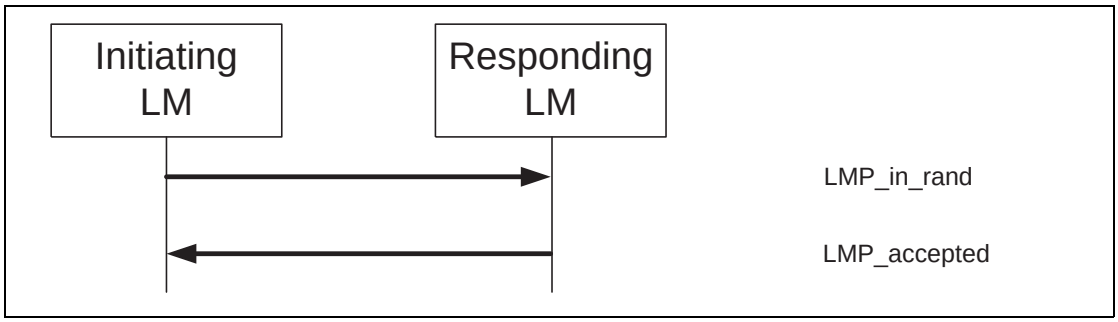
Table 44—PDUs used for pairing

M/O	PDU	Contents
M	LMP_in_rand	random number
M	LMP_au_rand	random number
M	LMP_sres	authentication response
M	LMP_comb_key	random number
M	LMP_unit_key	key

Note that all sequences described in 9.3, including the mutual authentication after the link key has been created, form a single transaction. The transaction ID from the first LMP_in_rand will be used for all subsequent sequences.

9.3.3.1 Responder accepts pairing

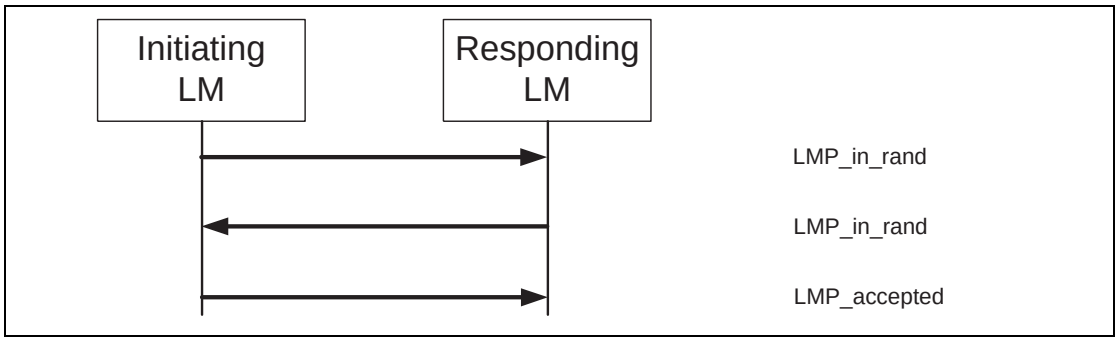
The initiator sends LMP_in_rand and the responder replies with LMP_accepted. Both devices calculate K_{init} based on the BD address of the responder and the procedure continues with creation of the link key (see 9.3.3.4).



Sequence 3: Pairing accepted. Responder has a variable PIN. Initiator has a variable or fixed PIN.

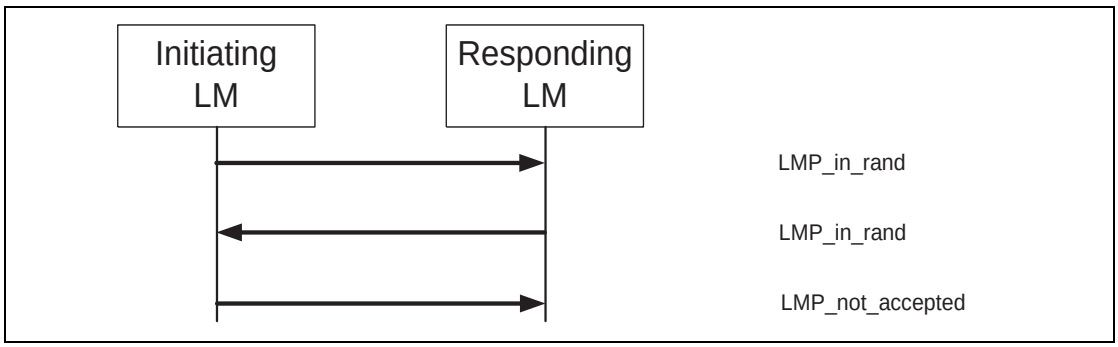
9.3.3.2 Responder has a fixed PIN

If the responder has a fixed PIN, it shall generate a new random number and send it back in LMP_in_rand. If the initiator has a variable PIN, it shall accept this and respond with LMP_accepted. Both sides then calculate K_{init} based on the last IN_RAND and the BD address of the initiator. Thereafter the procedure continues with creation of the link key (see 9.3.3.4).



Sequence 4: Responder has a fixed PIN and initiator has a variable PIN.

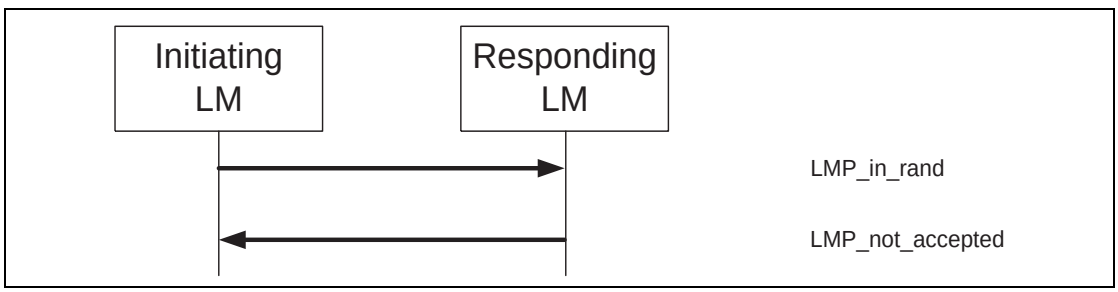
If the responder has a fixed PIN and the initiator also has a fixed PIN, the second LMP_in_rand is rejected by the initiator sending LMP_not_accepted with the reason code *pairing not allowed*; the pairing procedure is then ended.



Sequence 5: Both devices have a fixed PIN.

9.3.3.3 Responder rejects pairing

If the responder rejects pairing it sends LMP_not_accepted with the reason code *pairing not allowed* after receiving LMP_in_rand.



Sequence 6: Responder rejects pairing.

9.3.3.4 Creation of the link key

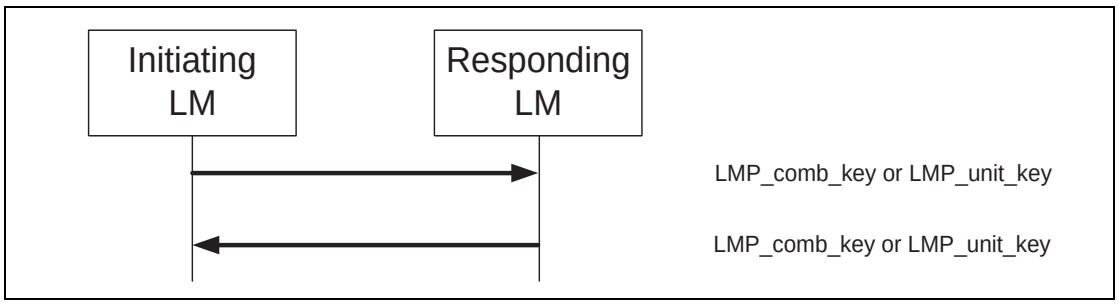
When K_{init} is calculated in both units the link key must be created. This link key will be used in the authentication between the two units for all subsequent connections until it is changed (see 9.3.4 and 9.3.5). The link key created in the pairing procedure will either be a combination key or one of the unit's unit keys. The following rules apply to the selection of the link key:

— If one unit sends LMP_unit_key and the other unit sends LMP_comb_key, the unit key will be the link key.

— If both units send LMP_unit_key, the master's unit key will be the link key.

— If both units send LMP_comb_key, the link key is calculated as described in 8.14.2.2.

The content of LMP_unit_key is the unit key bitwise XORed with K_{init}. The content of LMP_comb_key is LK_RAND bitwise XORed with K_{init}. Any device configured to use a combination key will store the link key.



Sequence 7: Creation of the link key.

9.3.3.5 Repeated attempts

When the authentication after creation of the link key fails because of a wrong authentication response, the same scheme as in 9.3.2.3 is applied. This prevents an intruder from trying a large number of different PINs in a relatively short time.

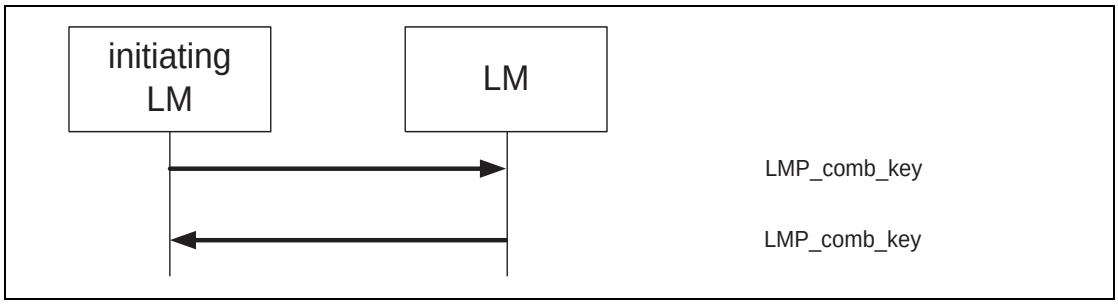
9.3.4 Change link key

If the link key is derived from combination keys and the current link is the semipermanent link key, the link key can be changed. If the link key is a unit key, the units shall go through the pairing procedure in order to change the link key. The contents of LMP_comb_key is protected by a bitwise XOR with the current link key and summarized in Table 45.

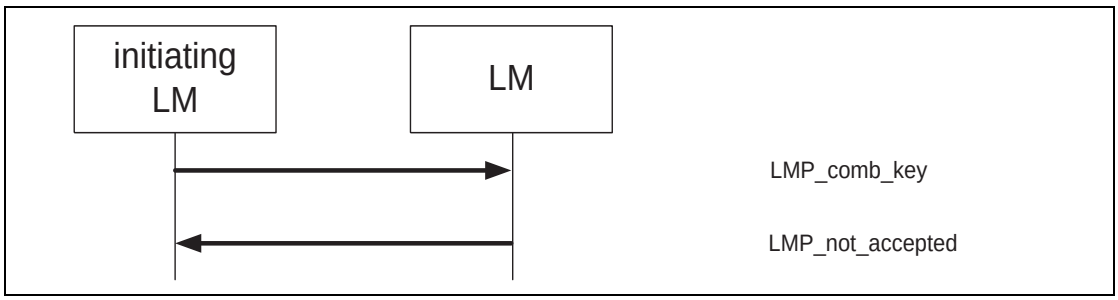
Table 45—PDUs used for change of link key

M/O	PDU	Contents
M	LMP_comb_key	random number

Note that all sequences described in 9.3.4, including the mutual authentication after the link key has been changed, form a single transaction. The transaction ID from the first LMP_comb_key will be used for all subsequent sequences.



Sequence 8: Successful change of the link key.



Sequence 9: Change of the link key not possible since the other unit uses a unit key.

If the change of link key is successful the new link key is stored and the old link key is discarded. The new link key will be used as link key for all the following connections between the two devices until the link key is changed again. The new link key also becomes the current link key. It will remain the current link key until the link key is changed again, or until a temporary link key is created (see 9.3.5).

When the new link key has been created mutual authentication shall be made to confirm that the same link key has been created in both units. The first authentication in the mutual authentication is made with the unit that initiated change link key as verifier. When finalized an authentication in the reversed direction is made.

9.3.5 Change the current link key

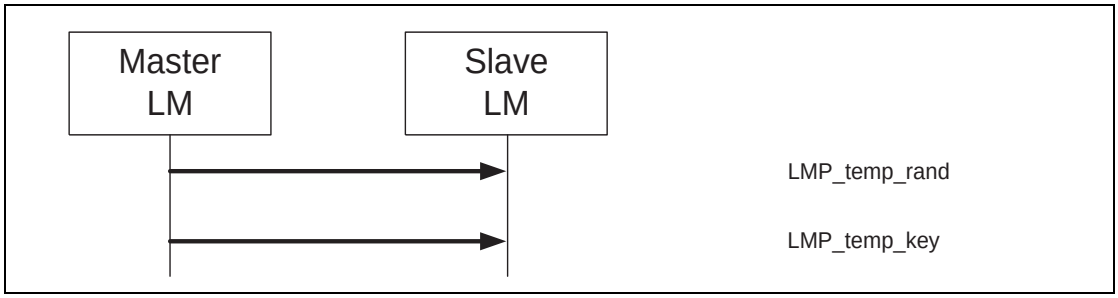
The current link key can be a semipermanent link key or a temporary link key key. It can be changed temporarily, but the change is only valid for the session (see 8.14.2.1). Changing to a temporary link key is necessary if the piconet is to support encrypted broadcast (see Table 46).

9.3.5.1 Change to a temporary link key

In the following, the same terms as in 8.14.2.2.8. The master starts by creating the master key K_{master} as described in Equation (26). Then the master issues a random number RAND and sends it to the slave in LMP_temp_rand. Both sides can then calculate an overlay denoted OVL as OVL= E_{22}(current link key, RAND, 16). Then the master sends K_{master} protected by a modulo-2 addition with OVL to the slave in LMP_temp_key. The slave, who knows OVL, calculates K_{master}. After this, K_{master} becomes the current link key. It will be the current link key until the units fall back to the semipermanent link key (see 9.3.5.2).

Table 46—PDUs used to change the current link key

M/O	PDU	Contents
M	LMP_temp_rand	random number
M	LMP_temp_key	key
M	LMP_use_semi_permanent_key	—



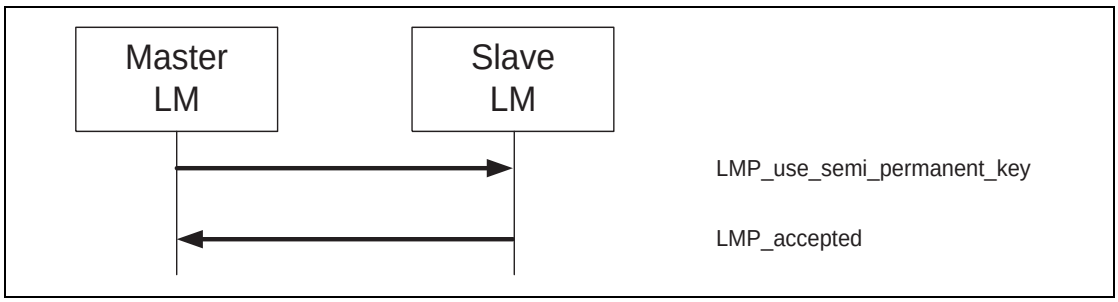
Sequence 10: Change to a temporary link key.

Note that all sequences described in 9.3.4, including the mutual authentication after K_{master} has been created, form a single transaction. The transaction ID is set to 0.

When the units have changed to the temporary key, a mutual authentication shall be made to confirm that the same link key has been created in both units. The first authentication in the mutual authentication is made with the master as verifier. When finalized an authentication in the reversed direction is made.

9.3.5.2 Make the semipermanent link key the current link key

After the current link key has been changed to K_{master}, this change can be undone and the semipermanent link key becomes the current link key again. If encryption is used on the link, the procedure of going back to the semipermanent link key shall be immediately followed by a stop of the encryption by the master invoking the procedure described in 9.3.6.4. Encryption can then be started again by the master according to the procedures in 9.3.6.1. This is to assure that encryption with encryption parameters known by other devices in the piconet is not used when the semipermanent link key is the current link key.



Sequence 11: Link key changed to the semipermanent link key.

9.3.6 Encryption

If at least one authentication has been performed encryption may be used. If the master wants all slaves in the piconet to use the same encryption parameters it must issue a temporary key (K_{master}) and make this key the current link key for all slaves in the piconet before encryption is started (see 9.3.5). This is necessary if broadcast packets should be encrypted (see Table 47).

Table 47—PDUs used for handling encryption

M/O	PDU	Contents
O	LMP_encryption_mode_req	encryption mode
O	LMP_encryption_key_size_req	key size
O	LMP_start_encryption_req	random number
O	LMP_stop_encryption_req	—

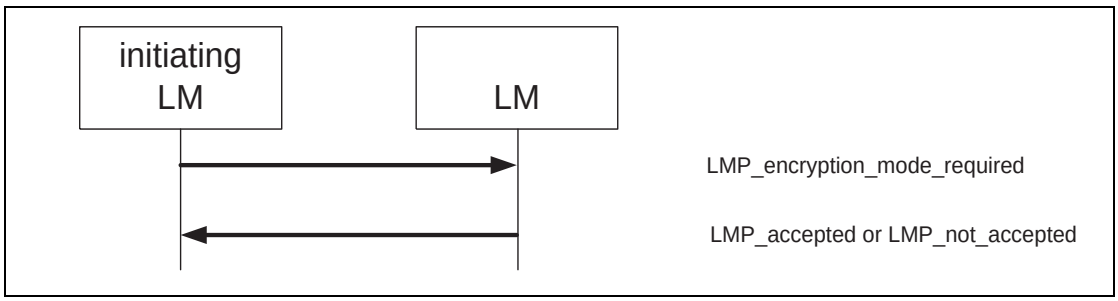
Note that all sequences described in 9.3.6 form a single transaction. The transaction ID from the LMP_encryption_mode_req will be used for all subsequent sequences.

9.3.6.1 Encryption mode

First of all the master and the slave must agree upon whether to use encryption or not and if encryption shall only apply to point-to-point packets or if encryption shall apply to both point-to-point packets and broadcast packets. If master and slave agree on the encryption mode, the master continues to give more detailed information about the encryption.

The initiating LM finalizes the transmission of the current ACL packet with L2CAP information, stops L2CAP transmission and sends LMP_encryption_mode_req. If the change in encryption mode is accepted then the other device finalizes the transmission of the current ACL packet with L2CAP information, stops L2CAP transmission and responds with LMP_accepted.

L2CAP transmission is re-enabled when the attempt to encrypt or decrypt the link is completed i.e., at the end of Sequence 14, 15, or 16.



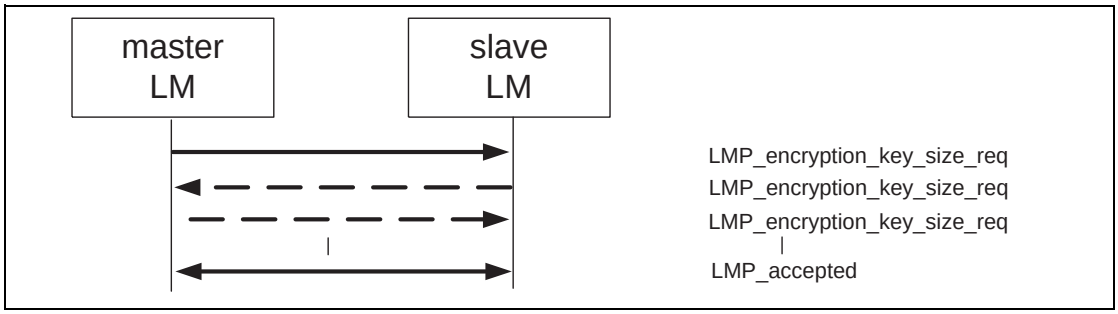
Sequence 12: Negotiation for encryption mode.

After a unit has sent LMP_encryption_mode_req, it is not allowed to send LMP_au_rand before encryption is actually switched on. After a unit has received LMP_encryption_mode_req and sent LMP_accepted it is not allowed to send LMP_au_rand before encryption is actually switched on. If an LMP_au_rand is still sent

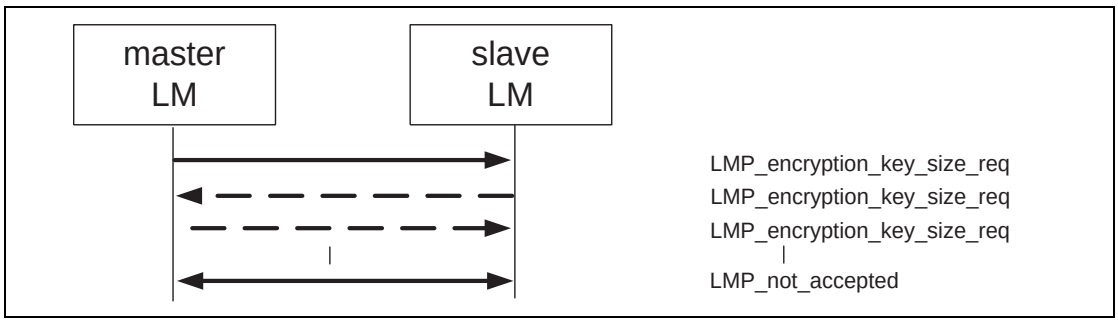
violating these rules, the claimant shall respond with LMP_not_accepted with the reason code *PDU not allowed*. This is to avoid that the units have different ACOs when they calculate the encryption key. If the encryption mode is not accepted or the encryption key size negotiation results in disagreement the units are allowed to send LMP_au_rand again.

9.3.6.2 Encryption key size

The next step is to determine the size of the encryption key. In the following we use the same terms as in 8.14.3.1. The master sends LMP_encryption_key_size_req including the suggested key size $L_{sug,\ m}$, which is initially equal to $L_{max,\ m}$. If $L_{min,\ s} \leq L_{sug,\ m}$ and the slave supports $L_{sug,\ m}$ it responds with LMP_accepted and $L_{sug,\ m}$ will be used as the key size. If both conditions are not fulfilled the slave sends back LMP_encryption_key_size_req including the slave's suggested key size $L_{sug,\ s}$. This value is the slave's largest supported key size that is less than $L_{sug,\ m}$. Then the master performs the corresponding test on the slave's suggestion. This procedure is repeated until a key size agreement is reached or it becomes clear that no such agreement can be reached. If an agreement is reached a unit sends LMP_accepted and the key size in the last LMP_encryption_key_size_req will be used. After this, the encryption is started; see 9.3.6.3. If an agreement is not reached a unit sends LMP_not_accepted with the reason code *Unsupported parameter value* and the units are not allowed to communicate using Bluetooth link encryption.



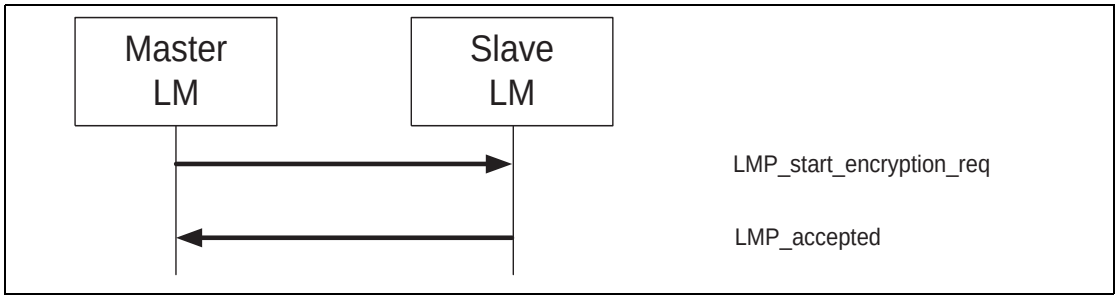
Sequence 13: Encryption key size negotiation successful.



Sequence 14: Encryption key size negotiation failed.

9.3.6.3 Start encryption

Finally, encryption is started. The master issues the random number EN_RAND and calculates the encryption key as $K_c=E_3$(current link key, EN_RAND, COF). See 8.14.2.2.5 for the definition of the COF. The random number must be the same for all slaves if the piconet should support encrypted broadcast. Then the master sends LMP_start_encryption_req, which includes EN_RAND. The slave calculates K_c when this message is received and acknowledges with LMP_accepted.



Sequence 15: Start of encryption.

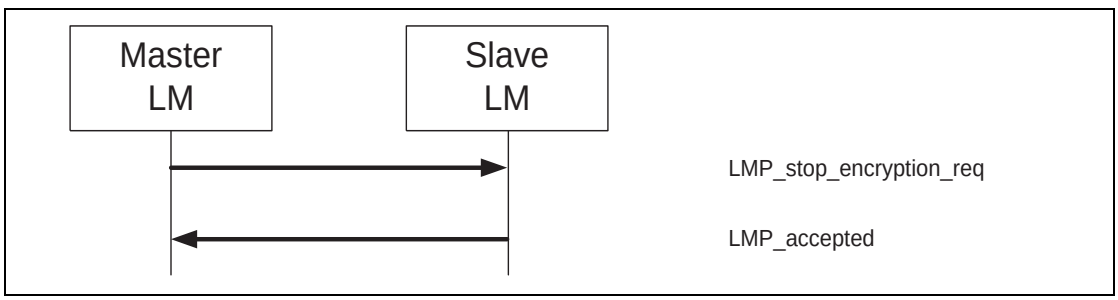
The start of encryption will be done in three steps:

1) Master is configured to transmit unencrypted packets, but to receive encrypted packets.
2) Slave is configured to transmit and receive encrypted packets.
3) Master is configured to transmit and receive encrypted packets.

Between step 1 and step 2, master-to-slave transmission is possible. This is when LMP_start_encryption_req is transmitted. Step 2 is triggered when the slave receives this message. Between step 2 and step 3, slave-to-master transmission is possible. This is when LMP_accepted is transmitted. Step 3 is triggered when the master receives this message.

9.3.6.4 Stop encryption

If a unit wants to stop encryption it sends LMP_encryption_mode_req with the parameter encryption mode equal to 0 (no encryption). The other device responds with LMP_accepted or LMP_not_accepted (the procedure is described in Sequence 12 in 9.3.6.1). If accepted encryption is stopped by the master sending LMP_stop_encryption_req and the slave responding with LMP_accepted according to Sequence 16.



Sequence 16: Stop of encryption.

Stopping of encryption is then done in three steps, similar to the procedure for starting encryption.

1) Master is configured to transmit encrypted packets, but to receive unencrypted packets.
2) Slave is configured to transmit and receive unencrypted packets.
3) Master is configured to transmit and receive unencrypted packets.

Between step 1 and step 2 master to slave transmission is possible. This is when LMP_stop_encryption_req is transmitted. Step 2 is triggered when the slave receives this message. Between step 2 and step 3 slave to

master transmission is possible. This is when LMP_accepted is transmitted. Step 3 is triggered when the master receives this message.

9.3.6.5 Change encryption mode, key or random number

If the encryption key or encryption random number need to be changed or if the encryption mode needs to be changed between 1 (point-to-point) and 2 (point-to-point and broadcast), encryption must first be stopped and then restarted with the new parameters according to the procedures in 9.3.6.

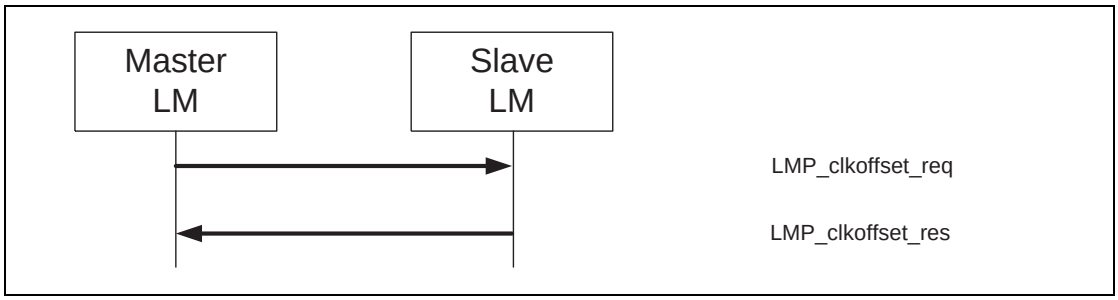
9.3.7 Clock offset request

When a slave receives the FHS packet, the difference is computed between its own clock and the master's clock included in the payload of the FHS packet. The clock offset is also updated each time a packet is received from the master. The master can request the clock offset at anytime following a successful baseband paging procedure (i.e., before, during, or after connection setup).

By saving this clock offset, the master knows on what RF channel the slave wakes up to PAGE SCAN after it has left the piconet. This can be used to speed up the paging time the next time the same device is paged (see Table 48).

Table 48—PDUs used for clock offset request

M/O	PDU	Contents
M	LMP_clkoffset_req	—
M	LMP_clkoffset_res	clock offset



Sequence 17: Clock offset requested.

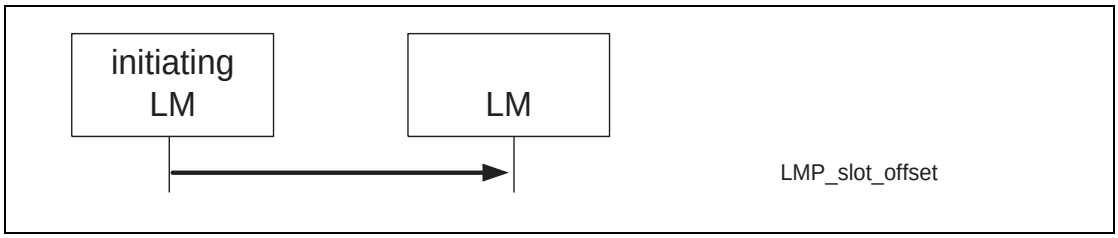
9.3.8 Slot offset information

With LMP_slot_offset the information about the difference between the slot boundaries in different piconets is transmitted. This PDU carries the parameters slot offset and BD_ADDR. The slot offset is the subtraction of time in µs of the start of the master's TX slot in the piconet where the PDU is transmitted from the time in µs of the start of the master's TX slot in the piconet where the BD_ADDR device is master modulo 1250 (see Table 49).

See 9.3.12 for the use of LMP_slot_offset in the context of the master-slave switch.

Table 49— PDU used for slot offset information

M/O	PDU	Contents
O	LMP_slot_offset	slot offset BD_ADDR



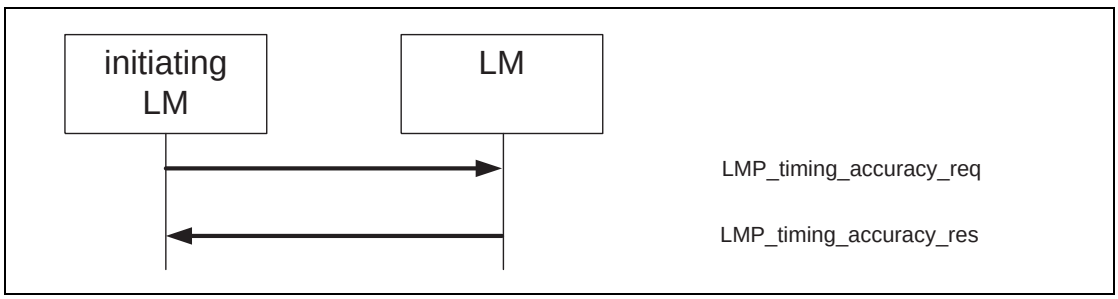
Sequence 18: Slot offset information is sent.

9.3.9 Timing accuracy information request

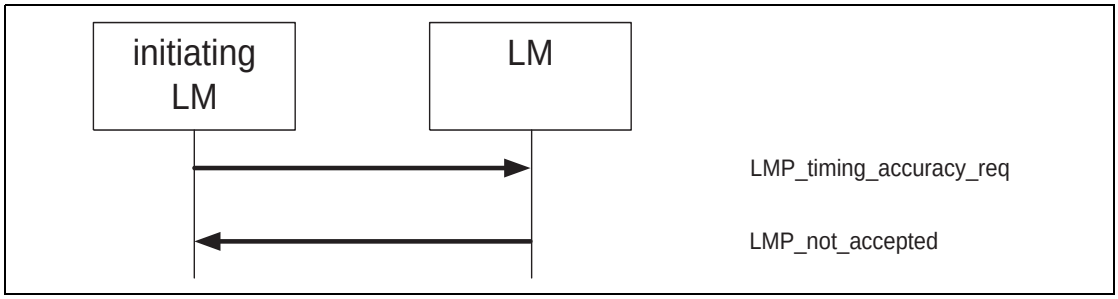
LMP supports requests for the timing accuracy. This information can be used to minimize the scan window for a given hold time when returning from hold and to extend the maximum hold time. It can also be used to minimize the scan window when scanning for the sniff mode slots or the park mode beacon packets. The timing accuracy parameters returned are the long-term drift measured in ppm and the long term jitter measured in μs of the clock used during hold, sniff, and park mode. These parameters are fixed for a certain device and shall be identical when requested several times. Timing accuracy can be requested at anytime following a successful baseband paging procedure, provided this PDU is shown as supported in the supported features list. If the timing accuracy request is not supported, the requesting device shall assume worst case values (drift = 250 ppm and jitter = 10 μs) (see Table 50).

Table 50—PDUs used for requesting timing accuracy information

M/O	PDU	Contents
O	LMP_timing_accuracy_req	—
O	LMP_timing_accuracy_res	drift jitter



Sequence 19: The requested device supports timing accuracy information.



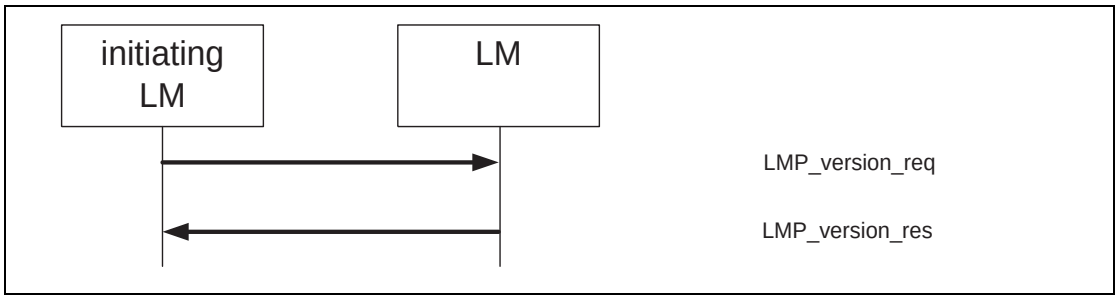
Sequence 20: The requested device does not support timing accuracy information.

9.3.10 LMP version

LMP supports requests for the version of the LM protocol. The requested device will send a response with three parameters: VersNr, CompId, and SubVersNr. VersNr specifies the version of the Bluetooth LMP specification that the device supports. CompId is used to track possible problems with the lower Bluetooth layers. All companies that create a unique implementation of the Link Manager shall have their own CompId. The same company is also responsible for the administration and maintenance of the SubVersNr. It is recommended that each company have a unique SubVersNr for each RF/BB/LM implementation. For a given VersNr and CompId, the values of the SubVersNr shall increase each time a new implementation is released. For both CompId and SubVersNr the value 0xFFFF means that no valid number applies. There is no ability to negotiate the version of the LMP. The sequence below (Sequence 21) is only used to exchange the parameters. LMP version can be requested at anytime following a successful baseband paging procedure (see Table 51).

Table 51—PDUs used for LMP version request

M/O	PDU	Contents
M	LMP_version_req	VersNr CompId SubVersNr
M	LMP_version_res	VersNr CompId SubVersNr



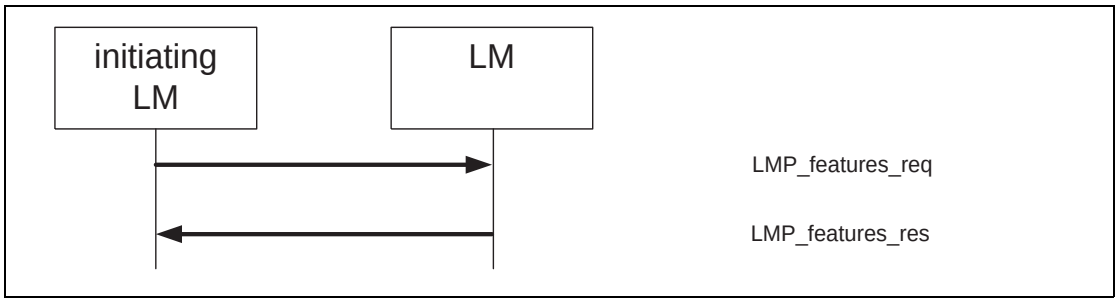
Sequence 21: Request for LMP version.

9.3.11 Supported features

The Bluetooth radio and link controller may support only a subset of the packet types and features described in Clause 8 and Physical layer. The PDU LMP_features_req and LMP_features_res are used to exchange this information. The supported features can be requested at anytime following a successful baseband paging procedure. A device may not send any packets other than ID, FHS, NULL, POLL, DM1 or DH1 before it is aware of the supported features of the other device. After the features request has been carried out, the intersection of the supported packet types for both sides may also be transmitted. Whenever a request is issued, it shall be compatible with the supported features of the other device. For instance, when establishing an SCO link the initiator may not propose to use HV3 packets if that packet type is not supported by the other device. Exceptions to this rule are LMP_switch_ req and LMP_slot_offset, which can be sent before the requesting side is aware of the other side's features (switch is an optional feature) (see Table 52).

Table 52—PDUs used for features request

M/O	PDU	Contents
M	LMP_features_req	features
M	LMP_features_res	features



Sequence 22: Request for supported features.

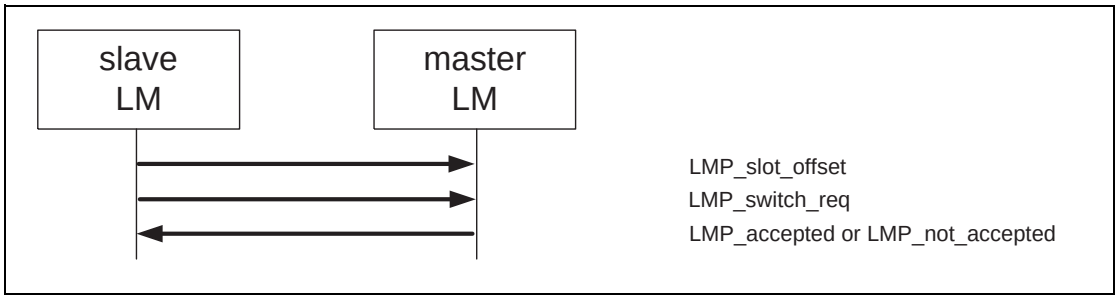
9.3.12 Switch of master-slave role

Since the paging device always becomes the master of the piconet, a switch of the master-slave role is sometimes needed (see 8.10.9.3). See also Table 53.

Table 53—PDUs used for master-slave switch

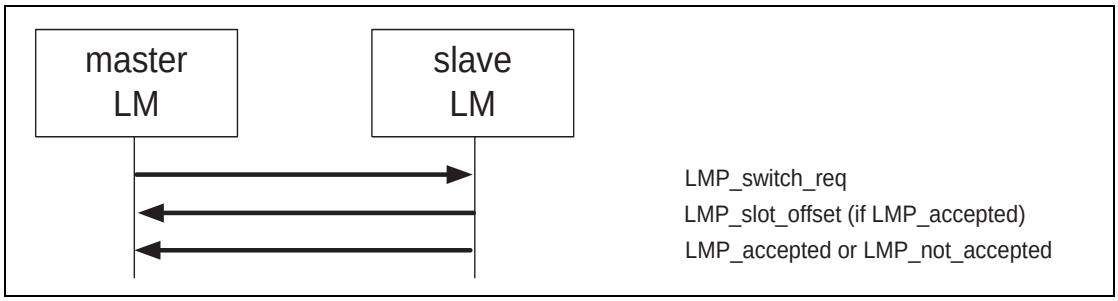
M/O	PDU	Contents
O	LMP_switch_req	switch instant
O	LMP_slot_offset	slot offset BD_ADDR

If the slave initiates the master-slave switch, it finalizes the transmission of the current ACL packet with L2CAP information, stops L2CAP transmission, and sends LMP_slot_offset immediately followed by LMP_switch_req. If the master accepts the master-slave switch, it finalizes the transmission of the current ACL packet with L2CAP information, stops L2CAP transmission, and responds with LMP_accepted. When the master-slave switch has been completed on Baseband level (successfully or not), both units re-enable L2CAP transmission. If the master rejects the master-slave switch, it responds with LMP_not_accepted and the slave re-enables L2CAP transmission. The transaction ID for all PDUs in the sequence is set to 1.



Sequence 23: Master-slave switch (slave initiated).

If the master initiates the master-slave switch, it finalizes the transmission of the current ACL packet with L2CAP information, stops L2CAP transmission, and sends LMP_switch_req. If the slave accepts the master-slave switch, it finalizes the transmission of the current ACL packet with L2CAP information, stops L2CAP transmission, and responds with LMP_slot_offset immediately followed by LMP_accepted. When the master-slave switch has been completed on Baseband level (successfully or not), both units re-enable L2CAP transmission. If the slave rejects the master-slave switch, it responds with LMP_not_accepted and the master re-enables L2CAP transmission. The transaction ID for all PDUs in the sequence is set to 0.



Sequence 24: Master-slave switch (master initiated).

The LMP_switch_req PDU contains a parameter, switch instant, which specifies the instant at which the TDD switch is made. This is specified as a Bluetooth clock value of the master's clock, which is available to both devices. This instant is chosen by the sender of the message and should be at least $2*T_{poll}$ or 32 (whichever is greater) slots in the future. The assumption is that the switch instant is always within 12 h of the current clock value, in order to accommodate clock wrap.

The sender of the LMP_switch_req selects the switch instant, queues the LMP_switch_req to LC for transmission, and starts a timer to expire at the switch instant. When the timer expires it initiates the mode switch. In the case of a master-initiated switch, if the LMP_slot_offset has not been received by the switch instant, the master slave switch is carried out without an estimate of the slave's slot offset. If LMP_not_accepted is received before the timer expires, then the timer is stopped, and the role switch is not initiated.

When the LMP_switch_req is received, the switch instant is compared with the current master clock value. If it is in the past, then the instant has been passed and LMP_not_accepted with the reason code Instant passed shall be returned. If it is in the future, then LMP_accepted is returned assuming the master-slave switch is allowed and a timer is started to expire at the switch instant. When this timer expires, it initiates the mode switch.

Support for LMP_slot_offset is mandatory if LMP_switch_req is supported.

9.3.13 Name request

LMP supports name request to another Bluetooth device. The name is a user-friendly name associated with the Bluetooth device and consists of a maximum of 248 bytes coded according to the UTF-8 standard. The name is fragmented over one or more DM1 packets. When the LMP_name_req is sent, a name offset indicates which fragment is expected. The corresponding LMP_name_res carries the same name offset, the name length indicating the total number of bytes in the name of the Bluetooth device and the name fragment, where:

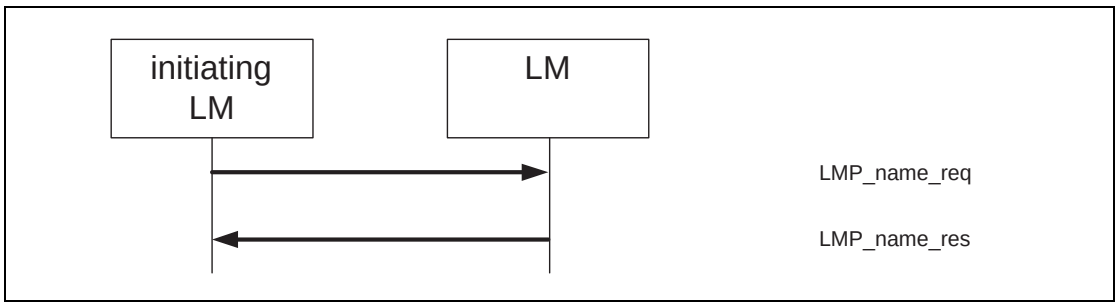
— name fragment(N) = name(N + name offset), if (N + name offset) < name length

— name fragment(N) = 0, otherwise.

Here $0 \leq N \leq 13$. In the first sent LMP_name_req, name offset = 0. Sequence 25 is then repeated until the initiator has collected all fragments of the name. The name request can be made at anytime following a successful baseband paging procedure (see Table 54).

Table 54—PDUs used for name request

M/O	PDU	Contents
M	LMP_name_req	name offset
M	LMP_name_res	name offset name length name fragment



Sequence 25: Device's name requested and it responses.

9.3.14 Detach

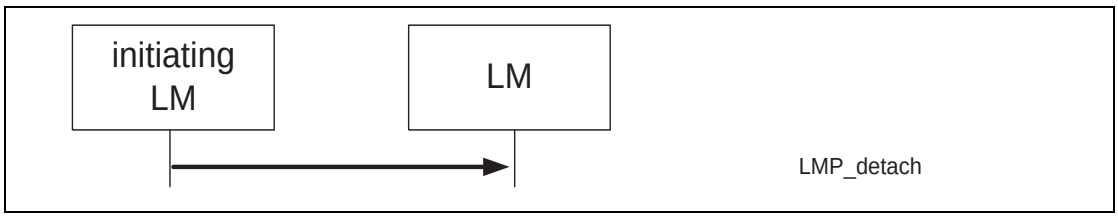
The connection between two Bluetooth devices can be closed anytime by the master or the slave. A reason parameter is included in the message to inform the other party of why the connection is closed (see Table 55).

Table 55—PDU used for detach

M/O	PDU	Contents
M	LMP_detach	reason

The initiating LM first finalizes the transmission of the current ACL packet with L2CAP information and stops L2CAP transmission. The initiating LM then queues the LMP_detach for transmission and starts a timer for $6*T_{poll}$ slots where T_{poll} is the poll interval for the connection. If the initiating LM receives the Baseband-level acknowledgement before the timer expires it now starts a timer for $3*T_{poll}$ slots. When this timer expires (and if the initiating LM is the master) the AM_ADDR can be re-used immediately. If the initial timer expires then the initiating LM drops the link and starts a timer for $T_{linksupervisiontimeout}$ slots after which the AM_ADDR can be re-used (if the initiating LM is the master).

When the receiving LM receives the LMP_detach, it starts a timer for $6*T_{poll}$ slots if it is the master and $3*T_{poll}$ if it is the slave. On timer expiration, the link is detached and (if the receiving LM is the master) the AM_ADDR can be re-used immediately. If the receiver never gets the LMP_detach then a link supervision timeout will occur and the link will be detached.



Sequence 26: Connection closed by sending LMP_detach.

9.3.15 Hold mode

The ACL link of a connection between two Bluetooth devices can be placed in hold mode for a specified hold time. During this time no ACL packets will be transmitted from the master. The hold mode is typically entered when there is no need to send data for a relatively long time. The transceiver can then be turned off in order to save power. But the hold mode can also be used if a device wants to discover or be discovered by other Bluetooth devices, or wants to join other piconets. What a device actually does during the hold time is not controlled by the hold message, but it is up to each device to decide (see Table 56).

The LMP_hold and LMP_hold_req PDUs both contain a parameter, hold instant, which specifies the instant at which the hold will become effective. This is specified as a Bluetooth clock value of the master's clock, which is available to both devices. This instant is chosen by the sender of the message and should be at least $6*T_{poll}$ slots in the future. The assumption is that the hold instant is always within 12 h of the current clock value, in order to accommodate clock wrap.

Table 56—PDUs used for hold mode

M/O	PDU	Contents
O	LMP_hold	hold time, hold instant
O	LMP_hold_req	hold time, hold instant

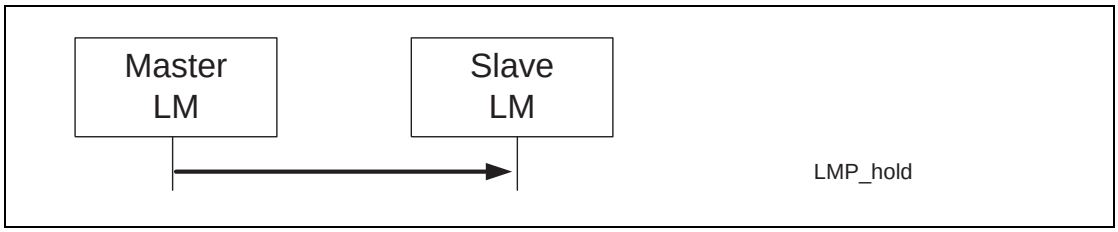
9.3.15.1 Master forces hold mode

The master can force hold mode if there has previously been a request for hold mode that has been accepted. The hold time included in the PDU when the master forces hold mode cannot be longer than any hold time the slave has previously accepted when there was a request for hold mode.

The master LM first finalizes the transmission of the current ACL packet with L2CAP information and stops L2CAP transmission. It selects the hold instant and queues the LMP_hold to its LC for transmission. It then starts a timer to wait until the hold instant occurs. When this timer expires then the connection enters hold mode. Note that the Baseband-level acknowledgement is ignored in this mechanism.

When the slave LM receives the LMP_hold it compares the hold instant with the current master clock value. If it is in the future then it starts a timer to expire at this instant and enters hold mode when it expires.

When the master LM exits from Hold mode it re-enables L2CAP transmission.



Sequence 27: Master forces slave into hold mode.

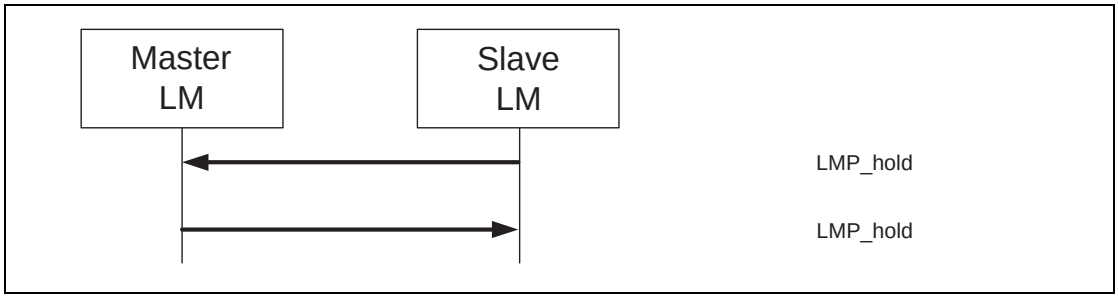
9.3.15.2 Slave forces hold mode

The slave can force hold mode if there has previously been a request for hold mode that has been accepted. The hold time included in the PDU when the slave forces hold mode cannot be longer than any hold time the master has previously accepted when there was a request for hold mode.

The slave LM first finalizes the transmission of the current ACL packet with L2CAP information and stops L2CAP transmission. It selects the hold instant and queues the LMP_hold to its LC for transmission. It then waits for the LMP_hold from the master acting according to the procedure described in 9.3.15.1.

When the master LM receives the LMP_hold it finalizes the transmission of the current ACL packet with L2CAP information and stops L2CAP transmission. It then inspects the hold instant. If this is less than $6*T_{poll}$ slots in the future it should modify the instant so that it is at least $6*T_{poll}$ slots in the future. Then it sends the LMP_hold using the mechanism described in 9.3.15.1.

When the master and slave LMs exit from Hold mode they re-enable L2CAP transmission.



Sequence 28: Slave forces master into hold mode.

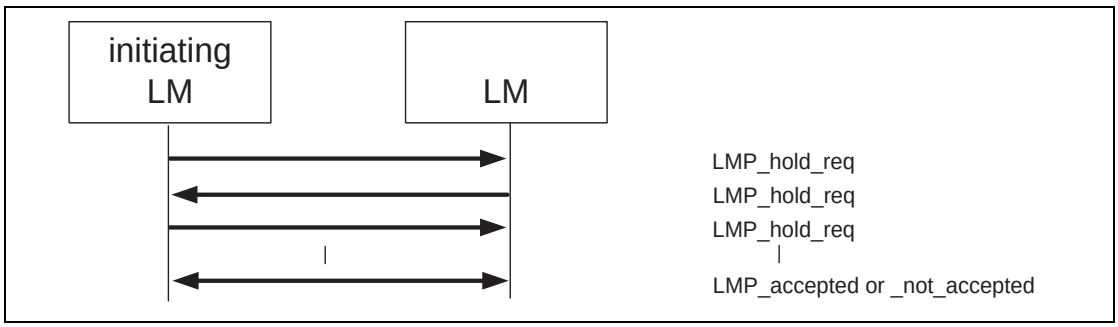
9.3.15.3 Master or slave requests hold mode

The master or the slave can request to enter hold mode. Upon receipt of the request, the same request with modified parameters can be returned or the negotiation can be terminated. If an agreement is seen, LMP_accepted terminates the negotiation and the ACL link is placed in hold mode. If no agreement is seen, LMP_not_accepted with the reason code *unsupported parameter value* terminates the negotiation and hold mode is not entered.

The initiating LM first finalizes the transmission of the current ACL packet with L2CAP information and stops L2CAP transmission. On receiving the LMP_hold_req, the receiving LM finalizes the transmission of the current ACL packet with L2CAP information and stops L2CAP transmission

The LM sending LMP_hold_req selects the hold instant, which should be at least $9*T_{poll}$ slots in the future. If this is a response to a previous LMP_hold_req and the contained hold instant is at least $9*T_{poll}$ slots in the future then this should be used. The LMP_hold_req is then queued to its LC for transmission and a timer is started to expire at this instant and the connection enters hold mode when it expires unless an LMP_not_accepted or LMP_hold_req is received by its LM before that point. If the LM receiving LMP_hold_req agrees to enter hold mode it returns LMP_accepted and starts a timer to expire at the hold instant. When this timer expires it enters hold mode.

When each LM exits from Hold mode it re-enables L2CAP transmission.



Sequence 29: Negotiation for hold mode.

9.3.16 Sniff mode

To enter sniff mode, master and slave negotiate a sniff interval T_{sniff} and a sniff offset, D_{sniff}, which specifies the timing of the sniff slots. The offset determines the time of the first sniff slot; after that the sniff slots follows periodically with the sniff interval T_{sniff}. To avoid problems with a clock wrap-around during the

initialization, one of two options is chosen for the calculation of the first sniff slot. A timing control flag in the message from the master indicates this. Note that only bit1 of the timing control flag is valid.

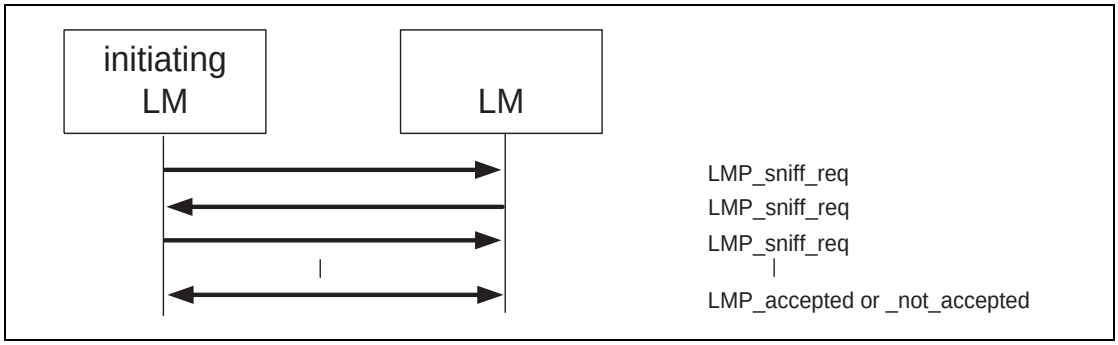
When the link is in sniff mode the master can only start a transmission in the sniff slot. Two parameters control the listening activity in the slave. The sniff attempt parameter determines for how many slots the slave shall listen, beginning at the sniff slot, even if it does not receive a packet with its own AM address. The sniff timeout parameter determines for how many additional slots the slave shall listen if it continues to receive only packets with its own AM address (see Table 57).

Table 57—PDUs used for sniff mode

M/O	PDU	Contents
O	LMP_sniff_req	timing control flags D_{sniff} T_{sniff} sniff attempt sniff timeout
O	LMP_unsniff_req	-

9.3.16.1 Master or slave requests sniff mode

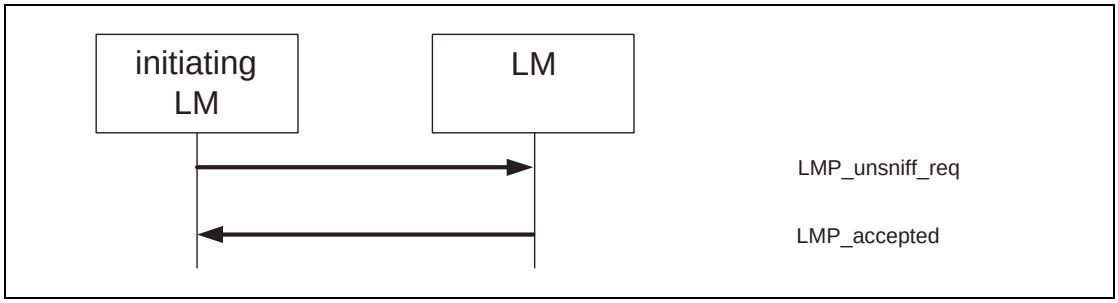
The master or the slave can request to enter sniff mode. Upon receipt of the request, the same request with modified parameters can be returned or the negotiation can be terminated. If an agreement is seen LMP_accepted terminates the negotiation and the ACL link is placed in sniff mode. If no agreement is seen, LMP_not_accepted with the reason code *unsupported parameter value* terminates the negotiation and sniff mode is not entered.



Sequence 30: Negotiation for sniff mode.

9.3.16.2 Moving a slave from sniff mode to active mode

Sniff mode is ended by sending the PDU LMP_unsniff_req. The requested device shall reply with LMP_accepted. If the slave requests, it will enter active mode after receiving LMP_accepted. If the master requests, the slave will enter active mode after receiving LMP_unsniff_req.



Sequence 31: Slave moved from sniff mode to active mode.

9.3.17 Park mode

If a slave does not need to participate in the channel, but still should be FH-synchronized, it can be placed in park mode. In this mode the device gives up its AM_ADDR but still re-synchronizes to the channel by waking up at the beacon instants separated by the beacon interval. The beacon interval, a beacon offset, and a flag indicating how the first beacon instant is calculated determine the first beacon instant. After this the beacon instants follow periodically at the predetermined beacon interval. At the beacon instant the parked slave can be activated again by the master, the master can change the park mode parameters, transmit broadcast information, or let the parked slaves request access to the channel.

All PDUs sent from the master to the parked slaves are broadcast. These PDUs (LMP_set_broadcast_scan_window, LMP_modify_beacon, LMP_unpark_BD_addr_req and LMP_unpark_PM_addr_req) are the only PDUs that can be sent to a slave in park mode and the only PDUs that can be broadcast. To increase reliability for broadcast, the packets are made as short as possible. Therefore, the format for these LMP PDUs are somewhat different. The parameters are not always byte-aligned and the length of the PDUs is variable.

The messages for controlling the park mode include many parameters, which are all defined in 8.10.8.4. When a slave is placed in park mode it is assigned a unique PM_ADDR, which can be used by the master to unpark that slave. The all-zero PM_ADDR has a special meaning; it is not a valid PM_ADDR. If a device is assigned this PM_ADDR, it shall be identified with its BD_ADDR when it is unparked by the master (see Table 58).

9.3.17.1 Master requests slave to enter park mode

The master can request park mode. The master finalizes the transmission of the current ACL packet with L2CAP information, stops point-to-point L2CAP transmission, and then sends LMP_park_req. If the slave accepts to enter park mode, it finalizes the transmission of the current ACL packet with L2CAP information, stops L2CAP transmission, and then responds with LMP_accepted.

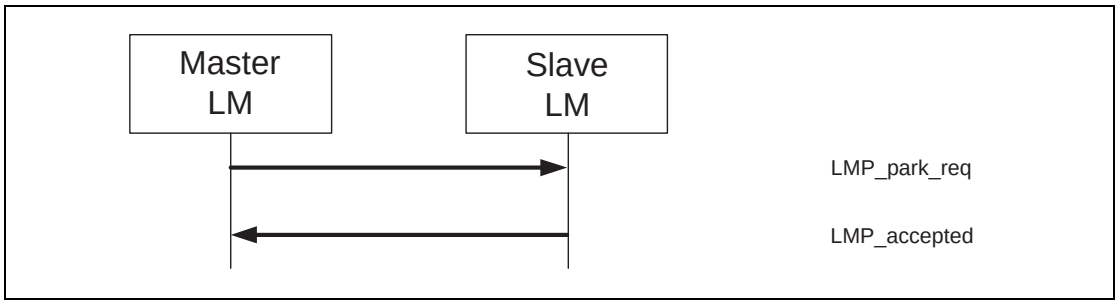
When the slave queues LMP_accepted, it starts a timer for $6*T_{poll}$ slots. If the Baseband-level acknowledgement is received before this timer expires, it enters park mode immediately, otherwise it enters park mode when the timer expires.

When the master receives LMP_accepted, it starts a timer for $6*T_{poll}$ slots. When this timer expires the slave is in park mode and the AM_ADDR can be re-used.

Note that if the master never receives the LMP_accepted then a link supervision timeout will occur.

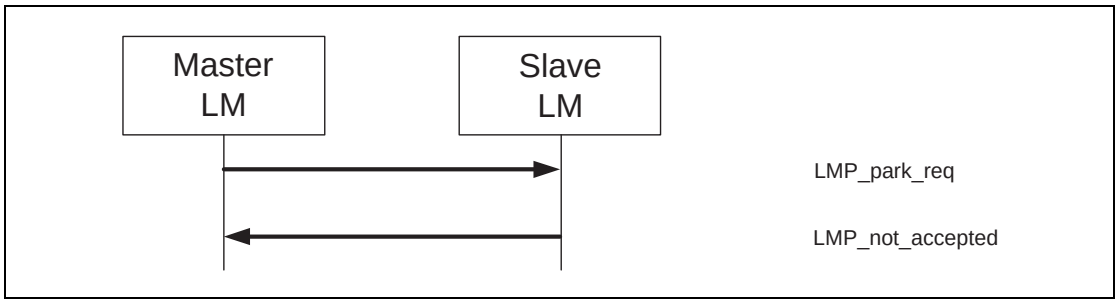
Table 58—PDUs used for park mode

M/O	PDU	Contents
O	LMP_park_req	timing control flags D_B T_B N_B Δ_B PM_ADDR AR_ADDR N_{Bsleep} D_{Bsleep} D_{access} T_{access} $N_{acc\text{-}slots}$ N_{poll} M_{access} access scheme
O	LMP_set_broadcast_scan_window	timing control flags D_B (optional) broadcast scan window
O	LMP_modify_beacon	timing control flags D_B (optional) T_B N_B Δ_B D_{access} T_{access} $N_{acc\text{-}slots}$ N_{poll} M_{access} access scheme
O	LMP_unpark_PM_ADDR_req	timing control flags D_B (optional) AM_ADDR PM_ADDR AM_ADDR (optional) PM_ADDR (optional) (totally 1–7 pairs of AM_ADDR, PM_ADDR)
O	LMP_unpark_BD_ADDR _req	timing control flags D_B (optional) AM_ADDR BD_ADDR AM_ADDR (optional) BD_ADDR (optional)



Sequence 32: Slave accepts to enter park mode.

If the slave rejects to enter park mode it responds with LMP_not_accepted and the master re-enables L2CAP transmission.



Sequence 33: Slave rejects to enter into park mode

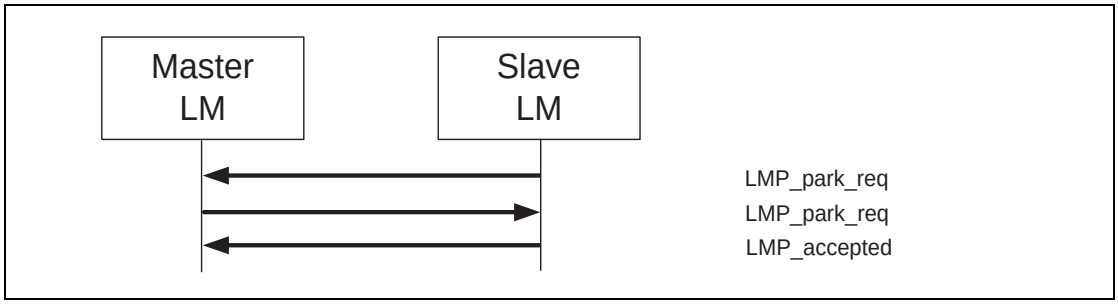
9.3.17.2 Slave requests to enter park mode

The slave can request park mode. The slave finalizes the transmission of the current ACL packet with L2CAP information, stops L2CAP transmission, and then sends LMP_park_req. The parameters PM_ADDR and AR_ADDR are not valid and the other parameters represent suggested values. If the master wants the slave to enter park mode, it finalizes the transmission of the current ACL packet with L2CAP information, stops point-to-point L2CAP transmission, and then sends LMP_park_req, where the parameter values may be different from the values in the PDU sent from the slave. If the slave can accept these parameters, it responds with LMP_accepted.

When the slave queues LMP_accepted, it starts a timer for $6*T_{poll}$ slots. If the Baseband-level acknowledgement is received before this timer expires, it enters park mode immediately, otherwise it enters park mode when the timer expires.

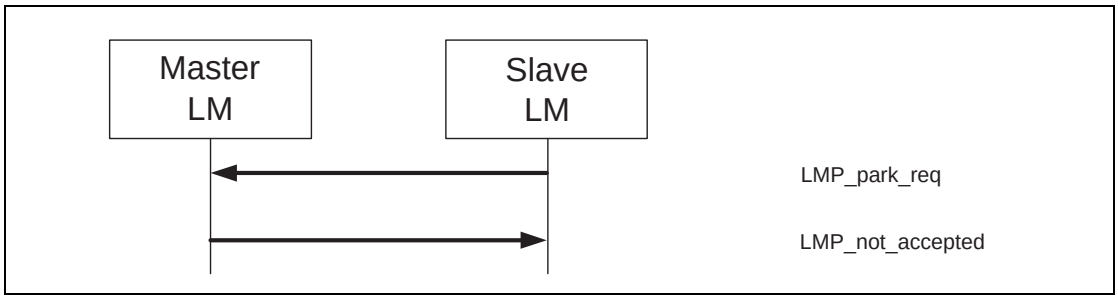
When the master receives LMP_accepted it starts a timer for $6*T_{poll}$ slots. When this timer expires the slave is in park mode and the AM_ADDR can be re-used.

Note that if the master never receives the LMP_accepted then a link supervision timeout will occur.



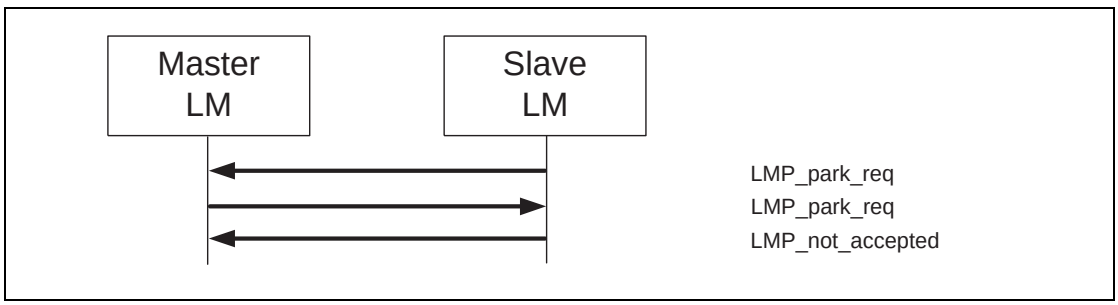
Sequence 34: Slave requests to enter park mode and accepts master's beacon parameters.

If the master does not accept that the slave enters park mode, it sends LMP_not_accepted. The slave then re-enables L2CAP transmission.



Sequence 35: Master rejects slave's request to enter park mode

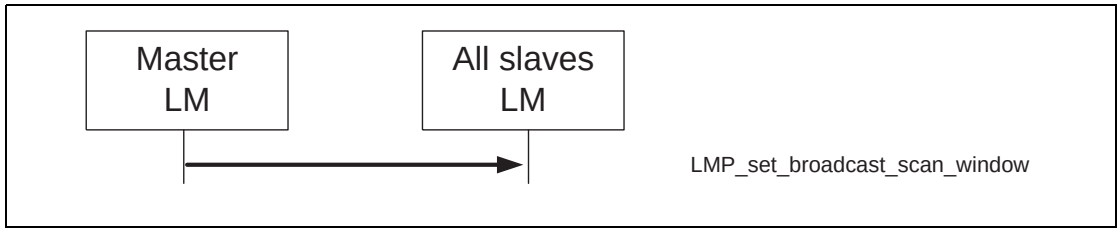
If the slave does not accept the parameters in LMP_park_req sent from the master, it responds with LMP_not_accepted and both units re-enable L2CAP transmission.



Sequence 36: Slave requests to enter park mode, but rejects master's beacon parameters.

9.3.17.3 Master sets up broadcast scan window

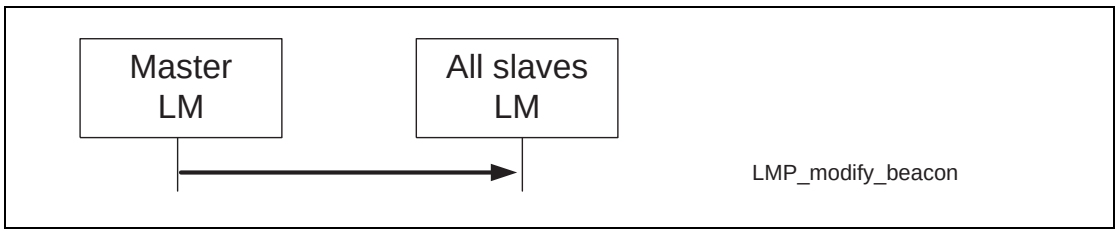
If more broadcast capacity is needed than the beacon train, the master can indicate to the slaves that more broadcast information will follow the beacon train by sending LMP_set_broadcast_scan_window. This message is always sent in a broadcast packet at the beacon slot(s). The scan window starts in the beacon instant and is only valid for the current beacon.



Sequence 37: Master notifies all slaves of increase in broadcast capacity.

9.3.17.4 Master modifies beacon parameters

When the beacon parameters change, the master notifies the parked slaves of this by sending LMP_modify_beacon. This message is always sent in a broadcast packet at the beacon slot(s).



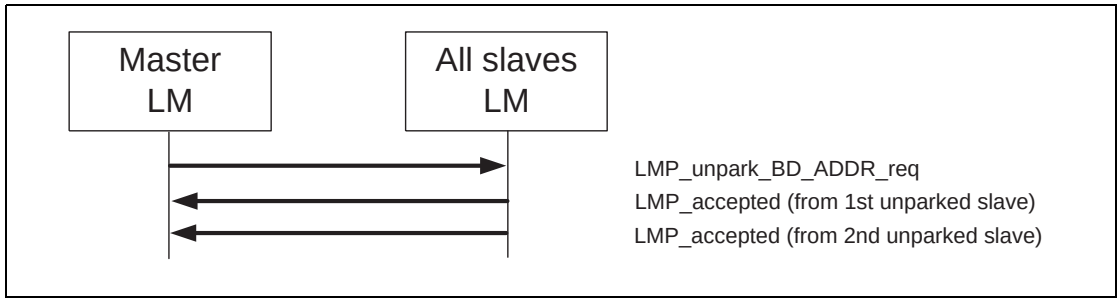
Sequence 38: Master modifies beacon parameters.

9.3.17.5 Unparking slaves

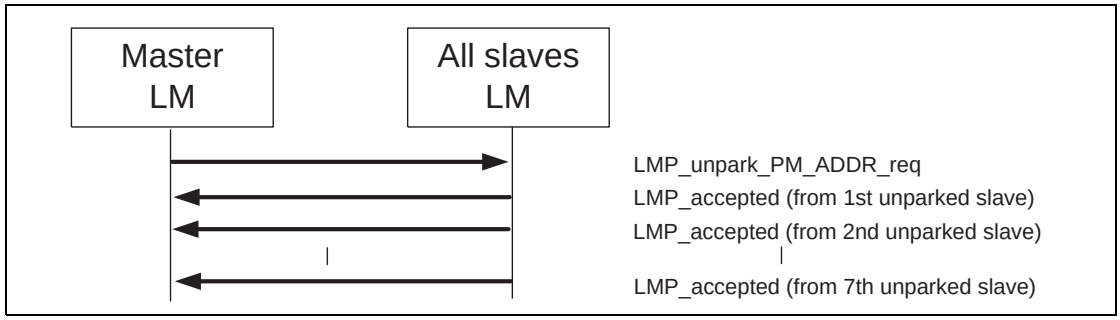
The master can unpark one or many slaves by sending a broadcast LMP message including the PM_ADDR or the BD_ADDR of the device(s) it wishes to unpark at the beacon slot(s). This message also includes the AM_ADDR that the master assigns to the slave(s). After sending this message, the master shall check the success of the unpark by polling each unparked slave, i.e., sending POLL packets, so that the slave is granted access to the channel. The unparked slave shall then send a response with LMP_accepted. If this message is not received from the slave within a certain time after the master sent the unpark message, the unpark failed and the master shall consider the slave as still being in park mode.

One message is used where the parked device is identified with the PM_ADDR, and another message is used where it is identified with the BD_ADDR. Both messages have variable length depending on the number of slaves the master unparks. For each slave the master wishes to unpark an AM_ADDR followed by the PM/BD_ADDR of the device that is assigned this AM_ADDR is included in the payload. If the slaves are identified with the PM_ADDR a maximum of seven slaves can be unparked with the same message. If they are identified with the BD_ADDR a maximum of two slaves can be unparked with the same message.

After a successful unparking, both units re-enable L2CAP transmission.



Sequence 39: Master unparks slaves addressed with their BD_ADDR.



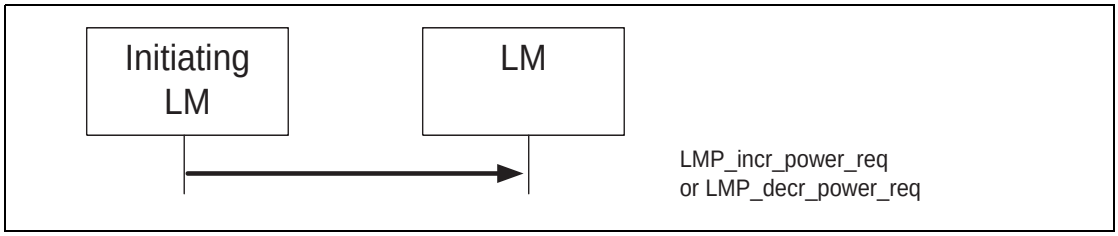
Sequence 40: Master unparks slaves addressed with their PM_ADDR.

9.3.18 Power control

If the RSSI value differs too much from the preferred value of a Bluetooth device, it can request an increase or a decrease of the other device's TX power. The power adjustment requests can be made at anytime following a successful baseband paging procedure. If a device does not support power control requests this is indicated in the supported features list and thus no power control requests shall be sent after the supported features response has been processed. Prior to this time, a power control adjustment might be sent and if the recipient does not support power control it is allowed to send LMP_max_power in response to LMP_incr_power_req and LMP_min_power in response to LMP_decr_power_req. Another possibility is to send LMP_not_accepted with the reason unsupported LMP feature. Upon receipt of this message, the output power is increased or decreased one step. See 7.3 for the definition of the step size. At the master side the TX power is completely independent for different slaves; a request from one slave can only effect the master's TX power for that same slave (see Table 59).

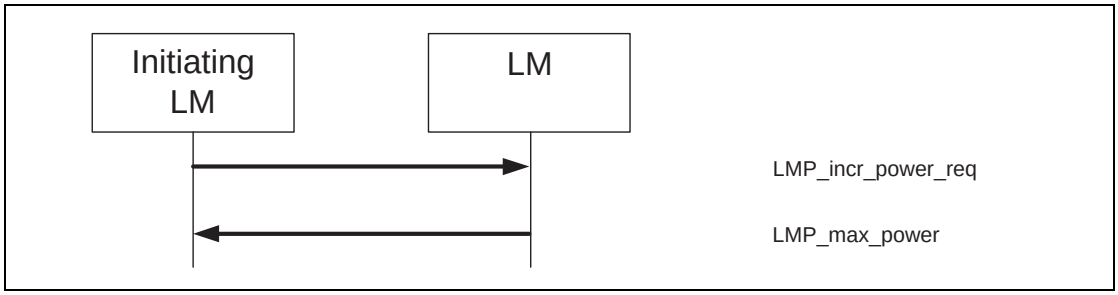
Table 59—PDUs used for power control

M/O	PDU	Contents
O	LMP_incr_power_req	for future use (1 Byte)
O	LMP_decr_power_req	for future use (1 Byte)
O	LMP_max_power	—
O	LMP_min_power	—

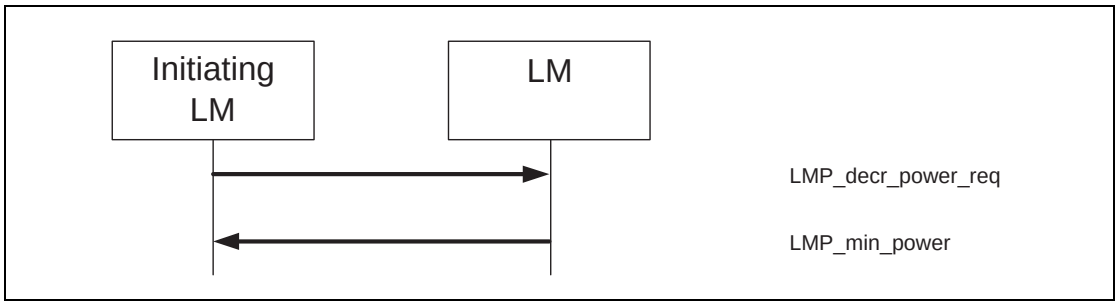


Sequence 41: A device requests a change of the other device's TX power.

If the receiver of LMP_incr_power_req already transmits at maximum power, LMP_max_power is returned. The device may then only request an increase again after having requested a decrease at least once. Similarly, if the receiver of LMP_decr_power_req already transmits at minimum power then LMP_min_power is returned and the device may only request a decrease again after having requested an increase at least once.



Sequence 42: The TX power cannot be increased.



Sequence 43: The TX power cannot be decreased.

One byte is reserved in LMP_incr/decr_power_req for future use. It could, for example, be the mismatch between preferred and measured RSSI. The receiver of LMP_incr/decr_power_req could then use this value to adjust to the correct power at once, instead of only changing it one step for each request. The parameter value shall be 0x00 for all versions of LMP where this parameter is not yet defined.

9.3.19 Channel quality-driven change between DM and DH

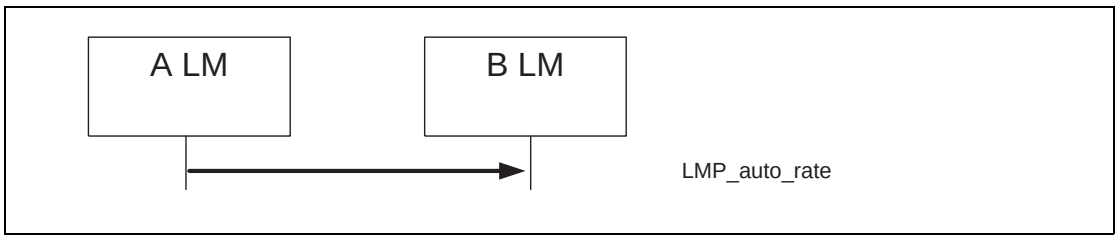
The data throughput for a given packet type depends on the quality of the RF channel. Quality measurements in the receiver of one device can be used to dynamically control the packet type transmitted from the remote device for optimization of the data throughput. If a device A wants the remote device B to have this control it sends LMP_auto_rate once. The device B can then send back LMP_preferred_rate to

device A whenever it wishes to change the packet type that A transmits. This PDU has a parameter which determines the preferred coding (with or without 2/3FEC) and the preferred size (in slots) of the packets. Device A is not required to change to the packet type specified by this parameter and may never send a packet that is larger than the maximum allowed number of slots even if the preferred size is greater than this value.

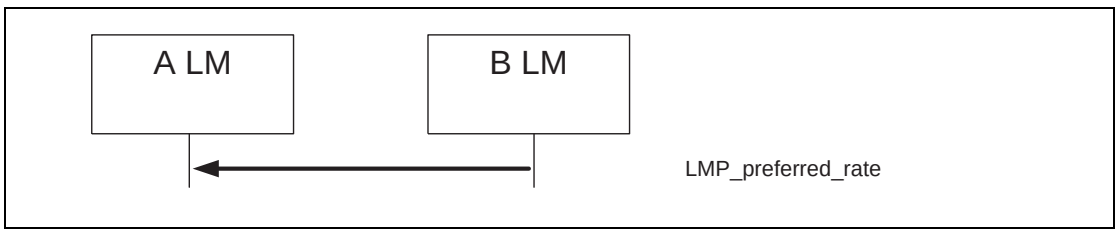
These PDUs can be sent at anytime after connection setup is completed. These PDUs shall not be sent to a given Bluetooth device if the supported features list of the device shows that it does not support the channel quality driven change requests (see Table 60).

Table 60—PDUs used for quality driven change of the data rate

M/O	PDU	Contents
O	LMP_auto_rate	—
O	LMP_preferred_rate	data rate



Sequence 44: A wants B to control A's packet type



Sequence 45: B changes A's packet type.

9.3.20 Quality of service (QoS)

The Link Manager provides QoS capabilities. A poll interval, which is defined as the maximum time between subsequent transmissions from the master to a particular slave on the ACL link, is used to support bandwidth allocation and latency control. The poll interval is guaranteed in the active mode except when there are collisions with page, page scan, inquiry, and inquiry scan. The poll interval is also known as T_{poll}. These PDUs can be sent at anytime after connection setup is completed.

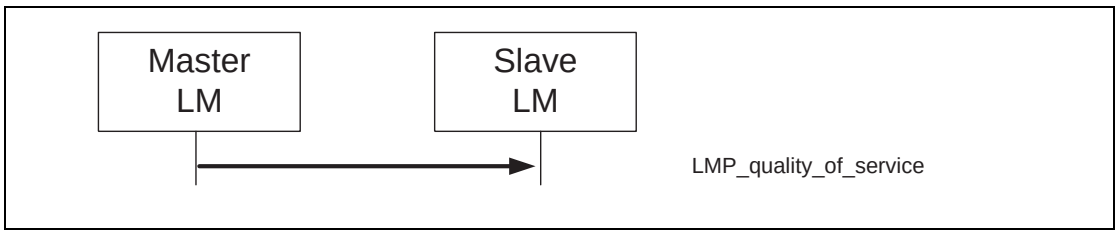
In addition, master and slave negotiate the number of repetitions for broadcast packets (N_{BC}); see 8.5.3.5 and Table 61.

Table 61— PDUs used for QoS

M/O	PDU	Contents
M	LMP_quality_of_service	poll interval N_{BC}
M	LMP_quality_of_service_req	poll interval N_{BC}

9.3.20.1 Master notifies slave of the QoS

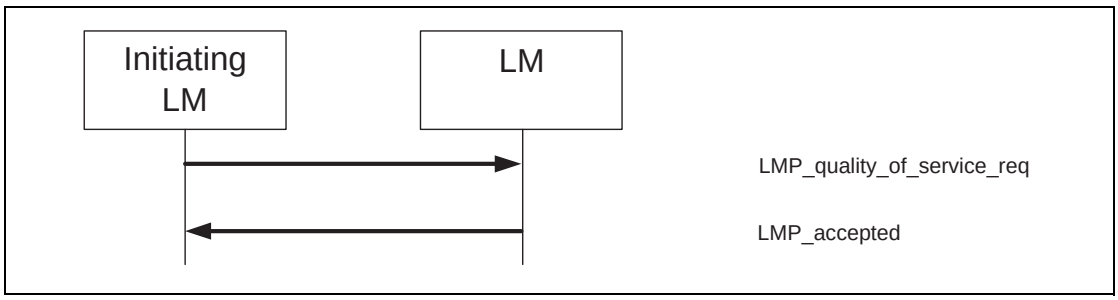
In this case the master notifies the slave of the new poll interval and N_{BC}. The slave cannot reject the notification.



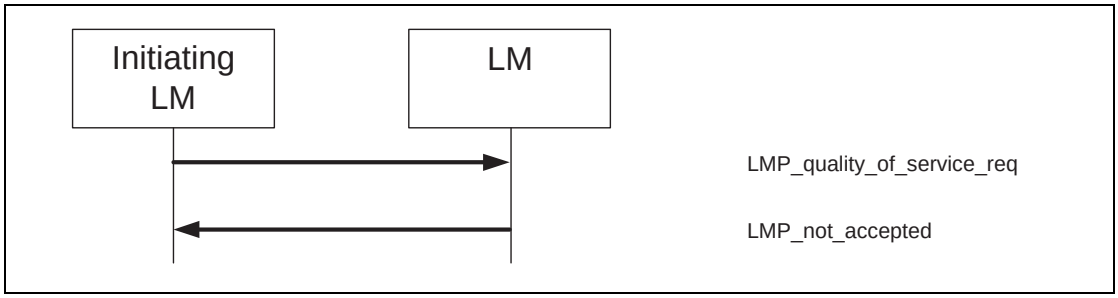
Sequence 46: Master notifies slave of the QoS.

9.3.20.2 Device requests new QoS

In this case the master or slave requests a new poll interval and N_{BC}. The parameter N_{BC} is meaningful only when it is sent by a master to a slave. For transmission of LMP_quality_of_service_req PDUs from a slave, this parameter is ignored by the master. The request can be accepted or rejected. This will allow the master and slave to dynamically negotiate the QoS as needed.



Sequence 47: Device accepts new QoS.



Sequence 48: Device rejects new QoS.

9.3.21 SCO Links

When a connection has been established between two Bluetooth devices, the connection consists of an ACL link. One or more SCO links can then be established. The SCO link reserves slots separated by the SCO interval, T_{sco}. The first slot reserved for the SCO link is defined by T_{sco} and the SCO offset, D_{sco}. After that the SCO slots follows periodically with the SCO interval. To avoid problems with a wrap-around of the clock during initialization of the SCO link, a flag indicating how the first SCO slot should be calculated is included in a message from the master. Note: Only bit 0 and bit 1 of this field is valid. Each SCO link is distinguished from all other SCO links by an SCO handle. The SCO handle zero is never used (see Table 62).

Table 62—PDUs used for managing the SCO links

M/O	PDU	Contents
O	LMP_SCO_link_req	SCO handle timing control flags D_{sco} T_{sco} SCO packet air mode
O	LMP_remove_SCO_link_req	SCO handle reason

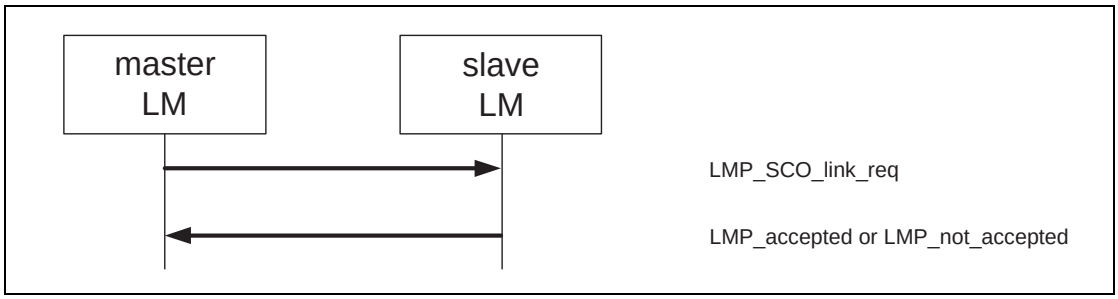
9.3.21.1 Master initiates an SCO link

When establishing an SCO link the master sends a request with parameters that specify the timing, packet type and coding that will be used on the SCO link. For each of the SCO packets, Bluetooth supports three different voice coding formats on the air-interface: μ-law log PCM, A-law log PCM, and CVSD. The air coding by log PCM or CVSD can be deactivated to achieve a transparent synchronous data link at 64 kbits/s.

The slots used for the SCO links are determined by three parameters controlled by the master: T_{sco}, $D_{sco,}$ and a flag indicating how the first SCO slot should be calculated. After the first slot, the SCO slots follows periodically with the T_{sco}.

If the slave does not accept the SCO link, but is willing to consider another possible set of SCO parameters, it can indicate what it does not accept in the error reason field of LMP_not_accepted. The master then has the possibility to issue a new request with modified parameters.

The SCO handle in the message shall be different from any already existing SCO link(s). Note that if the SCO packet type is HV1 the LMP_accepted shall be sent using the DM1 packet.

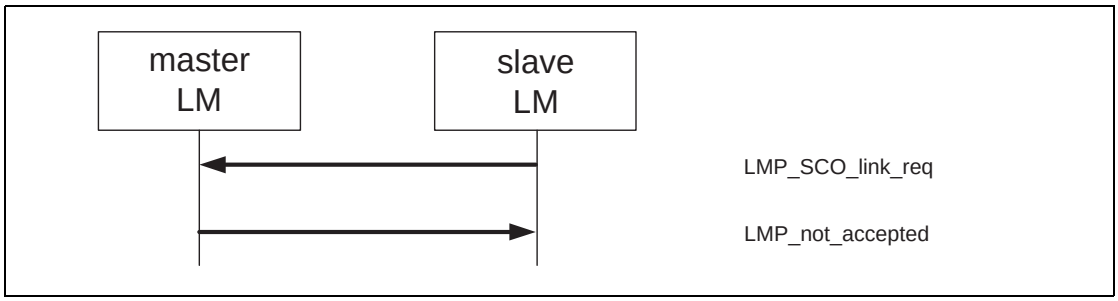


Sequence 49: Master requests an SCO link.

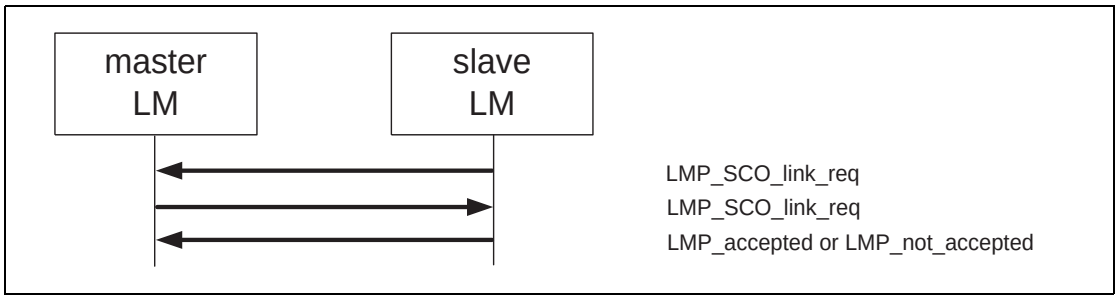
9.3.21.2 Slave initiates an SCO link

The slave can also initiate the establishment of an SCO link. The slave sends LMP_SCO_link_req, but the parameters timing control flags and D_{sco} are invalid as well as the SCO handle, which shall be zero. If the master is not capable of establishing an SCO link, it replies with LMP_not_accepted. Otherwise, it sends back LMP_SCO_link_req. This message includes the assigned SCO handle, $D_{sco,}$ and the timing control flags. For the other parameters, the master should try to use the same parameters as in the slave request; if the master cannot meet that request, it is allowed to use other values. The slave shall then reply with LMP_accepted or LMP_not_accepted.

Note that if the SCO packet type is HV1 the LMP_accepted shall be sent using the DM1 packet.



Sequence 50: Master rejects slave's request for an SCO link.



Sequence 51: Master accepts slave's request for an SCO link.

9.3.21.3 Master requests change of SCO parameters

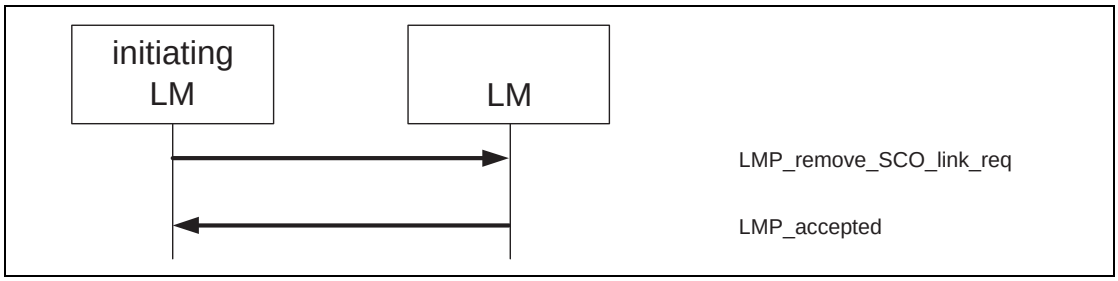
The master sends LMP_SCO_link_req, where the SCO handle is the handle of the SCO link the master wishes to change parameters for. If the slave accepts the new parameters, it replies with LMP_accepted and the SCO link will change to the new parameters. If the slave does not accept the new parameters, it replies with LMP_not_accepted and the SCO link is left unchanged. When the slave replies with LMP_not_accepted, it shall indicate in the error reason parameter what it does not accept. The master can then try to change the SCO link again with modified parameters. The sequence is the same as in 9.3.21.1.

9.3.21.4 Slave requests change of SCO parameters

The slave sends LMP_SCO_link_req, where the SCO handle is the handle of the SCO link the slave wishes to change parameters for. The parameters timing control flags and D_{sco} are not valid in this message. If the master does not accept the new parameters, it replies with LMP_not_accepted and the SCO link is left unchanged. If the master accepts the new parameters, it replies with LMP_SCO_link_req, where it shall use the same parameters as in the slave request. When receiving this message, the slave replies with LMP_not_accepted if it does not accept the new parameters. The SCO link is then left unchanged. If the slave accepts the new parameters, it replies with LMP_accepted and the SCO link will change to the new parameters. These sequences are the same as in 9.3.21.2.

9.3.21.5 Remove an SCO link

Master or slave can remove the SCO link by sending a request including the SCO handle of the SCO link to be removed and a reason indicating why the SCO link is removed. The receiving party shall respond with LMP_accepted.



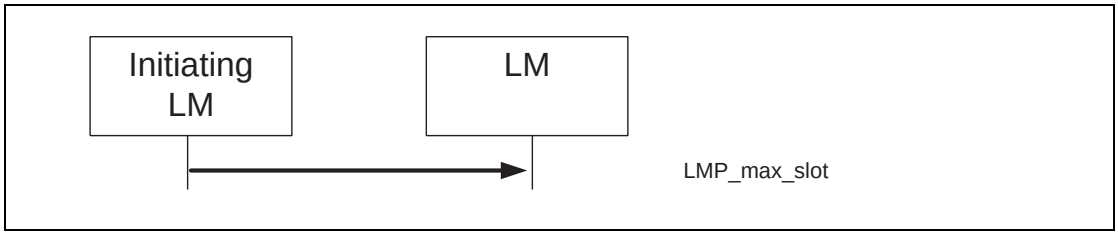
Sequence 52: SCO link removed.

9.3.22 Control of multislot packets

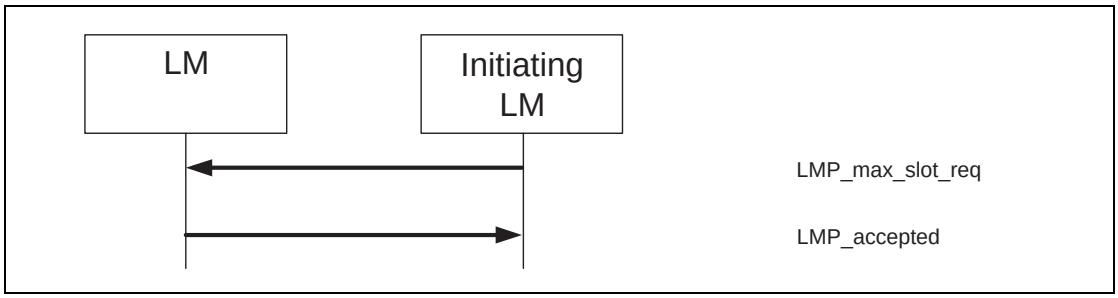
The number of slots used by a device can be limited. A device allows the remote device to use a maximal number of slots by sending the PDU LMP_max_slot providing max slots as parameter. Each device can request to use a maximal number of slots by sending the PDU LMP_max_slot_req providing max slots as parameter. After a new connection, as a result of page, page scan, master-slave switch, or unpark, the default value is 1 slot. Two PDUs are used for the control of multislot packets.These PDUs can be sent at anytime after connection setup is completed (see Table 63).

Table 63—PDUs used to control the use of multislot packets

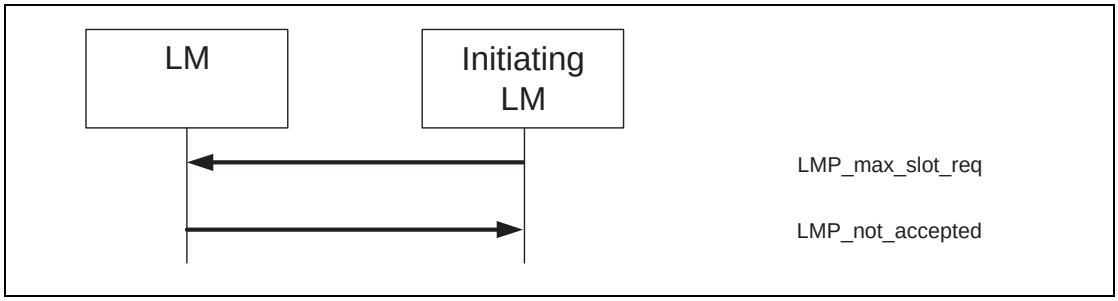
M/O	PDU	Contents
M	LMP_max_slot	max slots
M	LMP_max_slot_req	max slots



Sequence 53: Device allows Remote Device to use a maximum number of slots.



Sequence 54: Device requests a maximum number of slots. Remote Device accepts.



Sequence 55: Device requests a maximum number of slots. Remote Device rejects.

9.3.23 Paging scheme

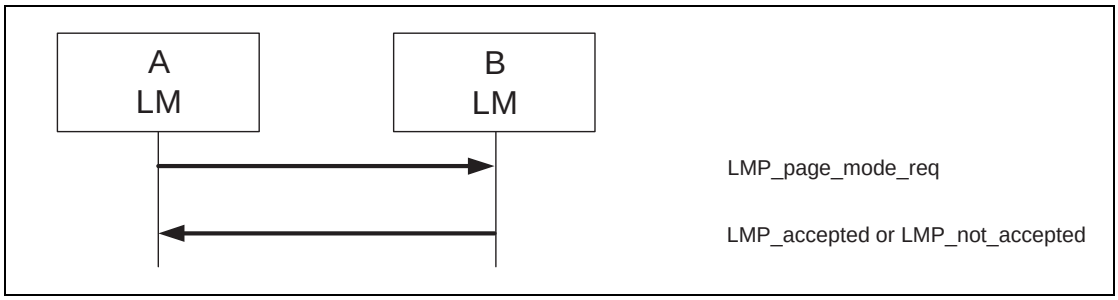
In addition to the mandatory paging scheme, Bluetooth defines optional paging schemes; see Annex D. LMP provides a means to negotiate the paging scheme, which is to be used the next time a unit is paged (see Table 64).

Table 64—PDUs used to request paging scheme

M/O	PDU	Contents
O	LMP_page_mode_req	paging scheme paging scheme settings
O	LMP_page_scan_mode_req	paging scheme paging scheme settings

9.3.23.1 Page mode

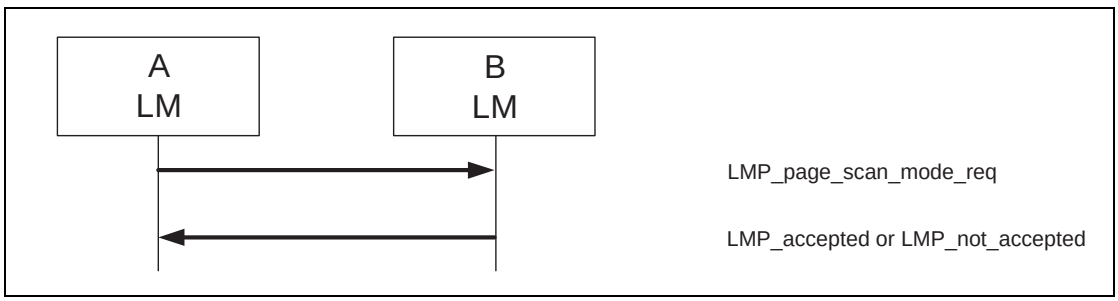
This procedure is initiated from device A and negotiates the paging scheme used when device A pages device B. Device A proposes a paging scheme including the parameters for this scheme and device B can accept or reject. On rejection the old setting is not changed. A request to switch back to the mandatory scheme may be rejected.



Sequence 56: Negotiation for page mode.

9.3.23.2 Page scan mode

This procedure is initiated from device A and negotiates the paging scheme used when device B pages device A. Device A proposes a paging scheme including the parameters for this scheme and device B can accept or reject. On rejection the old setting is not changed. A request to switch to the mandatory scheme shall be accepted.



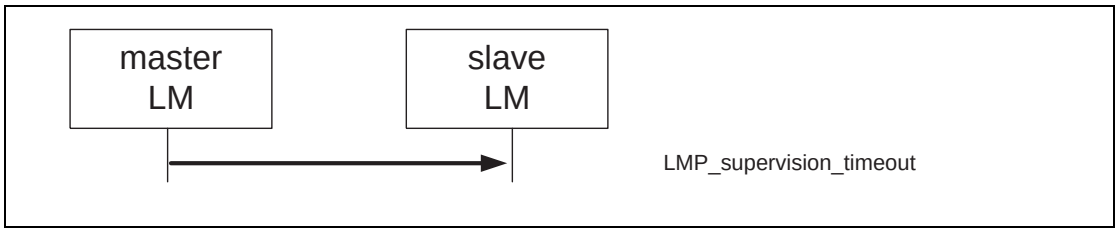
Sequence 57: Negotiation for page scan mode.

9.3.24 Link supervision

Each Bluetooth link has a timer that is used for link supervision. This timer is used to detect link loss caused by devices moving out of range, a device's power-down, or other similar failure cases. The scheme for link supervision is described in 8.10.11. An LMP procedure is used to set the value of the supervision timeout (see Table 65).

Table 65—PDU used to set the supervision timeout

M/O	PDU	Contents
M	LMP_supervision_timeout	supervision timeout



Sequence 58: Setting the link supervision timeout.

9.4 Connection Establishment

After the paging procedure, the master shall poll the slave with a max poll interval as defined in Table 71. LMP procedures for clock offset request, LMP version, supported features, name request and detach can then be carried out.

When the paging device wishes to create a connection involving layers above LM, it sends LMP_host_connection_req (see Table 66). When the other side receives this message, the host is informed about the incoming connection. The remote device can accept or reject the connection request by sending LMP_accepted or LMP_not_accepted. Alternatively, if the slave needs a master-slave switch (see 9.3.12), it sends LMP_slot_offset and LMP_switch_req after it has received LMP_host_connection_req. When the master-slave switch has been successfully completed, the old slave will reply with LMP_accepted or LMP_not_accpted to LMP_host_connection_req (with the transaction ID set to 0).

If LMP_host_connection_req is accepted, LMP security procedures (pairing, authentication, and encryption) can be invoked. When a device is not going to initiate any more security procedures during connection establishment, it sends LMP_setup_complete (see Table 66). When both devices have sent LMP_setup_complete the first packet on a logical channel different from LMP can then be transmitted.

Note that the transaction ID shall be 0 if LMP_setup_complete is sent from the master and 1 if it is sent from the slave. Figure 93 summarizes the LMP transactions for establishing a connection following the baseband page procedure.

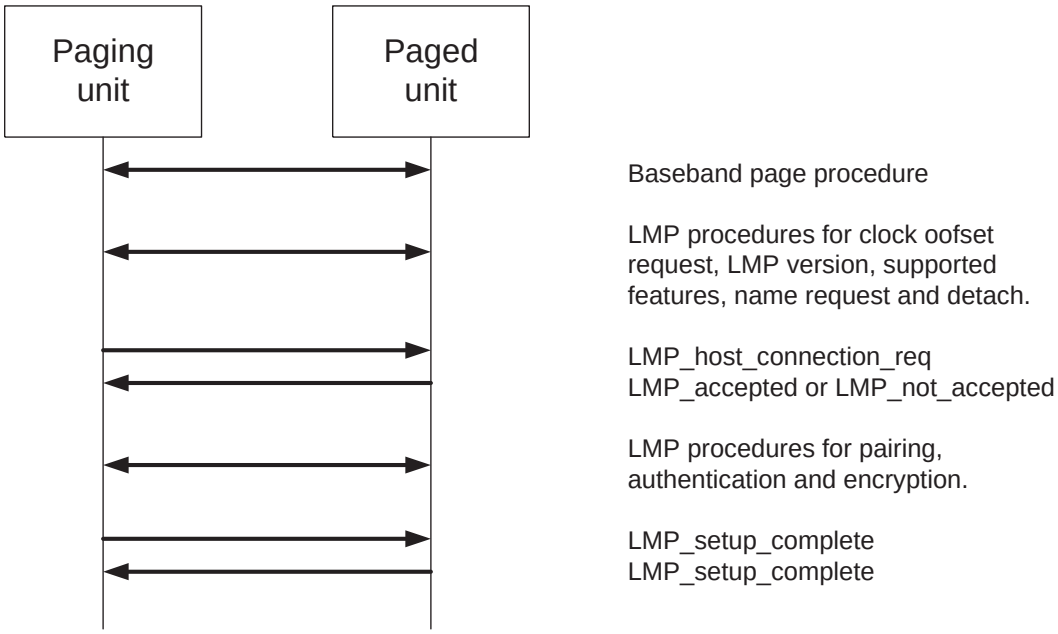


Figure 93—Connection establishment

Table 66—PDUs used for connection establishment

M/O	PDU	Contents
M	LMP_host_connection_req	—
M	LMP_setup_complete	—

9.5 Summary of PDUs

Table 67 provides the coding of the different LM PDUs.

Table 67—Coding of the different LM PDUs

LMP PDU	Length (bytes)	op code	Packet type	Possible direction	Contents	Position in payload
LMP_accepted	2	3	DM1/ DV	m ↔ s	op code	2
LMP_au_rand	17	11	DM1	m ↔ s	random number	2–17
LMP_auto_rate	1	35	DM1/ DV	m ↔ s	—	
LMP_clkoffset_req	1	5	DM1/ DV	m → s	—	
LMP_clkoffset_res	3	6	DM1/ DV	m ← s	clock offset	2–3
LMP_comb_key	17	9	DM1	m ↔ s	random number	2–17
LMP_decr_power_req	2	32	DM1/ DV	m ↔ s	for future use	2
LMP_detach	2	7	DM1/ DV	m ↔ s	reason	2
LMP_encryption_key_size_req	2	16	DM1/ DV	m ↔ s	key size	2
LMP_encryption_mode_ req	2	15	DM1/ DV	m ↔ s	encryption mode	2
LMP_features_req	9	39	DM1/ DV	m ↔ s	features	2–9
LMP_features_res	9	40	DM1/ DV	m ↔ s	features	2–9
LMP_host_connection_req	1	51	DM1/ DV	m ↔ s	—	
LMP_hold	7	20	DM1/ DV	m ↔ s	hold time, hold instant	4–7
LMP_hold_req	7	21	DM1/ DV	m ↔ s	hold time, hold instant	4–7
LMP_incr_power_req	2	31	DM1/ DV	m ↔ s	for future use	2
LMP_in_rand	17	8	DM1	m ↔ s	random number	2–17
LMP_max_power	1	33	DM1/ DV	m ↔ s	—	
LMP_max_slot	2	45	DM1/ DV	m ↔ s	max slots	2

Table 67—Coding of the different LM PDUs *(continued)*

LMP PDU	Length (bytes)	op code	Packet type	Possible direction	Contents	Position in payload
LMP_max_slot_req	2	46	DM1/ DV	m ↔ s	max slots	2
LMP_min_power	1	34	DM1/ DV	m ↔ s	—	
LMP_modify_beacon	11 or 13	28	DM1	m → s	timing control flags	2
					D_B	3–4
					T_B	5–6
					N_B	7
					Δ_B	8
					D_{access}	9
					T_{access}	10
					$N_{acc\text{-}slots}$	11
					N_{poll}	12
					M_{access}	13:0–3
					access scheme	13:4–7
LMP_name_req	2	1	DM1/ DV	m ↔ s	name offset	2
LMP_name_res	17	2	DM1	m ↔ s	name offset	2
					name length	3
					name fragment	4–17
LMP_not_accepted	3	4	DM1/ DV	m ↔ s	op code	2
					reason	3
LMP_page_mode_req	3	53	DM1/ DV	m ↔ s	paging scheme	2
					paging scheme settings	3
LMP_page_scan_mode_req	3	54	DM1/ DV	m ↔ s	paging scheme	2
					paging scheme settings	3

Table 67—Coding of the different LM PDUs *(continued)*

LMP PDU	Length (bytes)	op code	Packet type	Possible direction	Contents	Position in payload
LMP_park_req	17	25	DM	m → s	timing control flags	2
					D_B	3–4
					T_B	5–6
					N_B	7
					Δ_B	8
					PM_ADDR	9
					AR_ADDR	10
					N_{Bsleep}	11
					D_{Bsleep}	12
					D_{access}	13
					T_{access}	14
					$N_{acc\text{-}slots}$	15
					N_{poll}	16
					M_{access}	17:0–3
					access scheme	17:4–7
LMP_preferred_rate	2	36	DM1/ DV	m ↔ s	data rate	2
LMP_quality_of_service	4	41	DM1/ DV	m → s	poll interval	2–3
					N_{BC}	4
LMP_quality_of_service_req	4	42	DM1/ DV	m ↔ s	poll interval	2–3
					N_{BC}	4
LMP_remove_SCO_link_req	3	44	DM1/ DV	m ↔ s	SCO handle	2
					reason	3
LMP_SCO_link_req	7	43	DM1/ DV	m ↔ s	SCO handle	2
					timing control flags	3
					D_{sco}	4
					T_{sco}	5
					SCO packet	6
					air mode	7
LMP_set_broadcast_ scan_window	4 or 6	27	DM1	m → s	timing control flags	2
					D_B	3–4
					broadcast scan window	5–6

Table 67—Coding of the different LM PDUs *(continued)*

LMP PDU	Length (bytes)	op code	Packet type	Possible direction	Contents	Position in payload
LMP_setup_complete	1	49	DM1	m ↔ s	—	
LMP_slot_offset	9	52	DM1/ DV	m ↔ s	slot offset	2–3
					BD_ADDR	4–9
LMP_sniff_req	10	23	DM1	m ↔ s	timing control flags	2
					D_{sniff}	3–4
					T_{sniff}	5–6
					sniff attempt	7–8
					sniff timeout	9–10
LMP_sres	5	12	DM1/ DV	m ↔ s	authentication response	2–5
LMP_start_encryption_req	17	17	DM1	m → s	random number	2–17
LMP_stop_encryption_req.	1	18	DM1/ DV	m → s	—	
LMP_supervision_timeout	3	55	DM1/ DV	m →s	supervision timeout	2–3
LMP_switch_req	5	19	DM1/ DV	m ↔ s	switch instant	2–5
LMP_temp_rand	17	13	DM1	m → s	random number	2–17
LMP_temp_key	17	14	DM1	m → s	key	2–17
LMP_timing_accuracy_req	1	47	DM1/ DV	m ↔ s	—	
LMP_timing_accuracy_res	3	48	DM1/ DV	m ↔ s	drift	2
					jitter	3
LMP_unit_key	17	10	DM1	m ↔ s	key	2–17
LMP_unpark_BD_ADDR_req	variable	29	DM1	m → s	timing control flags	2
					D_B	3–4
					AM_ADDR 1st unpark	5:0–2
					AM_ADDR 2nd unpark	5:4–6
					BD_ADDR 1st unpark	6–11
					BD_ADDR 2nd unpark	12–17

Table 67—Coding of the different LM PDUs *(continued)*

LMP PDU	Length (bytes)	op code	Packet type	Possible direction	Contents	Position in payload
LMP_unpark_PM_ADDR_req	variable	30	DM1	m → s	timing control flags	2
					D_B	3–4
					AM_ADDR 1st unpark	5:0–3
					AM_ADDR 2nd unpark	5:4–7
					PM_ADDR 1st unpark	6
					PM_ADDR 2nd unpark	7
LMP_unsniff_req	1	24	DM1/ DV	m ↔ s	—	
LMP_use_semi_ permanent_key	1	50	DM1/ DV	m → s	—	
LMP_version_req	6	37	DM1/ DV	m ↔ s	VersNr	2
					CompId	3–4
					SubVersNr	5–6
LMP_version_res	6	38	DM1/ DV	m ↔ s	VersNr	2
					CompId	3–4
					SubVersNr	5–6

Note 1—For LMP_set_broadcast_scan_window, LMP_modify_beacon, LMP_unpark_BD_ADDR_req, and LMP_unpark_PM_ADDR_req, the parameter D_B is optional. This parameter is only present if bit 0 of *timing control flags* is 1. If the parameter is not included, the position in payload for all parameters following D_B are decreased by 2.

Note 2—For LMP_unpark_BD_ADDR, the AM_ADDR and the BD_ADDR of the 2nd unparked slave are optional. If only one slave is unparked, "AM_ADDR 2nd unpark" should be zero and "BD_ADDR 2nd unpark" is left out.

Note 3—For LMP_unpark_PM_ADDR, the AM_ADDR and the PM_ADDR of the 2nd through 7th unparked slaves are optional. If N slaves are unparked, the fields up to and including the N^{th} unparked slave are present. If N is odd, the "AM_ADDR (N+1)th unpark" shall be zero. The length of the message is $x + 3N/2$ if N is even and $x + 3(N+1)/2 - 1$ if N is odd, where $x = 2$ or 4 depending on if the D_B is included or not (see Note 1).

9.5.1 Description of parameters

Table 68 contains the parameters that appear in the various LM_PDUs shown in Table 67.

Table 68—Parameters in LM PDUs

Name	Length (bytes)	Type	Unit	Detailed
access scheme	1	u_int4		0: polling technique 1–15: Reserved
air mode	1	u_int8		0: μ-law log 1: A-law log 2: CVSD 3: transparent data 4–255: Reserved
AM_ADDR	1	u_int4		
AR_ADDR	1	u_int8		
authentication response	4	multiple bytes		
BD_ADDR	6	multiple bytes		
broadcast scan window	2	u_int16	slots	
clock offset	2	u_int16	1.25ms	($CLKN_{16-2}$ slave - $CLKN_{16-2}$ master) mod 2^{15} (MSbit of second byte not used)
CompId	2	u_int16		See 2.4.3
D_{access}	1	u_int8	slots	
D_B	2	u_int16	slots	
D_{Bsleep}	1	u_int8	slots	
data rate	1	u_int8		bit0 = 0: use FEC bit0 = 1: do not use FEC bit1-2=0: No packet-size preference available bit1-2=1: use 1-slot packets bit1-2=2: use 3-slot packets bit1-2=3: use 5-slot packets bit3-7: Reserved
drift	1	u_int8	ppm	
D_{sco}	1	u_int8	slots	
D_{sniff}	2	u_int16	slots	
encryption mode	1	u_int8		0: no encryption 1: point-to-point encryption 2: point-to-point and broadcast encryption 3–255: Reserved

Table 68—Parameters in LM PDUs *(continued)*

Name	Length (bytes)	Type	Unit	Detailed
features	8	multiple bytes		See Table 69
hold instant	4	u_int32	slots	Bits 27:1 of the master Bluetooth clock value
hold time	2	u_int16	slots	
jitter	1	u_int8	µs	
key	16	multiple bytes		
key size	1	u_int8	byte	
M_{access}	1	u_int4		number of access windows
max slots	1	u_int8	slots	
$N_{acc\text{-}slots}$	1	u_int8	slots	
name fragment	14	multiple bytes		UTF-8 characters.
name length	1	u_int8	bytes	
name offset	1	u_int8	bytes	
N_B	1	u_int8		
N_{BC}	1	u_int8		
N_{Bsleep}	1	u_int8	slots	
N_{poll}	1	u_int8	slots	
op code	1	u_int8		
paging scheme	1	u_int8		0: mandatory scheme 1: optional scheme I 2: optional scheme II 3: optional scheme III 4–255: Reserved
paging scheme settings	1	u_int8		For mandatory scheme: 0: R0 1: R1 2: R2 3–255: Reserved For optional scheme 1: 0: Reserved 1: R1 2: R2 3–255: Reserved
PM_ADDR	1	u_int8		
poll interval	2	u_int16	slots	
random number	16	multiple bytes		
reason	1	u_int8		See Table 70
SCO handle	1	u_int8		

Table 68—Parameters in LM PDUs *(continued)*

Name	Length (bytes)	Type	Unit	Detailed
SCO packet	1	u_int8		0: HV1 1: HV2 2: HV3 3–255: Reserved
slot offset	2	u_int16	μs	0 ≤ slot offset < 1250
sniff attempt	2	u_int16	slots	Number of receive slots
sniff timeout	2	u_int16	slots	Number of receive slots
SubVersNr	2	u_int16		Defined by each company
supervision timeout	2	u_int16	slots	0 means an infinite timeout
switch instant	4	u_int32	slots	Bits 27:1 of the master Bluetooth clock value
T_{access}	1	u_int8	slots	
T_B	2	u_int16	slots	
timing control flags	1	u_int8		bit0 = 0: no timing change bit0 = 1: timing change bit1 = 0: use initialization 1 bit1 = 1: use initialization 2 bit2 = 0: access window bit2 = 1: no access window bit3–7: Reserved
T_{sco}	1	u_int8	slots	
T_{sniff}	2	u_int16	slots	
VersNr	1	u_int8		See 2.4.3
Δ_B	1	u_int8	slots	

9.5.1.1 Coding of features

This parameter is a bitmap with information about the Bluetooth radio, baseband, and LMP features which a device supports. The bit shall be one if the feature is supported. In addition to the bitmap information the feature parameter has a 3-bit field denoted flow control lag. This is defined as the total amount of L2CAP data that can be sent following the receipt of a valid payload header with the payload header flow bit set to 0 and is in units of 256 bytes (see 8.4.5.2). The feature parameter bits that are not defined in Table 69 shall be 0.

Table 69—Coding of the parameter features

Byte	Bit	Supported feature
0	0	3–slot packets
	1	5–slot packets
	2	encryption
	3	slot offset
	4	timing accuracy
	5	switch
	6	hold mode
	7	sniff mode
1	0	park mode
	1	RSSI
	2	channel quality driven data rate
	3	SCO link
	4	HV2 packets
	5	HV3 packets
	6	u-law log
	7	A-law log
2	0	CVSD
	1	paging scheme
	2	power control
	3	transparent SCO data
	4	Flow control lag (bit0)
	5	Flow control lag (bit1)
	6	Flow control lag (bit2)

9.5.1.2 List of error reasons

Table 70 contains the codes of the different error reasons used in LMP.

Table 70—List of error reasons

Reason	Description
0x05	Authentication failure
0x06	Key missing
0x0A	Max number of SCO connections to a device. (The maximum number of SCO connections to a particle device has been reached. All allowed SCO connection handles to that device are used.)
0x0D	Host rejected due to limited resources. (The host at the remote side has rejected the connection because the remote host did not have enough additional resources to accept the connection.)
0x0E	Host rejected due to security reasons. (The host at the remote side has rejected the connection because the remote host determined that the local host did not meet its security criteria.)
0x0F	Host rejected due to remote device is only a personal device. (The host at the remote side has rejected the connection because the remote host is a personal device and will only accept the connection from one particle remote host.)
0x10	Host timeout. (Used at connection accept timeout, the host did not respond to an incoming connection attempt before the connection accept timer expired.)
0x13	Other end terminated connection: User ended connection
0x14	Other end terminated connection: Low resources
0x15	Other end terminated connection: About to power off
0x16	Connection terminated by Local Host
0x17	Repeated attempts. (An authentication or pairing attempt is made too soon after a previously failed authentication or pairing attempt.)
0x18	Pairing not allowed.
0x19	Unknown LMP PDU
0x1A	Unsupported LMP feature
0x1B	SCO offset rejected
0x1C	SCO interval rejected
0x1D	SCO air mode rejected
0x1E	Invalid LMP parameters
0x1F	Unspecified error
0x20	Unsupported parameter value
0x21	Switch not allowed
0x23	LMP error transaction collision
0x24	PDU not allowed
0x25	Encryption mode not acceptable
0x26	Unit key used
0x27	QoS not supported
0x28	Instant passed
0x29	Pairing with unit key not supported

9.5.2 Default values

The Bluetooth device shall use the values in Table 71 before anything else has been negotiated:

Table 71—Default values

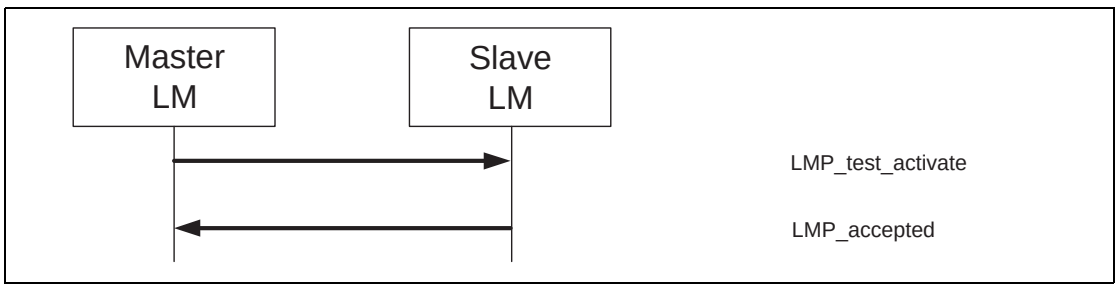
Parameter	Value
drift	250
jitter	10
max slots	1
poll interval	40

9.6 Test modes

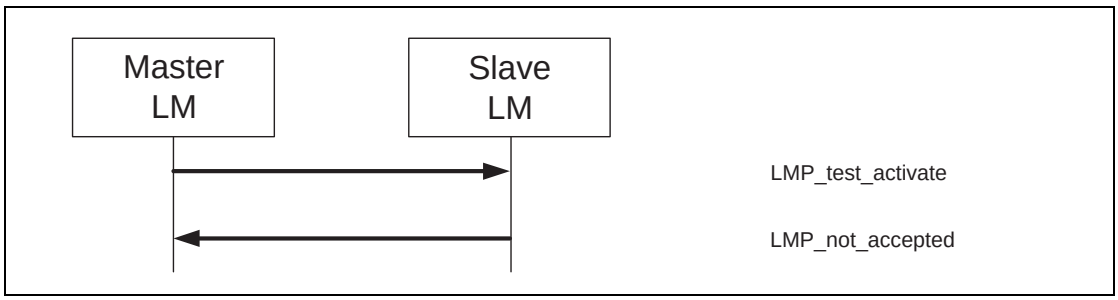
LMP has PDUs to support different Bluetooth test modes, which are used for certification and compliance testing of the Bluetooth radio and baseband. See Annex E for a detailed description of these test modes.

9.6.1 Activation and deactivation of test mode

The test mode is activated by sending LMP_test_activate to the device under test (DUT). The DUT is always the slave. The link manager shall be able to receive this message anytime. If entering test mode is locally enabled in the DUT it responds with LMP_accepted and test mode is entered. Otherwise, the DUT responds with LMP_not_accepted and the DUT remains in normal operation. The reason code in LMP_not_accepted shall be *PDU not allowed*.



Sequence 59: Activation of test mode successful.

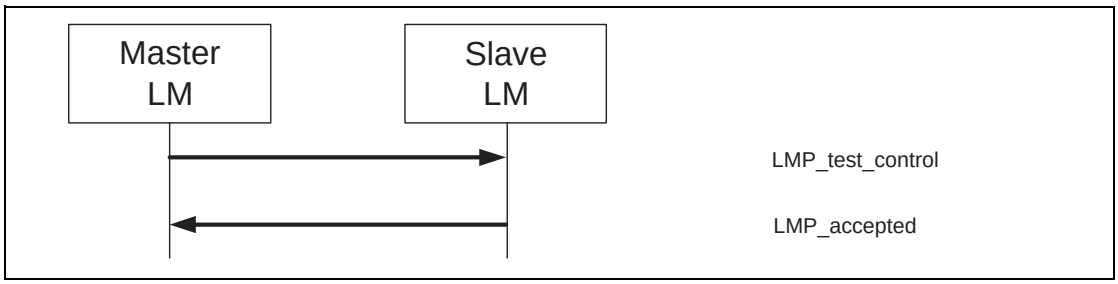


Sequence 60: Activation of test mode fails. Slave is not allowed to enter test mode.

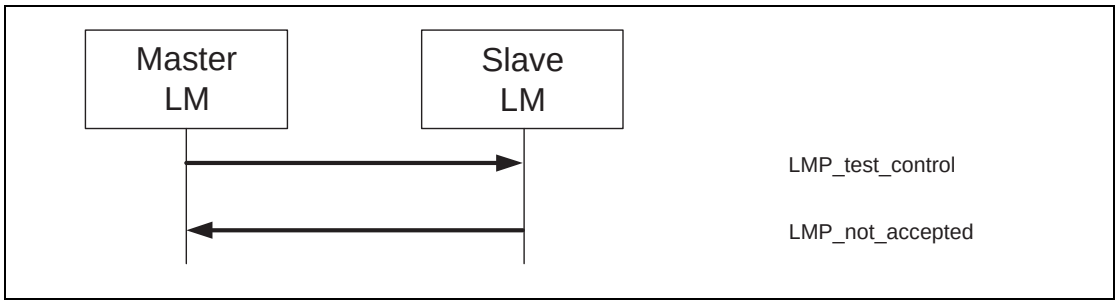
The test mode can be deactivated in two ways. Sending LMP_test_control with the test scenario set to “exit test mode” exits the test mode and the slave returns to normal operation still connected to the master. Sending LMP_detach to the DUT ends the test mode and the connection.

9.6.2 Control of test mode

When the DUT has entered test mode, the PDU LMP_test_control can be sent to the DUT to start a specific test. This PDU is acknowledged with LMP_accepted. If a device that is not in test mode receives LMP_test_control it responds with LMP_not_accepted, where the reason code shall be *PDU not allowed*.



Sequence 61: Control of test mode successful.



Sequence 62: Control of test mode rejected since slave is not in test mode.

9.6.3 Summary of test mode PDUs

The PDUs used for test purposes are summarized in Table 72. For a detailed description of the parameters, see Table E.2.

Table 72—Test mode PDUs

M/O	LMP PDU	Length	op code	Packet type	Possible direction	Contents	Position in payload
M	LMP_test_activate	1	56	DM1/DV	m → s	—	
M	LMP_test_control	10	57	DM1	m → s	test scenario	2
						hopping mode	3
						TX frequency	4
						RX frequency	5
						power control mode	6
						poll period	7
						packet type	8
						length of test data	9–10

9.7 Error Handling

If the Link Manager receives a PDU with unrecognized opcode, it responds with LMP_not_accepted with the reason code *unknown LMP PDU*. The opcode parameter that is echoed back is the unrecognized opcode.

If the Link Manager receives a PDU with invalid parameters, it responds with LMP_not_accepted with the reason code *invalid LMP parameters*.

If the maximum response time (see 9.1), is exceeded or if a link loss is detected (see 8.10.11), the party that waits for the response shall conclude that the procedure has terminated unsuccessfully.

Erroneous LMP messages can be caused by errors on the channel or systematic errors at the transmit side. To detect the latter case, the LM should monitor the number of erroneous messages and disconnect if it exceeds a threshold, which is implementation-dependent.

Since LMP PDUs are not interpreted in real time, collision situations can occur where both LMs initiate the same procedure and both cannot be completed. In this situation, the master shall reject the slave-initiated procedure by sending LMP_not_accepted with the reason code “LMP Error Transaction Collision”. The master-initiated procedure shall then be completed.

When the Link Manager receives a PDU that is not allowed, if the PDU normally expects a PDU reply, for example LMP_host_connection_req or LMP_unit_key, the PDU LMP_not_accepted with the reason code “PDU not allowed” will be returned. If the PDU normally does not expect a reply, for example LMP_sres or LMP_temp_key, the PDU will be ignored.

10. Logical Link Control and Adaptation Protocol Specification

Figure 94 indicates the relationship of the Bluetooth protocol stack to this clause. This clause describes the Bluetooth logical link control and adaptation protocol (L2CAP). This protocol supports higher level protocol multiplexing, packet segmentation, and reassembly, and the conveying of QoS information. It describes the protocol state machine, packet format, and composition.

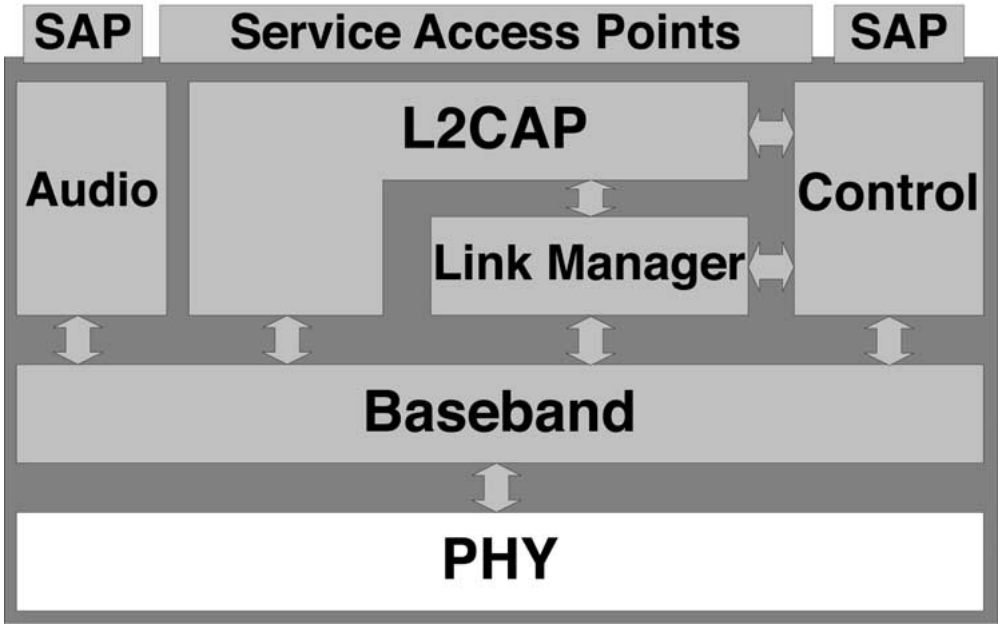


Figure 94—L2CAP interface relationships

10.1 Introduction

This section of the Bluetooth Specification defines the logical link control and adaptation layer protocol, referred to as L2CAP. L2CAP is layered over the baseband protocol and resides in the data link layer as shown in Figure 95. L2CAP provides connection-oriented and connectionless data services to upper layer protocols with protocol multiplexing capability, segmentation and reassembly operation, and group abstractions. L2CAP permits higher level protocols and applications to transmit and receive L2CAP data packets up to 64 kilobytes in length.

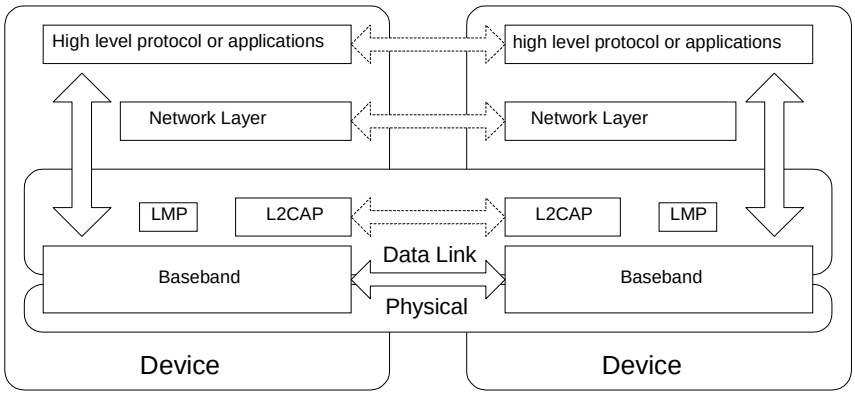


Figure 95—L2CAP within protocol layers

The Baseband specification (Clause 8) defines two link types: SCO links and ACL links. SCO links support real-time voice traffic using reserved bandwidth. ACL links support best effort traffic. The L2CAP specification is defined for only ACL links and no support for SCO links is planned.

For ACL links, use of the AUX1 packet on the ACL link is prohibited. This packet type supports no data integrity checks (no CRC). Because L2CAP depends on integrity checks in the Baseband to protect the transmitted information, AUX1 packets shall never be used to transport L2CAP packets.

The format of the ACL payload header is shown below. Figure 96 displays the payload header used for single-slot packets, and Figure 97 displays the header used in multislot packets. The only difference is the size of the length field. The packet type (a field in the baseband header) distinguishes single-slot packets from multislot packets.

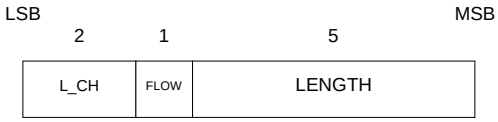


Figure 96—ACL payload header for single-slot packets

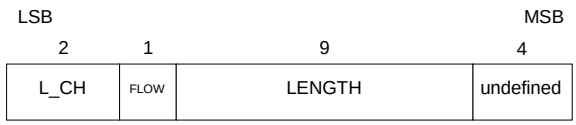


Figure 97—ACL payload header for multislot packets

The 2-bit logical channel (L_CH) field, defined in Table 73, distinguishes L2CAP packets from Link Manager Protocol () packets. The remaining code is reserved for future use.

Table 73—Logical channel L_CH field contents

L_CH code	Logical Channel	Information
00	RESERVED	Reserved for future use
01	L2CAP	Continuation of L2CAP packet
10	L2CAP	Start of L2CAP packet
11	LMP	Link Manager Protocol

The FLOW bit in the ACL header is managed by the LC, a baseband implementation entity, and is normally set to1 (“flow on”). It is set to 0 (“flow off”) when no further L2CAP traffic shall be sent over the ACL link. Sending an L2CAP packet with the FLOW bit set to 1 resumes the flow of incoming L2CAP packets. This is described in more detail in Clause 8.

10.1.1 L2CAP functional requirements

The functional requirements for L2CAP include protocol multiplexing, segmentation, and reassembly (SAR), and group management. Figure 98 illustrates how L2CAP fits into the Bluetooth Protocol Stack. L2CAP lies above the Baseband Protocol in Clause 8 and interfaces with other communication protocols such as the Bluetooth Service Discovery Protocol (SDP), RFCOMM, and Telephony Control (TCS). Voice-quality channels for audio and telephony applications are usually run over Baseband SCO links. Packetized audio data, such as IP Telephony, may be sent using communication protocols running over L2CAP.

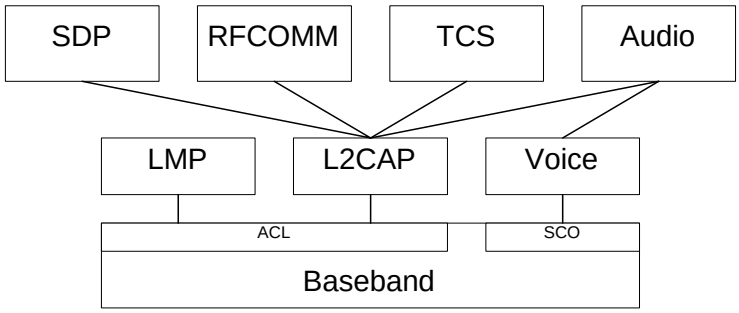


Figure 98—L2CAP in Bluetooth protocol architecture

Essential protocol requirements for L2CAP include simplicity and low overhead. Implementations of L2CAP should be applicable for devices with limited computational resources. L2CAP should not consume excessive power since that significantly sacrifices power efficiency achieved by the Bluetooth radio. Memory requirements for protocol implementation should also be kept to a minimum.

The protocol complexity should be acceptable to personal computers, PDAs, digital cellular phones, wireless headsets, joysticks and other wireless devices supported by Bluetooth. Furthermore, the protocol should be designed to achieve reasonably high bandwidth efficiency.

— *Protocol Multiplexing*

L2CAP supports protocol multiplexing because the Baseband Protocol does not support any “type” field identifying the higher layer protocol being multiplexed above it. L2CAP distinguishes between upper layer protocols such as the Service Discovery Protocol (2.4.2), RFCOMM (2.4.2), and Telephony Control (2.4.2).

— *Segmentation and Reassembly*

Compared to other wired physical media, the data packets defined by the Baseband Protocol (Clause 8) are limited in size. Exporting a maximum transmission unit (MTU) associated with the largest Baseband payload (341 bytes for DH5 packets) limits the efficient use of bandwidth for higher layer protocols that are designed to use larger packets. Large L2CAP packets must be segmented into multiple smaller Baseband packets prior to their transmission over the air. Similarly, multiple received Baseband packets may be reassembled into a single larger L2CAP packet following a simple integrity check (described in 10.2.4.2). The segmentation and reassembly (SAR) functionality is absolutely necessary to support protocols using packets larger than those supported by the Baseband.

— *Quality of Service*

The L2CAP connection establishment process allows the exchange of information regarding the QoS expected between two Bluetooth units. Each L2CAP implementation must monitor the resources used by the protocol and ensure that QoS contracts are honored.

— *Groups*

Many protocols include the concept of a group of addresses. The Baseband Protocol supports the concept of a piconet, a group of devices synchronously hopping together using the same clock. The L2CAP group abstraction permits implementations to efficiently map protocol groups on to piconets. Without a group abstraction, higher level protocols would need to be exposed to the Baseband Protocol and Link Manager functionality in order to manage groups efficiently.

10.1.2 Assumptions

The protocol is designed based on the following assumptions:

1) The ACL link between two units is set up using the Link Manager Protocol (Clause 9). The Baseband provides orderly delivery of data packets, although there might be individual packet corruption and duplicates. No more than one ACL link exists between any two devices.

2) The Baseband always provides the impression of full-duplex communication channels. This does not imply that all L2CAP communications are bi-directional. Multicasts and unidirectional traffic (e.g., video) do not require duplex channels.

3) L2CAP provides a reliable channel using the mechanisms available at the Baseband layer. The Baseband always performs data integrity checks when requested and resends data until it has been successfully acknowledged or a timeout occurs. Because acknowledgements may be lost, timeouts may occur even after the data has been successfully sent. The Baseband protocol uses a 1-bit sequence number that removes duplicates. Note that the use of Baseband broadcast packets is prohibited if reliability is required since all broadcasts start the first segment of an L2CAP packet with the same sequence bit.

10.1.3 Features not supported

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

— L2CAP does not transport audio designated for SCO links.

— L2CAP does not enforce a reliable channel or ensure data integrity, that is, L2CAP performs no retransmissions or checksum calculations.

— L2CAP does not support a reliable multicast channel (see 10.4.2).

— L2CAP does not support the concept of a global group name.

10.2 General operation

The L2CAP is based around the concept of *"channels"*. Each one of the end-points of an L2CAP channel is referred to by a *channel identifier.*

10.2.1 Channel identifiers

Channel identifiers (CIDs) are local names representing a logical channel end-point on the device. Identifiers from 0x0001 to 0x003F are reserved for specific L2CAP functions. The null identifier (0x0000) is defined as an illegal identifier and shall never be used as a destination end-point. Implementations are free to manage the remaining CIDs in a manner best suited for that particular implementation, with the provision that the same CID is not reused as a local L2CAP channel endpoint for multiple simultaneous L2CAP channels between a local device and some remote device. Table 74 summarizes the definition and partitioning of the CID name space.

CID assignment is relative to a particular device and a device can assign CIDs independently from other devices (unless it needs to use any of the reserved CIDs shown in Table 74). Thus, even if the same CID value has been assigned to (remote) channel endpoints by several remote devices connected to a single local device, the local device can still uniquely associate each remote CID with a different device.

Table 74—CID definitions

CID	Description
0x0000	Null identifier
0x0001	Signalling channel
0x0002	Connectionless reception channel
0x0003–0x003F	Reserved
0x0040–0xFFFF	Dynamically allocated

10.2.2 Operation between devices

Figure 99 illustrates the use of CIDs in a communication between corresponding peer L2CAP entities in separate devices. The connection-oriented data channels represent a connection between two devices, where a CID identifies each endpoint of the channel. The connectionless channels restrict data flow to a single direction. These channels are used to support a channel "group" where the CID on the source represents one or more remote devices. There are also a number of CIDs reserved for special purposes. The signalling

channel is one example of a reserved channel. This channel is used to create and establish connection-oriented data channels and to negotiate changes in the characteristics of these channels. Support for a signalling channel within an L2CAP entity is mandatory. Another CID is reserved for all incoming connectionless data traffic. In the example below, a CID is used to represent a group consisting of device #3 and #4. Traffic sent from this channel ID is directed to the remote channel reserved for connectionless data traffic.

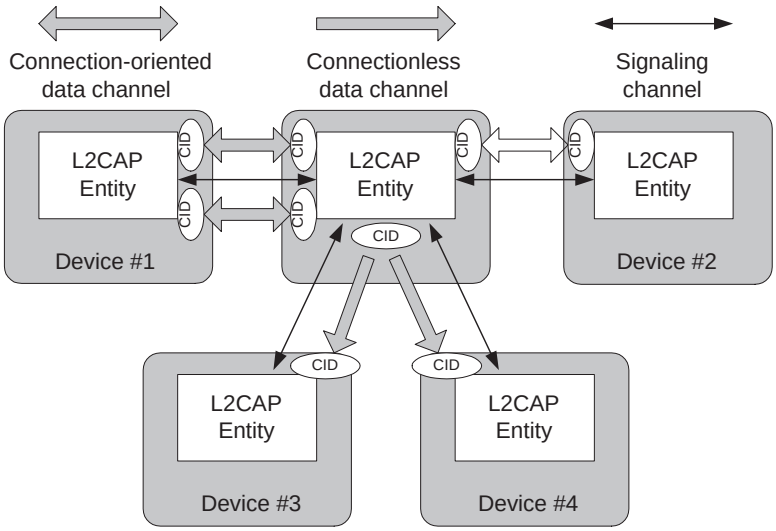


Figure 99—Channels between devices

Table 75 describes the various channels and their source and destination identifiers. An “allocated” channel is created to represent the local endpoint and should be in the range 0x0040 to 0xFFFF. 10.3 describes the state machine associated with each connection-orientedchannel. 10.4.1 describes the packet format associated with bi-directional channels and 10.4.2 describes the packet format associated with uni-directional channels.

Table 75—Types of channel identifiers

Channel type	Local CID	Remote CID
Connection-oriented	Dynamically allocated	Dynamically allocated
Connectionless data	Dynamically allocated	0x0002 (fixed)
Signalling	0x0001 (fixed)	0x0001 (fixed)

10.2.3 Operation between layers

L2CAP implementations should follow the general architecture described in Figure 100. L2CAP implementations must transfer data between higher layer protocols and the lower layer protocol. This document lists a number of services that should be exported by any L2CAP implementation. Each implementation must also support a set of signalling commands for use between L2CAP implementations. L2CAP implementations should also be prepared to accept certain types of events from lower layers and generate events to upper layers. How these events are passed between layers is an implementation-dependent process.

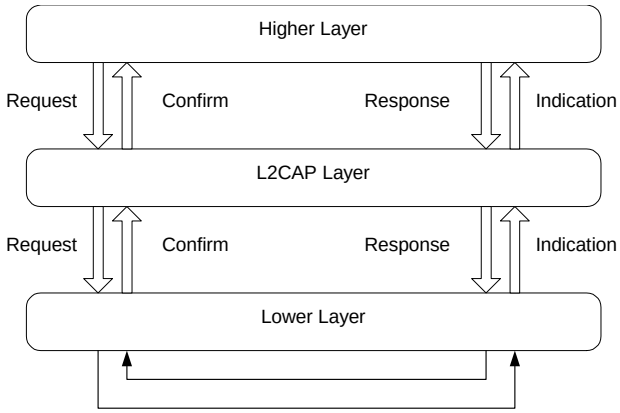


Figure 100—L2CAP architecture

10.2.4 Segmentation and reassembly

SAR operations are used to improve efficiency by supporting a maximum transmission unit (MTU) size larger than the largest baseband packet. This reduces overhead by spreading the network and transport packets used by higher layer protocols over several baseband packets. All L2CAP packets may be segmented for transfer over baseband packets. The protocol does not perform any segmentation and reassembly operations but the packet format supports adaptation to smaller physical frame sizes. An L2CAP implementation exposes the outgoing (i.e., the remote host's receiving) MTU and segments higher layer packets into "chunks" that can be passed to the Link Manager via the HCI, whenever one exists. On the receiving side, an L2CAP implementation receives "chunks" from the HCI and reassembles those chunks into L2CAP packets using information provided through the HCI and from the packet header (see Figure 101).

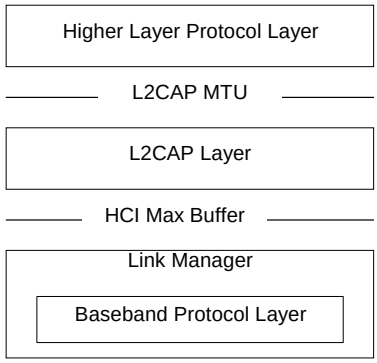


Figure 101—L2CAP SAR variables

Segmentation and reassembly is implemented using very little overhead in baseband packets. The two L_CH bits defined in the first byte of baseband payload (also called the frame header) are used to signal the start and continuation of L2CAP packets. L_CH shall be "10" for the first segment in an L2CAP packet and "01"' for a continuation segment. An example use of SAR is shown in Figure 102.

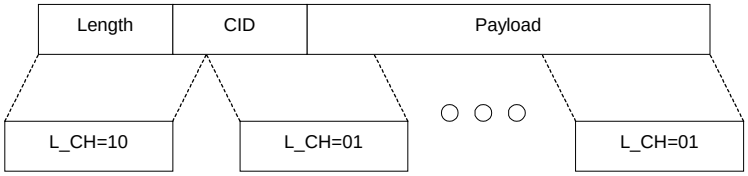


Figure 102—L2CAP segmentation

10.2.4.1 Segmentation procedures

The L2CAP maximum transmission unit (MTU) will be exported using an implementation-specific service interface. It is the responsibility of the higher layer protocol to limit the size of packets sent to the L2CAP layer below the MTU limit. An L2CAP implementation will segment the packet into protocol data units (PDUs) to send to the lower layer. If L2CAP runs directly over the Baseband Protocol, an implementation may segment the packet into Baseband packets for transmission over the air. If L2CAP runs above the host controller interface (typical scenario), an implementation may send block-sized chunks to the host controller where they will be converted into Baseband packets. All L2CAP segments associated with an L2CAP packet shall be passed through to the Baseband before any other L2CAP packet destined to the same unit may be sent.

10.2.4.2 Reassembly procedures

The Baseband Protocol delivers ACL packets in sequence and protects the integrity of the data using a 16-bit CRC. The Baseband also supports reliable connections using an automatic repeat request (ARQ) mechanism. As the Baseband controller receives ACL packets, it either signals the L2CAP layer on the arrival of each Baseband packets, or accumulates a number of packets before the receive buffer fills up or a timer expires before signalling the L2CAP layer.

L2CAP implementations shall use the length field in the header of L2CAP packets (see 10.4), as a consistency check and discard any L2CAP packets that fail to match the length field. If channel reliability is not needed, packets with improper lengths may be silently discarded. For reliable channels, L2CAP implementations shall indicate to the upper layer that the channel has become unreliable. Reliable channels are defined by having an infinite flush timeout value as specified in 10.6.2.

Figure 103 illustrates the use of segmentation and reassembly operations to transmit a single higher layer PDU.[17] Note that while there is a one-to-one mapping between a high layer PDU and an L2CAP packet, the segment size used by the segmentation and reassembly routines is left to the implementation and may differ from the sender to the receiver.

[17]For simplicity, the stripping of any additional HCI and USB specific information fields prior to the creation of the baseband packets (Air_1, Air_2, etc.) is not shown in Figure 103.

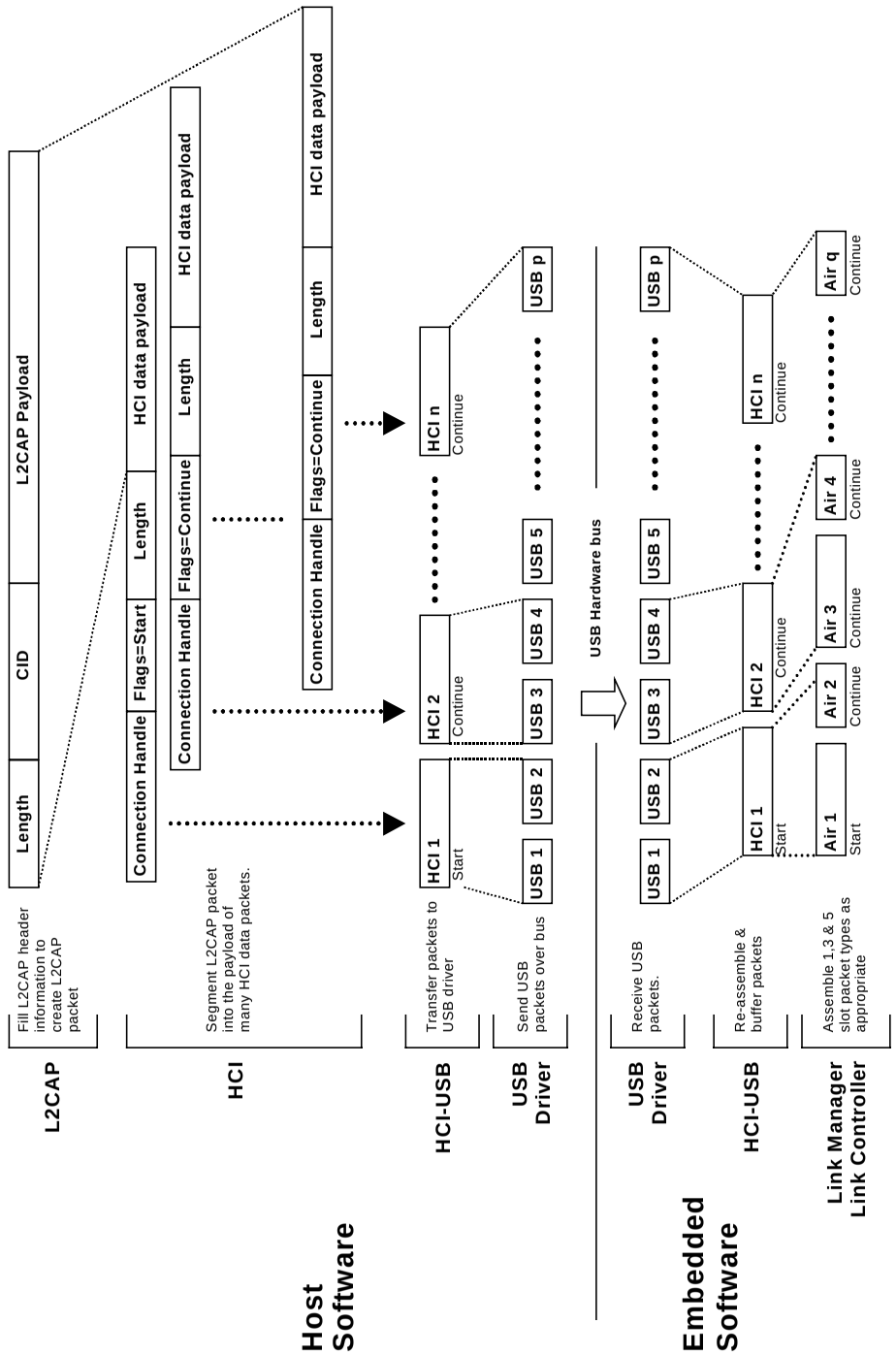


Figure 103—Segmentation and reassembly services in a unit with an HCI

10.3 State machine

This subclause describes the L2CAP connection-oriented channel state machine. The subclause defines the states, the events causing state transitions, and the actions to be performed in response to events. This state machine is only pertinent to bi-directional CIDs and is not representative of the signalling channel or the uni-directional channel.

Figure 104 illustrates the events and actions performed by an implementation of the L2CAP layer. Client and Server simply represent the initiator of the request and the acceptor of the request respectively. An application-level Client would both initiate and accept requests. The naming convention is as follows. The interface between two layers (vertical interface) uses the prefix of the lower layer offering the service to the higher layer, e.g., L2CA. The interface between two entities of the same layer (horizontal interface) uses the prefix of the protocol (adding a P to the layer identification), e.g., L2CAP. Events coming from above are called Requests (Req) and the corresponding replies are called Confirms (Cfm). Events coming from below are called Indications (Ind) and the corresponding replies are called Responses (Rsp). Responses requiring further processing are called Pending (Pnd). The notation for Confirms and Responses assumes positive replies. Negative replies are denoted by a “Neg” suffix such as L2CAP_ConnectCfmNeg.

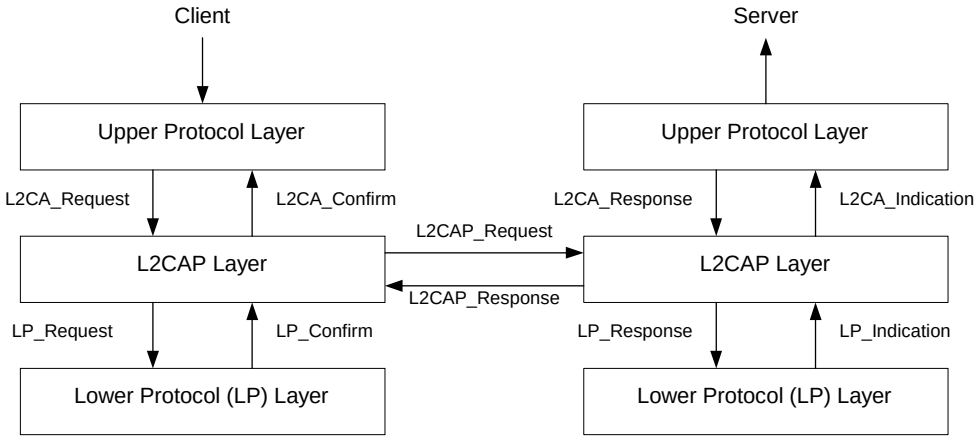


Figure 104—L2CAP layer interactions

While Requests for an action always result in a corresponding Confirmation (for the successful or unsuccessful satisfaction of the action), Indications do not always result into corresponding Responses. The latter is especially true, if the Indications are informative about locally triggered events, e.g., seeing the *LP_QoSViolationInd* in 10.3.1.1, or *L2CA_TimeOutInd* in 10.3.2.4.

Figure 105 uses a message sequence chart (MSC) to illustrate the normal sequence of events. The two outer vertical lines represent the L2CA interface on the initiator (the device issuing a request) and the acceptor (the device responding to the initiator’s request). Request commands at the L2CA interface result in Requests defined by the protocol. When the protocol communicates the request to the acceptor, the remote L2CA entity presents the upper protocol with an Indication. When the acceptor’s upper protocol responds, the response is packaged by the protocol and communicated back to the initiator. The result is passed back to the initiator’s upper protocol using a Confirm message.

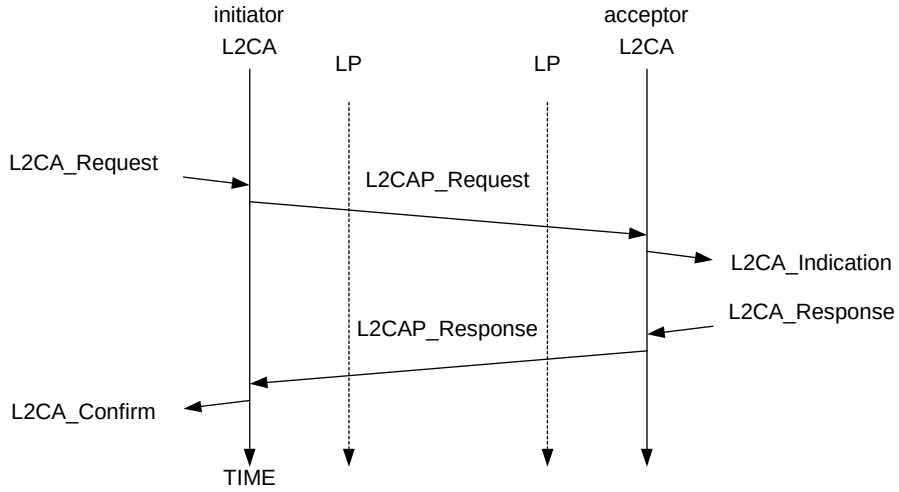


Figure 105—MSC of layer interactions

10.3.1 Events

Events are all incoming messages to the L2CAP layer along with timeouts. Events are partitioned into five categories: Indications and Confirms from lower layers, Requests and Responses from higher layers, data from peers, signal Requests and Responses from peers, and events caused by timer expirations.

10.3.1.1 Lower-layer protocol (LP) to L2CAP events

— *LP_ConnectCfm*

Confirms the request (see *LP_ConnectReq* in 10.3.2.1) to establish a lower layer (Baseband) connection. This includes passing the authentication challenge if authentication is required to establish the physical link.

— *LP_ConnectCfmNeg*

Confirms the failure of the request (see *LP_ConnectReq* in 10.3.2.1) to establish a lower layer (Baseband) connection failed. This could be because the device could not be contacted, refused the request, or the LMP authentication challenge failed.

— *LP_ConnectInd*

Indicates the lower protocol has successfully established connection. In the case of the Baseband, this will be an ACL link. An L2CAP entity may use this information to keep track of what physical links exist.

— *LP_DisconnectInd*

Indicates the lower protocol (Baseband) has been shut down by LMP commands or a timeout event.

— *LP_QoSCfm*

Confirms the request (see *LP_QoSReq* in 10.3.2.1) for a given QoS.

— *LP_QoSCfmNeg*

Confirms the failure of the request (see *LP_QoSReq* in 10.3.2.1) for a given QoS.

— *LP_QoSViolationInd*

Indicates the lower protocol has detected a violation of the QoS agreement specified in the previous *LP_QoSReq* (see 10.3.2.1).

10.3.1.2 L2CAP to L2CAP signalling events

L2CAP to L2CAP signalling events are generated by each L2CAP entity following the exchange of the corresponding L2CAP signalling PDUs (see 10.5). L2CAP signalling PDUs, like any other L2CAP PDUs, are received from a lower layer via a lower protocol indication event. For simplicity of the presentation, we avoid a detailed description of this process, and we assume that signalling events are exchanged directly between the L2CAP peer entities as shown in Figure 104.

— *L2CAP_ConnectReq*

A Connection Request packet has been received.

— *L2CAP_ConnectRsp*

A Connection Response packet has been received with a positive result indicating that the connection has been established.

— *L2CAP_ConnectRspPnd*

A Connection Response packet has been received indicating the remote endpoint has received the request and is processing it.

— *L2CAP_ConnectRspNeg*

A Connection Response packet has been received, indicating that the connection could not be established.

— *L2CAP_ConfigReq*

A Configuration Request packet has been received indicating the remote endpoint wishes to engage in negotiations concerning channel parameters.

— *L2CAP_ConfigRsp*

A Configuration Response packet has been received indicating the remote endpoint agrees with all the parameters being negotiated.

— *L2CAP_ConfigRspNeg*

A Configuration Response packet has been received indicating the remote endpoint does not agree to the parameters received in the response packet.

— *L2CAP_DisconnectReq*

A Disconnection Request packet has been received and the channel shall initiate the disconnection process. Following the completion of an L2CAP channel disconnection process, an L2CAP entity should return the corresponding local CID to the pool of “unassigned” CIDs.

— *L2CAP_DisconnectRsp*

A Disconnection Response packet has been received. Following the receipt of this signal, the receiving L2CAP entity may return the corresponding local CID to the pool of unassigned CIDs. There is no corresponding negative response because the Disconnect Request must succeed.

10.3.1.3 L2CAP to L2CAP data events

— *L2CAP_Data*

A Data packet has been received.

10.3.1.4 Upper-layer to L2CAP events

— *L2CA_ConnectReq*

Request from upper layer for the creation of a channel to a remote device.

— *L2CA_ConnectRsp*

Response from upper layer to the indication of a connection request from a remote device (see L2CA_ConnectInd in 10.3.2.4).

— *L2CA_ConnectRspNeg*

Negative response (rejection) from upper layer to the indication of a connection request from a remote device (see *L2CA_ConnectInd* in 10.3.2.4).

— *L2CA_ConfigReq*

Request from upper layer to (re)configure the channel.

— *L2CA_ConfigRsp*

Response from upper layer to the indication of a (re) configuration request (see *L2CA_ConfigInd* in 10.3.2.4).

— *L2CA_ConfigRspNeg*

A negative response from upper layer to the indication of a (re) configuration request (see *L2CA_ConfigInd* in 10.3.2.4).

— *L2CA_DisconnectReq*

Request from upper layer for the immediate disconnection of a channel.

— *L2CA_DisconnectRsp*

Response from upper layer to the indication of a disconnection request (see *L2CA_DisconnectInd* in 10.3.2.4). There is no corresponding negative response, the disconnect indication must always be accepted.

— *L2CA_DataRead*

Request from upper layer for the transfer of received data from L2CAP entity to upper layer.

— *L2CA_DataWrite*

Request from upper layer for the transfer of data from the upper layer to L2CAP entity for transmission over an open channel.

10.3.1.5 Timer events

— *RTX*

The Response Timeout eXpired (RTX) timer is used to terminate the channel when the remote endpoint is unresponsive to signalling requests. This timer is started when a signalling request (see 10.5) is sent to the remote device. This timer is disabled when the response is received. If the initial timer expires, a duplicate Request message may be sent or the channel identified in the request may be disconnected. If a duplicate Request message is sent, the RTX timeout value shall be reset to a new value at least double the previous value.

Implementations have the responsibility to decide on the maximum number of Request retransmissions performed at the L2CAP level before terminating the channel identified by the Requests. The one exception is the signalling CID that should never be terminated. The decision should be based on the flush timeout of the signalling link. The longer the flush timeout, the more retransmissions may be performed at the physical layer and the reliability of the channel improves, requiring fewer retransmissions at the L2CAP level. For example, if the flush timeout is infinite, no retransmissions should be performed at the L2CAP level. When terminating the channel, it is not

necessary to send a L2CAP DisconnectReq and enter disconnection state.Channels should be transitioned directly to the Closed state.

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

— *ERTX*

The Extended Response Timeout eXpired (ERTX) timer is used in place of the RTX timer when it is suspected the remote endpoint is performing additional processing of a request signal. This timer is started when the remote endpoint responds that a request is pending, e.g., when an *L2CAP_ConnectRspPnd* event is received. This timer is disabled when the formal response is received or the physical link is lost. If the initial timer expires, a duplicate Request may be sent or the channel may be disconnected. If a duplicate Request is sent, the particular ERTX timer disappears, replaced by a new RTX timer and the whole timing procedure restarts as described previously for the RTX timer.

The value of this timer is implementation-dependent but the minimum initial value is 60 s and the maximum initial value is 300 s. Similar to RTX, there shall be at least one ERTX timer for each outstanding request that received a Pending response. There should be at most one (RTX or ERTX) associated with each outstanding request. The maximum elapsed time between the initial start of this timer and the initiation of channel termination (if no response is received) is 300 s. When terminating the channel, it is not necessary to send a L2CAP DisconnectReq and enter disconnection state. Channels should be transitioned directly to the Closed state.

10.3.2 Actions

Actions are partitioned into five categories: Confirms and Indications to higher layers, Request and Responses to lower layers, Requests and Responses to peers, data transmission to peers, and setting timers.

10.3.2.1 L2CAP to lower layer actions

— *LP_ConnectReq*

L2CAP requests the lower protocol to create a connection. If a physical link to the remote device does not exist, this message shall be sent to the lower protocol to establish the physical connection. Since no more than a single ACL link between two devices is assumed, see 10.1.2, additional L2CAP channels between these two devices shall share the same baseband ACL link.

Following the processing of the request, the lower layer returns with an *LP_ConnectCfm* or an *LP_ConnectCfmNeg* to indicate whether the request has been satisfied or not, respectively.

— *LP_QoSReq*

L2CAP requests the lower protocol to accommodate a particular QoS parameter set. Following the processing of the request, the lower layer returns with an *LP_QoSCfm* or an *LP_QoSCfmNeg* to indicate whether the request has been satisfied or not, respectively.

— *LP_ConnectRsp*

A positive response accepting the previous connection indication request (see *LP_ConnectInd* in 10.3.1.1).

— *LP_ConnectRspNeg*

A negative response denying the previous connection indication request (see *LP_ConnectInd* in 10.3.1.1).

10.3.2.2 L2CAP to L2CAP signalling actions

This section contains the same names identified in 10.3.1.2 except the actions refer to the transmission, rather than reception, of these messages.

10.3.2.3 L2CAP to L2CAP data actions

This section is the counterpart of 10.3.1.3. Data transmission is the action performed here.

10.3.2.4 L2CAP to upper layer actions

— *L2CA_ConnectInd*

Indicates a Connection Request has been received from a remote device (see *L2CA_ConnectReq* in 10.3.1.4).

— *L2CA_ConnectCfm*

Confirms that a Connection Request has been accepted (see (*L2CAP_ConnectReq* in 10.3.1.4) following the receipt of a Connection message from the remote device.

— *L2CA_ConnectCfmNeg*

Negative confirmation (failure) of a Connection Request (see *L2CA_ConnectReq* in 10.3.1.4). An RTX timer expiration (see 10.3.1.5 and *L2CA_TimeOutInd* below) for an outstanding Connect Request can substitute for a negative Connect Response and result in this action.

— *L2CA_ConnectPnd*

Confirms that a Connection Response (pending) has been received from the remote device.

— *L2CA_ConfigInd*

Indicates a Configuration Request has been received from a remote device.

— *L2CA_ConfigCfm*

Confirms that a Configuration Request has been accepted (see *L2CA_ConfigReq* in 10.3.1.4) following the receipt of a Configuration Response from the remote device.

— *L2CA_ConfigCfmNeg*

Negative confirmation (failure) of a Configuration Request (see *L2CA_ConfigReq* in 10.3.1.4). An RTX timer expiration (see 10.3.1.5 and *L2CA_TimeOutInd* below) for an outstanding Connect Request can substitute for a negative Connect Response and result in this action.

— *L2CA_DisconnectInd*

Indicates a Disconnection Request has been received from a remote device or the remote device has been disconnected because it has failed to respond to a signalling request. See 10.3.1.5

— *L2CA_DisconnectCfm*

Confirms that a Disconnect Request has been processed by the remote device (see *L2CA_DisconnectReq* in 10.3.1.4) following the receipt of a Disconnection Response from the remote device. An RTX timer expiration (see 10.3.1.5 and *L2CA_TimeOutInd* below) for an outstanding Disconnect Request can substitute for a Disconnect Response and result in this action. Upon receiving this event the upper layer knows the L2CAP channel has been terminated. There is no corresponding negative confirm.

— *L2CA_TimeOutInd*

Indicates that a RTX or ERTX timer has expired. This indication will occur an implementation-dependant number of times before the L2CAP implementation will give up and send a L2CA_DisconnectInd.

— *L2CA_QoSViolationInd*

Indicates that the quality of service agreement has been violated.

10.3.3 Channel operational states

— *CLOSED*

In this state, there is no channel associated with this CID. This is the only state when a link level connection (Baseband*)* may not exist. Link disconnection forces all other states into the CLOSED state.

— *W4_L2CAP_CONNECT_RSP*

In this state, the CID represents a local end-point and an L2CAP_ConnectReq message has been sent referencing this endpoint and it is now waiting for the corresponding L2CAP_ConnectRsp message.

— *W4_L2CA_CONNECT_RSP*

In this state, the remote end-point exists and an *L2CAP_ConnectReq* has been received by the local L2CAP entity. An L2CA_ConnectInd has been sent to the upper layer and the part of the local L2CAP entity processing the received *L2CAP_ConnectReq* waits for the corresponding response. The response may require a security check to be performed.

— *CONFIG*

In this state, the connection has been established but both sides are still negotiating the channel parameters. The Configuration state may also be entered when the channel parameters are being renegotiated. Prior to entering the CONFIG state, all outgoing data traffic should be suspended since the traffic parameters of the data traffic are to be renegotiated. Incoming data traffic shall be accepted until the remote channel endpoint has entered the CONFIG state.

In the CONFIG state, both sides shall issue L2CAP_ConfigReq messages – if only defaults are being used, a null message should be sent (see 10.5.4). If a large amount of parameters need to be negotiated, multiple messages may be sent to avoid any MTU limitations and negotiate incrementally. See 10.6 for more details.

Moving from the CONFIG state to the OPEN state requires both sides to be ready. An L2CAP entity is ready when it has received a positive response to its final request and it has positively responded to the final request from the remote device.

— *OPEN*

In this state, the connection has been established and configured, and data flow may proceed.

— *W4_L2CAP_DISCONNECT_RSP*

In this state, the connection is shutting down and an L2CAP_DisconnectReq message has been sent. This state is now waiting for the corresponding response.

— *W4_L2CA_DISCONNECT_RSP*

In this state, the connection on the remote endpoint is shutting down and an L2CAP_DisconnectReq message has been received. An L2CA_DisconnectInd has been sent to the upper layer to notify the owner of the CID that the remote endpoint is being closed. This state is now waiting for the corresponding response from the upper layer before responding to the remote endpoint.

10.3.4 Mapping events to actions

Table 76 defines the actions taken in response to events that occur in a particular state. Events that are not listed in the table, nor have actions marked N/C (for no change), are assumed to be errors and silently discarded.

Data input and output events are only defined for the Open and Configuration states. Data may not be received during the initial Configuration state, but may be received when the Configuration state is re-entered due to a reconfiguration process. Data received during any other state should be silently discarded.

Table 76—L2CAP Channel State Machine

Event	Current state	Action	New state
LP_ConnectCfm	CLOSED	Flag physical link as up and initiate the L2CAP connection.	CLOSED
LP_ConnectCfmNeg	CLOSED	Flag physical link as down and fail any outstanding service connection requests by sending an L2CA_ConnectCfmNeg message to the upper layer.	CLOSED
LP_ConnectInd	CLOSED	Flag link as up.	CLOSED
LP_DisconnectInd	CLOSED	Flag link as down.	CLOSED
LP_DisconnectInd	Any except CLOSED	Send upper layer L2CA_DisconnectInd message.	CLOSED
LP_QoSViolationInd	Any but OPEN	Discard	N/C
LP_QoSViolationInd	OPEN	Send upper layer L2CA_QoSViolationInd message. If service level is guaranteed, terminate the channel.	OPEN or W4_L2CA_ DISCONNECT_ RSP
L2CAP_ConnectReq	CLOSED. (CID dynamically allocated from free pool.)	Send upper layer L2CA_ConnectInd. Optionally: Send peer L2CAP_ConnectRspPnd	W4_L2CA_ CONNECT_ RSP
L2CAP_ConnectRsp	W4_L2CAP_CONNECT_RSP	Send upper layer L2CA_ConnectCfm message. Disable RTX timer.	CONFIG
L2CAP_ConnectRspPnd	W4_L2CAP_CONNECT_RSP	Send upper layer L2CA_ConnectPnd message. Disable RTX timer and start ERTX timer.	N/C
L2CAP_ConnectRspNeg	W4_L2CAP_CONNECT_RSP	Send upper layer L2CA_ConnectCfmNeg message. Return CID to free pool. Disable RTX/ERTX timers.	CLOSED
L2CAP_ConfigReq	CLOSED	Send peer L2CAP_Reject message.	N/C

Table 76—L2CAP Channel State Machine *(continued)*

Event	Current state	Action	New state
L2CAP_ConfigReq	CONFIG	Send upper layer L2CA_ConfigInd message.	N/C
L2CAP_ConfigReq	OPEN	Suspend data transmission at a convenient point. Send upper layer L2CA_ConfigInd message.	CONFIG
L2CAP_ConfigRsp	CONFIG	Send upper layer L2CA_ConfigCfm message. Disable RTX timer. If an L2CAP_ConfigReq message has been received and positively responded to, then enter OPEN state, otherwise remain in CONFIG state.	N/C or OPEN
L2CAP_ConfigRsp Neg	CONFIG	Send upper layer L2CA_ConfigCfmNeg message. Disable RTX timer.	N/C
L2CAP_Disconnect Req	CLOSED	Send peer L2CAP_DisconnectRsp message.	N/C
L2CAP_Disconnect Req	Any except CLOSED	Send upper layer L2CA_ DisconnectInd message.	W4_L2CA_ DISCONNECT_ RSP
L2CAP_Disconnect Rsp	W4_L2CAP_ DISCONNECT_ RSP	Send upper layer L2CA_DisconnectCfm message. Disable RTX timer.	CLOSED
L2CAP_Data	OPEN or CONFIG	If complete L2CAP packet received, send upper layer L2CA_Read confirm.	N/C
L2CA_ConnectReq	CLOSED (CID dynamically allocated from free pool)	Send peer L2CAP_ConnectReq message. Start RTX timer.	W4_L2CAP_ CONNECT_RSP
L2CA_ConnectRsp	W4_L2CA_CONNEC T_RSP	Send peer L2CAP_ ConnectRsp message.	CONFIG
L2CA_ConnectRsp Neg	W4_L2CA_CONNEC T_RSP	Send peer L2CAP_ConnectRspNeg message. Return CID to free pool.	CLOSED
L2CA_ConfigReq	CLOSED	Send upper layer L2CA_ ConfigCfmNeg message.	N/C
L2CA_ConfigReq	CONFIG	Send peer L2CAP_ConfigReq message. Start RTX timer.	N/C
L2CA_ConfigReq	OPEN	Suspend data transmission at a convenient point. Send peer L2CAP_ConfigReq message. Start RTX timer.	CONFIG

Table 76—L2CAP Channel State Machine *(continued)*

Event	Current state	Action	New state
L2CA_ConfigRsp	CONFIG	Send peer L2CAP_ConfigRsp message. If all outstanding L2CAP_ConfigReq messages have received positive responses then move in OPEN state. Otherwise, remain in CONFIG state.	N/C or OPEN
L2CA_ConfigRspNeg	CONFIG	Send peer L2CAP_ ConfigRspNeg message.	N/C
L2CA_Disconnect Req	OPEN or CONFIG	Send peerF L2CAP_DisconnectReq message. Start RTX timer.	W4_L2CAP_ DISCONNECT_ RSP
L2CA_DisconnectRsp	W4_L2CA_ DISCONNECT_ RSP	Send peer L2CAP_DisconnectRsp message. Return CID to free pool.	CLOSED
L2CA_DataRead	OPEN	If payload complete, transfer payload to InBuffer.	OPEN
L2CA_DataWrite	OPEN	Send peer L2CAP_Data message.	OPEN
Timer_RTX	Any	Send upper layer L2CA_TimeOutInd message. If final expiration, return CID to free pool and go to CLOSE state, else re-send Request.	N/C or CLOSED
Timer_ERTX	Any	Send upper layer L2CA_TimeOutInd message. If final expiration, return CID to free pool and go to CLOSE state, else re-send Request.	N/C or CLOSED

Figure 106 illustrates a simplified state machine and typical transition path taken by an initiator and acceptor. The state machine shows what events cause state transitions and what actions are also taken while the transitions occur. Not all the events listed in Table 76 are included in the simplified State Machine to avoid cluttering the figure. Also, W2_L2CA_DISCON_RSP stands for W2_L2CA_DISCONNECT_RSP; W2_L2CAP_DISCON_RSP stands for W2_L2CAP_DISCONNECT_RSP.

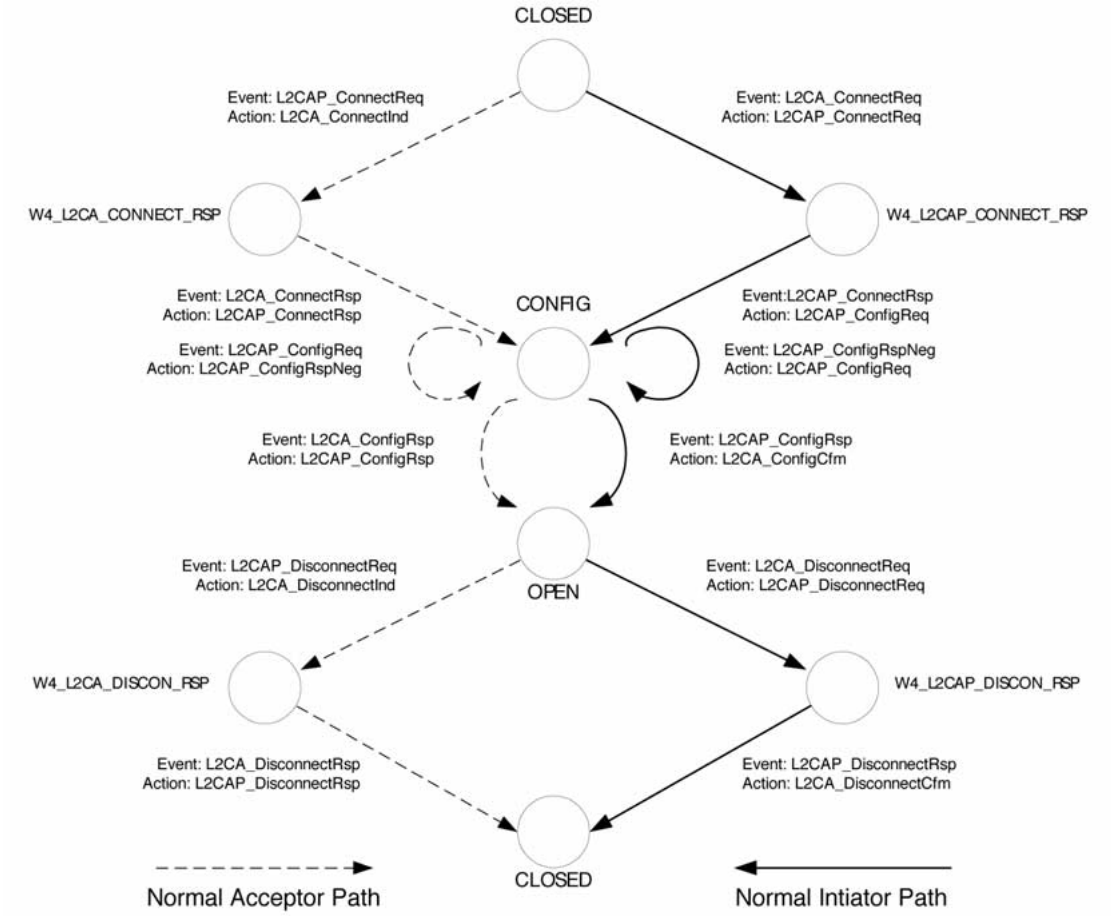


Figure 106—State machine example

Figure 107 presents another illustration of the events and actions based around the messages sequences being communicated between two devices. In this example, the initiator is creating the first L2CAP channel between two devices. Both sides start in the CLOSED state. After receiving the request from the upper layer, the entity requests the lower layer to establish a physical link. If no physical link exists, LMP commands are used to create the physical link between the devices. Once the physical link is established, L2CAP signals may be sent over it.

Figure 107 is an example and not all setup sequences will be identical to the one illustrated in the figure.

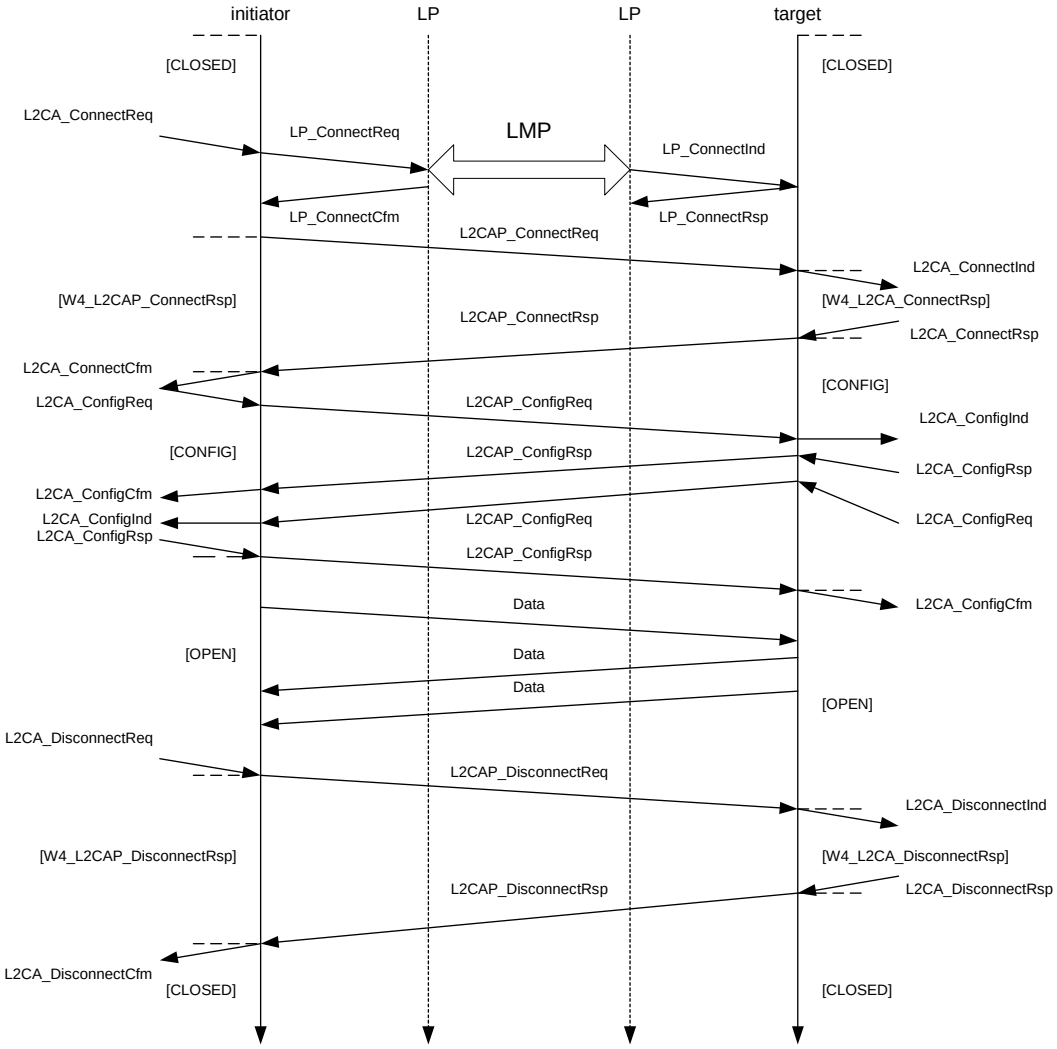


Figure 107—Message sequence chart of basic operation

10.4 Data packet format

L2CAP is packet-based but follows a communication model based on *channels*. A channel represents a data flow between L2CAP entities in remote devices. Channels may be connection-oriented or connectionless. All packet fields use Little Endian byte order.

10.4.1 Connection-oriented channel

Figure 108 illustrates the format of the L2CAP packet (also referred to as the L2CAP PDU) within a connection-oriented channel.

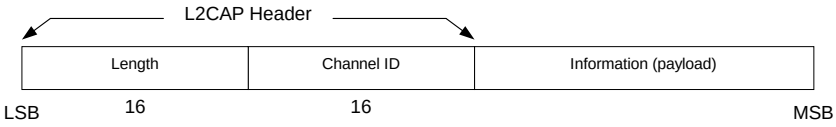


Figure 108—L2CAP packet (field sizes in bits)

The fields shown in Figure 108 are as follows:

— *Length: 2 octets (16 bits)*

Length indicates the size of information payload in bytes, excluding the length of the L2CAP header. The length of an information payload can be up to 65535 bytes. The Length field serves as a simple integrity check of the reassembled L2CAP packet on the receiving end.

— *Channel ID: 2 octets*

The channel ID identifies the destination channel endpoint of the packet. The scope of the channel ID is relative to the device the packet is being sent to.

— *Information: 0 to 65535 octets*

This contains the payload received from the upper layer protocol (outgoing packet), or delivered to the upper layer protocol (incoming packet). The minimum supported MTU for connection-oriented packets (MTU_{cno}) is negotiated during channel configuration (see 10.6.1). The minimum supported MTU for the signalling packet (MTU_{sig}) is 48 bytes (see 10.5).

10.4.2 Connectionless data channel

In addition to connection-oriented channels, L2CAP also exports the concept of a group-oriented channel. Data sent to the “group” channel is sent to all members of the group in a best-effort manner. Groups have no QoS associated with them. Group channels are unreliable; L2CAP makes no guarantee that data sent to the group successfully reaches all members of the group. If reliable group transmission is required, it must be implemented at a higher layer.

Transmissions to a group must be nonexclusively sent to all members of that group. The local device cannot be a member of the group, and higher layer protocols are expected to loopback any data traffic being sent to the local device. Nonexclusive implies non-group members may receive group transmissions and higher level (or link level) encryption can be used to support private communication. Figure 109 illustrates the format of the L2CAP PDU within a connectionless channel.

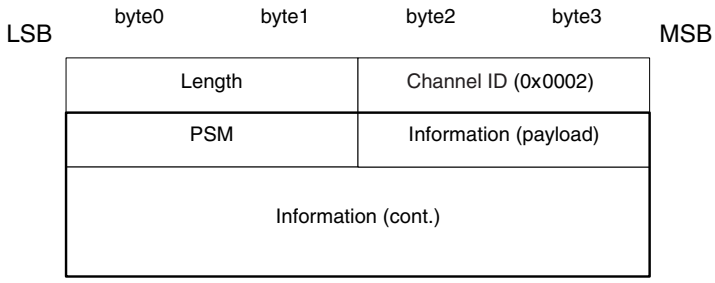


Figure 109—Connectionless packet

The fields shown in Figure 109 are as follows:

— *Length: 2 octets*

Length indicates the size of information payload plus the PSM field in bytes, excluding the length of the L2CAP header.

— *Channel ID: 2 octets*

Channel ID (0x0002) reserved for connectionless traffic.

— *Protocol/Service Multiplexer (PSM): 2 octets (minimum)*

The PSM field is based on the ISO 3309 extension mechanism for address fields. All content of the PSM field, referred to as the PSM value, shall be ODD, that is, the least significant bit of the least significant octet must be “1”. Also, all PSM values shall be assigned such that the least significant bit of the most significant octet equals “0”. This allows the PSM field to be extended beyond 16 bits. The PSM value definitions are specific to L2CAP and assigned by the Bluetooth SIG. For more information on the PSM field see 10.5.2.

— *Information: 0 to 65533 octets*

The payload information to be distributed to all members of the group. Implementations shall support a minimum connectionless MTU (MTU_{cnl}) of 670 octets, unless explicitly agreed upon otherwise, e.g., for single operation devices that are built to comply to a specific Bluetooth profile that dictates the use of a specific MTU for connectionless traffic that is less than MTU_{cnl}.

The L2CAP group service interface provides basic group management mechanisms including creating a group, adding members to a group, and removing members from a group. There are no predefined groups such as “all radios in range.”

10.5 Signalling

This subclause describes the signalling commands passed between two L2CAP entities on remote devices. All signalling commands are sent to CID 0x0001. The L2CAP implementation must be able to determine the Bluetooth address (BD_ADDR) of the device that sent the commands. Figure 110 illustrates the general format of all L2CAP packets containing signalling commands. Multiple commands may be sent in a single (L2CAP) packet and packets are sent to CID 0x0001. Commands take the form of Requests and Responses. All L2CAP implementations shall support the reception of signalling packets whose MTU (MTU_{sig}) does not exceed 48 bytes. L2CAP implementations should not use signalling packets beyond this size without first testing whether the implementation can support larger signalling packets. Implementations must be able to handle the reception of multiple commands in an L2CAP packet as long as the MTU is not exceeded.

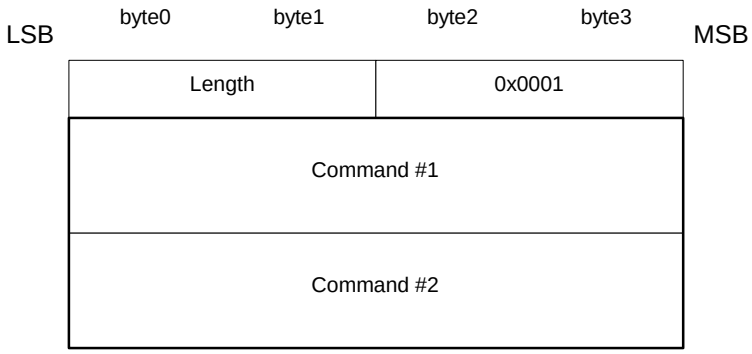


Figure 110—Signalling command packet format

Figure 111 displays the general format of all signalling commands.

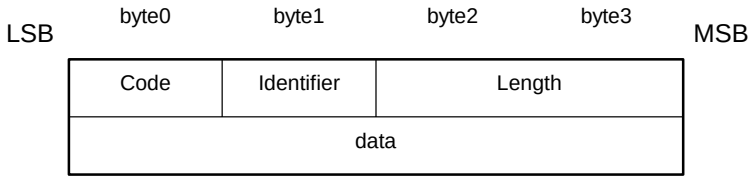


Figure 111—Command format

The fields shown in Figure 111 are as follows:

— *Code: 1 octet*

The Code field is one octet long and identifies the type of command. When a packet is received with an unknown Code field, a Command Reject packet (defined in 10.5.1) is sent in response. Up-to-date values of assigned Codes are specified in the latest Bluetooth Assigned Numbers document 2.4.3. Table 77 lists the codes defined by this document. All codes are specified with the most significant bit in the left-most position.

— *Identifier: 1 octet*

The Identifier field is one octet long and helps matching a request with the reply. The requesting device sets this field and the responding device uses the same value in its response. A different Identifier shall be used for each original command. Identifiers should not be recycled until a period of 360 s has elapsed from the initial transmission of the command using the identifier. On the expiration of a RTX or ERTX timer, the same identifier should be used if a duplicate Request is resent as stated in 10.3.1.5. A device receiving a duplicate request should reply with a duplicate response. A command response with an invalid identifier is silently discarded. Signalling identifier 0x0000 is defined to be an illegal identifier and shall never be used in any command.

— *Length: 2 octets*

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

— *Data: 0 or more octets*

The Data field is variable in length and discovered using the Length field. The Code field determines the format of the Data field.

Table 77—Signalling command codes

Code	Description
0x00	RESERVED
0x01	Command reject
0x02	Connection request
0x03	Connection response
0x04	Configure request
0x05	Configure response
0x06	Disconnection request
0x07	Disconnection response
0x08	Echo request
0x09	Echo response
0x0A	Information request
0x0B	Information response

10.5.1 Command reject (code 0x01)

A Command Reject packet is sent in response to a command packet with an unknown command code or when sending the corresponding Response is inappropriate. Figure 112 displays the format of the packet. The Identifier should match the Identifier of the packet containing the unidentified code field. Implementations shall always send these packets in response to unidentified signalling packets. Command Reject packets should not be sent in response to an identified Response packet.

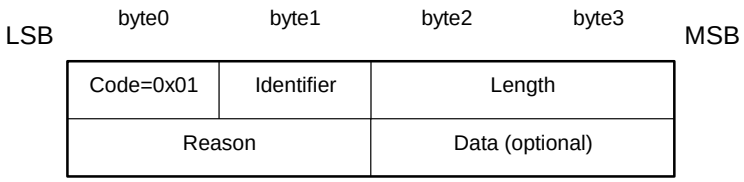


Figure 112—Command reject packet

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

— *Length = 0x0002 or more octets*

— *Reason: 2 octets*

The Reason field describes why the Request packet was rejected (see Table 78).

Table 78—Reason code descriptions

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

— *Data: 0 or more octets*

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

If a command refers to an invalid channel then the Reason code 0x0002 will be returned. Typically a channel is invalid because it does not exist. A 4-octet data field on the command reject will contain the local (first) and remote (second) channel endpoints (relative to the sender of the Command Reject) of the disputed channel. The latter endpoints are obtained from the corresponding rejected command. If the rejected command contains only one of the channel endpoints, the other one is replaced by the null CID 0x0000 (see Table 79).

Table 79—Reason data values

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

10.5.2 Connection request (code 0x02)

Connection request packets are sent to create a channel between two devices. The channel connection shall be established before configuration may begin. Figure 113 illustrates a minimum size Connection Request packet.

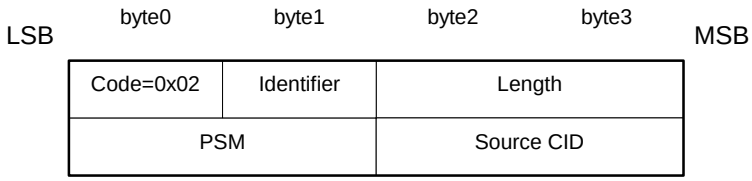


Figure 113—Connection request packet

— *Length = 0x0004 or more octets*

— *Protocol/Service Multiplexor (PSM): 2 octets (minimum)*

The PSM field is two octets (minimum) in length. The structure of the PSM field is based on the ISO 3309 extension mechanism for address fields. All PSM values shall be ODD, that is, the least significant bit of the least significant octet must be “1”. Also, all PSM values shall be assigned such that the least significant bit of the most significant octet equals “0”. This allows the PSM field to be extended beyond 16 bits. PSM values are separated into two ranges. Values in the first range are assigned by the Bluetooth SIG and indicate protocols. The second range of values are dynamically allocated and used in conjunction with the SDP; see 2.4.1, Part E, of the Bluetooth specification. The dynamically assigned values may be used to support multiple implementations of a particular protocol (see Table 80).

Table 80—Defined PSM values

PSM value	Description
0x0001	Service Discovery Protocol
0x0003	RFCOMM
0x0005	Telephony Control Protocol
<0x1000	RESERVED
[0x1001-0xFFFF]	DYNAMICALLY ASSIGNED

— *Source CID (SCID): 2 octets*

The source local CID is two octets in length and represents a channel end-point on the device sending the request. Once the channel has been configured, data packets flowing tothe sender of the request shall be sent to this CID. In this section, the Source CID represents the channel endpoint on the device sending the request and receiving the response.

10.5.3 Connection response (code 0x03)

When a unit receives a Connection Request packet, it shall send a Connection Response packet. The format of the connection response packet is shown in Figure 114.

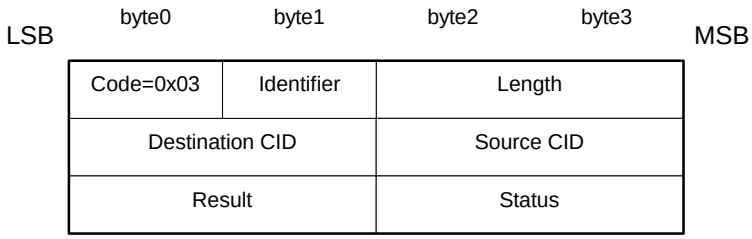


Figure 114—Connection response packet

— *Length = 0x0008 octets*

— *Destination Channel Identifier (DCID): 2 octets*

The field contains the channel end-point on the device sending this Response packet. In this section, the Destination CID represents the channel endpoint on the device receiving the request and sending the response.

— *Source Channel Identifier (SCID): 2 octets*

The field contains the channel end-point on the device receiving this Response packet.

— *Result: 2 octets*

The result field indicates the outcome of the connection request. The result value of 0x0000 indicates success while a nonzero value indicates the connection request failed or is pending. A logical channel is established on the receipt of a successful result. Table 81 defines values for this field. If the result field is not zero. The DCID and SCID fields should be ignored when the result field indicates the connection was refused.

Table 81—Result values

Value	Description
0x0000	Connection successful
0x0001	Connection pending
0x0002	Connection refused – PSM not supported
0x0003	Connection refused – security block
0x0004	Connection refused – no resources available
Other	Reserved

— *Status: 2 octets*

Only defined for Result = Pending. Indicates the status of the connection (see Table 82).

Table 82—Status values

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

10.5.4 Configuration request (code 0x04)

Configuration Request packets are sent to establish an initial logical link transmission contract between two L2CAP entities and also to renegotiate this contract whenever appropriate. During a renegotiation session, all data traffic on the channel should be suspended pending the outcome of the negotiation. Each configuration parameter in a Configuration Request is related exclusively either with the outgoing or the incoming data traffic but not both of them. In 10.6, the various configuration parameters and their relation to the outgoing or incoming data traffic are presented. If an L2CAP entity receives a Configuration Request while it is waiting for a response it shall not block sending the Configuration Response, otherwise the configuration process may deadlock.

If no parameters need to be negotiated, no options need to be inserted and the C-bit should be cleared. L2CAP entities in remote devices shall negotiate all parameters defined in this document whenever the default values are not acceptable. Any missing configuration parameters are assumed to have their most recently (mutually) explicitly or implicitly accepted values. Even if all default values are acceptable, a Configuration Request packet with no options shall be sent. Implicitly accepted values are any default values for the configuration parameters specified in this document that have not been explicitly negotiated for the specific channel under configuration.

Each configuration parameter is one-directional and relative to the direction implied by the sender of a Configuration Request. If a device needs to establish the value of a configuration parameter in the opposite direction than the one implied by a Configuration Request, a new Configuration Request with the desired value of the configuration parameter in it needs to be sent in the direction opposite the one used for the original Configuration Request.

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

Figure 115 defines the format of the Configuration Request packet.

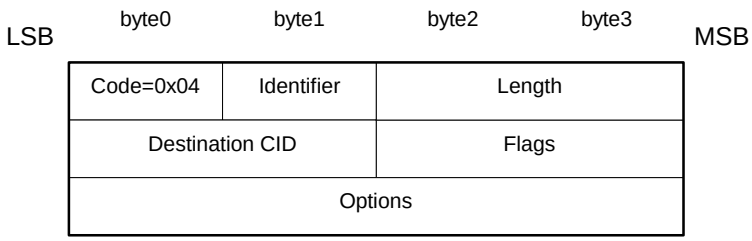


Figure 115—Configuration request packet

— *Length = 0x0004 or more octets*

— *Destination CID (DCID): 2 octets*

The field contains the channel end-point on the device receiving this Request packet.

— *Flags: 2 octets*

Figure 116 displays the two-octet Flags field. Note the most significant bit is shown on the left.

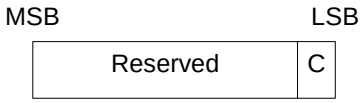


Figure 116—Configuration request flags field format

In Figure 116, C represents the continuation flag. When all configuration options cannot fit into the receiver's MTU_{sig}, the options must be passed in multiple configuration command packets. If all options fit into the receiver's MTU, then the continuation bit should not be used. Each Configuration Request should contain in integral number of options, partially formed options shall not be sent. Each Request will be tagged with a different Identifier and be matched with a Response with the same Identifier.

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

When used in the Configuration Response, the continuation flag shall be set if the flag is set in the Request. If the configuration flag is set in the Response when the matching Request does not set the flag, it indicates the responder has additional options to send to the requestor. In this situation, the requestor shall send null-option Configuration Requests (with cleared C-flag) to the responder until the responder replies with a Configuration Response where the continuation flag is clear. The Configuration Response packet with the configuration flag cleared shall be treated as the Configuration Response event in the channel state machine.

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

Other flags are reserved and should be cleared. L2CAP implementations should ignore these bits.

— *Configuration Options*

The list of the parameters and their values to be negotiated. These are defined in 10.6. Configuration Requests may contain no options (referred to as an empty or null configuration request) and can be used to request a response. For an empty configuration request the length field is set to 0x0004.

10.5.5 Configure response (code 0x05)

Configure Response packets shall be sent in reply to Configuration Request packets except when the error condition id covered by a Command Reject response. Each configuration parameter value (if any is present) in a Configuration Response reflects an "adjustment" to a configuration parameter value that has been sent (or, in case of default values, implied) in the corresponding Configuration Request. Thus, for example, if a

configuration parameter in a Configuration Request relates to traffic flowing from device A to device B, the sender of the Configuration Response will only adjust (if needed) this value again for the same traffic flowing from device A to device B. The options sent in the Response depend on the value in the Result field. Figure 117 defines the format of the Configuration Response packet.

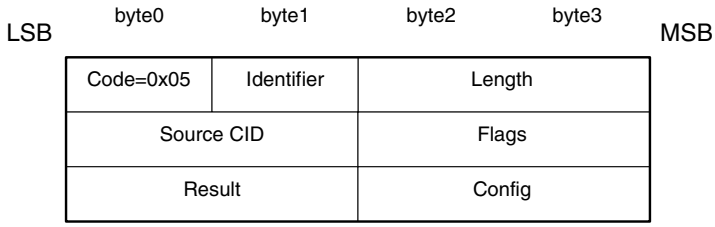


Figure 117—Configuration response packet

— *Length = 0x0006 or more octets*

— *Source CID (SCID): 2 octets*

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

— *Flags: 2 octets*

Figure 118 displays the two-octet Flags field. Note that the most significant bit is shown on the left.

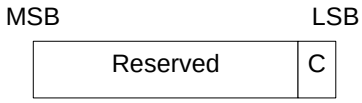


Figure 118—Configuration response flags field format

In Figure 118, C represents that more configuration responses will follow when set to 1. This flag indicates that the parameters included in the response are a partial subset of parameters being sent by the device sending the Response packet.
Other flags are reserved and should be cleared. L2CAP implementations should ignore these bits.

— *Result: 2 octets*

The Result field indicates whether or not the Request was acceptable. See Table 83 for possible result codes.

— *Configuration Options*

This field contains the list of parameters being negotiated. These are defined in 10.6. On a successful result, these parameters contain the return values for any wild card parameters (see 10.6.3) contained in the request.

On an unacceptable parameters failure (Result = 0x0001) the rejected parameters should be sent in the response with the values that would have been accepted if sent in the original request. Any missing configuration parameters are assumed to have their most recently (mutually) accepted values and they too can be included in the Configuration Response if need to be changed. Recall that, each configuration parameter is one-directional and relative to the direction implied by the sender of a Configuration Request. Thus, if the sender of the Configuration Response needs to

establish the value of a configuration parameter in the opposite direction than the one implied by an original Configuration Request, a new Configuration Request with the desired value of the configuration parameter in it needs to be sent in the direction opposite the one used for the original Connection Request.

On an unknown option failure (Result = 0x0003), the option types not understood by the recipient of the Request shall be included in the Response. Note that hints (defined in 10.6), those options in the Request that are skipped if not understood, shall not be included in the Response and shall not be the sole cause for rejecting the Request.

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

Table 83—Configuration response result codes

Result	Description
0x0000	Success
0x0001	Failure – unacceptable parameters
0x0002	Failure – rejected (no reason provided)
0x0003	Failure – unknown options
Other	RESERVED

10.5.6 Disconnection request (code 0x06)

Terminating an L2CAP channel requires that a disconnection request packet be sent and acknowledged by a disconnection response packet. Disconnection is requested using the signalling channel since all other L2CAP packets sent to the destination channel automatically get passed up to the next protocol layer. Figure 119 displays a disconnection packet request. The receiver shall ensure both source and destination CIDs match before initiating a connection disconnection. Once a Disconnection Request is issued, all incoming data in transit on this L2CAP channel will be discarded and any new additional outgoing data is not allowed. Once a disconnection request for a channel has been received, all data queued to be sent out on that channel may be discarded.

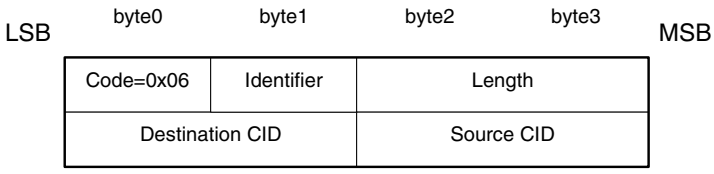


Figure 119—Disconnection request packet

— *Length = 0x0004 octets*

— *Destination CID (DCID): 2 octets*

This field specifies the end-point of the channel to be shutdown on the device receiving this request.

— *Source CID (SCID): 2 octets*

This field specifies the end-point of the channel to be shutdown on the device sending this request. The SCID and DCID are relative to the sender of this request and shall match those of the channel to be disconnected. If the DCID is not recognized by the receiver of this message, a Command Reject message with “invalid CID” result code shall be sent in response. If the receivers finds a DCID match but the SCID fails to find the same match, the request should be silently discarded.

10.5.7 Disconnection response (code 0x07)

Disconnection responses should be sent in response to each disconnection request (see Figure 120).

Figure 120—Disconnection response packet

— *Length = 0x0004 octets*

— *Destination CID (DCID): 2 octets*

This field identifies the channel end-point on the device sending the response.

— *Source CID (SCID): 2 octets*

This field identifies the channel end-point on the device receiving the response.

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

10.5.8 Echo request (code 0x08)

Echo requests are used to solicit a response from a remote L2CAP entity. These requests may be used for testing the link or passing vendor specific information using the optional data field. L2CAP entities shall respond to well-formed Echo Request packets with an Echo Response packet. The Data field is optional and implementation-dependent. L2CAP entities should ignore the contents of this field (see Figure 121).

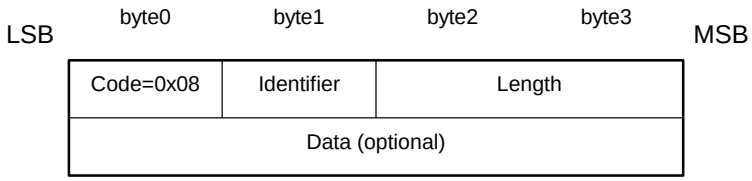


Figure 121—Echo request packet

10.5.9 Echo response (code 0x09)

Echo responses are sent upon receiving Echo Request packets. The identifier in the response shall match the identifier sent in the Request. The optional and implementation-dependent data field may contain the contents of the data field in the Request, different data, or no data at all (see Figure 122).

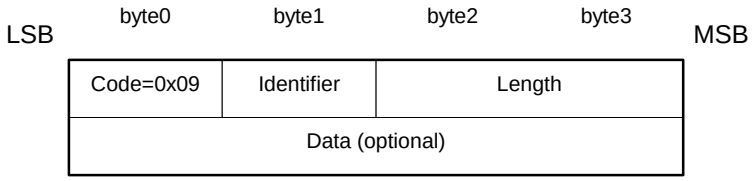


Figure 122—Echo response packet

10.5.10 Information Request (CODE 0x0A)

Information requests are used to solicit implementation-specific information from a remote L2CAP entity. L2CAP entities shall respond to well-formed Information Request packets with an Information Response packet (see Figure 123).

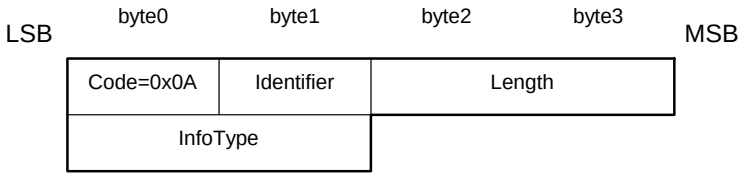


Figure 123—Information request packet

— *Length = 0x0002 octets*

— *InfoType: 2 octets*

The InfoType defines the type of implementation-specific information being solicited (see Table 84).

Table 84—InfoType definitions

Value	Description
0x0001	Connectionless MTU
Other	Reserved

10.5.11 Information response (CODE 0x0B)

Information responses are sent upon receiving Information Request packets. The identifier in the response shall match the identifier sent in the Request. The optional data field may contain the contents of the data field in the Request, different data, or no data at all (see Figure 124).

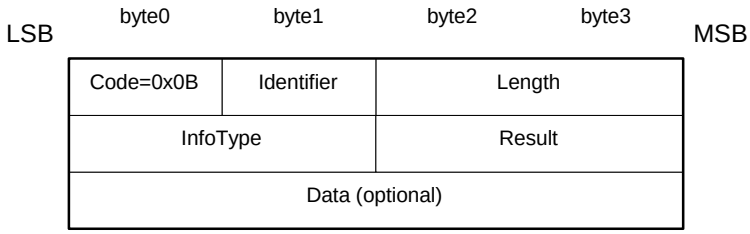


Figure 124—Information response packet

— *Length = 0x0004 or more octets*

— *InfoType: 2 octets*

Same value sent in the request.

— *Result: 2 octets*

The Result contains information about the success of the request. If result is “Success”, the data field contains the information as specified in Table 85. If result is “Not supported”, no data should be returned.

Table 85—Information response result values

Value	Description
0x0000	Success
0x0001	Not supported
Other	Reserved

— *Data: 0 or more octets*

The contents of the Data field depends on the InfoType. For the Connection MTU request, the data field contains the remote entity’s 2-octet acceptable connectionless MTU (see Table 86).

Table 86—Information response data fields

InfoType	Data	Data length (in octets)
0x0001	Connectionless MTU	2

10.6 Configuration parameter options

Options are a mechanism to extend the ability to negotiate different connection requirements. Options are transmitted in the form of information elements comprises an option type, an option length, and one or more option data fields. Figure 125 illustrates the format of an option.

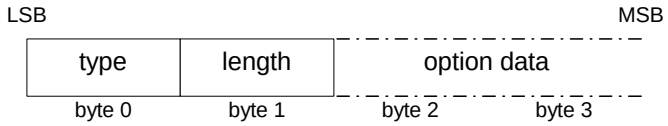


Figure 125—Configuration option format

— *Type: 1 octet*

The option type field defines the parameters being configured. The most significant bit of the type determines the action taken if the option is not recognized. The semantics assigned to the bit are defined as follows:

0–Option shall be recognized; refuse the configuration request

1–Option is a hint; skip the option and continue processing

— *Length: 1 octet*

The length field defines the number of octets in the option payload. So an option type with no payload has a length of 0.

— *Option data*

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

10.6.1 Maximum transmission unit (MTU)

This option specifies the payload size the sender is capable of accepting. The type is 0x01, and the payload length is 2 bytes, carrying the two-octet MTU size value as the only information element (see Figure 126).

Since all L2CAP implementations are capable to support a minimum L2CAP packet size (see 10.4), MTU is not really a negotiated value but rather an informational parameter to the remote device that the local device can accommodate in this channel an MTU larger than the minimum required. In the unlikely case that the remote device is only willing to send L2CAP packets in this channel that are larger than the MTU announced by the local device, then this Configuration Request will receive a negative response in which the remote device will include the value of MTU that is indented to transmit. In this case, it is implementation specific on whether the local device will continue the configuration process or even maintain this channel.

The remote device in its positive Configuration Response will include the actual MTU to be used on this channel for traffic flowing into the local device which is minimum{ MTU in configReq, outgoing MTU capability of remote device }. The MTU to be used on this channel but for the traffic flowing in the opposite

direction will be established when the remote device (with respect to this discussion) sends its own Configuration Request as explained in 10.5.4.

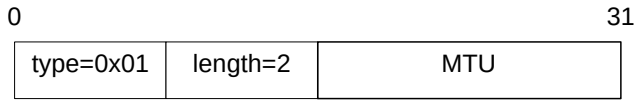


Figure 126—MTU Option Format

— MTU Size: 2 octets

The MTU field represents the largest L2CAP packet payload, in bytes, that the originator of the Request can accept for that channel. The MTU is asymmetric and the sender of the Request shall specify the MTU it can receive on this channel if it differs from the default value. L2CAP implementations shall support a minimum MTU size of 48 bytes. The default value is 672 bytes.[18]

10.6.2 Flush timeout option

This option is used to inform the recipient of the amount of time the originator's link controller/link manager will attempt to successfully transmit an L2CAP segment before giving up and flushing the packet. The type is 0x02 and the payload size is 2 octets (see Figure 127).

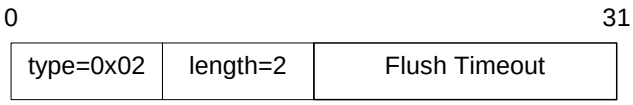


Figure 127—Flush timeout

— *Flush Timeout*

This value represents units of time measured in milliseconds. The value of 1 implies no retransmissions at the Baseband level should be performed since the minimum polling interval is 1.25 ms. The value of all 1's indicates an infinite amount of retransmissions. This is also referred to as "reliable channel". In this case, the link manager shall continue retransmitting a segment until physical link loss occurs. This is an asymmetric value and the sender of the Request shall specify its flush timeout value if it differs from the default value of 0xFFFF.

10.6.3 QoS option

This option specifies a flow specification (flowSpec) similar to RFC 1363 (see 2.5). If no QoS configuration parameter is negotiated the link should assume the default parameters discussed below. The QoS option is type 0x03.

When included in a Configuration Request, this option describes the outgoing traffic flow from the device sending the request to the device receiving it. When included in a positive Configuration Response, this option describes the incoming traffic flow agreement as seen from the device sending the response. When

[18] The default MTU was selected based on the payload carried by two Baseband DH5 packets (2 * 341 = 682) minus the Baseband ACL headers (2 * 2 = 4) and L2CAP header (6).

included in a negative Configuration Response, this option describes the preferred incoming traffic flow from the perspective of the device sending the response.

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

The remote device places information that depends on the value of the result field, see 10.5.5, in its Configuration Response. If the request was for Guaranteed Service, the response shall include specific values for any wild card parameters (see Token Rate and Token Bucket Size descriptions) contained in the request. If the result is “Failure – unacceptable parameters”, the response may include a list of outgoing flowspec parameters and parameter values that would make a new Connection Request from the local device acceptable by the remote device. Both explicitly referenced in a Configuration Request or implied configuration parameters can be included in a Configuration Response. Recall that any missing configuration parameters from a Configuration Request are assumed to have their most recently (mutually) accepted values. For both Best effort and Guaranteed service, when the QoS option appears in the Configuration Response, "do not cares" shall be present where they appeared in the Configuration Request (see Figure 128).

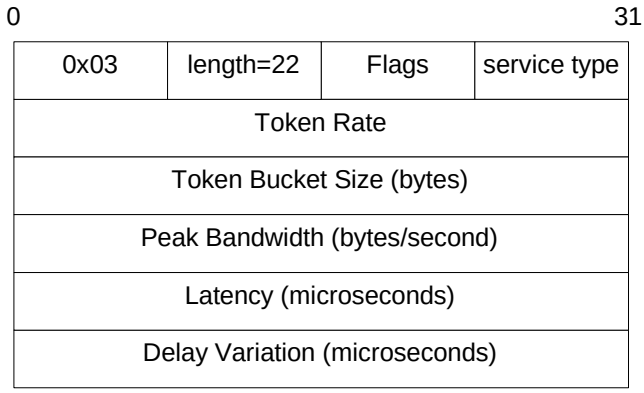


Figure 128—QoS flow specification

— *Flags: 1 octet*

Reserved for future use and shall be set to 0.

— *Service Type: 1 octet*

This field indicates the level of service required. Table 87 defines the different services available. If “No traffic” is selected, the remainder of the fields may be ignored because there is no data being sent across the channel in the outgoing direction.

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

— *Token Rate: 4 octets*

The value of this field represents the rate at which traffic credits are granted in bytes per second. An application may send data at this rate continuously. Burst data may be sent up to the token bucket size (see below). Until that data burst has been drained, an application must limit itself to the token rate. The value 0x00000000 indicates no token rate is specified. This is the default value and implies

indifference to token rate. The value 0xFFFFFFFF represents a wild card matching the maximum token rate available. The meaning of this value depends on the semantics associated with the service type. For best effort, the value is a hint that the application wants as much bandwidth as possible. For Guaranteed service the value represents the maximum bandwidth available at the time of the request.

— *Token Bucket Size: 4 octets*

The value of this field represents the size of the token bucket in bytes. If the bucket is full, then applications must either wait or discard data. The value of 0x00000000 represents no token bucket is needed; this is the default value. The value 0xFFFFFFFF represents a wild card matching the maximum token bucket available. The meaning of this value depends on the semantics associated with the service type. For best effort, the value indicates the application wants a bucket as big as possible. For Guaranteed service the value represents the maximum buffer space available at the time of the request.

— *Peak Bandwidth: 4 octets*

The value of this field, expressed in bytes per second, limits how fast packets may be sent back-to-back from applications. Some intermediate systems can take advantage of this information, resulting in more efficient resource allocation. The value of 0x00000000 states that the maximum bandwidth is unknown, which is the default value.

— *Latency: 4 octets*

The value of this field represents the maximum acceptable delay between transmission of a bit by the sender and its initial transmission over the air, expressed in microseconds. The precise interpretation of this number depends on the level of guarantee specified in the Class of Service. The value 0xFFFFFFFF represents a do not care and is the default value.

— *Delay Variation: 4 octets*

The value of this field is the difference, in microseconds, between the maximum and minimum possible delay that a packet will experience. This value is used by applications to determine the amount of buffer space needed at the receiving side in order to restore the original data transmission pattern. The value 0xFFFFFFFF represents a “do not care” and is the default value.

Table 87—Service type definitions

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

10.6.4 Configuration process

Negotiating the channel parameters involves three steps:

1) Informing the remote side of the nondefault parameters that the local side will accept using a Configuration Request
2) Remote side responds, agreeing or disagreeing to these values, including the default ones, using a Configuration Response. The local and remote devices repeat steps 1) and 2) as needed.
3) Repeat steps 1) and 2) exactly once more for the reverse direction.

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

10.6.4.1 Request Path

The Request Path negotiates the incoming MTU, flush timeout, and outgoing flowspec. Table 88 defines the configuration options that may be placed in the Configuration Request message and their semantics.

Table 88—Parameters allowed in request

Parameter	Description
MTU	Incoming MTU information
FlushTO	Outgoing flush timeout
OutFlow	Outgoing flow information

10.6.4.2 Response path

The Response Path negotiates the outgoing MTU (remote side's incoming MTU), the remote side's flush timeout, and incoming flowspec (remote side's outgoing flowspec). If a request-oriented parameter is not present in the Request message (reverts to default value), the remote side may negotiate for a nondefault value by including the proposed value in a negative Response message (see Table 89).

Table 89—Parameters allowed in response

Parameter	Description
MTU	Outgoing MTU information
FlushTO	Incoming flush timeout
InFlow	Incoming flow information

10.6.4.3 Configuration state machine

The configuration state machine shown in Figure 129 depicts two paths. Before leaving the CONFIG state and moving into the OPEN state, both paths shall reach closure. The request path requires the local device to receive a positive response to reach closure while the response path requires the local device to send a positive response to reach closure.

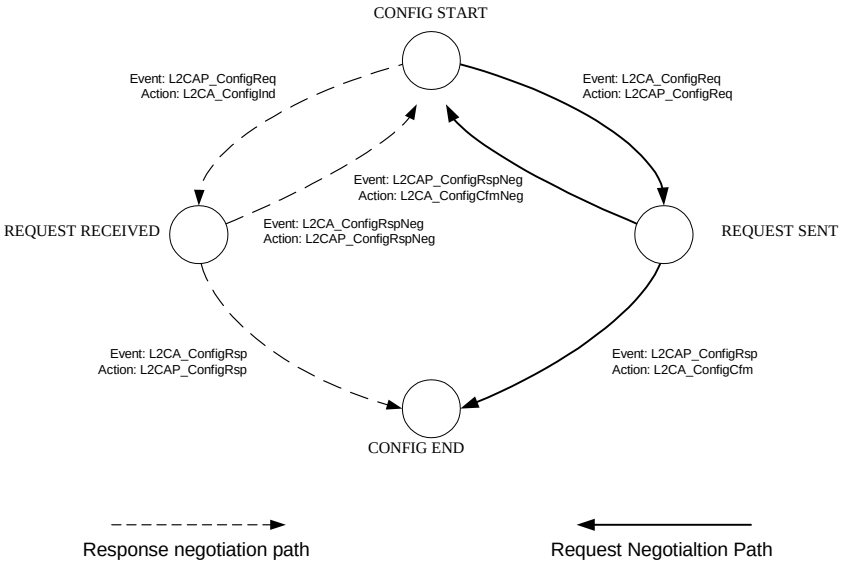


Figure 129—Configuration state machine

Subclause 10.8.1 provides some configuration examples.

10.7 Service primitives

This subclause presents an abstract description of the services offered by L2CAP in terms of service primitives and parameters. The service interface is required for testing. The interface is described independently of any platform specific implementation. All data values use Little Endian byte ordering.

10.7.1 Event indication

Table 90 provides the event indication service primative.

Table 90—Event indication

Service	Input parameters	Output parameters
EventIndication	Event, Callback	Result

Description:

The use of this primitive requests a callback when the selected indication Event occurs.

Input Parameters:

Event *Type: uint* *Size: 1 octet*

Value	Description
0x00	Reserved
0x01	L2CA_ConnectInd
0x02	L2CA_ConfigInd
0x03	L2CA_DisconnectInd
0x04	L2CA_QoSViolationInd
other	Reserved for future use

Callback *Type: function* *Size: N/A*

Event	Callback function input parameters
L2CA_ConnectInd	BD_ADDR, CID, PSM, Identifier
L2CA_ConfigInd	CID, OutMTU, InFlow, InFlushTO
L2CA_DisconnectInd	CID
L2CA_QoSViolationInd	BD_ADDR

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Event successfully registered
0x0001	Event registration failed

10.7.1.1 L2CA_ConnectInd callback

This callback function includes the parameters for the address of the remote device that issued the connection request, the local CID representing the channel being requested, the Identifier contained in the request, and the PSM value the request is targeting.

10.7.1.2 L2CA_ConfigInd callback

This callback function includes the parameters indicating the local CID of the channel the request has been sent to, the outgoing MTU size (maximum packet that can be sent across the channel) and the flowspec describing the characteristics of the incoming data. All other channel parameters are set to their default values if not provided by the remote device.

10.7.1.3 L2CA_DisconnectInd callback

This callback function includes the parameter indicating the local CID the request has been sent to.

10.7.1.4 L2CA_QoSViolationInd callback

This callback function includes the parameter indicating the address of the remote Bluetooth device where the QoS contract has been violated.

10.7.2 Connect

Table 91 shows the parameters associated with the L2CA connection request primitive.

Table 91—Connect request

Service	Input parameters	Output parameters
L2CA_ConnectReq	PSM, BD_ADDR	LCID, Result, Status

Description:

This primitive initiates the sending of an L2CAP_ConnectReq message and blocks until a corresponding L2CA_ConnectCfm(Neg) or L2CA_TimeOutInd eventis received.

The use of this primitive requests the creation of a channel representing a logical connection to a physical address. Input parameters are the target protocol (*PSM*) and remote device's 48-bit address (*BD_ADDR*). Output parameters are the local CID *(LCID)* allocated by the local L2CAP entity, and *Result* of the request. If the *Result* indicates success, the *LCID* value contains the identification of the local endpoint. Otherwise the *LCID* returned should be set to 0. If *Result* indicates a pending notification, the *Status* value may contain more information of what processing is delaying the establishment of the connection. Otherwise the *Status* value should be ignored.

Input Parameters:

PSM *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Target PSM provided for the connection

BD_ADDR *Type: uint* *Size: 6 octets*

Value	Description
0xXXXXXXXXXXXX	Unique Bluetooth address of target device

Output Parameters:

LCID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel if Result = 0x0000, otherwise set to 0.

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Connection successful and the CID identifies the local endpoint. Ignore Status parameter
0x0001	Connection pending. Check Status parameter for more information
0x0002	Connection refused because no service for the PSM has been registered
0x0003	Connection refused because the security architecture on the remote side has denied the request
0xEEEE	Connection timeout occurred. This is a result of a timer expiration indication being included in the connection confirm message

Status *Type: uint* *Size: 2 octets*

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

10.7.3 Connect response

Table 92 shows the parameters associated with the L2CA connection response primitive.

Table 92—Connect response

Service	Input parameters	Output parameters
L2CA_ConnectRsp	BD_ADDR, Identifier, LCID, Response, Status	Result

Description:

This primitive represents the L2CAP_ConnectRsp.

The use of this primitive issues a response to a connection request event indication. Input parameters are the remote device's 48-bit address, Identifier sent in the request, local CID, the Response code, and the Status attached to the Response code. The output parameter is the Result of the service request.

This primitive shall be called no more than once after receiving the callback indication. This primitive returns once the local L2CAP entity has validated the request. A successful return does indicate the response has been sent over the air interface.

Input Parameters:

BD_ADDR *Type: uint* *Size: 6 octets*

Value	Description
0xXXXXXXXXXXXX	Unique Bluetooth address of target device

Identifier *Type: uint* *Size: 1 octets*

Value	Description
0xXX.	This value shall match the value received in the L2CA_ConnectInd event described in 10.7.1.1

LCID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

Response *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Connection successful
0x0001	Connection pending
0x0002	Connection refused – PSM not supported
0x0003	Connection refused – security block
0x0004	Connection refused – no resources available
0xXXXX	Other connection response code

Status *Type: uint* *Size: 2 octets*

Value	Description
0x0000	No further information available
0x0001	Authentication pending
0x0002	Authorization pending
0xXXXX	Other status code

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Response successfully sent
0x0001	Failure to match any outstanding connection request

10.7.4 Configure

Table 93 shows the parameters associated with the L2CA configure primitive.

Table 93—Configure

Service	Input parameters	Output parameters
L2CA_ConfigReq	CID, InMTU, OutFlow, OutFlushTO, LinkTO	Result, InMTU, OutFlow, OutFlushTO

Description:

This primitive initiates the sending of an L2CAP_ConfigReq message and blocks until a corresponding L2CA_ConfigCfm(Neg) or L2CA_TimeOutInd eventis received.

The use of this primitive requests the initial configuration (or reconfiguration) of a channel to a new set of channel parameters. Input parameters are the local CID endpoint, new incoming receivable MTU (InMTU), new outgoing flow specification, and flush and link timeouts. Output parameters composing the L2CA_ConfigCfm(Neg) message are the Result, accepted incoming MTU(InMTU), the remote side's flow requests, and flush and link timeouts. Note that the output results are returned only after the local L2CAP entity transitions out of the CONFIG state (even if this transition is back to the CONFIG state).

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Local CID

InMTU *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Maximum transmission unit this channel can accept

OutFlow *Type: Flow* *Size: x octets*

Value	Description
flowspec	Quality of service parameters dealing with the traffic characteristics of the outgoing data flow

OutFlushTO *Size: 2 octets*

Value	Description
0xXXXX	Number of milliseconds to wait before an L2CAP packet that cannot be acknowledged at the physical layer is dropped
0x0000	Request to use the existing flush timeout value if one exists, otherwise the default value (0xFFFF) will be used
0x0001	Perform no retransmissions at the Baseband layer
0xFFFF	Perform retransmission at the Baseband layer until the link timeout terminates the channel

LinkTO *Size: 2 octets*

Value	Description
0xXXXX	Number of milliseconds to wait before terminating an unresponsive link

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Configuration is successful. Parameters contain agreed upon values
0x0001	Failure – invalid CID
0x0002	Failure – unacceptable parameters
0x0003	Failure – signalling MTU exceeded
0x0004	Failure – unknown options
0xEEEE	Configuration timeout occurred. This is a result of a timer expiration indication being included in the configuration confirm

InMTU *Size: 2 octets*

Value	Description
0xXXXX	Maximum transmission unit that the remote unit will send across this channel (maybe less or equal to the InMTU input parameter).

OutFlow *Size: 2 octets*

Value	Description
FlowSpec	Quality of service parameters dealing with the traffic characteristics of the agreed-upon out-going data flow if Result is successful. Otherwise this represents the requested Quality of Service

OutFlushTO *Size: 2 octets*

Value	Description
0xXXXX	Number of milliseconds before an L2CAP packet that cannot be acknowledged at the physical layer is dropped. This value is informative of the actual value that will be used for outgoing packets. It may be less or equal to the OutFlushTO parameter given as input.

10.7.5 Configuration response

Table 94 shows the parameters associated with the L2CA configuration response primitive.

Table 94—Configuration response

Service	Input parameters	Output parameters
L2CA_ConfigRsp	CID, OutMTU, InFlow	Result

Description:

This primitive represents the L2CAP_ConfigRsp.

The use of this primitive issues a response to a configuration request event indication. Input parameters include the local CID of the endpoint being configured, outgoing transmit MTU (which may be equal or less to the OutMTU parameter in the L2CA_ConfigInd event) and the accepted flowspec for incoming traffic. The output parameter is the Result value.

Input Parameters:

LCID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Local channel identifier

OutMTU *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Maximum transmission unit this channel will send

InFlow *Type: Flow* *Size: x octets*

Value	Description
FlowSpec	Quality of service parameters dealing with the traffic characteristics of the incoming data flow

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Configuration is successful. Parameters contain agreed upon values
0x0001	Configuration failed – unacceptable parameters
0x0002	Configuration failed – rejected
0x0003	Configuration failed – invalid CID
0x0004	Configuration failed – unknown options
0xXXXX	Reserved

10.7.6 Disconnect

Table 95 shows the parameters associated with the L2CA disconnect response primitive.

Table 95—Disconnect

Service	Input parameters	Output parameters
L2CA_DisconnectReq	CID	Result

Description:

This primitive represents the L2CAP_DisconnectReq and the returned output parameters represent the corresponding L2CAP_DisconnectRsp or the RTX timer expiration.

The use of this primitive requests the disconnection of the channel. Input parameter is the *CID* representing the local channel endpoint. Output parameter is *Result*. *Result* is zero if a L2CAP_DisconnectRsp is received, otherwise a non-zero value is returned. Once disconnection has been requested, no process will be able to successfully read or write from the CID. Writes in progress should continue to be processed.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Disconnection successful. This is a result of the receipt of a disconnection response message
0xEEEE	Disconnection timeout occurred.

10.7.7 Write

Table 96 shows the parameters associated with the L2CA write primitive.

Table 96—Write

Service	Input parameters	Output parameters
L2CA_DataWrite	CID, Length, OutBuffer	Size, Result

Description:

The use of this primitive requests the transfer of data across the channel. If the length of the data exceeds the OutMTU then only the first OutMTU bytes are sent This command may be used for both connection-oriented and connectionless traffic.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

Length *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Size, in bytes, of the buffer where data to be transmitted are stored

OutBuffer *Type: pointer* *Size: N/A*

Value	Description
N/A	Address of the input buffer used to store the message

Output Parameters:

Size *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	The number of bytes transferred

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Successful write
0x0001	Error – Flush timeout expired
0x0002	Error – Link termination (perhaps this should be left to the indication)

10.7.8 Read

Table 97 shows the parameters associated with the L2CA read primitive.

Table 97—Read

Service	Input parameters	Output parameters
L2CA_DataRead	CID, Length, InBuffer	Result, N

Description:

The use of this primitive requests for the reception of data. This request returns when data is available or the link is terminated. The data returned represents a single L2CAP payload. If not enough data is available, the command will block until the data arrives or the link is terminated. If the payload is bigger than the buffer, only the portion of the payload that fits into the buffer will be returned, and the remainder of the payload will be discarded. This command may be used for both connection-oriented and connectionless traffic.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	CID

Length *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Size, in bytes, of the buffer where received data are to be stored

InBuffer *Type: pointer* *Size: N/A*

Value	Description
N/A	Address of the buffer used to store the message

Output parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Success
0x0001	Unsuccessful

N *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Number of bytes transferred to InBuffer

10.7.9 Group create

Table 98 shows the parameters associated with the L2CA group create primitive.

Table 98—Group create

Service	Input parameters	Output parameters
L2CA_GroupCreate	PSM	CID

Description:

The use of this primitive requests the creation of a CID to represent a logical connection to multiple devices. Input parameter is the *PSM* value that the outgoing connectionless traffic is labelled with, and the filter used for incoming traffic. Output parameter is the *CID* representing the local endpoint. On creation, the group is empty but incoming traffic destined for the PSM value is readable.

Input Parameters:

PSM *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Protocol/service multiplexer value

Output Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

10.7.10 Group close

Table 99 shows the parameters associated with the L2CA group close primitive.

Table 99—Group close

Service	Input parameters	Output parameters
L2CA_GroupClose	CID	Result

Description:

The use of this primitive closes down a Group.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Successful closure of the channel
0x0001	Invalid CID

10.7.11 Group add member

Table 100 shows the parameters associated with the L2CA group add member primitive.

Table 100—Group add member

Service	Input parameters	Output parameters
L2CA_GroupAddMember	CID, BD_ADDR	Result

Description:

The use of this primitive requests the addition of a member to a group. The input parameter includes the CID representing the group and the BD_ADDR of the group member to be added. The output parameter Result confirms the success or failure of the request.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

BD_ADDR *Type: uint* *Size: 6 octets*

Value	Description
0xXXXXXXXXXXXX	Remote device address

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Success
0x0001	Failure to establish connection to remote device
Other	Reserved

10.7.12 Group remove member

Table 101 shows the parameters associated with the L2CA group remove member primitive.

Table 101—Group remove member

Service	Input parameters	Output parameters
L2CA_GroupRemoveMember	CID, BD_ADDR	Result

Description:

The use of this primitive requests the removal of a member from a group. The input parameters include the CID representing the group and BD_ADDR of the group member to be removed. The output parameter Result confirms the success or failure of the request.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

BD_ADDR *Type: uint* *Size: 6 octets*

Value	Description
0xXXXXXXXXXXXX	Unique Bluetooth address device to be removed

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Success
0x0001	Failure – device not a member of the group
Other	Reserved

10.7.13 Get group membership

Table 102 shows the parameters associated with the L2CA get group membership primitive.

Table 102—Get group membership

Service	Input parameters	Output parameters
L2CA_GroupMembership	CID	Result, N, BD_ADDR_Lst

Description:

The use of this primitive requests a report of the members of a group. The input parameter CID represents the group being queried. The output parameter Result confirms the success or failure of the operation. If the Result is successful, BD_ADDR_Lst is a list of the Bluetooth addresses of the N members of the group.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel.

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Success
0x0001	Failure – group does not exist
Other	Reserved

N *Type: uint* *Size: 2 octets*

Value	Description
0x0000-0xFFFF	The number of devices in the group identified by the channel end-point CID. If Result indicates failure, N should be set to 0.

BD_ADDR_Lst *Type: pointer* *Size: N/A*

Value	Description
N/A	List of N unique Bluetooth addresses of the devices in the group identified by the channel end-point CID. If Result indicates failure, the all-zero address is the only address that should be returned.

10.7.14 Ping

Table 103 shows the parameters associated with the L2CA ping primitive.

Table 103—Ping

Service	Input parameters	Output parameters
L2CA_Ping	BD_ADDR, ECHO_DATA, Length	Result, ECHO_DATA, Size

Description:

This primitive represents the initiation of an L2CA_EchoReq message and the reception of the corresponding L2CAP_EchoRsp message.

Input Parameters:

BD_ADDR *Type: uint* *Size: 6 octets*

Value	Description
0xXXXXXXXXXXXX	Unique Bluetooth address of target device.

ECHO_DATA *Type: pointer* *Size: N/A*

Value	Description
N/A	The buffer containing the contents to be transmitted in the data payload of the Echo Request command.

Length *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Size, in bytes, of the data in the buffer.

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Response received.
0x0001	Timeout occurred.

ECHO_DATA *Type: pointer* *Size: N/A*

Value	Description
N/A	The buffer containing the contents received in the data payload of the Echo Response command.

Size *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Size, in bytes, of the data in the buffer.

10.7.15 GetInfo

Table 104 shows the parameters associated with the L2CA GetInfo primitive.

Table 104—GetInfo

Service	Input parameters	Output parameters
L2CA_GetInfo	BD_ADDR, InfoType	Result, InfoData, Size

Description:

This primitive represents the initiation of an L2CA_InfoReq messageand the reception of the corresponding L2CAP_InfoRsp message.

Input Parameters:

BD_ADDR *Type: uint* *Size: 6 octets*

Value	Description
0xXXXXXXXXXXXX	Unique Bluetooth address of target device

InfoType *Type: uint* *Size: 2 octets*

Value	Description
0x0001	Maximum connectionless MTU size

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Response received
0x0001	Not supported
0x0002	Informational PDU rejected, not supported by remote device
0x0003	Timeout occurred

InfoData *Type: pointer* *Size: N/A*

Value	Description
N/A	The buffer containing the contents received in the data payload of the Information Response command.

Size *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Size, in bytes, of the data in the InfoData buffer.

10.7.16 Disable connectionless traffic

Table 105 shows the parameters associated with the L2CA disable connectionless traffic primitive.

Table 105—Disable connectionless traffic

Service	Input parameters	Output parameters
L2CA_DisableCLT	PSM	Result

Description:

General request to disable the reception of connectionless packets. The input parameter is the *PSM* value indicating service that should be blocked. This command may be used to incrementally disable a set of PSM values. The use of the “invalid” PSM 0x0000 blocks all connectionless traffic. The output parameter *Result* indicates the success or failure of the command. A limited device might support only general blocking rather than PSM-specific blocks and would fail to block a single nonzero PSM value.

Input Parameters:

PSM *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Block all connectionless traffic
0xXXXX	Protocol/Service Multiplexer field to be blocked

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Successful
0x0001	Failure – not supported

10.7.17 Enable connectionless traffic

Table 106 shows the parameters associated with the L2CA enable connectionless traffic primitive.

Table 106—Enable connectionless traffic

Service	Input parameters	Output parameters
L2CA_EnableCLT	PSM	Result

Description:

General request to enable the reception of connectionless packets. The input parameter is the *PSM* value indicating the service that should be unblocked. This command may be used to incrementally enable a set of PSM values. The use of the 'invalid' PSM 0x0000 enables all connectionless traffic. The output parameter *Result* indicates the success or failure of the command. A limited device might support only general enabling rather than PSM-specific filters, and would fail to enable a single non-zero PSM value.

Input Parameters:

PSM *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Enable all connectionless traffic
0xXXXX	Protocol/Service Multiplexer field to enable

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Successful
0x0001	Failure – not supported

10.8 Summary

L2CAP is one of two link level protocols running over the Baseband. L2CAP is responsible for higher level protocol multiplexing, MTU abstraction, group management, and conveying quality of service information to the link level.

Protocol multiplexing is supported by defining channels. Each channel is bound to a single protocol in a many-to-one fashion. Multiple channels can be bound to the same protocol, but a channel cannot be bound to multiple protocols. Each L2CAP packet received on a channel is directed to the appropriate higher level protocol.

L2CAP abstracts the variable-sized packets used by the Baseband Protocol (see Clause 8). It supports large packet sizes up to 64 kilobytes using a low-overhead segmentation-and-reassembly mechanism.

Group management provides the abstraction of a group of units allowing more efficient mapping between groups and members of the Bluetooth piconet. Group communication is connectionless and unreliable. When composed of only a pair of units, groups provide connectionless channel alternative to L2CAP's connection-oriented channel.

L2CAP conveys QoS information across channels and provides some admission control to prevent additional channels from violating existing QoS contracts.

10.8.1 Example configuration MSCs

Figure 130 illustrates the basic configuration process. In this example, the devices exchange MTU information. All other values are assumed to be default.

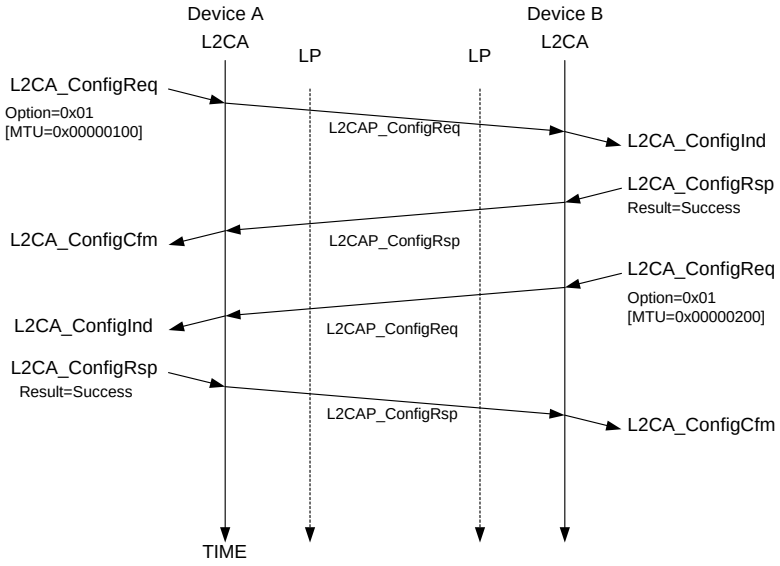


Figure 130—Basic MTU exchange

Figure 131 illustrates how two devices interoperate even though one device supports more options than the other does. Device A is an upgraded version. It uses a hypothetically defined option type 0x20 for link-level security. Device B rejects the command using the Configuration Response packet with result "unknown parameter" informing Device A that option 0x20 is not understood. Device A then resends the request omitting option 0x20. Device B notices that it does not need to such a large MTU and accepts the request but includes in the response the MTU option informing Device A that Device B will not send an L2CAP packet with a payload larger than 0x80 octets over this channel. On receipt of the response, Device A could reduce the buffer allocated to hold incoming traffic.

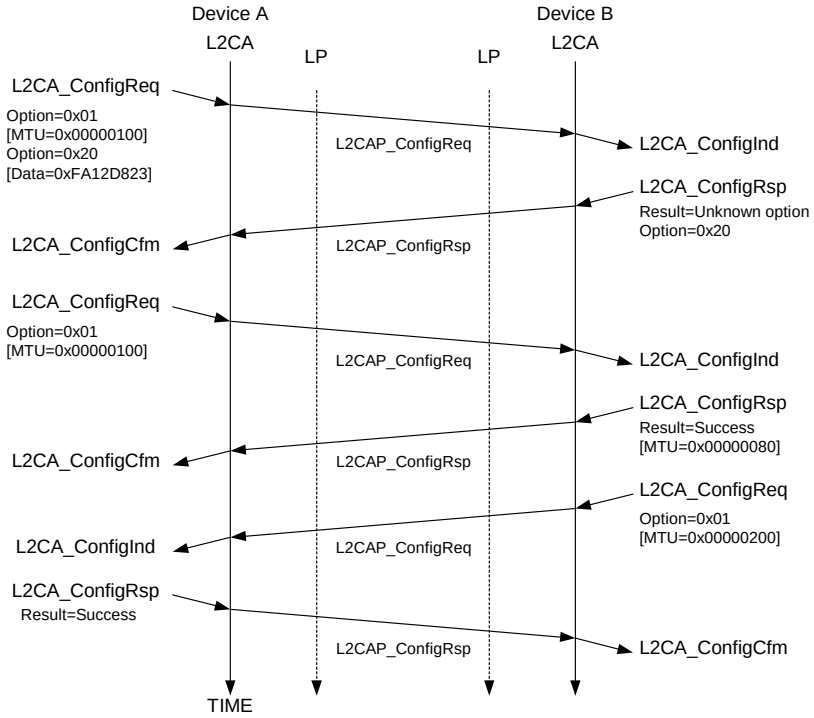


Figure 131—Dealing with unknown options

Figure 132 illustrates an unsuccessful configuration request. There are two problems described by this example. The first problem is that the configuration request is placed in an L2CAP packet that cannot be accepted by the remote device, due to its size. The remote device informs the sender of this problem using the Command Reject message. Device A then resends the configuration options using two smaller L2CAP_ConfigReq messages.

The second problem is an attempt to configure a channel with an invalid CID. For example, device B may not have an open connection on that CID (0x01234567 in this example case).

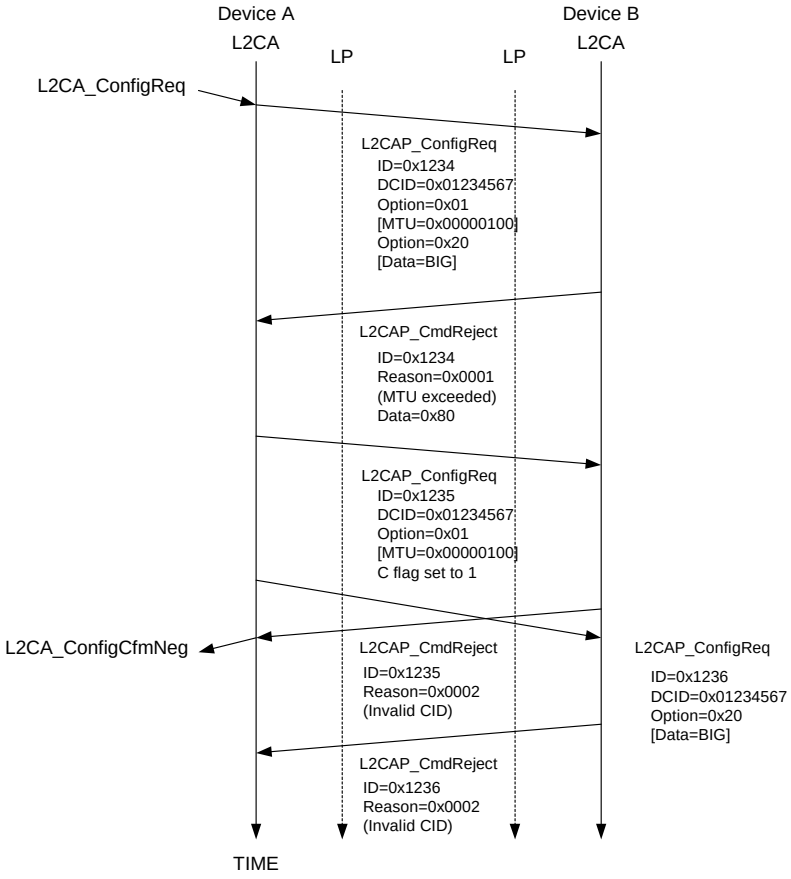


Figure 132—Unsuccessful configuration request

10.8.2 Implementation guidelines

This subclause contains some guidelines for implementations. These guidelines are not part of the compliance tests. At the moment they are simply suggestions on how to solve some difficult problems.

10.8.3 RTX timer

Implementations should not start this timer on an L2CAP Connection Request packet unless the physical link has been established. Otherwise the Baseband paging mechanism might increase the cost of the request beyond that of the minimal timeout value. If an implementation performs some form of security check it is recommended that the connection pending response be sent back prior to any consultation with a security manager that might perform Baseband authentication commands. If any security check requires user interaction, the link might timeout waiting for the user to enter a PIN.

10.8.4 QoS mapping to LM and L2CAP implementations

10.8.4.1 Token rate

The LM should ensure data is removed from the transmission buffer at this rate. The LM should ensure the polling interval is fast enough to support this data rate. The polling interval should be adjusted if the packet

type changes. If the buffer overflows, and the service type is Guaranteed, a QoS violation should be reported. If the service type is Best Effort, and a Token Rate was non-zero, a QoS violation should also be reported.

Given a Token Rate of 0xFFFFFFFF, and Service Type of Guaranteed, the LM should refuse any additional connections from remote devices and disable all periodic scans.

10.8.4.2 Token bucket size

L2CAP implementations should ensure that a buffer meeting the size request is allocated for the channel. If no buffer is available, and the service type is Guaranteed, the request should be rejected. If no appropriately sized buffer is available, and the service type is Best Effort, the largest available buffer should be allocated.

10.8.4.3 Peak bandwidth

If the token bucket buffer overflows, a QoS violation should be raised.

10.8.4.4 Latency

The LM should ensure the polling interval is at least this value. If the polling interval necessary to support the token rate is less than this value, the smaller interval should be used. If this interval cannot be supported, a QoS violation should be raised.

10.8.4.5 Delay variation

The LM may ignore this value because there is no clear mapping between L2CAP packet delays and the necessary polling interval without requiring the LM to comprehend the length field in L2CAP packets.

10.8.5 Collision tables

Table I: Result of Second Link Timeout Request

Current value	Requested value	Result
X	X	X
X	Y	If (X < Y) then X, else Y

Table II: Result of Second Flush Timeout Request

Current value	Requested value	Result
N	0	N
N	N	N
N	M ≠ N	Reject

11. Control interface

Figure 133 indicates the relationship of the Bluetooth protocol stack to this clause. This clause describes the functional specification for the Control interface for IEEE Std 802.15.1-2002. It is based upon the HCI section of the Bluetooth Specification. With the exception of 11.1, the text and graphics are solely an extraction from the original document. Normally this function would be refered to as a Management function in the IEEE nomenclature. The term Control interface was used to make clear the origin of the material.

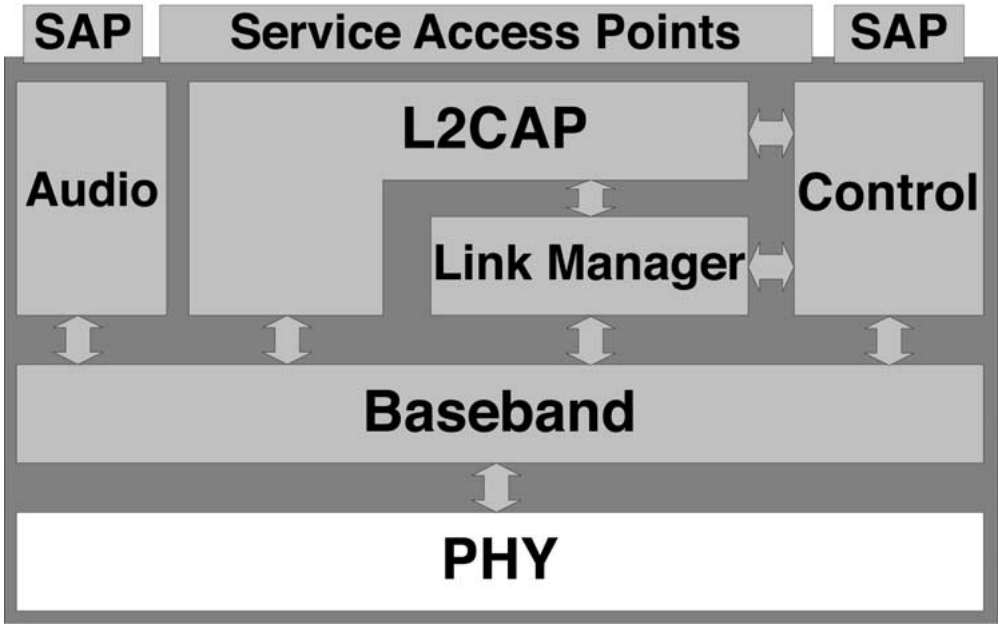


Figure 133—Control interface relationships

11.1 IEEE introduction

The HCI provides a command interface to the baseband controller and link manager, and access to hardware status and control registers. This interface provides a uniform method of accessing the Bluetooth baseband capabilities. The HCI section has two functions in the Bluetooth Specification:

1) Define a basis for a physical interface for a Bluetooth external module
2) Define the control functions necessary for all Bluetooth implementations

Since IEEE 802 Standards describe protocols and not implementations, the physical interface portions of the original document were omitted. Those portions that describe the logical relationship between the lower software layers and the controlling entity above it have been preserved.

The Host receives asynchronous notifications of HCI (management) events independent of which host controller transport layer is used. HCI events are used for notifying the Host when something occurs. When the Host discovers that an event has occurred, it will then parse the received event packet to determine which event occurred.

11.1.1 Differences between IEEE Std 802.15.1-2002 and the Bluetooth Specification

Table 107—Bluetooth commands omitted from IEEE Std 802.15.1-2002

Command	Meaning
Set_Host_Controller_To_Host_Flow_Control	The Set_Host_Controller_To_Host_Flow_Control command is used by the Host to turn flow control on or off in the direction from the Host Controller to the Host.
Host_Buffer_Size	The Host_Buffer_Size command is used by the Host to notify the Host Controller about its buffer sizes for ACL and SCO data. The Host Controller will segment the data to be transmitted from the Host Controller to the Host, so that data contained in HCI Data Packets will not exceed these sizes.
Host_Number_Of_Completed_Packets	The Host_Number_Of_Completed_Packets command is used by the Host to indicate to the Host Controller when the Host is ready to receive more HCI packets for any connection handle.
Read_SCO_Flow_Control_Enable	The Read_SCO_Flow_Control_Enable command provides the ability to read the SCO_Flow_Control_Enable setting. By using this setting, the Host can decide if the Host Controller will send Number Of Completed Packets events for SCO Connection Handles.
Write_SCO_Flow_Control_Enable	The Write_SCO_Flow_Control_Enable command provides the ability to write the SCO_Flow_Control_Enable setting. By using this setting, the Host can decide if the Host Controller will send Number Of Completed Packets events for SCO Connection Handles.

NOTE—The commands in Table 107 have been omitted from this standard, since they represent the physical interface implementation.

11.2 HCI commands

11.2.1 Introduction

The HCI provides a uniform command method of accessing the Bluetooth hardware capabilities. The HCI Link commands provide the Host with the ability to control the link layer connections to other Bluetooth devices. These commands typically involve the LM to exchange LMP commands with remote Bluetooth devices. For details, see Clause 9.

The HCI Policy commands are used to affect the behavior of the local and remote LM. These Policy commands provide the Host with methods of influencing how the LM manages the piconet. The Host Controller and Baseband, Informational, and Status commands provide the Host access to various registers in the Host Controller.

HCI commands may take different amounts of time to be completed. Therefore, the results of commands will be reported back to the Host in the form of an event. For example, for most HCI commands the Host Controller will generate the Command Complete event when a command is completed. This event contains the return parameters for the completed HCI command. For enabling the Host to detect errors on the HCI-Transport Layer, there needs to be a timeout between the transmission of the Host's command and the reception of the Host Controller's response (e.g., a Command Complete or Command Status event). Since the maximum response timeout is strongly dependent on the HCI-Transport Layer used, it is recommended to use a default value of 1 s for this timer. This amount of time is also dependent on the number of commands unprocessed in the command queue.

11.2.2 Terminology

Baseband packet: The smallest unit of data that is transmitted by one device to another, as defined in Clause 8.

Packet: A higher-level protocol message than the baseband packet, currently only L2CAP (see Clause 10) is defined, but additional packet types may be defined later.

Connection handle: A connection handle is a 12-bit identifier which is used to uniquely address a data/voice connection from one Bluetooth device to another. The connection handles can be visualized as identifying a unique data pipe that connects two Bluetooth devices. The connection handle is maintained for the lifetime of a connection, including when a device enters Park, Sniff, or Hold mode. The Connection Handle value has local scope between Host and Host Controller. There can be multiple connection handles for any given pair of Bluetooth devices but only one ACL connection.

Event: A mechanism that the HCI uses to notify the Host for command completion, link layer status changes, etc.

11.2.3 Data and parameter formats

- All values are in Binary and Hexadecimal Little Endian formats unless otherwise noted.
- In addition, all parameters which can have negative values shall use 2's complement when specifying values
- Arrayed parameters are specified using the following notation: ParameterA[i]. If more than one set of arrayed parameters are specified (e.g,. ParameterA[i], ParameterB[i]), then the order of the parameters are as follows: ParameterA[0], ParameterB[0], ParameterA[1], ParameterB[1], ParameterA[2], ParameterB[2], … ParameterA[n], ParameterB[n].
- Unless noted otherwise, all parameter values are sent and received in Little Endian format [i.e., for multibyte parameters the rightmost (least significant byte) is transmitted first].
- All command and event parameters that are not arrayed and all elements in an arrayed parameter have fixed sizes (an integer number of bytes). The parameters and the size of each not arrayed parameter (or of each element in an arrayed parameter) contained in a command or an event is specified for each command or event. The number of elements in an arrayed parameter is not fixed.
- Where bit strings are specified, the low order bit is the right hand bit, e.g., 0 is the low order bit in "10".

11.2.4 Exchange of HCI-specific information

The Host Controller Transport Layer provides transparent exchange of HCI-specific information. These transporting mechanisms provide the ability for the Host to send HCI commands, ACL data, and SCO data to the Host Controller. These transport mechanisms also provide the ability for the Host to receive HCI events, ACL data, and SCO data from the Host Controller.

Since the Host Controller Transport Layer provides transparent exchange of HCI-specific information, the HCI specification specifies the format of the commands, events, and data exchange between the Host and the Host Controller. The next sections specify the HCI packet formats.

11.2.4.1 HCI command packet

The HCI Command Packet is used to send commands to the Host Controller from the Host. The format of the HCI Command Packet is shown in Figure 134, and the definition of each field is explained below. When the Host Controller completes most of the commands, a Command Complete event is sent to the Host. Some commands do not receive a Command Complete event when they have been completed. Instead, when the Host Controller receives one of these commands the Host Controller sends a Command Status event back to the Host when it has begun to execute the command. Later on, when the actions associated with the command have finished, an event that is associated with the sent command will be sent by the Host Controller to the Host. However, if the command does not begin to execute (there may be a parameter error or the command may currently not be allowed), the event associated with the sent command will not be returned. The Command Status event will, in this case, return the appropriate error code in the Status parameter. On initial power-on, and after a reset, the Host can send a maximum of one outstanding HCI Command Packet until a Command Complete or Command Status event has been received. If an error occurs for a command for which a Command Complete event is returned, the Return_Parameters field may not contain all the return parameters specified for the command. The Status parameter, which explains the error reason and which is the first return parameter, will always be returned. If there is a Connection_Handle parameter or a BD_ADDR parameter right after the Status parameter, this parameter will also be returned so that the Host can identify to which instance of a command the Command Complete event belongs. In this case, the Connection_Handle or BD_ADDR parameter will have exactly the same value as that in the corresponding command parameter. It is implementation specific whether more parameters will be returned in case of an error.

Note that the BD_ADDR return parameter of the command Read_BD_ADDR is not used to identify to which instance of the Read_BD_ADDR command the Command Complete event belongs. It is therefore not mandatory for the Host Controller to return this parameter in case of an error.

If an error occurs for a command for which no Command Complete event is returned, all parameters returned with the event associated with this command may not be valid. The Host shall take care as to which parameters may have valid values depending on the value of the Status parameter of the Complete event associated with the given command. The Command Complete and Command Status events contain a parameter called Num_HCI_Command_Packets, which indicates the number of HCI Command Packets the Host is currently allowed to send to the Host Controller. The Host Controller may buffer one or more HCI command packets, but the Host Controller shall start performing the commands in the order in which they are received. The Host Controller can start performing a command before it completes previous commands. Therefore, the commands do not always complete in the order they are started. The Host Controller shall be able to accept HCI Command Packets with up to 255 bytes of data excluding the HCI Command Packet header.

Each command is assigned a 2 byte Opcode used to uniquely identify different types of commands. The Opcode parameter is divided into two fields, called the OpCode Group Field (OGF) and OpCode Command Field (OCF). The OGF occupies the upper 6 bits of the Opcode, while the OCF occupies the remaining 10 bits. The OGF of 0x3F is reserved for vendor-specific debug commands. The OGF of 0x3E is reserved for

Bluetooth Logo Testing. The organization of the Opcodes allows additional information to be inferred without fully decoding the entire Opcode.

Note that the OGF composed of all “ones” has been reserved for vendor-specific debug commands. These commands are vendor-specific and are used during manufacturing, for a possible method for updating firmware, and for debugging.

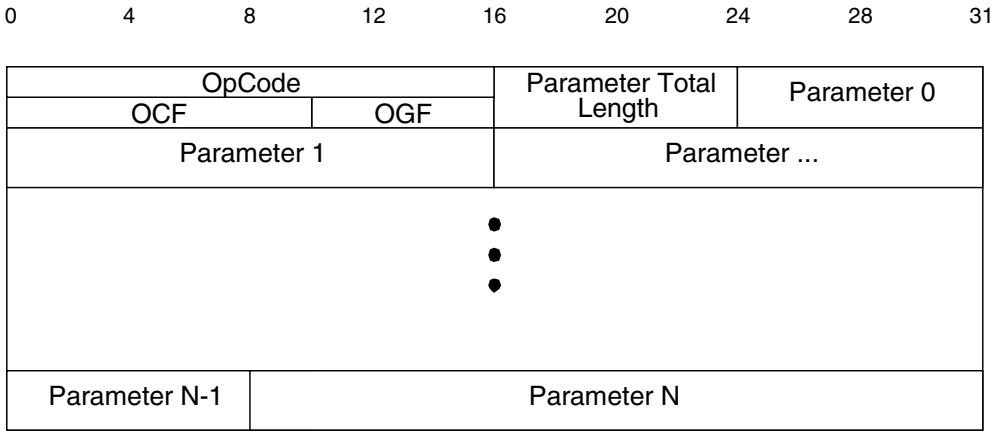


Figure 134—HCI command packet

Op_Code: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	OGFRange (6 bits): 0x00–0x3F (0x3E reserved for Bluetooth logo testing and 0x3F reserved for vendor-specific debug commands). OCF Range (10 bits): 0x0000–0x03FF.

Parameter_Total_Length: *Size: 1 Byte*

Value	Parameter description
0xXX	Lengths of all of the parameters contained in this packet measured in bytes. (N.B.: total length of parameters, not number of parameters).

Parameter 0–N: *Size: Parameter Total Length*

Value	Parameter description
0xXX	Each command has a specific number of parameters associated with it. These parameters and the size of each of the parameters are defined for each command. Each parameter is an integer number of bytes in size.

11.2.4.2 HCI event packet

The HCI Event Packet is used by the Host Controller to notify the Host when events occur. The Host shall be able to accept HCI Event Packets with up to 255 bytes of data excluding the HCI Event Packet header. The format of the HCI Event Packet is shown in Figure 135, and the definition of each field is explained below.

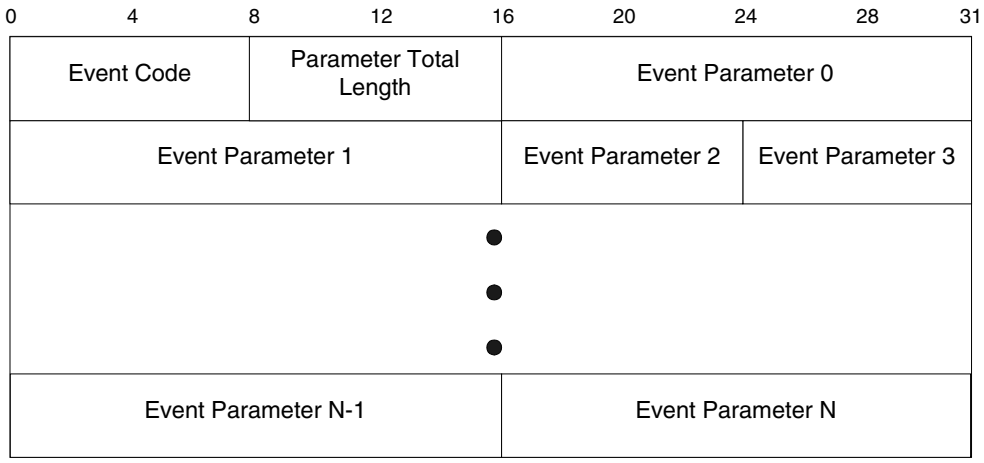


Figure 135—HCI event packet

Event_Code: *Size: 1 Byte*

Value	Parameter description
0xXX	Each event is assigned a 1-Byte event code used to uniquely identify different types of events. Range: 0x00–0xFF (The event code 0xFF is reserved for the event code used for vendor-specific debug events. In addition, the event code 0xFE is also reserved for Bluetooth Logo Testing).

Parameter_Total_Length: *Size: 1 Byte*

Value	Parameter description
0xXX	Length of all of the parameters contained in this packet, measured in bytes.

Event_Parameter 0–N: *Size: Parameter Total Length*

Value	Parameter description
0xXX	Each event has a specific number of parameters associated with it. These parameters and the size of each of the parameters are defined for each event. Each parameter is an integer number of bytes in size.

11.2.4.3 HCI data packets

HCI Data Packets are used to exchange data between the Host and Host Controller. The data packets are defined for both ACL and SCO data types. The format of the HCI ACL Data Packet is shown in Figure 136, and the format of the SCO Data Packet is shown in Figure 137. The definition for each of the fields in the data packets is explained below.

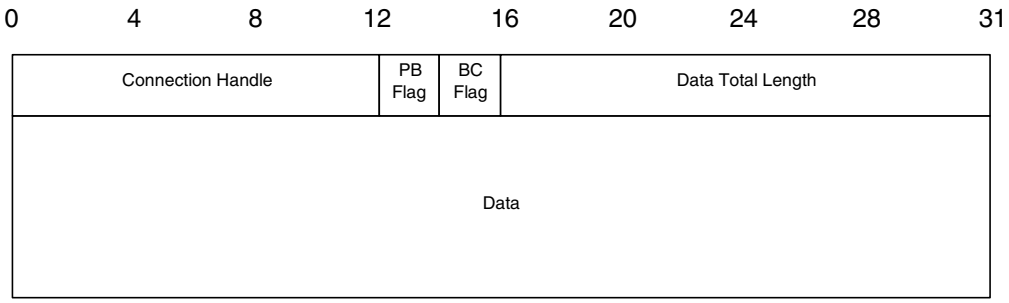


Figure 136—HCI ACL data packet

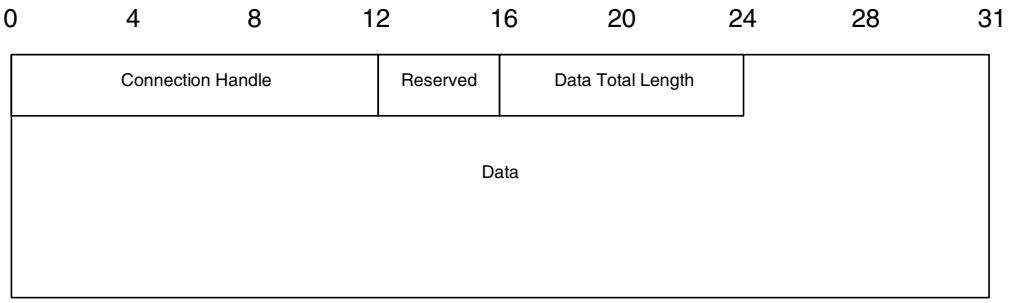


Figure 137—HCI SCO data packet

Connection_Handle: *Size: 12 Bits*

Value	Parameter description
0xXXX	Connection Handle to be used for transmitting a data packet or segment. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use). The first time the Host sends an HCI Data Packet with Broadcast_Flag set to 01b (active broadcast) or 10b (piconet broadcast) after a power-on or a reset, the value of the Connection_Handle parameter shall be a value which is not currently assigned by the Host Controller. The Host shall use different connection handles for active broadcast and piconet broadcast. The Host Controller must then continue to use the same connection handles for each type of broadcast until a reset is made. Note: The Host Controller shall not send a Connection Complete event containing a new Connection_Handle that it knows is used for broadcast. Note: In some situations, it may happen that the Host Controller sends a Connection Complete event before having interpreted a Broadcast packet received from the Host, and that the Connection_Handles of both Connection Complete event and HCI Data packet are the same. This conflict has to be avoided as follows: If a Connection Complete event is received containing one of the connection handles used for broadcast, the Host has to wait before sending any packets for the new connection until it receives a Number Of Completed Packets event indicating that there are no pending broadcast packets belonging to the connection handle. In addition, the Host shall change the Connection_Handle used for the corresponding type of broadcast to a Connection_Handle which is currently not assigned by the Host Controller. This Connection_Handle must then be used for all the following broadcasts of that type until a reset is performed or the same conflict situation happens again. However, this will occur very rarely. The Host Controller shall, in the above conflict case, be able to distinguish between the Broadcast message sent by the Host and the new connection made (this could be even a new SCO link) even though the connection handles are the same. For an HCI Data Packet sent from the Host Controller to the Host where the Broadcast_Flag is 01 or 10, the Connection_Handle parameter should contain the connection handle for the ACL connection to the master that sent the broadcast. Note: Connection handles used for Broadcast do not identify an ACL point-to-point connection, so they shall not be used in any command having a Connection_Handle parameter and they will not be returned in any event having a Connection_Handle parameter except the Number Of Completed Packets event.

Flags: *Size: 2 Bits*

The Flag Bits consist of the Packet_Boundary_Flag and Broadcast_Flag. The Packet_Boundary_Flag is located in bit 4 and bit 5, and the Broadcast_Flag is located in bit 6 and 7 in the second byte of the HCI ACL Data packet.

Packet_Boundary_Flag: *Size: 2 Bits*

Value	Parameter description
00	Reserved for future use.
01	Continuing fragment packet of Higher Layer Message.
10	First packet of Higher Layer Message (i.e., start of an L2CAP packet).
11	Reserved for future use.

Broadcast_Flag (in packet from Host to Host Controller): *Size: 2 Bits*

Value	Parameter description
00	No broadcast. Only point-to-point.
01	Active Broadcast: packet is sent to all active slaves (i.e., packet is usually not sent during park beacon slots), and it may be received by slaves in sniff or park mode. See note below.
10	Piconet Broadcast: packet is sent to all slaves and all slaves in park mode (i.e. packet is sent during park beacon slots if there are parked slaves), and it may be received by slaves in sniff mode. See note below.
11	Reserved for future use.

Broadcast_Flag (in packet from Host Controller to Host): *Size: 2 Bits*

Value	Parameter description
00	Point-to-point.
01	Packet received as a slave not in park mode (either Active Broadcast or Piconet Broadcast).
10	Packet received as a slave in park mode (Piconet Broadcast).
11	Reserved for future use.

NOTE—Active broadcast packets may be sent in park beacon slots for synchronization since a slave can synchronize to any baseband packet that is preceded by the proper channel access code.

Slaves in sniff mode may or may not receive an active or piconet broadcast packet depending on whether they happen to be listening at sniff slots when the packet is sent.

Data_Total_Length: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Length of data measured in bytes.

Connection_Handle: *Size: 12 Bits*

Value	Parameter description
0xXXX	Connection handle to be used to for transmitting a SCO data packet or segment. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

The Reserved Bits consist of four bits which are located from bit 4 to bit 7 in the second byte of the HCI SCO Data packet.

Reserved: *Size: 4 Bits*

Value	Parameter description
XXXX	Reserved for future use.

Data_Total_Length: *Size: 1 Byte*

Value	Parameter description
0xXX	Length of SCO data measured in bytes.

11.2.5 Link Control Commands

The Link Control commands allow the Host Controller to control connections to other Bluetooth devices. When the Link Control commands are used, the LM controls how the Bluetooth piconets and scatternets are established and maintained. These commands instruct the LM to create and modify link layer connections with Bluetooth remote devices, perform Inquiries of other Bluetooth devices in range, and other LMP commands. For the Link Control commands, the OGF is defined as 0x01.

Command	Command summary description
Inquiry	The Inquiry command will cause the Bluetooth device to enter Inquiry Mode. Inquiry Mode is used to discovery other nearby Bluetooth devices.
Inquiry_Cancel	The Inquiry_Cancel command will cause the Bluetooth device to stop the current Inquiry if the Bluetooth device is in Inquiry Mode.
Periodic_Inquiry_Mode	The Periodic_Inquiry_Mode command is used to configure the Bluetooth device to perform an automatic Inquiry based on a specified period range.
Exit_Periodic_Inquiry_Mode	The Exit_Periodic_Inquiry_Mode command is used to end the Periodic Inquiry mode when the local device is in Periodic Inquiry Mode.
Create_Connection	The Create_Connection command will cause the link manager to create an ACL connection to the Bluetooth device with the BD_ADDR specified by the command parameters.

Command	Command summary description
Disconnect	The Disconnect command is used to terminate an existing connection.
Add_SCO_Connection	The Add_SCO_Connection command will cause the link manager to create a SCO connection using the ACL connection specified by the Connection Handle command parameter.
Accept_Connection_Request	The Accept_Connection_Request command is used to accept a new incoming connection request.
Reject_Connection_Request	The Reject_Connection_Request command is used to decline a new incoming connection request.
Link_Key_Request_Reply	The Link_Key_Request_Reply command is used to reply to a Link Key Request event from the Host Controller, and specifies the Link Key stored on the Host to be used as the link key for the connection with the other Bluetooth device specified by BD_ADDR.
Link_Key_Request_Negative_Reply	The Link_Key_Request_Negative_Reply command is used to reply to a Link Key Request event from the Host Controller if the Host does not have a stored Link Key for the connection with the other Bluetooth Device specified by BD_ADDR.
PIN_Code_Request_Reply	The PIN_Code_Request_Reply command is used to reply to a PIN Code Request event from the Host Controller and specifies the PIN code to use for a connection.
PIN_Code_Request_Negative_Reply	The PIN_Code_Request_Negative_Reply command is used to reply to a PIN Code Request event from the Host Controller when the Host cannot specify a PIN code to use for a connection.
Change_Connection_Packet_Type	The Change_Connection_Packet_Type command is used to change which packet types can be used for a connection that is currently established.
Authentication_Requested	The Authentication_Requested command is used to establish authentication between the two devices associated with the specified Connection Handle.
Set_Connection_Encryption	The Set_Connection_Encryption command is used to enable and disable the link level encryption.
Change_Connection_Link_Key	The Change_Connection_Link_Key command is used to force both devices of a connection associated to the connection handle, to generate a new link key.

Command	Command summary description
Master_Link_Key	The Master_Link_Key command is used to force both devices of a connection associated to the connection handle to use the temporary link key of the Master device or the regular link keys.
Remote_Name_Request	The Remote_Name_Request command is used to obtain the user-friendly name of another Bluetooth device.
Read_Remote_Supported_Features	The Read_Remote_Supported_Features command requests a list of the supported features of a remote device.
Read_Remote_Version_Information	The Read_Remote_Version_Information command will read the values for the version information for the remote Bluetooth device.
Read_Clock_Offset	The Read_Clock_Offset command allows the Host to read the clock offset of remote devices.

11.2.5.1 Inquiry

Command	OCF	Command parameters	Return parameters
HCI_Inquiry	0x0001	LAP, Inquiry_Length, Num_Responses	—

Description:

This command will cause the Bluetooth device to enter Inquiry Mode. Inquiry Mode is used to discover other nearby Bluetooth devices. The LAP input parameter contains the LAP from which the inquiry access code shall be derived when the inquiry procedure is made. The Inquiry_Length parameter specifies the total duration of the Inquiry Mode and, when this time expires, Inquiry will be halted. The Num_Responses parameter specifies the number of responses that can be received before the Inquiry is halted. A Command Status event is sent from the Host Controller to the Host when the Inquiry command has been started by the Bluetooth device. When the Inquiry process is completed, the Host Controller will send an Inquiry Complete event to the Host indicating that the Inquiry has finished. The event parameters of Inquiry Complete event will have a summary of the result from the Inquiry process, which reports the number of nearby Bluetooth devices that responded. When a Bluetooth device responds to the Inquiry message, an Inquiry Result event will occur to notify the Host of the discovery.

A device that responds during an inquiry or inquiry period should always be reported to the Host in an Inquiry Result event if the device has not been reported earlier during the current inquiry or inquiry period and the device has not been filtered out using the command Set_Event_Filter. If the device has been reported earlier during the current inquiry or inquiry period, it may or may not be reported depending on the implementation (depending on if earlier results have been saved in the Host Controller and in that case how many responses that have been saved). It is recommended that the Host Controller tries to report a particular device only once during an inquiry or inquiry period.

Command Parameters:

LAP: *Size: 3 Bytes*

Value	Parameter description
0x9E8B00– 0X9E8B3F	This is the LAP from which the inquiry access code should be derived when the inquiry procedure is made; see 2.4.3.

Inquiry_Length: *Size: 1 Byte*

Value	Parameter description
N = 0xXX	Maximum amount of time specified before the Inquiry is halted. Size: 1 byte Range: 0x01–0x30 Time = *N* * 1.28 s Range: 1.28–61.44 s

Num_Responses: *Size: 1 Byte*

Value	Parameter description
0x00	Unlimited number of responses.
0xXX	Maximum number of responses from the Inquiry before the Inquiry is halted. Range: 0x01–0xFF

Return Parameters:

None.

Event(s) generated (unless masked away):

A Command Status event is sent from the Host Controller to the Host when the Host Controller has started the Inquiry process. An Inquiry Result event will be created for each Bluetooth device that responds to the Inquiry message. In addition, multiple Bluetooth devices which respond to the Inquire message may be combined into the same event. An Inquiry Complete event is generated when the Inquiry process has completed.

Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Inquiry Complete event will indicate that this command has been completed. No Inquiry Complete event will be generated for the canceled Inquiry process.

11.2.5.2 Inquiry_Cancel

Command	OCF	Command parameters	Return parameters
HCI_Inquiry_Cancel	0x0002		Status

Description:

This command will cause the Bluetooth device to stop the current Inquiry if the Bluetooth device is in Inquiry Mode. This command allows the Host to interrupt the Bluetooth device and request the Bluetooth device to perform a different task. The command should only be issued after the Inquiry command has been issued, a Command Status event has been received for the Inquiry command, and before the Inquiry Complete event occurs.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Inquiry_Cancel command succeeded.
0x01–0xFF	Inquiry_Cancel command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Inquiry Cancel command has completed, a Command Complete event will be generated. No Inquiry Complete event will be generated for the canceled Inquiry process.

11.2.5.3 Periodic_Inquiry_Mode

Command	OCF	Command parameters	Return parameters
HCI_Periodic_ Inquiry_Mode	0x0003	Max_Period_Length, Min_Period_Length, LAP, Inquiry_Length, Num_Responses	Status

Description:

The Periodic_Inquiry_Mode command is used to configure the Bluetooth device to enter the Periodic Inquiry Mode that performs an automatic Inquiry. Max_Period_Length and Min_Period_Length define the time range between two consecutive inquiries, from the beginning of an inquiry until the start of the next inquiry. The Host Controller will use this range to determine a new random time between two consecutive inquiries for each Inquiry. The LAP input parameter contains the LAP from which the inquiry access code shall be derived when the inquiry procedure is made. The Inquiry_Length parameter specifies the total duration of the Inquiry Mode and, when time expires, Inquiry will be halted. The Num_Responses parameter specifies the number of responses that can be received before the Inquiry is halted. This command is completed when the Inquiry process has been started by the Bluetooth device, and a Command Complete event is sent from the Host Controller to the Host. When each of the periodic Inquiry processes are completed, the Host Controller will send an Inquiry Complete event to the Host indicating that the latest periodic Inquiry process has finished. The event parameters of Inquiry Complete event will have a summary of the result from the previous Periodic Inquiry process, which reports the number of nearby Bluetooth devices that responded. When a Bluetooth device responds to the Inquiry message an Inquiry Result event will occur to notify the Host of the discovery. Note that Max_Period_Length > Min_Period_Length > Inquiry_Length.

A device which responds during an inquiry or inquiry period should always be reported to the Host in an Inquiry Result event if the device has not been reported earlier during the current inquiry or inquiry period and the device has not been filtered out using the command Set_Event_Filter. If the device has been reported earlier during the current inquiry or inquiry period, it may or may not be reported depending on the implementation (depending on if earlier results have been saved in the Host Controller and in that case how many responses that have been saved). It is recommended that the Host Controller tries to report a particular device only once during an inquiry or inquiry period.

Command Parameters:

Max_Period_Length: *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Maximum amount of time specified between consecutive inquiries. Size: 2 bytes Range: 0x03–0xFFFF Time = *N* * 1.28 s Range: 3.84–83884.8 s 0.0–23.3 h

Min_Period_Length: *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Minimum amount of time specified between consecutive inquiries. Size: 2 bytes Range: 0x02–0xFFFE Time = *N* * 1.28 s Range: 2.56–83883.52 s 0.0–23.3 h

LAP: *Size: 3 Bytes*

Value	Parameter description
0x9E8B00– 0X9E8B3F	This is the LAP from which the inquiry access code should be derived when the inquiry procedure is made, see 2.4.3.

Inquiry_Length: *Size: 1 Byte*

Value	Parameter description
N = 0xXX	Maximum amount of time specified before the Inquiry is halted. Size: 1 byte Range: 0x01–0x30 Time = *N* * 1.28 s Range: 1.28–61.44 s

Num_Responses: *Size: 1 Byte*

Value	Parameter description
0x00	Unlimited number of responses.
0xXX	Maximum number of responses from the Inquiry before the Inquiry is halted. Range: 0x01–0xFF

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Periodic Inquiry Mode command succeeded.
0x01–0xFF	Periodic Inquiry Mode command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

The Periodic Inquiry Mode begins when the Host Controller sends the Command Complete event for this command to the Host. An Inquiry Result event will be created for each Bluetooth device that responds to the Inquiry message. In addition, multiple Bluetooth devices that respond to the Inquiry message may be combined into the same event. An Inquiry Complete event is generated when each of the periodic Inquiry processes has completed. No Inquiry Complete event will be generated for the canceled Inquiry process.

11.2.5.4 Exit_Periodic_Inquiry_Mode

Command	OCF	Command Parameters	Return parameters
HCI_Exit_Periodic_Inquiry_Mode	0x0004		Status

Description:

The Exit Periodic Inquiry Mode command is used to end the Periodic Inquiry mode when the local device is in Periodic Inquiry Mode. If the local device is currently in an Inquiry process, the Inquiry process will be stopped directly and the Host Controller will no longer perform periodic inquiries until the Periodic Inquiry Mode command is reissued.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Exit Periodic Inquiry Mode command succeeded.
0x01–0xFF	Exit Periodic Inquiry Mode command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

A Command Complete event for this command will occur when the local device is no longer in Periodic Inquiry Mode. No Inquiry Complete event will be generated for the canceled Inquiry process.

11.2.5.5 Create_Connection

Command	OCF	Command parameters	Return Parameters
HCI_Create_Connection	0x0005	BD_ADDR, Packet_Type, Page_Scan_Repetition_Mode, Page_Scan_Mode, Clock_Offset, Allow_Role_Switch	

Description:

This command will cause the Link Manager to create a connection to the Bluetooth device with the BD_ADDR specified by the command parameters. This command causes the local Bluetooth device to begin the Page process to create a link level connection. The Link Manager will determine how the new ACL connection is established. This ACL connection is determined by the current state of the device, its piconet, and the state of the device to be connected. The Packet_Type command parameter specifies which packet types the Link Manager shall use for the ACL connection. The Link Manager must use only the packet type(s) specified by the Packet_Type command parameter for sending HCI ACL Data Packets. Multiple packet types may be specified for the Packet Type parameter by performing a bit-wise OR operation of the different packet types. The Link Manager may choose which packet type to be used from the list of acceptable packet types. The Page_Scan_Repetition_Mode and Page_Scan_Mode parameters specify the page scan modes supported by the remote device with the BD_ADDR. This is the information that was acquired during the inquiry process. The Clock_Offset parameter is the difference between its own clock and the clock of the remote device with BD_ADDR. Only bits 2 through 16 of the difference are used, and they are mapped to this parameter as bits 0 through 14, respectively. A Clock_Offset_Valid_Flag, located in bit 15 of the Clock_Offset parameter, is used to indicate if the Clock Offset is valid or not. A Connection handle for this connection is returned in the Connection Complete event (see below). The Allow_Role_Switch parameter specifies if the local device accepts or rejects the request of a master-slave role switch when the remote device requests it at the connection setup (in the Role parameter of the

Accept_Connection_Request command) (before the local Host Controller returns a Connection Complete event). For a definition of the different packet types, see Clause 8.

Note that at least one packet type shall be specified. The Host should enable as many packet types as possible for the Link Manager to perform efficiently. However, the Host shall not enable packet types that the local device does not support.

Command Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device to be connected.

Packet_Type: *Size: 2 Bytes*

Value	Parameter description
0x0001	Reserved for future use.
0x0002	Reserved for future use.
0x0004	Reserved for future use.
0x0008	DM1
0x0010	DH1
0x0020	Reserved for future use.
0x0040	Reserved for future use.
0x0080	Reserved for future use.
0x0100	Reserved for future use.
0x0200	Reserved for future use.
0x0400	DM3
0x0800	DH3
0x1000	Reserved for future use.
0x2000	Reserved for future use.
0x4000	DM5
0x8000	DH5

Page_Scan_Repetition_Mode: *Size: 1 Byte*

Value	Parameter description
0x00	R0
0x01	R1
0x02	R2
0x03–0xFF	Reserved.

Page_Scan_Mode: *Size: 1 Byte*

Value	Parameter description
0x00	Mandatory Page Scan Mode.
0x01	Optional Page Scan Mode I.
0x02	Optional Page Scan Mode II.
0x03	Optional Page Scan Mode III.
0x04–0xFF	Reserved.

Clock_Offset: *Size: 2 Bytes*

Bit format	Parameter description
Bit 14–0	Bit 16–2 of CLKslave-CLKmaster.
Bit 15	Clock_Offset_Valid_Flag Invalid Clock Offfset = 0 Valid Clock Offset = 1

Allow_Role_Switch: *Size: 1 Byte*

Value	Parameter description
0x00	The local device will be a master, and will not accept a master-slave switch requested by the remote device at the connection setup.
0x01	The local device may be a master, or may become a slave after accepting a master-slave switch requested by the remote device at the connection setup.
0x02–0xFF	Reserved for future use.

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Create Connection command, the Host Controller sends the Command Status event to the Host. In addition, when the LM determines the connection is established, the Host Controller, on both Bluetooth devices that form the connection, will send a Connection Complete event to each Host. The Connection Complete event contains the Connection Handle if this command is successful.

Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Connection Complete event will indicate that this command has been completed.

11.2.5.6 Disconnect

Command	OCF	Command parameters	Return parameters
HCI_Disconnect	0x0006	Connection_Handle, Reason	

Description:

The Disconnection command is used to terminate an existing connection. The Connection_Handle command parameter indicates which connection is to be disconnected. The Reason command parameter indicates the reason for ending the connection. The remote Bluetooth device will receive the Reason command parameter in the Disconnection Complete event. All SCO connections on a physical link should be disconnected before the ACL connection on the same physical connection is disconnected.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle for the connection being disconnected. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Reason: *Size: 1 Byte*

Value	Parameter description
0x05, 0x13–0x15, 0x1A, 0x29	Authentication Failure error code (0x05), Other End Terminated Connection error codes (0x13–0x15). Unsupported Remote Feature error code (0x1A) and Pairing with Unit Key Not Supported error code (0x29). See Table 109 for list of Error Codes.

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Disconnect command, it sends the Command Status event to the Host. The Disconnection Complete event will occur at each Host when the termination of the connection has completed, and indicates that this command has been completed.

Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Disconnection Complete event will indicate that this command has been completed.

11.2.5.7 Add_SCO_Connection

Command	OCF	Command parameters	Return parameters
HCI_Add_SCO_Connection	0x0007	Connection_Handle, Packet_Type	—

Description:

This command will cause the link manager to create a SCO connection using the ACL connection specified by the Connection_Handle command parameter. This command causes the local Bluetooth device to create a SCO connection. The Link Manager will determine how the new connection is established. This connection is determined by the current state of the device, its piconet, and the state of the device to be connected. The Packet_Type command parameter specifies which packet types the Link Manager should use for the connection. The Link Manager must only use the packet type(s) specified by the Packet_Type command parameter for sending HCI SCO Data Packets. Multiple packet types may be specified for the Packet_Type command parameter by performing a bitwise OR operation of the different packet types. The Link Manager may choose which packet type is to be used from the list of acceptable packet types. A Connection Handle for this connection is returned in the Connection Complete event (see below).

Note that an SCO connection can only be created when an ACL connection already exists and when it is not put in park mode. For a definition of the different packet types, see Clause 8.

Note also that at least one packet type shall be specified. The Host should enable as many packet types as possible for the Link Manager to perform efficiently. However, the Host shall not enable packet types that the local device does not support.

Command Parameters:

Connection_Handle *Size 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle for the ACL connection being used to create an SCO connection. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Packet_Type: *Size: 2 Bytes*

Value	Parameter description
0x0001	Reserved for future use.
0x0002	Reserved for future use.
0x0004	Reserved for future use.
0x0008	Reserved for future use.
0x0010	Reserved for future use.
0x0020	HV1
0x0040	HV2
0x0080	HV3
0x0100	Reserved for future use.
0x0200	Reserved for future use.
0x0400	Reserved for future use.
0x0800	Reserved for future use.
0x1000	Reserved for future use.
0x2000	Reserved for future use.
0x4000	Reserved for future use.
0x8000	Reserved for future use.

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Add_SCO_Connection command, it sends the Command Status event to the Host. In addition, when the LM determines the connection is established, the Host Controller, on both Bluetooth devices that form the connection, will send a Connection Complete event to each Host. The Connection Complete event contains the Connection Handle if this command is successful. Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Connection Complete event will indicate that this command has been completed.

11.2.5.8 Accept_Connection_Request

Command	OCF	Command parameters	Return parameters
HCI_Accept_Connection Request	0x0009	BD_ADDR, Role	—

Description:

The Accept_Connection_Request command is used to accept a new incoming connection request. The Accept_Connection_Request command shall only be issued after a Connection Request event has occurred. The Connection Request event will return the BD_ADDR of the device that is requesting the connection. This command will cause the Link Manager to create a connection to the Bluetooth device, with the BD_ADDR specified by the command parameters. The Link Manager will determine how the new connection will be established. This will be determined by the current state of the device, its piconet, and the state of the device to be connected. The Role command parameter allows the Host to specify if the Link Manager shall perform a Master-Slave switch, and become the Master for this connection. Also, the decision to accept a connection shall be completed before the connection accept timeout expires on the local Bluetooth Module.

Note that when accepting an SCO connection request, the Role parameter is not used and will be ignored by the Host Controller.

Command Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device to be connected.

Role: *Size: 1 Byte*

Value	Parameter description
0x00	Become the Master for this connection. The LM will perform the Master/ Slave switch.
0x01	Remain the Slave for this connection. The LM will **NOT** perform the Master/Slave switch.

Return Parameters:

None.

Event(s) generated (unless masked away):

The Accept_Connection_Request command will cause the Command Status event to be sent from the Host Controller when the Host Controller begins setting up the connection. In addition, when the Link Manager determines the connection is established, the Host Controllers on both Bluetooth devices that form the connection will send a Connection Complete event to each Host. The Connection Complete event contains the Connection Handle if this command is successful.

Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Connection Complete event will indicate that this command has been completed.

11.2.5.9 Reject_Connection_Request

Command	OCF	Command parameters	Return parameters
HCI_Reject_Connection_Request	0x000A	BD_ADDR, Reason	—

Description:

The Reject_Connection_Request command is used to decline a new incoming connection request. The Reject_Connection_Request command shall only be called after a Connection Request event has occurred. The Connection Request event will return the BD_ADDR of the device that is requesting the connection. The Reason command parameter will be returned to the connecting device in the Status parameter of the Connection Complete event returned to the Host of the connection device, to indicate why the connection was declined.

Command Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device to reject the connection from.

Reason: *Size: 1 Byte*

Value	Parameter description
0x0D-0x0F	Host Reject Error Code. See Table 109 for list of Error Codes and descriptions.

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Reject_Connection_Request command, the Host Controller sends the Command Status event to the Host. A Connection Complete event will then be sent, both to the local Host and to the Host of the device attempting to make the connection. The Status parameter of the Connection Complete event, which is sent to the Host of the device attempting to make the connection, will contain the Reason command parameter from this command.

Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Connection Complete event will indicate that this command has been completed.

11.2.5.10 Link_Key_Request_Reply

Command	OCF	Command parameters	Return parameters
HCI_Link_Key_Request_Reply	0x000B	BD_ADDR, Link_Key	Status, BD_ADDR

Description:

The Link_Key_Request_Reply command is used to reply to a Link Key Request event from the Host Controller, and specifies the Link Key stored on the Host to be used as the link key for the connection with the other Bluetooth Device specified by BD_ADDR. The Link Key Request event will be generated when the Host Controller needs a Link Key for a connection.

When the Host Controller generates a Link Key Request event in order for the local Link Manager to respond to the request from the remote Link Manager (as a result of a Create_Connection or Authentication_Requested command from the remote Host), the local Host shall respond with either a Link_Key_Request_Reply or Link_Key_Request_Negative_Reply command before the remote Link Manager detects LMP response timeout (see Clause 9).

Command Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device of which the Link Key is for.

Link_Key: *Size: 16 Bytes*

Value	Parameter description
0xXXXXXXXXXXXXXXXXXXXXXXXXXXXX	Link Key for the associated BD_ADDR.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Link_Key_Request_Reply command succeeded.
0x01–0xFF	Link_Key_Request_Reply command failed. See Table 109 for list of Error Codes.

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device of which the Link Key request reply has completed.

Event(s) generated (unless masked away):

The Link_Key_Request_Reply command will cause a Command Complete event to be generated.

11.2.5.11 Link_Key_Request_Negative_Reply

Command	OCF	Command parameters	Return parameters
HCI_Link_Key_Request_Negative_Reply	0x000C	BD_ADDR	Status, BD_ADDR

Description:

The Link_Key_Request_Negative_Reply command is used to reply to a Link Key Request event from the Host Controller if the Host does not have a stored Link Key for the connection with the other Bluetooth Device specified by BD_ADDR. The Link Key Request event will be generated when the Host Controller needs a Link Key for a connection.

When the Host Controller generates a Link Key Request event in order for the local Link Manager to respond to the request from the remote Link Manager (as a result of a Create_Connection or Authentication_Requested command from the remote Host), the local Host shall respond with either a Link_Key_Request_Reply or Link_Key_Request_Negative_Reply command before the remote Link Manager detects LMP response timeout (see Clause 9).

Command Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device which the Link Key is for.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Link_Key_Request_Negative_Reply command succeeded.
0x01–0xFF	Link_Key_Request_Negative_Reply command failed. See Table 109 for list of Error Codes.

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device which the Link Key request negative reply has completed.

Event(s) generated (unless masked away):

The Link_Key_Request_Negative_Reply command will cause a Command Complete event to be generated.

11.2.5.12 PIN_Code_Request_Reply

Command	OCF	Command parameters	Return parameters
HCI_PIN_Code_Request_Reply	0x000D	BD_ADDR, PIN_Code_Length, PIN_Code	Status, BD_ADDR

Description:

The PIN_Code_Request_Reply command is used to reply to a PIN Code request event from the Host Controller, and specifies the PIN code to use for a connection. The PIN Code Request event will be generated when a connection with remote initiating device has requested pairing.

When the Host Controller generates a PIN Code Request event in order for the local Link Manager to respond to the request from the remote Link Manager (as a result of a Create_Connection or Authentication_Requested command from the remote Host), the local Host shall respond with either a PIN_Code_Request_Reply or PIN_Code_Request_Negative_Reply command before the remote Link Manager detects LMP response timeout (see Clause 9).

Command Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device which the PIN code is for.

PIN_Code_Length: *Size: 1 Byte*

Value	Parameter description
0xXX	The PIN code length specifics the length, in bytes, of the PIN code to be used. Range: 0x01–0x10

PIN_Code: *Size: 16 Bytes*

Value	Parameter description
0xXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	PIN code for the device that is to be connected. The Host should insure that strong PIN Codes are used. PIN Codes can be up to a maximum of 128 bits. Note: the PIN_Code Parameter is a string parameter. Endianess does therefore not apply to the PIN_Code Parameter. The first byte of the PIN code should be transmitted first.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	PIN_Code_Request_Reply command succeeded.
0x01-0xFF	PIN_Code_Request_Reply command failed. See Table 109 for list of Error Codes.

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device which the PIN Code request reply has completed.

Event(s) generated (unless masked away):

The PIN_Code_Request_Reply command will cause a Command Complete event to be generated.

11.2.5.13 PIN_Code_Request_Negative_Reply

Command	OCF	Command parameters	Return parameters
HCI_PIN_Code_ Request_Negative Reply	0x000E	BD_ADDR	Status, BD_ADDR

Description:

The PIN_Code_Request_Negative_Reply command is used to reply to a PIN Code request event from the Host Controller when the Host cannot specify a PIN code to use for a connection. This command will cause the pair request with remote device to fail.

When the Host Controller generates a PIN Code Request event in order for the local Link Manager to respond to the request from the remote Link Manager (as a result of a Create_Connection or Authentication_Requested command from the remote Host), the local Host shall respond with either a PIN_Code_Request_Reply or PIN_Code_Request_Negative_Reply command before the remote Link Manager detects LMP response timeout. (See Clause 9.)

Command Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device which this command is responding to.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	PIN_Code_Request_Negative_Reply command succeeded.
0x01–0xFF	PIN_Code_Request_Negative_Reply command failed. See Table 109 for list of Error Codes.

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device which the PIN Code request negative reply has completed.

Event(s) generated (unless masked away):

The PIN_Code_Request_Negative_Reply command will cause a Command Complete event to be generated.

11.2.5.14 Change_Connection_Packet_Type

Command	OCF	Command parameters	Return parameters
HCI_Change_Connection_Packet_Type	0x000F	Connection_Handle, Packet_Type	

Description:

The Change_Connection_Packet_Type command is used to change which packet types can be used for a connection that is currently established. This allows current connections to be dynamically modified to support different types of user data. The Packet_Type command parameter specifies which packet types the Link Manager can use for the connection. The Link Manager must only use the packet type(s) specified by the Packet_Type command parameter for sending HCI Data Packets. The interpretation of the value for the Packet_Type command parameter will depend on the Link_Type command parameter returned in the Connection Complete event at the connection setup. Multiple packet types may be specified for the Packet_Type command parameter by bitwise OR operation of the different packet types. For a definition of the different packet types, see Clause 8.

Note that at least one packet type shall be specified. The Host should enable as many packet types as possible for the Link Manager to perform efficiently. However, the Host shall not enable packet types that the local device does not support.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to for transmitting and receiving voice or data. Returned from creating a connection. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Packet_Type: *Size: 2 Bytes*

For ACL Link_Type

Value	Parameter description
0x0001	Reserved for future use.
0x0002	Reserved for future use.
0x0004	Reserved for future use.
0x0008	DM1
0x0010	DH1
0x0020	Reserved for future use.
0x0040	Reserved for future use.
0x0080	Reserved for future use.
0x0100	Reserved for future use.
0x0200	Reserved for future use.
0x0400	DM3
0x0800	DH3
0x1000	Reserved for future use.
0x2000	Reserved for future use.
0x4000	DM5
0x8000	DH5

For SCO Link Type

Value	Parameter description
0x0001	Reserved for future use.
0x0002	Reserved for future use.
0x0004	Reserved for future use.
0x0008	Reserved for future use.
0x0010	Reserved for future use.
0x0020	HV1
0x0040	HV2
0x0080	HV3
0x0100	Reserved for future use.
0x0200	Reserved for future use.
0x0400	Reserved for future use.
0x0800	Reserved for future use.
0x1000	Reserved for future use.
0x2000	Reserved for future use.
0x4000	Reserved for future use.
0x8000	Reserved for future use.

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Change Connection Packet Type command, the Host Controller sends the Command Status event to the Host. In addition, when the Link Manager determines the packet type has been changed for the connection, the Host Controller on the local device will send a Connection Packet Type Changed event to the Host. This will be done at the local side only. Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Connection Packet Type Changed event will indicate that this command has been completed.

11.2.5.15 Authentication_Requested

Command	OCF	Command parameters	Return parameters
HCI_Authentication_ Requested	0x0011	Connection_Handle	—

Description:

The Authentication_Requested command is used to try to authenticate the remote device associated with the specified Connection Handle. The Host shall not issue the Authentication_Requested command with a Connection_Handle corresponding to an encrypted link. On an authentication failure, the Host Controller or Link Manager shall not automatically detach the link. The Host is responsible for issuing a Disconnect command to terminate the link if the action is appropriate. Note that the Connection_Handle command parameter is used to identify the other Bluetooth device, which forms the connection. The Connection Handle should be a Connection Handle for an ACL connection.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to set up authentication for all Connection Handles with the same Bluetooth device end-point as the specified Connection Handle. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Authentication_Requested command, it sends the Command Status event to the Host. The Authentication Complete event will occur when the authentication has been completed for the connection and is indication that this command has been completed. Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Authentication Complete event will indicate that this command has been completed.

Note that when the local or remote Host Controller does not have a link key for the specified Connection_Handle, it will request the link key from its Host, before the local Host finally receives the Authentication Complete event.

11.2.5.16 Set_Connection_Encryption

Command	OCF	Command parameters	Return parameters
HCI_Set_Connection_ Encryption	0x0013	Connection_Handle, Encryption_Enable	—

Description:

The Set_Connection_Encryption command is used to enable and disable the link level encryption. Note that the Connection_Handle command parameter is used to identify the other Bluetooth device which forms the connection. The Connection Handle should be a Connection Handle for an ACL connection. While the encryption is being changed, all ACL traffic shall be turned off for all Connection Handles associated with the remote device.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to enable/disable the link layer encryption for all Connection Handles with the same Bluetooth device end-point as the specified Connection Handle. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Encryption_Enable: *Size: 1 Byte*

Value	Parameter description
0x00	Turn Link Level Encryption OFF.
0x01	Turn Link Level Encryption ON.

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Set_Connection_Encryption command, the Host Controller sends the Command Status event to the Host. When the Link Manager has completed enabling/disabling encryption for the connection, the Host Controller on the local Bluetooth device will send an Encryption Change event to the Host, and the Host Controller on the remote device will also generate an Encryption Change event. Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Encryption Change event will indicate that this command has been completed.

11.2.5.17 Change_Connection_Link_Key

Command	OCF	Command parameters	Return parameters
HCI_Change_Connection_ Link_Key	0x0015	Connection_Handle	—

Description:

The Change_Connection_Link_Key command is used to force both devices of a connection associated with the connection handle to generate a new link key. The link key is used for authentication and encryption of connections.

Note that the Connection_Handle command parameter is used to identify the other Bluetooth device forming the connection. The Connection Handle should be a Connection Handle for an ACL connection. If the connection encryption is enabled, and the temporary link key is currently used, then the Bluetooth master device will automatically restart the encryption.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle used to identify a connection. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Change_Connection_Link_Key command, the Host Controller sends the Command Status event to the Host. When the Link Manager has changed the Link Key for the connection, the Host Controller on the local Bluetooth device will send a Link Key Notification event and a Change Connection Link Key Complete event to the Host, and the Host Controller on the remote device will also generate a Link Key Notification event. The Link Key Notification event indicates that a new connection link key is valid for the connection.

Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Change Connection Link Key Complete event will indicate that this command has been completed.

11.2.5.18 Master_Link_Key

Command	OCF	Command parameters	Return parameters
HCI_Master_Link_Key	0x0017	Key_Flag	—

Description:

The Master Link Key command is used to force the device that is master of the piconet to use the temporary link key of the master device, or the semipermanent link keys. The temporary link key is used for encryption of broadcast messages within a piconet, and the semipermanent link keys are used for private encrypted point-to-point communication. The Key_Flag command parameter is used to indicate which Link Key (temporary link key of the Master, or the semipermanent link keys) shall be used.

Command Parameters:

Key_Flag: *Size: 1 Byte*

Value	Parameter description
0x00	Use semipermanent Link Keys.
0x01	Use Temporary Link Key.

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Master_Link_Key command, the Host Controller sends the Command Status event to the Host. When the Link Manager has changed link key, the Host Controller on both the local and the remote device will send a Master Link Key Complete event to the Host. The Connection Handle on the master side will be a Connection Handle for one of the existing connections to a slave. On the slave side, the Connection Handle will be a Connection Handle to the initiating master.

The Master Link Key Complete event contains the status of this command.

Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Master Link Key Complete event will indicate that this command has been completed.

11.2.5.19 Remote_Name_Request

Command	OCF	Command parameters	Return Parameters
HCI_Remote_Name_Request	0x0019	BD_ADDR, Page_Scan_Repetition_Mode, Page_Scan_Mode, Clock_Offset	—

Description:

The Remote_Name_Request command is used to obtain the user-friendly name of another Bluetooth device. The user-friendly name is used to enable the user to distinguish one Bluetooth device from another. The BD_ADDR command parameter is used to identify the device for which the user-friendly name is to be obtained. The Page_Scan_Repetition_Mode and Page_Scan_Mode command parameters specify the page scan modes supported by the remote device with the BD_ADDR. This is the information that was acquired during the inquiry process. The Clock_Offset parameter is the difference between its own clock and the clock of the remote device with BD_ADDR. Only bits 2 through 16 of the difference are used and they are mapped to this parameter as bits 0 through 14, respectively. A Clock_Offset_Valid_Flag, located in bit 15 of the Clock_Offset command parameter, is used to indicate if the Clock Offset is valid or not.

Note that if no connection exists between the local device and the device corresponding to the BD_ADDR, a temporary link layer connection will be established to obtain the name of the remote device.

Command Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR for the device whose name is requested.

Page_Scan_Repetition_Mode: *Size: 1 Byte*

Value	Parameter description
0x00	R0
0x01	R1
0x02	R2
0x03–0xFF	Reserved.

Page_Scan_Mode: *Size: 1 Byte*

Value	Parameter description
0x00	Mandatory Page Scan Mode.
0x01	Optional Page Scan Mode I.
0x02	Optional Page Scan Mode II.
0x03	Optional Page Scan Mode III.
0x04–0xFF	Reserved.

Clock_Offset: *Size: 2 Bytes*

Bit format	Parameter description
Bit 14–0	Bit 16–2 of CLKslave-CLKmaster.
Bit 15	Clock_Offset_Valid_Flag Invalid Clock Offfset = 0 Valid Clock Offset = 1

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Remote_Name_Request command, the Host Controller sends the Command Status event to the Host. When the Link Manager has completed the LMP messages to obtain the remote name, the Host Controller on the local Bluetooth device will send a Remote Name Request Complete event to the Host. Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, only the Remote Name Request Complete event will indicate that this command has been completed.

11.2.5.20 Read_Remote_Supported_Features

Command	OCF	Command parameters	Return parameters
HCI_Read_Remote_Supported_Features	0x001B	Connection_Handle	—

Description:

This command requests a list of the supported features for the remote device identified by the Connection_Handle parameter. The Connection_Handle shall be a Connection_Handle for an ACL connection. The Read Remote Supported Features Complete event will return a list of the LMP features. For details, see Clause 9.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Specifies which Connection Handle's LMP-supported features list to get. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Read_Remote_Supported_Features command, the Host Controller sends the Command Status event to the Host. When the Link Manager has completed the LMP messages to determine the remote features, the Host Controller on the local Bluetooth device will send a Read Remote Supported Features Complete event to the Host. The Read Remote Supported Features Complete event contains the status of this command, and parameters describing the supported features of the remote device. Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Read Remote Supported Features Complete event will indicate that this command has been completed.

11.2.5.21 Read_Remote_Version_Information

Command	OCF	Command parameters	Return parameters
HCI_Read_Remote_Version_ Information	0x001D	Connection_Handle	—

Description:

This command will obtain the values for the version information for the remote Bluetooth device identified by the Connection_Handle parameter. The Connection_Handle shall be a Connection_Handle for an ACL connection.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Specifies which Connection Handle's version information to get. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Read_Remote_Version_Information command, the Host Controller sends the Command Status event to the Host. When the Link Manager has completed the LMP messages to determine the remote version information, the Host Controller on the local Bluetooth device will send a Read Remote Version Information Complete event to the Host. The Read Remote Version Information Complete event contains the status of this command, and parameters describing the version and subversion of the LMP used by the remote device.

Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Read Remote Version Information Complete event will indicate that this command has been completed.

11.2.5.22 Read_Clock_Offset

Command	OCF	Command parameters	Return parameters
HCI_Read_Clock_Offset	0x001F	Connection_Handle	—

Description:

Both the System Clock and the clock offset to a remote device are used to determine what hopping frequency is used by a remote device for page scan. This command allows the Host to read clock offset to remote devices. The clock offset can be used to speed up the paging procedure when the local device tries to establish a connection to a remote device, for example, when the local Host has issued Create_Connection or Remote_Name_Request. The Connection_Handle shall be a Connection_Handle for an ACL connection.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 bits meaningful)*

Value	Parameter description
0xXXXX	Specifies which Connection Handle's Clock Offset parameter is returned. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Read_Clock_Offset command, the Host Controller sends the Command Status event to the Host. If this command was requested at the master and the Link Manager has completed the LMP messages to obtain the Clock Offset information, the Host Controller on the local Bluetooth device will send a Read Clock Offset Complete event to the Host. Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, only the Read Clock Offset Complete event will indicate that this command has been completed. If the command is requested at the slave, the LM will immediately send a Command Status event and a Read Clock Offset Complete event to the Host, without an exchange of LMP PDU.

11.2.6 Link Policy Commands

The Link Policy Commands provide methods for the Host to affect how the Link Manager manages the piconet. When Link Policy Commands are used, the LM still controls how Bluetooth piconets and scatternets are established and maintained, depending on adjustable policy parameters. These policy commands modify the Link Manager behavior that can result in changes to the link layer connections with Bluetooth remote devices.

Note that only one ACL connection can exist between two Bluetooth Devices, and therefore there can only be one ACL HCI Connection Handle for each physical link layer Connection. The Bluetooth Host Controller provides policy adjustment mechanisms to provide support for a number of different policies. This capability allows one Bluetooth module to be used to support many different usage models, and the same Bluetooth module can be incorporated in many different types of Bluetooth devices. For the Link Policy Commands, the OGF is defined as 0x02.

Command	Command summary description
Hold_Mode	The Hold_Mode command is used to alter the behavior of the LM and have the LM place the local or remote device into the hold mode.
Sniff_Mode	The Sniff_Mode command is used to alter the behavior of the LM and have the LM place the local or remote device into the sniff mode.
Exit_Sniff_Mode	The Exit_Sniff_Mode command is used to end the sniff mode for a connection handle which is currently in sniff mode.
Park_Mode	The Park_Mode command is used to alter the behavior of the LM and have the LM place the local or remote device into the Park mode.
Exit_Park_Mode	The Exit_Park_Mode command is used to switch the Bluetooth device from park mode back to active mode.
QoS_Setup	The QoS_Setup command is used to specify Quality of Service parameters for a connection handle.

Command	Command summary description
Role_Discovery	The Role_Discovery command is used for a Bluetooth device to determine which role the device is performing for a particular Connection Handle.
Switch_Role	The Switch_Role command is used for a Bluetooth device switch the current role the device is performing for a particular connection with the specified Bluetooth device
Read_Link_Policy_Settings	The Read_Link_Policy_Settings command will read the Link Policy settings for the specified Connection Handle. The Link Policy settings allow the Host to specify which Link Modes the LM can use for the specified Connection Handle.
Write_Link_Policy_Settings	The Write_Link_Policy_Settings command will write the Link Policy settings for the specified Connection Handle. The Link Policy settings allow the Host to specify which Link Modes the LM can use for the specified Connection Handle.

11.2.6.1 Hold_Mode

Command	OCF	Command parameters	Return parameters
HCI_Hold_Mode	0x0001	Connection_Handle, Hold_Mode_Max_Interval, Hold_Mode_Min_Interval	—

Description:

The Hold_Mode command is used to alter the behavior of the Link Manager, and have it place the ACL baseband connection associated by the specified Connection Handle into the hold mode. The Hold_Mode_Max_Interval and Hold_Mode_Min_Interval command parameters specify the length of time the Host wants to put the connection into the hold mode. The local and remote devices will negotiate the length in the hold mode. The Hold_Mode_Max_Interval parameter is used to specify the maximum length of the Hold interval for which the Host may actually enter into the hold mode after negotiation with the remote device. The Hold interval defines the amount of time between when the Hold Mode begins and when the Hold Mode is completed. The Hold_Mode_Min_Interval parameter is used to specify the minimum length of the Hold interval for which the Host may actually enter into the hold mode after the negotiation with the remote device. Therefore the Hold_Mode_Min_Interval cannot be greater than the Hold_Mode_Max_Interval. The Host Controller will return the actual Hold interval in the Interval parameter of the Mode Change event, if the command is successful. This command enables the Host to support a low-power policy for itself or several other Bluetooth devices, and allows the devices to enter Inquiry Scan, Page Scan, and a number of other possible actions.

Note that the connection handle cannot be of the SCO link type If the Host sends data to the Host Controller with a Connection_Handle corresponding to a connection in hold mode, the Host Controller will keep the data in its buffers until either the data can be transmitted (the hold mode has ended) or a flush, a flush

timeout or a disconnection occurs. This is valid even if the Host has not yet been notified of the hold mode through a Mode Change event when it sends the data.

Note that the above is not valid for an HCI Data Packet sent from the Host to the Host Controller on the master side where the Connection_Handle is a Connection_Handle used for broadcast and the Broadcast_Flag is set to Active Broadcast or Piconet Broadcast. The broadcast data will then never be received by slaves in hold mode.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to identify a connection. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Hold_Mode_Max_Interval: *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Maximum acceptable number of Baseband slots to wait in Hold Mode. Time Length of the Hold = *N* * 0.625 ms (1 Baseband slot) Range for *N*: 0x0001–0xFFFF Time Range: 0.625 ms–40.9 s

Hold_Mode_Min_Interval: *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Minimum acceptable number of Baseband slots to wait in Hold Mode. Time Length of the Hold = *N* * 0.625 msec (1 Baseband slot) Range for *N*: 0x0001–0xFFFF Time Range: 0.625 ms–40.9 s

Return Parameters:

None**.**

Event(s) generated (unless masked away):

The Host Controller sends the Command Status event for this command to the Host when it has received the Hold_Mode command. The Mode Change event will occur when the Hold Mode has started and the Mode Change event will occur again when the Hold Mode has completed for the specified connection handle. The Mode Change event signaling the end of the Hold Mode is an estimation of the hold mode ending if the event is for a remote Bluetooth device.

Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, only the Mode Change event will indicate that this command has been completed. If an error occurs after the Command Status event has occurred, then the status in the Mode Change event will indicate the error.

11.2.6.2 Sniff_Mode

Command	OCF	Command parameters	Return parameters
HCI_Sniff_Mode	0x0003	Connection_Handle, Sniff_Max_Interval, Sniff_Min_Interval, Sniff_Attempt, Sniff_Timeout	—

Description:

The Sniff Mode command is used to alter the behavior of the Link Manager and have it place the ACL base-band connection associated with the specified Connection Handle into the sniff mode. The Connection_Handle command parameter is used to identify which ACL link connection is to be placed in sniff mode. The Sniff_Max_Interval and Sniff_Min_Interval command parameters are used to specify the requested acceptable maximum and minimum periods in the Sniff Mode. The Sniff_Min_Interval cannot be greater than the Sniff_Max_Interval. The sniff interval defines the amount of time between each consecutive sniff period. The Host Controller will return the actual sniff interval in the Interval parameter of the Mode Change event, if the command is successful. For a description of the meaning of the Sniff_Attempt and Sniff_Timeout parameters, see 8.10.8.2. Sniff_Attempt is there called $N_{\text{sniff attempt}}$ and Sniff_Timeout is called $N_{\text{sniff timeout}}$. This command enables the Host to support a low-power policy for itself or several other Bluetooth devices, and allows the devices to enter Inquiry Scan, Page Scan, and a number of other possible actions.

Note that in addition, the connection handle cannot be one of SCO link type. If the Host sends data to the Host Controller with a Connection_Handle corresponding to a connection in sniff mode, the Host Controller will keep the data in its buffers until either the data can be transmitted or a flush, a flush timeout or a disconnection occurs. This is valid even if the Host has not yet been notified of the sniff mode through a Mode Change event when it sends the data. Note that it is possible for the master to transmit data to a slave without exiting sniff mode (see description in 10.8.2).

Note that the above is not valid for an HCI Data Packet sent from the Host to the Host Controller on the master side where the Connection_Handle is a Connection_Handle used for broadcast and the Broadcast_Flag is set to Active Broadcast or Piconet Broadcast. In that case, the broadcast data will only be received by a slave in sniff mode if that slave happens to listen to the master when the broadcast is made.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to identify a connection. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Sniff_Max_Interval: *Size: 2 Byte*

Value	Parameter description
N = 0xXXXX	Maximum acceptable number of Baseband slots between each sniff period. (Sniff_Max_Interval >= Sniff_Min_Interval) Length = *N* * 0.625 ms (1 Baseband slot) Range for *N*: 0x0001–0xFFFF Time Range: 0.625 ms–40.9 s

Sniff_Min_Interval: *Size: 2 Byte*

Value	Parameter description
N = 0xXXXX	Minimum acceptable number of Baseband slots between each sniff period. (Sniff_Max_Interval >= Sniff_Min_Interval) Length = *N* * 0.625 ms (1 Baseband slot) Range for *N*: 0x0001–0xFFFF Time Range: 0.625 ms–40.9 s

Sniff_Attempt: *Size: 2 Byte*

Value	Parameter description
N = 0xXXXX	Number of Baseband receive slots for sniff attempt. Length = (2 * *N* – 1)* 0.625 ms Range for *N*: 0x0001–0x7FFF Time Range: 0.625 ms–40.9 s

Sniff_Timeout: *Size: 2 Byte*

Value	Parameter description
N = 0xXXXX	Number of Baseband recieve slots for sniff timeout. Length = (2 **N* – 1) * 0.625 ms if *N* > 0, Length = 0 if *N* = 0 Range for *N*: 0x0000–0x7FFF Time Range: 0 ms–40.9 s

Return Parameters:

None.

Event(s) generated (unless masked away):

The Host Controller sends the Command Status event for this command to the Host when it has received the Sniff_Mode command. The Mode Change event will occur when the Sniff Mode has started for the specified connection handle. Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, only the Mode Change event will indicate that this

command has been completed. If an error occurs after the Command Status event has occurred, then the status in the Mode Change event will indicate the error.

11.2.6.3 Exit_Sniff_Mode

Command	OCF	Command parameters	Return parameters
HCI_Exit_Sniff_Mode	0x0004	Connection_Handle	—

Description:

The Exit_Sniff_Mode command is used to end the sniff mode for a connection handle, which is currently in sniff mode. The Link Manager will determine and issue the appropriate LMP commands to remove the sniff mode for the associated Connection Handle. Note that in addition, the connection handle cannot be one of SCO link type.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to identify a connection. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Return Parameters:

None.

Event(s) generated (unless masked away):

A Command Status event for this command will occur when Host Controller has received the Exit_Sniff_Mode command. The Mode Change event will occur when the Sniff Mode has ended for the specified connection handle. Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, only the Mode Change event will indicate that this command has been completed.

11.2.6.4 Park_Mode

Command	OCF	Command parameters	Return parameters
HCI_Park_Mode	0x0005	Connection_Handle, Beacon_Max_Interval, Beacon_Min_Interval	—

Description:

The Park Mode command is used to alter the behavior of the Link Manager, and have the LM place the baseband connection associated by the specified Connection Handle into the Park mode. The Connection_Handle command parameter is used to identify which connection is to be placed in Park mode. The Connection_Handle shall be a Connection_Handle for an ACL connection. The Beacon Interval

command parameters specify the acceptable length of the interval between beacons. However, the remote device may request shorter interval. The Beacon_Max_Interval parameter specifies the acceptable longest length of the interval between beacons. The Beacon_Min_Interval parameter specifies the acceptable shortest length of the interval between beacons. Therefore, the Beacon Min Interval cannot be greater than the Beacon Max Interval. The Host Controller will return the actual Beacon interval in the Interval parameter of the Mode Change event, if the command is successful. This command enables the Host to support a low-power policy for itself or several other Bluetooth devices, allows the devices to enter Inquiry Scan, Page Scan, provides support for large number of Bluetooth Devices in a single piconet, and a number of other possible activities.

Note that when the Host issues the Park_Mode command, no Connection Handles for SCO connections are allowed to exist to the remote device that is identified by the Connection_Handle parameter. If one or more Connection Handles for SCO connections exist to that device, depending on the implementation, a Command Status event or a Mode Change event (following a Command Status event where Status=0x00) will be returned with the error code 0x0C “Command Disallowed”. If the Host sends data to the Host Controller with a Connection_Handle corresponding to a parked connection, the Host Controller will keep the data in its buffers until either the data can be transmitted (after unpark) or a flush, a flush timeout or a disconnection occurs. This is valid even if the Host has not yet been notified of the park mode through a Mode Change event when it sends the data.

Note that the above is not valid for an HCI Data Packet sent from the Host to the Host Controller on the master side where the Connection_Handle is a Connection_Handle used for Piconet Broadcast and the Broadcast_Flag is set to Piconet Broadcast. In that case, slaves in park mode will also receive the broadcast data. (If the Broadcast_Flag is set to Active Broadcast, the broadcast data will usually not be received by slaves in park mode.) It is possible for the Host Controller to do an automatic unpark to transmit data and then park the connection again depending on the value of the Link_Policy_Settings parameter (see Write_Link_Policy_Settings) and depending on whether the implementation supports this or not (optional feature). The optional feature of automatic unpark/park can also be used for link supervision. Whether Mode Change events are returned or not at automatic unpark/park if this is implemented, is vendor specific. This could be controlled by a vendor-specific HCI command.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to identify a connection. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Beacon_Max_Interval: *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Maximum acceptable number of Baseband slots between consecutive beacons. Interval Length = *N* * 0.625 ms (1 Baseband slot) Range for *N*: 0x0001–0xFFFF Time Range: 0.625 ms–40.9 s

Beacon_Min_Interval *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Minimum acceptable number of Baseband slots between consecutive beacons Interval Length = *N* * 0.625 ms(1 Baseband slot) Range for *N*: 0x0001–0xFFFF Time Range: 0.625 ms–40.9 s

Return Parameters:

None.

Event(s) generated (unless masked away):

The Host Controller sends the Command Status event for this command to the Host when it has received the Park_Mode command. The Mode Change event will occur when the Park Mode has started for the specified connection handle. Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, only the Mode Change event will indicate that this command has been completed. If an error occurs after the Command Status event has occurred, then the status in the Mode Change event will indicate the error.

11.2.6.5 Exit_Park_Mode

Command	OCF	Command parameters	Return parameters
HCI_Exit_Park_Mode	0x0006	Connection_Handle	—

Description:

The Exit_Park_Mode command is used to switch the Bluetooth device from park mode back to active mode. This command may only be issued when the device associated with the specified Connection Handle is in Park Mode. The Connection_Handle shall be a Connection_Handle for an ACL connection. This function does not complete immediately.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to identify a connection. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Return Parameters:

None.

Event(s) generated (unless masked away):

A Command Status event for this command will occur when the Host Controller has received the Exit_Park_Mode command. The Mode Change event will occur when the Park Mode has ended for the specified connection handle. Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, only the Mode Change event will indicate that this command has been completed.

11.2.6.6 QoS_Setup

Command	OCF	Command parameters	Return parameters
HCI_QoS_Setup	0x0007	Connection_Handle, Flags, Service_Type, Token_Rate, Peak_Bandwidth, Latency, Delay_Variation	—

Description:

The QoS_Setup command is used to specify Quality of Service parameters for a connection handle. The Connection_Handle shall be a Connection_Handle for an ACL connection. These QoS parameter are the same parameters as L2CAP QoS. For more detail see Clause 10, “Logical Link Control and Adaptation Protocol Specification”. This allows the Link Manager to have all of the information about what the Host is requesting for each connection. The LM will determine if the QoS parameters can be met. Bluetooth devices that are both slaves and masters can use this command. When a device is a slave, this command will trigger an LMP request to the master to provide the slave with the specified QoS as determined by the LM. When a device is a master, this command is used to request a slave device to accept the specified QoS as determined by the LM of the master. The Connection_Handle command parameter is used to identify for which connection the QoS request is requested.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to identify which connection for the QoS Setup. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Flags: *Size: 1 Byte*

Value	Parameter description
0x00–0xFF	Reserved for future use.

Service_Type: *Size: 1 Byte*

Value	Parameter description
0x00	No Traffic.
0x01	Best Effort.
0x02	Guaranteed.
0x03–0xFF	Reserved for future use.

Token_Rate: *Size: 4 Bytes*

Value	Parameter description
0xXXXXXXXX	Token Rate in bytes per second.

Peak_Bandwidth: *Size: 4 Bytes*

Value	Parameter description
0xXXXXXXXX	Peak Bandwidth in bytes per second.

Latency: *Size: 4 Bytes*

Value	Parameter description
0xXXXXXXXX	Latency in microseconds.

Delay_Variation: *Size: 4 Bytes*

Value	Parameter description
0xXXXXXXXX	Delay Variation in microseconds.

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the QoS_Setup command, the Host Controller sends the Command Status event to the Host. When the Link Manager has completed the LMP messages to establish the requested QoS parameters, the Host Controller on the local Bluetooth device will send a QoS Setup Complete event to the Host, and the event may also be generated on the remote side if there was LMP negotiation. The values of the parameters of the QoS Setup Complete event may, however, be different on the initiating and the remote side. The QoS Setup Complete event returned by the Host Controller on the local side contains the status of this command, and returned QoS parameters describing the supported QoS for the connection.

Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the QoS Setup Complete event will indicate that this command has been completed.

11.2.6.7 Role_Discovery

Command	OCF	Command parameters	Return parameters
HCI_Role_Discovery	0x0009	Connection_Handle	Status, Connection_Handle, Current_Role

Description:

The Role_Discovery command is used for a Bluetooth device to determine which role the device is performing for a particular Connection Handle. The Connection_Handle shall be a Connection_Handle for an ACL connection.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to identify a connection. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Role_Discovery command succeeded.
0x01–0xFF	Role_Discovery command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to identify which connection the Role_Discovery command was issued on. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Current_Role: *Size: 1 Byte*

Value	Parameter description
0x00	Current Role is Master for this Connection Handle.
0x01	Current Role is Slave for this Connection Handle.

Event(s) generated (unless masked away):

When the Role_Discovery command has completed, a Command Complete event will be generated.

11.2.6.8 Switch_Role

Command	OCF	Command parameters	Return parameters
HCI_Switch_Role	0x000B	BD_ADDR, Role	—

Description:

The Switch_Role command is used for a Bluetooth device to switch the current role the device is performing for a particular connection with another specified Bluetooth device. The BD_ADDR command parameter indicates for which connection the role switch is to be performed. The Role indicates the requested new role that the local device performs. Note that the BD_ADDR command parameter shall specify a Bluetooth device for which a connection already exists.

Command Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR for the connected device with which a role switch is to be performed.

Role: *Size: 1 Byte*

Value	Parameter description
0x00	Change own Role to Master for this BD_ADDR.
0x01	Change own Role to Slave for this BD_ADDR.

Return Parameters:

None.

Event(s) generated (unless masked away):

A Command Status event for this command will occur when the Host Controller has received the Switch_Role command. When the role switch is performed, a Role Change event will occur to indicate that the roles have been changed, and will be communicated to both Hosts.

Note that no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, only the Role Change event will indicate that this command has been completed.

11.2.6.9 Read_Link_Policy_Settings

Command	OCF	Command parameters	Return parameters
HCI_Read_Link_Policy_ Settings	0x000C	Connection_Handle	Status, Connection_Handle Link_Policy_Settings

Description:

This command will read the Link Policy setting for the specified Connection Handle. The Link_Policy_Settings parameter determines the behavior of the local Link Manager when it receives a request from a remote device or it determines itself to change the master-slave role or to enter the hold, sniff, or park mode. The local Link Manager will automatically accept or reject such a request from the remote device, and may even autonomously request itself, depending on the value of the Link_Policy_Settings parameter for the corresponding Connection_Handle. When the value of the Link_Policy_Settings parameter is changed for a certain Connection_Handle, the new value will only be used for requests from a remote device or from the local Link Manager itself made after this command has been completed. The Connection_Handle shall be a Connection_Handle for an ACL connection. By enabling each mode individually, the Host can choose any combination needed to support various modes of operation. Multiple LM policies may be specified for the Link_Policy_Settings parameter by performing a bitwise OR operation of the different activity types.

Note that the local device may be forced into hold mode (regardless of whether the local device is master or slave) by the remote device regardless of the value of the Link_Policy_Settings parameter. The forcing of hold mode can, however, only be done once the connection has already been placed into hold mode through an LMP request (the Link_Policy_Settings determine if requests from a remote device should be accepted or rejected). The forcing of hold mode can after that be done as long as the connection lasts regardless of the setting for the hold mode in the Link_Policy_Settings parameter.

Note that the previous description implies that if the implementation in the remote device is a “polite” implementation that does not force another device into hold mode via LMP PDUs, then the Link_Policy_Settings will never be overruled.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to identify a connection. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Link_Policy_Settings command succeeded.
0x01–0xFF	Read_Link_Policy_Settings command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to identify a connection. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Link_Policy_Settings *Size: 2 Bytes*

Value	Parameter description
0x0000	Disable All LM Modes.
0x0001	Enable Master Slave Switch.
0x0002	Enable Hold Mode.
0x0004	Enable Sniff Mode.
0x0008	Enable Park Mode.
0x0010– 0x8000	Reserved for future use.

Event(s) generated (unless masked away):

When the Read_Link_Policy_Settings command has completed, a Command Complete event will be generated.

11.2.6.10 Write_Link_Policy_Settings

Command	OCF	Command parameters	Return parameters
HCI_Write_Link_Policy_ Settings	0x000D	Connection_Handle, Link_Policy_Settings	Status, Connection_Handle

Description:

This command will write the Link Policy setting for the specified Connection Handle. The Link_Policy_Settings parameter determines the behavior of the local Link Manager when it receives a request from a remote device or it determines itself to change the master-slave role or to enter the hold, sniff, or park mode. The local Link Manager will automatically accept or reject such a request from the remote device, and may even autonomously request itself, depending on the value of the Link_Policy_Settings parameter for the corresponding Connection_Handle. When the value of the Link_Policy_Settings parameter is changed for a certain Connection_Handle, the new value will only be used for requests from a remote device or from the local Link Manager itself made after this command has been completed. The Connection_Handle shall be a Connection_Handle for an ACL connection. By enabling each mode individually, the Host can choose any combination needed to support various modes of operation. Multiple LM policies may be specified for the Link_Policy_Settings parameter by performing a bitwise OR operation of the different activity types.

Note that the local device may be forced into hold mode (regardless of whether the local device is master or slave) by the remote device regardless of the value of the Link_Policy_Settings parameter. The forcing of hold mode can however only be done once the connection has already been placed into hold mode through

an LMP request (the Link_Policy_Settings determine if requests from a remote device should be accepted or rejected). The forcing of hold mode can after that be done as long as the connection lasts regardless of the setting for the hold mode in the Link_Policy_Settings parameter.

Note that the previous description implies that if the implementation in the remote device is a “polite” implementation that does not force another device into hold mode via LMP PDUs, then the Link_Policy_Settings will never be overruled.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to identify a connection. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use)

Link_Policy_Settings *Size: 2 Bytes*

Value	Parameter description
0x0000	Disable All LM Modes **Default**.
0x0001	Enable Master Slave Switch.
0x0002	Enable Hold Mode.
0x0004	Enable Sniff Mode.
0x0008	Enable Park Mode.
0x0010–0x8000	Reserved for future use.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Link_Policy_Settings command succeeded.
0x01–0xFF	Write_Link_Policy_Settings command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to identify a connection. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use)

Event(s) generated (unless masked away):

When the Write_Link_Policy_Settings command has completed, a Command Complete event will be generated.

11.2.7 Host controller and baseband commands

The Host Controller and Baseband Commands provide access and control to various capabilities of the Bluetooth hardware. These parameters provide control of Bluetooth devices and of the capabilities of the Host Controller, Link Manager, and Baseband. The host device can use these commands to modify the behavior of the local device. For the HCI Control and Baseband Commands, the OGF is defined as 0x03.

Command	Command summary description
Set_Event_Mask	The Set_Event_Mask command is used to control which events are generated by the HCI for the Host.
Reset	The Reset command will reset the Bluetooth Host Controller, Link Manager, and the radio module.
Set_Event_Filter	The Set_Event_Filter command is used by the Host to specify different event filters. The Host may issue this command multiple times to request various conditions for the same type of event filter and for different types of event filters.
Flush	The Flush command is used to discard all data that is currently pending for transmission in the Host Controller for the specified connection handle, even if there currently are chunks of data that belong to more than one L2CAP packet in the Host Controller.
Read_PIN_Type	The Read_PIN_Type command is used for the Host to read the value that is specified to indicate whether the Host supports variable PIN or only fixed PINs.
Write_PIN_Type	The Write_PIN_Type command is used for the Host to specify whether the Host supports variable PIN or only fixed PINs.
Create_New_Unit_Key	The Create_New_Unit_Key command is used to create a new unit key.
Read_Stored_Link_Key	The Read_Stored_Link_Key command provides the ability to read one or more link keys stored in the Bluetooth Host Controller.
Write_Stored_Link_Key	The Write_Stored_Link_Key command provides the ability to write one or more link keys to be stored in the Bluetooth Host Controller.

Command	Command summary description
Delete_Stored_Link_Key	The Delete_Stored_Link_Key command provides the ability to remove one or more of the link keys stored in the Bluetooth Host Controller.
Change_Local_Name	The Change_Local_Name command provides the ability to modify the user-friendly name for the Bluetooth device.
Read_Local_Name	The Read_Local_Name command provides the ability to read the stored user-friendly name for the Bluetooth device.
Read_Connection_Accept_Timeout	The Read_Connection_Accept_Timeout command will read the value for the Connection_Accept_Timeout configuration parameter, which allows the Bluetooth hardware to automatically deny a connection request after a specified period has occurred, and to refuse a new connection.
Write_Connection_Accept_Timeout	The Write_Connection_Accept_Timeout will write the value for the Connection_Accept_ Timeout configuration parameter, which allows the Bluetooth hardware to automatically deny a connection request after a specified period has occurred, and to refuse a new connection.
Read_Page_Timeout	The Read_Page_Timeout command will read the value for the Page_Reply_Timeout configuration parameter, which allows the Bluetooth hardware to define the amount of time a connection request will wait for the remote device to respond before the local device returns a connection failure.
Write_Page_Timeout	The Write_Page_Timeout command will write the value for the Page_Reply_Timeout configuration parameter, which allows the Bluetooth hardware to define the amount of time a connection request will wait for the remote device to respond before the local device returns a connection failure.
Read_Scan_Enable	The Read_Scan_Enable command will read the value for the Scan_Enable configuration parameter, which controls whether or not the Bluetooth device will periodically scan for page attempts and/or inquiry requests from other Bluetooth devices.

Command	Command summary description
Write_Scan_Enable	The Write_Scan_Enable command will write the value for the Scan_Enable configuration parameter, which controls whether or not the Bluetooth device will periodically scan for page attempts and/or inquiry requests from other Bluetooth devices.
Read_Page_Scan_Activity	The Read_Page_Scan_Activity command will read the values for the Page_Scan_Interval and Page_Scan_Window configuration parameters. Page_Scan_Interval defines the amount of time between consecutive page scans. Page_Scan_Window defines the duration of the page scan.
Write_Page_Scan_Activity	The Write_Page_Scan_Activity command will write the value for Page_Scan_Interval and_ Page_Scan_Window configuration parameters. Page_Scan_Interval defines the amount of time between consecutive page scans. Page_Scan_Window defines the duration of the page scan.
Read_Inquiry_Scan_Activity	The Read_Inquiry_Scan_Activity command will read the value for Inquiry_Scan_Interval and Inquiry_Scan_Window configuration parameters. Inquiry_Scan_Interval defines the amount of time between consecutive inquiry scans. Inquiry_Scan_Window defines the amount of time for the duration of the inquiry scan.
Write_Inquiry_Scan_Activity	The Write_Inquiry_Scan_Activity command will write the value for Inquiry_Scan_Interval and Inquiry_Scan_Window configuration parameters. Inquiry_Scan_Interval defines the amount of time between consecutive inquiry scans. Inquiry_Scan_Window defines the amount of time for the duration of the inquiry scan.
Read_Authentication_Enable	The Read_Authentication_Enable command will read the value for the Authentication_ Enable parameter, which controls whether the Bluetooth device will require authentication for each connection with other Bluetooth devices.
Write_Authentication_Enable	The Write_Authentication_Enable command will write the value for the Authentication_ Enable parameter, which controls whether the Bluetooth device will require authentication for each connection with other Bluetooth devices.

Command	Command summary description
Read_Encryption_Mode	The Read_Encryption_Mode command will read the value for the Encryption_Mode parameter, which controls whether the Bluetooth device will require encryption for each connection with other Bluetooth devices.
Write_Encryption_Mode	The Write_Encryption_Mode command will write the value for the Encryption_Mode parameter, which controls whether the Bluetooth device will require encryption for each connection with other Bluetooth devices.
Read_Class_of_Device	The Read_Class_of_Device command will read the value for the Class_of_Device parameter, which is used to indicate its capabilities to other devices.
Write_Class_of_Device	The Write_Class_of_Device command will write the value for the Class_of_Device parameter, which is used to indicate its capabilities to other devices.
Read_Voice_Setting	The Read_Voice_Setting command will read the values for the Voice_Setting parameter, which controls all the various settings for the voice connections.
Write_Voice_Setting	The Write_Voice_Setting command will write the values for the Voice_Setting parameter, which controls all the various settings for the voice connections.
Read_Automatic_Flush_Timeout	The Read_Automatic_Flush_Timeout will read the value for the Flush_Timeout parameter for the specified connection handle. The Flush_ Timeout parameter is only used for ACL connections.
Write_Automatic_Flush_Timeout	The Write_Automatic_Flush_Timeout will write the value for the Flush_Timeout parameter for the specified connection handle. The Flush_ Timeout parameter is only used for ACL connections.
Read_Num_Broadcast_Retransmissions	The Read_Num_Broadcast_Retransmissions command will read the parameter value for the Number of Broadcast Retransmissions for the device. Broadcast packets are not acknowledged and are unreliable. This parameter is used to increase the reliability of a broadcast message by retransmitting the broadcast message multiple times.

Command	Command summary description
Write_Num_Broadcast_Retransmissions	The Write_Num_Broadcast_Retransmissions command will write the parameter value for the Number of Broadcast Retransmissions for the device. Broadcast packets are not acknowledged and are unreliable. This parameter is used to increase the reliability of a broadcast message by retransmitting the broadcast message multiple times.
Read_Hold_Mode_Activity	The Read_Hold_Mode_Activity command will read the value for the Hold_Mode_Activity parameter. This value is used to determine what activity the device should do when it is in hold mode.
Write_Hold_Mode_Activity	The Write_Hold_Mode_Activity command will write the value for the Hold_Mode_Activity parameter. This value is used to determine what activity the device should do when it is in hold mode.
Read_Transmit_Power_Level	The Read_Transmit_Power_Level command will read the values for the Transmit_Power_ Level parameter for the specified Connection Handle.
Read_SCO_Flow_Control_Enable	The Read_SCO_Flow_Control_Enable command provides the ability to read the SCO_Flow_Control_Enable setting. By using this setting, the Host can decide if the Host Controller will send Number Of Completed Packets events for SCO Connection Handles.
Write_SCO_Flow_Control_Enable	The Write_SCO_Flow_Control_Enable command provides the ability to write the SCO_Flow_Control_Enable setting. By using this setting, the Host can decide if the Host Controller will send Number Of Completed Packets events for SCO Connection Handles.
Read_Link_Supervision_Timeout	The Read_Link_Supervision_Timeout command will read the value for the Link_Supervision_Timeout parameter for the device. This parameter is used by the master or slave Bluetooth device to monitor link loss.
Write_Link_Supervision_Timeout	The Write_Link_Supervision_Timeout command will write the value for the Link_Supervision_ Timeout parameter for the device. This parameter is used by the master or slave Bluetooth device to monitor link loss.

Command	Command summary description
Read_Number_Of_Supported_IAC	The Read_Number_Of_Supported_IAC command will read the value for the number of Inquiry Access Codes (IAC) that the local Bluetooth device can simultaneously listen for during an Inquiry Scan.
Read_Current_IAC_LAP	The Read_Current_IAC_LAP command will read the LAP(s) used to create the Inquiry Access Codes (IAC) that the local Bluetooth device is simultaneously scanning for during Inquiry Scans.
Write_Current_IAC_LAP	The Write_Current_IAC_LAP will write the LAP(s) used to create the Inquiry Access Codes (IAC) that the local Bluetooth device is simultaneously scanning for during Inquiry Scans.
Read_Page_Scan_Period_Mode	The Read_Page_Scan_Period_Mode command is used to read the mandatory Page_Scan_Period_Mode of the local Bluetooth device.
Write_Page_Scan_Period_Mode	The Write_Page_Scan_Period_Mode command is used to write the mandatory Page_Scan_Period_Mode of the local Bluetooth device.
Read_Page_Scan_Mode	The Read_Page_Scan_Mode command is used to read the default Page_Scan_Mode of the local Bluetooth device.
Write_Page_Scan_Mode	The Write_Page_Scan_Mode command is used to write the default Page_Scan_Mode of the local Bluetooth device.

11.2.7.1 Set_Event_Mask

Command	OCF	Command parameters	Return parameters
HCI_Set_Event_Mask	0x0001	Event_Mask	Status

Description:

The Set_Event_Mask command is used to control which events are generated by the HCI for the Host. If the bit in the Event_Mask is set to a one, then the event associated with that bit will be enabled. The Host has to deal with each event that occurs by the Bluetooth devices. The event mask allows the Host to control how much it is interrupted.

Note that the Command Complete event, Command Status event and Number Of Completed Packets event cannot be masked. These events always occur. The Event_Mask is a bit mask of all of the events specified in Table 108.

Command Parameters:

Event_Mask: *Size: 8 Bytes*

Value	Parameter description
0x0000000000000000	No events specified.
0x0000000000000001	Inquiry Complete event.
0x0000000000000002	Inquiry Result event.
0x0000000000000004	Connection Complete event.
0x0000000000000008	Connection Request event.
0x0000000000000010	Disconnection Complete event.
0x0000000000000020	Authentication Complete event.
0x0000000000000040	Remote Name Request Complete event.
0x0000000000000080	Encryption Change event.
0x0000000000000100	Change Connection Link Key Complete event.
0x0000000000000200	Master Link Key Complete event.
0x0000000000000400	Read Remote Supported Features Complete event.
0x0000000000000800	Read Remote Version Information Complete event.
0x0000000000001000	QoS Setup Complete event.
0x0000000000002000	Command Complete event.
0x0000000000004000	Command Status event.
0x0000000000008000	Hardware Error event.
0x0000000000010000	Flush Occurred event.
0x0000000000020000	Role Change event.

Value	Parameter description
0x0000000000040000	Number Of Completed Packets event.
0x0000000000080000	Mode Change event.
0x0000000000100000	Return Link Keys event.
0x0000000000200000	PIN Code Request event.
0x0000000000400000	Link Key Request event.
0x0000000000800000	Link Key Notification event.
0x0000000001000000	Loopback Command event.

Value	Parameter description
0x0000000002000000	Data Buffer Overflow event.
0x0000000004000000	Max Slots Change event.
0x0000000008000000	Read Clock Offset Complete event.
0x0000000010000000	Connection Packet Type Changed even.t
0x0000000020000000	QoS Violation event.
0x0000000040000000	Page Scan Mode Change event .
0x0000000080000000	Page Scan Repetition Mode Change event.
0x0000000100000000 to 0x8000000000000000	Reserved for future use.
0x00000000FFFFFFFF	**Default** (All events enabled).

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Set_Event_Mask command succeeded.
0x01-0xFF	Set_Event_Mask command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Set_Event_Mask command has completed, a Command Complete event will be generated.

11.2.7.2 Reset

Command	OCF	Command parameters	Return parameters
HCI_Reset	0x0003	—	Status

Description:

The Reset command will reset the Host Controller and the Link Manager. The reset command should not affect the used HCI transport layer since the HCI transport layers have reset mechanisms of their own. After the reset is completed, the current operational state will be lost, the Bluetooth device will enter standby mode and the Host Controller will automatically revert to the default values for the parameters for which default values are defined in the specification.

Note that the Host is not allowed to send additional HCI commands before the Command Complete event related to the Reset command has been received.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Reset command succeeded, was received and will be executed.
0x01–0xFF	Reset command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the reset has been performed, a Command Complete event will be generated.

11.2.7.3 Set_Event_Filter

Command	OCF	Command parameters	Return parameters
HCI_Set_Event_Filter	0x0005	Filter_Type, Filter_Condition_Type, Condition	Status

Description:

The Set_Event_Filter command is used by the Host to specify different event filters. The Host may issue this command multiple times to request various conditions for the same type of event filter and for different types of event filters. The event filters are used by the Host to specify items of interest, which allow the Host Controller to send only events that interest the Host. Only some of the events have event filters. By default (before this command has been issued after power-on or Reset), no filters are set, and the Auto_Accept_Flag is off (incoming connections are not automatically accepted). An event filter is added each time this command is sent from the Host and the Filter_Condition_Type is not equal to 0x00. (The old event filters will not be overwritten). To clear all event filters, the Filter_Type = 0x00 is used. The Auto_Accept_Flag will then be set to off. To clear event filters for only a certain Filter_Type, the Filter_Condition_Type = 0x00 is used. The Inquiry Result filter allows the Host Controller to filter out Inquiry Result events. The Inquiry Result filter allows the Host to specify that the Host Controller only sends Inquiry Results to the Host if the Inquiry Result event meets one of the specified conditions set by the Host. For the Inquiry Result filter, the Host can specify one or more of the following Filter Condition Types:

1) A new device responded to the Inquiry process
2) A device with a specific Class of Device responded to the Inquiry process
3) A device with a specific BD_ADDR responded to the Inquiry process

The Inquiry Result filter is used in conjunction with the Inquiry and Periodic Inquiry command. The Connection Setup filter allows the Host to specify that the Host Controller only sends a Connection Complete or Connection Request event to the Host if the event meets one of the specified conditions set by the Host. For the Connection Setup filter, the Host can specify one or more of the following Filter Condition Types:

1) Allow Connections from all devices
2) Allow Connections from a device with a specific Class of Device
3) Allow Connections from a device with a specific BD_ADDR

For each of these conditions, an Auto_Accept_Flag parameter allows the Host to specify what action should be done when the condition is met. The Auto_Accept_Flag allows the Host to specify if the incoming connection should be auto accepted (in which case the Host Controller will send the Connection Complete event to the Host when the connection is completed) or if the Host should make the decision (in which case the Host Controller will send the Connection Request event to the Host, to elicit a decision on the connection).

The Connection Setup filter is used in conjunction with the Read/Write_Scan_Enable commands. If the local device is in the process of a page scan, and is paged by another device which meets one on the conditions set by the Host, and the Auto_Accept_Flag is off for this device, then a Connection Request event will be sent to the Host by the Host Controller. A Connection Complete event will be sent later on after the Host has responded to the incoming connection attempt. In this same example, if the Auto_Accept_Flag is on, then a Connection Complete event will be sent to the Host by the Host Controller. (No Connection Request event will be sent in that case.)

The Host Controller will store these filters in volatile memory until the Host clears the event filters using the Set_Event_Filter command or until the Reset command is issued. The number of event filters the Host Controller can store is implementation dependent. If the Host tries to set more filters than the Host Controller can store, the Host Controller will return the “Memory Full” error code and the filter will not be installed.

Note that the Clear All Filters has no Filter Condition Types or Conditions.

Note that in the condition that a connection is auto accepted, a Link Key Request event and possibly also a PIN Code Request event and a Link Key Notification event could be sent to the Host by the Host Controller before the Connection Complete event is sent.

If there is a contradiction between event filters, the latest set event filter will override older ones. An example is an incoming connection attempt where more than one Connection Setup filter matches the incoming connection attempt, but the Auto-Accept_Flag has different values in the different filters.

Command Parameters:

Filter_Type: *Size: 1 Byte*

Value	Parameter description
0x00	Clear All Filters (Note—In this case, the Filter_Condition_type and Condition parameters should not be given, they should have a length of 0 bytes. Filter_Type should be the only parameter.)
0x01	Inquiry Result.
0x02	Connection Setup.
0x03–0xFF	Reserved for future use.

Filter Condition Types: For each Filter Type one or more Filter Condition types exists.

Inquiry_Result_Filter_Condition_Type: *Size: 1 Byte*

Value	Parameter description
0x00	A new device responded to the Inquiry process. (Note that a device may be reported to the Host in an Inquiry Result event more than once during an inquiry or inquiry period depending on the implementation. See description in 11.2.5.1 and 11.2.5.3.)
0x01	A device with a specific Class of Device responded to the Inquiry process.
0x02	A device with a specific BD_ADDR responded to the Inquiry process.
0x03–0xFF	Reserved for future use.

Connection_Setup_Filter_Condition_Type: *Size: 1 Byte*

Value	Parameter description
0x00	Allow Connections from all devices.
0x01	Allow Connections from a device with a specific Class of Device.
0x02	Allow Connections from a device with a specific BD_ADDR.
0x03–0xFF	Reserved for future use.

Condition: For each Filter Condition Type defined for the Inquiry Result Filter and the Connection Setup Filter, zero or more Condition parameters are required, depending on the filter condition type and filter type.

Condition for Inquiry_Result_Filter_Condition_Type = 0x00

Condition: *Size: 0 Byte*

Value	Parameter description
	The Condition parameter is not used.

Condition for Inquiry_Result_Filter_Condition_Type = 0x01

Condition:

Size: 6 Bytes

Class_of_Device: *Size: 3 Bytes*

Value	Parameter description
0x000000	Default, Return All Devices.
0xXXXXXX	*Class of Device* of Interest.

Class_of_Device_Mask: *Size: 3 Bytes*

Value	Parameter description
0xXXXXXX	Bit Mask used to determine which bits of the Class of Device parameter are “don’t care”. Zero-value bits in the mask indicate the “don’t care” bits of the Class of Device.

Condition for Inquiry_Result_Filter_Condition_Type = 0x02

Condition:

Size: 6 Bytes

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device of Interest.

Condition for Connection_Setup_Filter_Condition_Type = 0x00

Condition:

Size: 1 Byte

Auto_Accept_Flag: *Size:1 Byte*

Value	Parameter description
0x01	Do NOT Auto accept the connection. (Auto accept is off.)
0x02	Do Auto accept the connection with role switch disabled. (Auto accept is on.)
0x03	Do Auto accept the connection with role switch enabled. (Auto accept is on). Note: When auto accepting an incoming SCO connection, no role switch will be performed. The value 0x03 of the Auto_Accept_Flag will then get the same effect as if the value had been 0x02.
0x04–0xFF	Reserved for future use.

Condition for Connection_Setup_Filter_Condition_Type = 0x01

Condition:

Size: 7 Bytes

Class_of_Device: *Size: 3 Bytes*

Value	Parameter description
0x000000	Default, Return All Devices.
0xXXXXXX	*Class of Device* of Interest.

Class_of_Device_Mask: *Size: 3 Bytes*

Value	Parameter description
0xXXXXXX	Bit Mask used to determine which bits of the Class of Device parameter are “don’t care”. Zero-value bits in the mask indicate the “don’t care” bits of the Class of Device.

Auto_Accept_Flag: *Size: 1 Byte*

Value	Parameter description
0x01	Do NOT Auto accept the connection. (Auto accept is off.)
0x02	Do Auto accept the connection with role switch disabled. (Auto accept is on.)
0x03	Do Auto accept the connection with role switch enabled. (Auto accept is on). Note—When auto accepting an incoming SCO connection, no role switch will be performed. The value 0x03 of the Auto_Accept_Flag will then get the same effect as if the value had been 0x02.
0x04–0xFF	Reserved for future use.

Condition for Connection_Setup_Filter_Condition_Type = 0x02

Condition:

Size: 7 Bytes

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device of Interest.

Auto_Accept_Flag: *Size: 1 Byte*

Value	Parameter description
0x01	Do NOT Auto accept the connection. (Auto accept is off.)
0x02	Do Auto accept the connection with role switch disabled. (Auto accept is on.)
0x03	Do Auto accept the connection with role switch enabled. (Auto accept is on). Note: When auto accepting an incoming SCO connection, no role switch will be performed. The value 0x03 of the Auto_Accept_Flag will then get the same effect as if the value had been 0x02.
0x04–0xFF	Reserved for future use.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Set_Event_Filter command succeeded.
0x01–0xFF	Set_Event_Filter command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

A Command Complete event for this command will occur when the Host Controller has enabled the filtering of events. When one of the conditions are met, a specific event will occur.

11.2.7.4 Flush

Command	OCF	Command parameters	Return parameters
HCI_Flush	0x0008	Connection_Handle	Status, Connection_Handle

Description:

The Flush command is used to discard all data that is currently pending for transmission in the Host Controller for the specified connection handle, even if there currently are chunks of data that belong to more than one L2CAP packet in the Host Controller. After this, all data that is sent to the Host Controller for the same connection handle will be discarded by the Host Controller until an HCI Data Packet with the start Packet_Boundary_Flag (0x02) is received. When this happens, a new transmission attempt can be made. This command will allow higher-level software to control how long the baseband should try to retransmit a baseband packet for a connection handle before all data that is currently pending for transmission in the Host Controller should be flushed. Note that the Flush command is used for ACL connections ONLY. In addition to the Flush command, the automatic flush timers (see 11.2.7.31) can be used to automatically flush the L2CAP packet that is currently being transmitted after the specified flush timer has expired.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to identify which connection to flush. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Flush command succeeded.
0x01–0xFF	Flush command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to identify which connection the flush command was issued on. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Event(s) generated (unless masked away):

The Flush Occurred event will occur once the flush is completed. A Flush Occurred event could be from an automatic Flush or could be caused by the Host issuing the Flush command. When the Flush command has completed, a Command Complete event will be generated, to indicate that the Host caused the Flush.

11.2.7.5 Read_PIN_Type

Command	OCF	Command parameters	Return parameters
HCI_Read_PIN_Type	0x0009	—	Status, PIN_Type

Description:

The Read_PIN_Type command is used for the Host to read whether the Link Manager assumes that the Host supports variable PIN codes only a fixed PIN code. The Bluetooth hardware uses the PIN-type information during pairing.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_PIN_Type command succeeded.
0x01-0xFF	Read_PIN_Type command failed. See Table 109 for list of Error Codes.

PIN_Type: *Size: 1 Byte*

Value	Parameter description
0x00	Variable PIN.
0x01	Fixed PIN.

Event(s) generated (unless masked away):

When the Read_PIN_Type command has completed, a Command Complete event will be generated.

11.2.7.6 Write_PIN_Type

Command	OCF	Command parameters	Return parameters
HCI_Write_PIN_Type	0x000A	PIN_Type	Status

Description:

The Write_PIN_Type command is used for the Host to write to the Host Controller whether the Host supports variable PIN codes or only a fixed PIN code. The Bluetooth hardware uses the PIN-type information during pairing.

Command Parameters:

PIN_Type: *Size: 1 Byte*

Value	Parameter description
0x00	Variable PIN.
0x01	Fixed PIN.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write PIN Type command succeeded.
0x01–0xFF	Write PIN Type command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_PIN_Type command has completed, a Command Complete event will be generated.

11.2.7.7 Create_New_Unit_Key

Command	OCF	Command parameters	Return parameters
HCI_Create_New_Unit_Key	0x000B	—	Status

Description:

The Create_New_Unit_Key command is used to create a new unit key. The Bluetooth hardware will generate a random seed that will be used to generate the new unit key. All new connection will use the new unit key, but the old unit key will still be used for all current connections.

Note that this command will not have any effect for a device that does not use unit keys (i.e., a device that uses only combination keys).

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Create New Unit Key command succeeded.
0x01–0xFF	Create New Unit Key command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Create_New_Unit_Key command has completed, a Command Complete event will be generated.

11.2.7.8 Read_Stored_Link_Key

Command	OCF	Command parameters	Return parameters
HCI_Read_Stored_ Link_Key	0x000D	BD_ADDR, Read_All_Flag	Status, Max_Num_Keys, Num_Keys_Read

Description:

The Read_Stored_Link_Key command provides the ability to read one or more link keys stored in the Bluetooth Host Controller. The Bluetooth Host Controller can store a limited number of link keys for other Bluetooth devices. Link keys are shared between two Bluetooth devices, and are used for all security transactions between the two devices. A Host device may have additional storage capabilities, which can be

used to save additional link keys to be reloaded to the Bluetooth Host Controller when needed. The Read_All_Flag parameter is used to indicate if all of the stored Link Keys should be returned. If Read_All_Flag indicates that all Link Keys are to be returned, then the BD_ADDR command parameter shall be ignored The BD_ADDR command parameter is used to identify which link key to read. The stored Link Keys are returned by one or more Return Link Keys events.

Command Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR for the stored link key to be read.

Read_All_Flag: *Size: 1 Byte*

Value	Parameter description
0x00	Return Link Key for specified BD_ADDR.
0x01	Return all stored Link Keys.
0x02–0xFF	Reserved for future use.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Stored_Link_Key command succeeded.
0x01–0xFF	Read_Stored_Link_Key command failed. See Table 109 for list of Error Codes.

Max_Num_Keys: *Size: 2 Byte*

Value	Parameter description
0xXXXX	Total Number of Link Keys that the Host Controller can store. Range: 0x0000–0xFFFF

Num_Keys_Read: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Number of Link Keys Read. Range: 0x0000–0xFFFF

Event(s) generated (unless masked away):

Zero or more instances of the Return Link Keys event will occur after the command is issued. When there are no link keys stored, no Return Link Keys events will be returned. When there are link keys stored, the

number of link keys returned in each Return Link Keys event is implementation specific. When the Read Stored Link Key command has completed a Command Complete event will be generated.

11.2.7.9 Write_Stored_Link_Key

Command	OCF	Command parameters	Return parameters
HCI_Write_Stored_Link_Key	0x0011	Num_Keys_To_Write, *BD_ADDR*[i], Link_Key[i]	Status, Num_Keys_Written

Description:

The Write_Stored_Link_Key command provides the ability to write one or more link keys to be stored in the Bluetooth Host Controller. The Bluetooth Host Controller can store a limited number of link keys for other Bluetooth devices. If no additional space is available in the Bluetooth Host Controller then no additional link keys will be stored. If space is limited and if all the link keys to be stored will not fit in the limited space, then the order of the list of link keys without any error will determine which link keys are stored. Link keys at the beginning of the list will be stored first. The Num_Keys_Written parameter will return the number of link keys that were successfully stored. If no additional space is available, then the Host must delete one or more stored link keys before any additional link keys are stored. The link key replacement algorithm is implemented by the Host and not the Host Controller. Link keys are shared between two Bluetooth devices and are used for all security transactions between the two devices. A Host device may have additional storage capabilities, which can be used to save additional link keys to be reloaded to the Bluetooth Host Controller when needed.

Note that Link Keys are only stored by issuing this command.

Command Parameters:

Num_Keys_To_Write: *Size: 1 Byte*

Value	Parameter description
0xXX	Number of Link Keys to Write. Range: 0x01–0x0B

BD_ADDR [i]: *Size: 6 Bytes * Num_Keys_To_Write*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR for the associated Link Key.

Link_Key[i]: *Size: 16 Bytes * Num_Keys_To_Write*

Value	Parameter description
0xXXXXXXXXXX XXXXXXXXXX XXXXXXXXXX	Link Key for the associated BD_ADDR.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Stored_Link_Key command succeeded.
0x01–0xFF	Write_Stored_Link_Key command failed. See Table 109 for list of Error Codes.

Num_Keys_Written: *Size: 1 Bytes*

Value	Parameter description
0xXX	Number of Link Keys successfully written. Range: 0x00–0x0B

Event(s) generated (unless masked away):

When the Write_Stored_Link_Key command has completed, a Command Complete event will be generated.

11.2.7.10 Delete_Stored_Link_Key

Command	OCF	Command parameters	Return parameters
HCI_Delete_Stored_ Link_Key	0x0012	BD_ADDR, Delete_All_Flag	Status, Num_Keys_Deleted

Description:

The Delete_Stored_Link_Key command provides the ability to remove one or more of the link keys stored in the Bluetooth Host Controller. The Bluetooth Host Controller can store a limited number of link keys for other Bluetooth devices. Link keys are shared between two Bluetooth devices and are used for all security transactions between the two devices. The Delete_All_Flag parameter is used to indicate if all of the stored Link Keys should be deleted. If the Delete_All_Flag indicates that all Link Keys are to be deleted, then the BD_ADDR command parameter shall be ignored. This command provides the ability to negate all security agreements between two devices. The BD_ADDR command parameter is used to identify which link key to delete. If a link key is currently in use for a connection, then the link key will be deleted when all of the connections are disconnected.

Command Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR for the link key to be deleted.

Delete_All_Flag: *Size: 1 Byte*

Value	Parameter description
0x00	Delete only the Link Key for specified BD_ADDR.
0x01	Delete all stored Link Keys.
0x02–0xFF	Reserved for future use.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Delete_Stored_Link_Key command succeeded.
0x01–0xFF	Delete_Stored_Link_Key command failed. See Table 109 for list of Error Codes.

Num_Keys_Deleted: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Number of Link Keys Deleted.

Event(s) generated (unless masked away):

When the Delete_Stored_Link_Key command has completed, a Command Complete event will be generated.

11.2.7.11 Change_Local_Name

Command	OCF	Command parameters	Return parameters
HCI_Change_Local_Name	0x0013	Name	Status

Description:

The Change_Local_Name command provides the ability to modify the user-friendly name for the Bluetooth device. A Bluetooth device may send a request to get the user-friendly name of another Bluetooth device. The user-friendly name provides the user with the ability to distinguish one Bluetooth device from another. The Name command parameter is a UTF-8 encoded string with up to 248 bytes in length. The Name command parameter should be null-terminated (0x00) if the UTF-8 encoded string is less than 248 bytes.

Note that the Name Parameter is a string parameter. Endianess does therefore not apply to the Name Parameter. The first byte of the name should be transmitted first.

Command Parameters:

Name: *Size: 248 Bytes*

Value	Parameter description
	A UTF-8 encoded User-Friendly Descriptive Name for the device. If the name contained in the parameter is shorter than 248 bytes, the end of the name is indicated by a NULL byte (0x00), and the following bytes (to fill up 248 bytes, which is the length of the parameter) do not have valid values.
	Null terminated Zero length String. **Default.**

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Change_Local_Name command succeeded.
0x01–0xFF	Change_Local_Name command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Change_Local_Name command has completed, a Command Complete event will be generated.

11.2.7.12 Read_Local_Name

Command	OCF	Command parameters	Return parameters
HCI_Read_Local_Name	0x0014	—	Status, Name

Description:

The Read_Local_Name command provides the ability to read the stored user-friendly name for the Bluetooth device. The user-friendly name provides the user the ability to distinguish one Bluetooth device from another. The Name return parameter is a UTF-8 encoded string with up to 248 bytes in length. The Name return parameter will be null terminated (0x00) if the UTF-8 encoded string is less than 248 bytes.

Note that the Name Parameter is a string parameter. Endianess does therefore not apply to the Name Parameter. The first byte of the name is received first.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Local_Name command succeeded.
0x01–0xFF	Read_Local_Name command failed. See Table 109 for list of Error Codes

Name: *Size: 248 Bytes*

Value	Parameter description
	A UTF-8 encoded User Friendly Descriptive Name for the device. If the name contained in the parameter is shorter than 248 bytes, the end of the name is indicated by a NULL byte (0x00), and the following bytes (to fill up 248 bytes, which is the length of the parameter) do not have valid values.

Event(s) generated (unless masked away):

When the Read_Local_Name command has completed a Command Complete event will be generated.

11.2.7.13 Read_Connection_Accept_Timeout

Command	OCF	Command parameters	Return parameters
HCI_Read_Connection_Accept_Timeout	0x0015	—	Status, Conn_Accept_Timeout

Description:

This command will read the value for the Connection_Accept_Timeout configuration parameter. The Connection_Accept_Timeout configuration parameter allows the Bluetooth hardware to automatically deny a connection request after a specified time period has occurred and the new connection is not accepted. The parameter defines the time duration from when the Host Controller sends a Connection Request event until the Host Controller will automatically reject an incoming connection.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Connection_Accept_Timeout command succeeded.
0x01–0xFF	Read_Connection_Accept_Timeout command failed. See Table 109 for list of Error Codes.

Conn_Accept_Timeout: *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Connection Accept Timeout measured in Number of Baseband slots. Interval Length = *N* * 0.625 ms (1 Baseband slot) Range for *N*: 0x0001–0xB540 Time Range: 0.625 ms–29 s

Event(s) generated (unless masked away):

When the Read_Connection_Timeout command has completed, a Command Complete event will be generated.

11.2.7.14 Write_Connection_Accept_Timeout

Command	OCF	Command parameters	Return parameters
HCI_Write_Connection_Accept_Timeout	0x0016	Conn_Accept_Timeout	Status

Description:

This command will write the value for the Connection_Accept_Timeout configuration parameter. The Connection_Accept_Timeout configuration parameter allows the Bluetooth hardware to automatically deny a connection request after a specified time interval has occurred and the new connection is not accepted. The parameter defines the time duration from when the Host Controller sends a Connection Request event until the Host Controller will automatically reject an incoming connection.

Command Parameters:

Conn_Accept_Timeout: *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Connection Accept Timeout measured in Number of Baseband slots. Interval Length = *N* * 0.625 ms (1 Baseband slot) Range for *N*: 0x0001–0xB540 Time Range: 0.625 ms–29 s Default: *N* = 0x1F40 Time = 5 s

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Connection_Accept_Timeout command succeeded.
0x01–0xFF	Write_Connection_Accept_Timeout command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Connection_Accept_Timeout command has completed, a Command Complete event will be generated.

11.2.7.15 Read_Page_Timeout

Command	OCF	Command parameters	Return parameters
HCI_Read_Page_Timeout	0x0017	—	Status, Page_Timeout

Description:

This command will read the value for the Page_Timeout configuration parameter. The Page_Timeout configuration parameter defines the maximum time the local Link Manager will wait for a baseband page response from the remote device at a locally initiated connection attempt. If this time expires and the remote device has not responded to the page at baseband level, the connection attempt will be considered to have failed.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Page_Timeout command succeeded.
0x01–0xFF	Read_Page_Timeout command failed. See Table 109 for list of Error Codes.

Page_Timeout: *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Page Timeout measured in Number of Baseband slots. Interval Length = *N* * 0.625 ms (1 Baseband slot) Range for *N*: 0x0001–0xFFFF Time Range: 0.625 ms–40.9 s

Event(s) generated (unless masked away):

When the Read_Page_Timeout command has completed, a Command Complete event will be generated.

11.2.7.16 Write_Page_Timeout

Command	OCF	Command parameters	Return parameters
HCI_Write_Page_Timeout	0x0018	Page_Timeout	Status

Description:

This command will write the value for the Page_Timeout configuration parameter. The Page_Timeout configuration parameter defines the maximum time the local Link Manager will wait for a baseband page response from the remote device at a locally initiated connection attempt. If this time expires and the remote device has not responded to the page at baseband level, the connection attempt will be considered to have failed.

Command Parameters:

Page_Timeout: *Size: 2 Bytes*

Value	Parameter description
0	Illegal Page Timeout. Shall be larger than 0.
N = 0xXXXX	Page Timeout measured in Number of Baseband slots. Interval Length = *N* * 0.625 ms (1 Baseband slot) Range for *N*: 0x0001–0xFFFF Time Range: 0.625 ms–40.9 s Default: *N* = 0x2000 Time = 5.12 s

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Page_Timeout command succeeded.
0x01–0xFF	Write_Page_Timeout command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Page_Timeout command has completed, a Command Complete event will be generated.

11.2.7.17 Read_Scan_Enable

Command	OCF	Command parameters	Return parameters
HCI_Read_Scan_Enable	0x0019	—	Status, Scan_Enable

Description:

This command will read the value for the Scan_Enable parameter. The Scan_Enable parameter controls whether or not the Bluetooth device will periodically scan for page attempts and/or inquiry requests from other Bluetooth devices. If Page_Scan is enabled, then the device will enter page scan mode based on the value of the Page_Scan_Interval and Page_Scan_Window parameters. If Inquiry_Scan is enabled, then the device will enter Inquiry Scan mode based on the value of the Inquiry_Scan_Interval and Inquiry_Scan_ Window parameters.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Scan_Enable command succeeded.
0x01–0xFF	Read_Scan_Enable command failed. See Table 109 for list of Error Codes.

Scan_Enable: *Size: 1 Byte*

Value	Parameter description
0x00	No Scans enabled.
0x01	Inquiry Scan enabled. Page Scan disabled.
0x02	Inquiry Scan disabled. Page Scan enabled.
0x03	Inquiry Scan enabled. Page Scan enabled.
0x04–0xFF	Reserved

Event(s) generated (unless masked away):

When the Read_Scan_Enable command has completed, a Command Complete event will be generated.

11.2.7.18 Write_Scan_Enable

Command	OCF	Command parameters	Return parameters
HCI_Write_Scan_Enable	0x001A	Scan_Enable	Status

Description:

This command will write the value for the Scan_Enable parameter. The Scan_Enable parameter controls whether or not the Bluetooth device will periodically scan for page attempts and/or inquiry requests from other Bluetooth devices. If Page_Scan is enabled, then the device will enter page scan mode based on the value of the Page_Scan_Interval and Page_Scan_Window parameters. If Inquiry_Scan is enabled, then the device will enter Inquiry Scan mode based on the value of the Inquiry_Scan_Interval and Inquiry_Scan_ Window parameters.

Command Parameters:

Scan_Enable: *Size: 1 Byte*

Value	Parameter description
0x00	No Scans enabled. **Default.**
0x01	Inquiry Scan enabled. Page Scan disabled.
0x02	Inquiry Scan disabled. Page Scan enabled.
0x03	Inquiry Scan enabled. Page Scan enabled.
0x04–0xFF	Reserved.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Scan_Enable command succeeded.
0x01–0xFF	Write_Scan_Enable command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Scan_Enable command has completed, a Command Complete event will be generated.

11.2.7.19 Read_Page_Scan_Activity

Command	OCF	Command parameters	Return parameters
HCI_Read_Page_Scan_ Activity	0x001B	—	Status, Page_Scan_Interval, Page_Scan_Window

Description:

This command will read the value for Page_Scan_Activity configuration parameters. The Page_Scan_Interval configuration parameter defines the amount of time between consecutive page scans. This time interval is defined from when the Host Controller started its last page scan until it begins the next page scan. The Page_Scan_Window configuration parameter defines the amount of time for the duration of the page scan. The Page_Scan_Window can only be less than or equal to the Page_Scan_Interval.

Note that Page Scan is only performed when Page_Scan is enabled (see 11.2.7.17 and 11.2.7.18).

A changed Page_Scan_Interval could change the local Page_Scan_Repetition_Mode (see Clause 8, Keyword: SR-Mode).

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Page_Scan_Activity command succeeded.
0x01–0xFF	Read_Page_Scan_Activity command failed. See Table 109 for list of Error Codes.

Page_Scan_Interval: *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Size: 2 Bytes Range: 0x0012–0x1000 Time = *N* * 0.625 ms Range: 11.25 ms– 2560 ms

Page_Scan_Window: *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Size: 2 Bytes Range: 0x0012–0x1000 Time = *N* * 0.625 ms Range: 11.25 ms–2560 ms

Event(s) generated (unless masked away):

When the Read_Page_Scan_Activity command has completed, a Command Complete event will be generated.

11.2.7.20 Write_Page_Scan_Activity

Command	OCF	Command parameters	Return parameters
HCI_Write_Page_Scan_Activity	0x001C	Page_Scan_Interval, Page_Scan_Window	Status

Description:

This command will write the value for Page_Scan_Activity configuration parameter. The Page_Scan_Interval configuration parameter defines the amount of time between consecutive page scans. This is defined as the time interval from when the Host Controller started its last page scan until it begins the next page scan. The Page_Scan_Window configuration parameter defines the amount of time for the duration of the page scan. The Page_Scan_Window can only be less than or equal to the Page_Scan_Interval.

Note that Page Scan is only performed when Page_Scan is enabled (see 11.2.7.17 and 11.2.7.18). A changed Page_Scan_Interval could change the local Page_Scan_Repetition_Mode (see Clause 8, Keyword: SR-Mode).

Command Parameters:

Page_Scan_Interval: *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Size: 2 Bytes Range: 0x0012–0x1000 Time = *N* * 0.625 ms Range: 11.25 ms–2560 ms Default: *N* = 0x0800 Time = 1.28 s

Page_Scan_Window: *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Size: 2 Bytes Range: 0x0012–0x1000 Time = *N* * 0.625 ms Range: 11.25 ms– 2560 ms Default: *N* = 0x0012 Time = 11.25 ms

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Page_Scan_Activity command succeeded.
0x01–0xFF	Write_Page_Scan_Activity command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Page_Scan_Activity command has completed, a Command Complete event will be generated.

11.2.7.21 Read_Inquiry_Scan_Activity

Command	OCF	Command parameters	Return parameters
HCI_Read_ Inquiry_Scan_Activity	0x001D	—	Status, Inquiry_Scan_Interval, Inquiry_Scan_Window

Description:

This command will read the value for Inquiry_Scan_Activity configuration parameter. The Inquiry_Scan_Interval configuration parameter defines the amount of time between consecutive inquiry scans. This is defined as the time interval from when the Host Controller started its last inquiry scan until it begins the next inquiry scan. The Inquiry_Scan_Window configuration parameter defines the amount of time for the duration of the inquiry scan. The Inquiry_Scan_Window can only be less than or equal to the Inquiry_Scan_Interval.

Note that Inquiry Scan is only performed when Inquiry_Scan is enabled (see 11.2.7.17 and 11.2.7.18).

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Inquiry_Scan_Activity command succeeded.
0x01–0xFF	Read_Inquiry_Scan_Activity command failed. See Table 109 for list of Error Codes.

Inquiry_Scan_Interval: *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Size: 2 Bytes Range: 0x0012–0x1000 Time = *N* * 0.625 ms Range: 11.25–2560 ms

Inquiry_Scan_Window: *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Size: 2 Bytes Range: 0x0012–0x1000 Time = *N* * 0.625 ms Range: 0.625 ms–2560 ms

Event(s) generated (unless masked away):

When the Read_Inquiry_Scan_Activity command has completed, a Command Complete event will be generated.

11.2.7.22 Write_Inquiry_Scan_Activity

Command	OCF	Command parameters	Return parameters
HCI_Write_Inquiry_ Scan_Activity	0x001E	Inquiry_Scan_Interval, Inquiry_Scan_Window	Status

Description:

This command will write the value for Inquiry_Scan_Activity configuration parameter. The Inquiry_Scan_Interval configuration parameter defines the amount of time between consecutive inquiry scans. This is defined as the time interval from when the Host Controller started its last inquiry scan until it begins the next inquiry scan. The Inquiry_Scan_Window configuration parameter defines the amount of time for the duration of the inquiry scan. The Inquiry_Scan_Window can only be less than or equal to the Inquiry_Scan_Interval.

Note that Inquiry Scan is only performed when Inquiry_Scan is enabled (see 11.2.7.17 and 11.2.7.18).

Command Parameters:

Inquiry_Scan_Interval: *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Size: 2 Bytes Range: 0x0012–0x1000 Time = *N* * 0.625 ms Range: 11.25–2560 ms Default: *N* = 0x0800 Time = 1.28 s

Inquiry_Scan_Window: *Size: 2 Bytes*

Value	Parameter description
N = 0xXXXX	Size: 2 Bytes Range: 0x0012–0x1000 Time = *N* * 0.625 ms Range: 11.25 ms–2560 ms Default: *N* = 0x0012 Time = 11.25 ms

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Inquiry_Scan_Activity command succeeded.
0x01–0xFF	Write_Inquiry_Scan_Activity command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Inquiry_Scan_Activity command has completed, a Command Complete event will be generated.

11.2.7.23 Read_Authentication_Enable

Command	OCF	Command parameters	Return parameters
HCI_Read_ Authentication_Enable	0x001F	—	Status, Authentication_Enable

Description:

This command will read the value for the Authentication_Enable parameter. The Authentication_Enable parameter controls if the local device requires to authenticate the remote device at connection setup (between the Create_Connection command or acceptance of an incoming ACL connection and the corresponding Connection Complete event). At connection setup, only the device(s) with the Authentication_Enable parameter enabled will try to authenticate the other device.

Note that changing this parameter does not affect existing connections.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Authentication_Enable command succeeded.
0x01–0xFF	Read_Authentication_Enable command failed. See Table 109 for list of Error Codes.

Authentication_Enable: *Size: 1 Byte*

Value	Parameter description
0x00	Authentication disabled.
0x01	Authentication enabled for all connections.
0x02–0xFF	Reserved.

Event(s) generated (unless masked away):

When the Read_Authentication_Enable command has completed, a Command Complete event will be generated.

11.2.7.24 Write_Authentication_Enable

Command	OCF	Command parameters	Return parameters
HCI_Write_ Authentication_Enable	0x0020	Authentication_Enable	Status

Description:

This command will write the value for the Authentication_Enable parameter. The Authentication_Enable parameter controls if the local device requires to authenticate the remote device at connection setup (between the Create_Connection command or acceptance of an incoming ACL connection and the corresponding Connection Complete event). At connection setup, only the device(s) with the Authentication_Enable parameter enabled will try to authenticate the other device.

Note that changing this parameter does not affect existing connections.

Command Parameters:

Authentication_Enable: *Size: 1 Byte*

Value	Parameter description
0x00	Authentication disabled. **Default.**
0x01	Authentication enabled for all connection.
0x02–0xFF	Reserved.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write Authentication_Enable command succeeded.
0x01–0xFF	Write Authentication_Enable command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Authentication_Enable command has completed, a Command Complete event will be generated.

11.2.7.25 Read_Encryption_Mode

Command	OCF	Command parameters	Return parameters
HCI_Read_Encryption_Mode	0x0021	—	Status, Encryption_Mode

Description:

This command will read the value for the Encryption_Mode parameter. The Encryption_Mode parameter controls if the local device requires encryption to the remote device at connection setup (between the Create_Connection command or acceptance of an incoming ACL connection and the corresponding Connection Complete event). At connection setup, only the device(s) with the Authentication_Enable parameter enabled and Encryption_Mode parameter enabled will try to encrypt the connection to the other device.

Note that changing this parameter does not affect existing connections.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Encryption_Mode command succeeded.
0x01–0xFF	Read_Encryption_Mode command failed. See Table 109 for list of Error Codes.

Encryption_Mode: *Size: 1 Byte*

Value	Parameter description
0x00	Encryption disabled.
0x01	Encryption only for point-to-point packets.
0x02	Encryption for both point-to-point and broadcast packets.
0x03–0xFF	Reserved.

Event(s) generated (unless masked away):

When the Read_Encryption_Mode command has completed, a Command Complete event will be generated.

11.2.7.26 Write_Encryption_Mode

Command	OCF	Command parameters	Return parameters
HCI_Write_Encryption_Mode	0x0022	Encryption_Mode	Status

Description:

This command will write the value for the Encryption_Mode parameter. The Encryption_Mode parameter controls if the local device requires encryption to the remote device at connection setup (between the Create_Connection command or acceptance of an incoming ACL connection and the corresponding Connection Complete event). At connection setup, only the device(s) with the Authentication_Enable parameter enabled and Encryption_Mode parameter enabled will try to encrypt the connection to the other device.

Note that changing this parameter does not affect existing connections. A temporary link key shall be used when both broadcast and point-to-point traffic shall be encrypted.

The Host shall not specify the Encryption_Mode parameter with more encryption capability than its local device currently supports, although the parameter is used to request the encryption capability to the remote device. Note that the Host shall not request the command with the Encryption_Mode parameter set to either 0x01 or 0x02, when the local device does not support encryption. Also note that the Host shall not request the command with the parameter set to 0x02, when the local device does not support broadcast encryption.

Note that the actual Encryption_Mode to be returned in an event for a new connection (or in a Connection Complete event) will only support a part of the capability, when the local device requests more encryption capability than the current remote device supports. For example, 0x00 will always be returned in the event

when the remote device supports no encryption, and either 0x00 or 0x01 will be returned when it supports only point-to-point encryption.

Command Parameters:

Encryption_Mode: *Size: 1 Byte*

Value	Parameter description
0x00	Encryption disabled. **Default.**
0x01	Encryption only for point-to-point packets.
0x02	Encryption for both point-to-point and broadcast packets.
0x03–0xFF	Reserved.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Encryption_Mode command succeeded.
0x01–0xFF	Write_Encryption_Mode command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Encryption_Mode command has completed, a Command Complete event will be generated.

11.2.7.27 Read_Class_of_Device

Command	OCF	Command parameters	Return parameters
HCI_Read_Class_of_Device	0x0023	—	Status, Class_of_Device

Description:

This command will read the value for the Class_of_Device parameter. The Class_of_Device parameter is used to indicate the capabilities of the local device to other devices.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Class_of_Device command succeeded.
0x01–0xFF	Read_Class_of_Device command failed. See Table 109 for list of Error Codes.

Class_of_Device: *Size: 3 Bytes*

Value	Parameter description
0xXXXXXX	Class of Device for the device.

Event(s) generated (unless masked away):

When the Read_Class_of_Device command has completed, a Command Complete event will be generated.

11.2.7.28 Write_Class_of_Device

Command	OCF	Command parameters	Return Parameters
HCI_Write_Class_of_Device	0x0024	Class_of_Device	Status

Description:

This command will write the value for the Class_of_Device parameter. The Class_of_Device parameter is used to indicate the capabilities of the local device to other devices.

Command Parameters:

Class_of_Device: *Size: 3 Bytes*

Value	Parameter description
0xXXXXXX	Class of Device for the device.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Class_of_Device command succeeded.
0x01–0xFF	Write_Class_of_Device command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Class_of_Device command has completed, a Command Complete event will be generated.

11.2.7.29 Read_Voice_Setting

Command	OCF	Command parameters	Return parameters
HCI_Read_Voice_Setting	0x0025	—	Status, Voice_Setting

Description:

This command will read the values for the Voice_Setting parameter. The Voice_Setting parameter controls all the various settings for voice connections. These settings apply to all voice connections, and **cannot** be set for individual voice connections. The Voice_Setting parameter controls the configuration for voice connections: Input Coding, Air coding format, input data format, Input sample size, and linear PCM parameter.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Voice_Setting command succeeded.
0x01–0xFF	Read_Voice_Setting command failed. See Table 109 for list of Error Codes.

Voice_Setting: *Size: 2 Bytes (10 Bits meaningful)*

Value	Parameter description
00XXXXXXXX	Input Coding: Linear.
01XXXXXXXX	Input Coding: μ-law Input Coding.
10XXXXXXXX	Input Coding: A-law Input Coding.
11XXXXXXXX	Reserved for future use.
XX00XXXXXX	Input Data Format: 1’s complement.
XX01XXXXXX	Input Data Format: 2’s complement.
XX10XXXXXX	Input Data Format: Sign-Magnitude.
XX11XXXXXX	Reserved for future use.
XXXX0XXXXX	Input Sample Size: 8-bit (only for Liner PCM).
XXXX1XXXXX	Input Sample Size: 16-bit (only for Liner PCM).

Value	Parameter description
XXXXXnnnXX	Linear_PCM_Bit_Pos: number of bit positions that MSB of sample is away from starting at MSB (only for Liner PCM).
XXXXXXXX00	Air Coding Format: CVSD.
XXXXXXXX01	Air Coding Format: μ-law.
XXXXXXXX10	Air Coding Format: A-law.
XXXXXXXX11	Reserved.

Event(s) generated (unless masked away):

When the Read_Voice_Setting command has completed, a Command Complete event will be generated.

11.2.7.30 Write_Voice_Setting

Command	OCF	Command parameters	Return parameters
HCI_Write_Voice_Setting	0x0026	Voice_Setting	Status

Description:

This command will write the values for the Voice_Setting parameter. The Voice_Setting parameter controls all the various settings for the voice connections. These settings apply to all voice connections and **cannot** be set for individual voice connections. The Voice_Setting parameter controls the configuration for voice connections: Input Coding, Air coding format, input data format, Input sample size, and linear PCM parameter.

Command Parameters:

Voice_Setting: *Size: 2 Bytes (10 Bits meaningful)*

Value	Parameter description
00XXXXXXXX	Input Coding: Linear.
01XXXXXXXX	Input Coding: μ-law Input Coding.
10XXXXXXXX	Input Coding: A-law Input Coding.
11XXXXXXXX	Reserved for future use.
XX00XXXXXX	Input Data Format: 1's complement.
XX01XXXXXX	Input Data Format: 2's complement.
XX10XXXXXX	Input Data Format: Sign-Magnitude.
XX11XXXXXX	Reserved for future use.
XXXX0XXXXX	Input Sample Size: 8 bit (only for Liner PCM).

Value	Parameter description
XXXX1XXXXX	Input Sample Size: 16 bit (only for Liner PCM).
XXXXXnnnXX	Linear_PCM_Bit_Pos: number of bit positions that MSB of sample is away from starting at MSB (only for Liner PCM).
XXXXXXXX00	Air Coding Format: CVSD.
XXXXXXXX01	Air Coding Format: μ-law.
XXXXXXXX10	Air Coding Format: A-law.
XXXXXXXX11	Reserved.
00011000XX	Default Condition. (X means that there is no default value for the corresponding bit. The manufacturer may use any value.)

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Voice_Setting command succeeded.
0x01–0xFF	Write_Voice_Setting command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Voice_Setting command has completed, a Command Complete event will be generated.

11.2.7.31 Read_Automatic_Flush_Timeout

Command	OCF	Command parameters	Return parameters
HCI_Read_Automatic_Flush_Timeout	0x0027	Connection_Handle	Status, Connection_Handle, Flush_Timeout

Description:

This command will read the value for the Flush_Timeout parameter for the specified connection handle. The Flush_Timeout parameter is used for ACL connections ONLY. The Flush_Timeout parameter defines the amount of time before all chunks of the L2CAP packet, of which a baseband packet is currently being transmitted, are automatically flushed by the Host Controller. The timeout period starts when a transmission attempt is made for the first baseband packet of an L2CAP packet. This allows ACL packets to be automatically flushed without the Host device issuing a Flush command. The Read_Automatic_Flush_Timeout command provides support for isochronous data, such as video. When the L2CAP packet that is currently being transmitted is automatically “flushed,” the Failed Contact Counter is incremented by one. The first chunk of the next L2CAP packet to be transmitted for the specified connection handle may already be stored in the Host Controller. In that case, the transmission of the first baseband packet containing data from that L2CAP packet can begin immediately. If the next L2CAP packet is not

stored in the Host Controller, all data that is sent to the Host Controller after the flush for the same connection handle will be discarded by the Host Controller until an HCI Data Packet having the start Packet_Boundary_Flag (0x02) is received. When this happens, a new transmission attempt will be made.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Specifies which Connection Handle's Flush Timeout to read. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Automatic_Flush_Timeout command succeeded.
0x01–0xFF	Read_Automatic_Flush_Timeout command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Specifies which Connection Handle's Flush Timeout has been read. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Flush_Timeout: *Size: 2 Bytes*

Value	Parameter description
0	Timeout = ∞; No Automatic Flush.
N = 0xXXXX	Flush Timeout = *N* * 0.625 ms Size: 11 bits Range: 0x0001–0x07FF

Event(s) generated (unless masked away):

When the Read_Automatic_Flush_Timeout command has completed, a Command Complete event will be generated.

11.2.7.32 Write_Automatic_Flush_Timeout

Command	OCF	Command parameters	Return parameters
HCI_Write_Automatic_Flush_ Timeout	0x0028	Connection_Handle, Flush_Timeout	Status, Connection_Handle

Description:

This command will write the value for the Flush_Timeout parameter for the specified connection handle. The Flush_Timeout parameter is used for ACL connections ONLY. The Flush_Timeout parameter defines the amount of time before all chunks of the L2CAP packet, of which a baseband packet is currently being transmitted, are automatically flushed by the Host Controller. The timeout period starts when a transmission attempt is made for the first baseband packet of an L2CAP packet. This allows ACL packets to be automatically flushed without the Host device issuing a Flush command. The Write_Automatic_Flush_Timeout command provides support for isochronous data, such as video. When the L2CAP packet that is currently being transmitted is automatically "flushed," the Failed Contact Counter is incremented by one. The first chunk of the next L2CAP packet to be transmitted for the specified connection handle may already be stored in the Host Controller. In that case, the transmission of the first baseband packet containing data from that L2CAP packet can begin immediately. If the next L2CAP packet is not stored in the Host Controller, all data that is sent to the Host Controller after the flush for the same connection handle will be discarded by the Host Controller until an HCI Data Packet having the start Packet_Boundary_Flag (0x02) is received. When this happens, a new transmission attempt will be made.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Specifies which Connection Handle's Flush Timeout to write to. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Flush_Timeout: *Size: 2 Bytes*

Value	Parameter description
0	Timeout = ∞; No Automatic Flush. **Default.**
N = 0xXXXX	Flush Timeout = *N* * 0.625 ms Size: 11 bits Range: 0x0001–0x07FF

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Automatic_Flush_Timeout command succeeded.
0x01–0xFF	Write_Automatic_Flush_Timeout command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Specifies which Connection Handle's Flush Timeout has been written. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Event(s) generated (unless masked away):

When the Write_Automatic_Flush_Timeout command has completed, a Command Complete event will be generated.

11.2.7.33 Read_Num_Broadcast_Retransmissions

Command	OCF	Command Parameters	Return parameters
HCI_Read_Num_Broadcast_ Retransmissions	0x0029	—	Status, Num_Broadcast_Retran

Description:

This command will read the device's parameter value for the Number of Broadcast Retransmissions. Broadcast packets are not acknowledged and are unreliable. The Number of Broadcast Retransmissions parameter is used to increase the reliability of a broadcast message by retransmitting the broadcast message multiple times. This parameter defines the number of times the device will retransmit a broadcast data packet. This parameter should be adjusted as the link quality measurement changes.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Num_Broadcast_Retransmissions command succeeded.
0x01–0xFF	Read_Num_Broadcast_Retransmissions command failed. See Table 109 for list of Error Codes.

Num_Broadcast_Retran: *Size: 1 Byte*

Value	Parameter description
N = 0xXX	Number of Broadcast Retransmissions = *N* Range 0x00–0xFF

Event(s) generated (unless masked away):

When the Read_Num_Broadcast_Retransmission command has completed, a Command Complete event will be generated.

11.2.7.34 Write_Num_Broadcast_Retransmissions

Command	OCF	Command parameters	Return parameters
HCI_Write_Num_Broadcast_ Retransmissions	0x002A	Num_Broadcast_Retran	Status

Description:

This command will write the device's parameter value for the Number of Broadcast Retransmissions. Broadcast packets are not acknowledged and are unreliable. The Number of Broadcast Retransmissions parameter is used to increase the reliability of a broadcast message by retransmitting the broadcast message multiple times. This parameter defines the number of times the device will retransmit a broadcast data packet. This parameter should be adjusted as link quality measurement changes.

Command Parameters:

Num_Broadcast_Retran: *Size: 1 Byte*

Value	Parameter description
N = 0xXX	Number of Broadcast Retransmissions = *N* Range 0x00–0xFF Default: *N* = 0x01

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Num_Broadcast_Retransmissions command succeeded.
0x01–0xFF	Write_Num_Broadcast_Retransmissions command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Num_Broadcast_Retransmissions command has completed, a Command Complete event will be generated.

11.2.7.35 Read_Hold_Mode_Activity

Command	OCF	Command Parameters	Return parameters
HCI_Read_Hold_Mode_Activity	0x002B	—	Status, Hold_Mode_Activity

Description:

This command will read the value for the Hold_Mode_Activity parameter. The Hold_Mode_Activity value is used to determine what activities should be suspended when the device is in hold mode. After the hold period has expired, the device will return to the previous mode of operation. Multiple hold mode activities may be specified for the Hold_Mode_Activity parameter by performing a bitwise OR operation of the different activity types. If no activities are suspended, then all of the current Periodic Inquiry, Inquiry Scan, and Page Scan settings remain valid during the Hold Mode. If the Hold_Mode_Activity parameter is set to Suspend Page Scan, Suspend Inquiry Scan, and Suspend Periodic Inquiries, then the device can enter a low-power state during the Hold Mode period, and all activities are suspended. Suspending multiple activities can be specified for the Hold_Mode_Activity parameter by performing a bitwise OR operation of the different activity types.The Hold Mode Activity is only valid if all connections are in Hold Mode.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Hold_Mode_Activity command succeeded.
0x01–0xFF	Read_Hold_Mode_Activity command failed. See Table 109 for list of Error Codes.

Hold_Mode_Activity: *Size: 1 Byte*

Value	Parameter description
0x00	Maintain current Power State.
0x01	Suspend Page Scan.
0x02	Suspend Inquiry Scan.
0x04	Suspend Periodic Inquiries.
0x08–0xFF	Reserved for future use.

Event(s) generated (unless masked away):

When the Read_Hold_Mode_Activity command has completed, a Command Complete event will be generated.

11.2.7.36 Write_Hold_Mode_Activity

Command	OCF	Command parameters	Return parameters
HCI_Write_Hold_Mode_Activity	0x002C	Hold_Mode_Activity	Status

Description:

This command will write the value for the Hold_Mode_Activity parameter. The Hold_Mode_Activity value is used to determine what activities should be suspended when the device is in hold mode. After the hold period has expired, the device will return to the previous mode of operation. Multiple hold mode activities may be specified for the Hold_Mode_Activity parameter by performing a bitwise OR operation of the different activity types. If no activities are suspended, then all of the current Periodic Inquiry, Inquiry Scan, and Page Scan settings remain valid during the Hold Mode. If the Hold_Mode_Activity parameter is set to Suspend Page Scan, Suspend Inquiry Scan, and Suspend Periodic Inquiries, then the device can enter a low power state during the Hold Mode period and all activities are suspended. Suspending multiple activities can be specified for the Hold_Mode_Activity parameter by performing a bitwise OR operation of the different activity types. The Hold Mode Activity is only valid if all connections are in Hold Mode.

Command Parameters:

Hold_Mode_Activity: *Size: 1 Byte*

Value	Parameter description
0x00	Maintain current Power State. **Default.**
0x01	Suspend Page Scan.
0x02	Suspend Inquiry Scan.
0x04	Suspend Periodic Inquiries.
0x08–0xFF	Reserved for future use.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Hold_Mode_Activity command succeeded.
0x01–0xFF	Write_Hold_Mode_Activity command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Hold_Mode_Activity command has completed, a Command Complete event will be generated.

11.2.7.37 Read_Transmit_Power_Level

Command	OCF	Command parameters	Return parameters
HCI_Read_Transmit_ Power_Level	0x002D	Connection_Handle, Type	Status, Connection_Handle, Transmit_Power_Level

Description:

This command will read the values for the Transmit_Power_Level parameter for the specified Connection Handle. The Connection_Handle shall be a Connection_Handle for an ACL connection.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Specifies which Connection Handle's Transmit Power Level setting to read. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Type: *Size: 1 Byte*

Value	Parameter description
0x00	Read Current Transmit Power Level.
0x01	Read Maximum Transmit Power Level.
0x02–0xFF	Reserved.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Transmit_Power_Level command succeeded.
0x01–0xFF	Read_Transmit_Power_Level command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Specifies which Connection Handle's Transmit Power Level setting is returned. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Transmit_Power_Level: *Size: 1 Byte*

Value	Parameter description
N = 0xXX	Size: 1 Byte (signed integer) Range: -30 ≤ *N* ≤ 20 Units: dBm

Event(s) generated (unless masked away):

When the Read_Transmit_Power_Level command has completed, a Command Complete event will be generated.

11.2.7.38 Read_SCO_Flow_Control_Enable

Command	OCF	Command parameters	Return parameters
HCI_Read_SCO_ Flow_Control_Enable	0x002E	—	Status, SCO_Flow_Control_Enable

Description:

The Read_SCO_Flow_Control_Enable command provides the ability to read the SCO_Flow_Control_Enable setting. By using this setting, the Host can decide if the Host Controller will send Number Of Completed Packets events for SCO Connection Handles. This setting allows the Host to enable and disable SCO flow control.

Note that the SCO_Flow_Control_Enable setting can only be changed if no connections exist.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_SCO_Flow_Control_Enable command succeeded.
0x01–0xFF	Read_SCO_Flow_Control_Enable command failed see Table 109 for list of Error Codes.

SCO_Flow_Control_Enable: *Size: 1 Byte*

Value	Parameter description
0x00	SCO Flow Control is disabled. No Number of Completed Packets events will be sent from the Host Controller for SCO Connection Handles.
0x01	SCO Flow Control is enabled. Number of Completed Packets events will be sent from the Host Controller for SCO Connection Handles.

Event(s) generated (unless masked away):

When the Read_SCO_Flow_Control_Enable command has completed a Command Complete event will be generated.

11.2.7.39 Write_SCO_Flow_Control_Enable

Command	OCF	Command parameters	Return parameters
HCI_Write_SCO_Flow_Control_Enable	0x002F	SCO_Flow_Control_Enable	Status

Description:

The Write_SCO_Flow_Control_Enable command provides the ability to write the SCO_Flow_Control_Enable setting. By using this setting, the Host can decide if the Host Controller will send Number Of Completed Packets events for SCO Connection Handles. This setting allows the Host to enable and disable SCO flow control.

Note that the SCO_Flow_Control_Enable setting can only be changed if no connections exist.

Command Parameters:

SCO_Flow_Control_Enable: *Size: 1 Byte*

Value	Parameter description
0x00	SCO Flow Control is disabled. No Number of Completed Packets events will be sent from the Host Controller for SCO Connection Handles. Default.
0x01	SCO Flow Control is enabled. Number of Completed Packets events will be sent from the Host Controller for SCO Connection Handles.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_SCO_Flow_Control_Enable command succeeded.
0x01–0xFF	Write_SCO_Flow_Control_Enable command failed see Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_SCO_Flow_Control_Enable command has completed a Command Complete event will be generated.

11.2.7.40 Set_Host_Controller_To_Host_Flow_Control

Command	OCF	Command parameters	Return parameters
HCI_Set_Host_Controller_To_Host_Flow_Control	0x0031	Flow_Control_Enable	Status

Description:

This command is used by the Host to turn flow control on or off for data and/or voice sent in the direction from the Host Controller to the Host. If flow control is turned off, the Host should not send the Host_Number_Of_Completed_Packets command. That command will be ignored by the Host Controller if it is sent by the Host and flow control is off. If flow control is turned on for HCI ACL Data Packets and off for HCI SCO Data Packets, Host_Number_Of_Completed_Packets commands sent by the Host should only contain Connection Handles for ACL connections. If flow control is turned off for HCI ACL Data Packets and on for HCI SCO Data Packets, Host_Number_Of_Completed_Packets commands sent by the Host should only contain Connection Handles for SCO connections. If flow control is turned on for HCI ACL Data Packets and HCI SCO Data Packets, the Host will send Host_Number_Of_Completed_Packets commands both for ACL connections and SCO connections. Note that the Flow_Control_Enable setting must only be changed if no connections exist.

Command Parameters:

Flow_Control_Enable: *Size: 1 Byte*

Value	Parameter description
0x00	Flow control **off** in direction from Host Controller to Host. **Default.**
0x01	Flow control **on** for HCI ACL Data Packets and **off** for HCI SCO Data Packets in direction from Host Controller to Host.
0x02	Flow control **off** for HCI ACL Data Packets and **on** for HCI SCO Data Packets in direction from Host Controller to Host.
0x03	Flow control **on** both for HCI ACL Data Packets and HCI SCO Data Packets in direction from Host Controller to Host.
0x04–0xFF	Reserved.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Set_Host_Controller_To_Host_Flow_Control command succeeded.
0x01–0xFF	Set_Host_Controller_To_Host_Flow_Control command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Set_Host_Controller_To_Host_Flow_Control command has completed, a Command Complete event will be generated.

11.2.7.41 Host_Buffer_Size

Command	OCF	Command parameters	Return Parameters
HCI_Host_Buffer_Size	0x0033	Host_ACL_Data_Packet_Length, Host_SCO_Data_Packet_Length, Host_Total_Num_ACL_Data_Packets, Host_Total_Num_SCO_Data_Packets	Status

Description:

The Host_Buffer_Size command is used by the Host to notify the Host Controller about the maximum size of the data portion of HCI ACL and SCO Data Packets sent from the Host Controller to the Host. The Host Controller will segment the data to be transmitted from the Host Controller to the Host according to these sizes, so that the HCI Data Packets will contain data with up to these sizes. The Host_Buffer_Size command also notifies the Host Controller about the total number of HCI ACL and SCO Data Packets that can be stored in the data buffers of the Host. If flow control from the Host Controller to the Host is turned off, and the Host_Buffer_Size command has not been issued by the Host, this means that the Host Controller will send HCI Data Packets to the Host with any lengths the Host Controller wants to use, and it is assumed that the data buffer sizes of the Host are unlimited. If flow control from the Host controller to the Host is turned on, the Host_Buffer_Size command must after a power-on or a reset always be sent by the Host before the first Host_Number_Of_Completed_Packets command is sent.

(The Set_Host_Controller_To_Host_Flow_Control command is used to turn flow control on or off.) The Host_ACL_Data_Packet_Length command parameter will be used to determine the size of the L2CAP segments contained in ACL Data Packets, which are transferred from the Host Controller to the Host. The Host_SCO_Data_Packet_Length command parameter is used to determine the maximum size of HCI SCO Data Packets. Both the Host and the Host Controller shall support command and event packets, where the data portion (excluding header) contained in the packets is 255 bytes in size.

The Host_Total_Num_ACL_Data_Packets command parameter contains the total number of HCI ACL Data Packets that can be stored in the data buffers of the Host. The Host Controller will determine how the buffers are to be divided between different Connection Handles. The Host_Total_Num_SCO_Data_Packets command parameter gives the same information for HCI SCO Data Packets.

Note that the Host_ACL_Data_Packet_Length and Host_SCO_Data_Packet_Length command parameters do not include the length of the HCI Data Packet header.

Command Parameters:

Host_ACL_Data_Packet_Length: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Maximum length (in bytes) of the data portion of each HCI ACL Data Packet that the Host is able to accept.

Host_SCO_Data_Packet_Length: *Size: 1 Byte*

Value	Parameter description
0xXX	Maximum length (in bytes) of the data portion of each HCI SCO Data Packet that the Host is able to accept.

Host_Total_Num_ACL_Data_Packets: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Total number of HCI ACL Data Packets that can be stored in the data buffers of the Host.

Host_Total_Num_SCO_Data_Packets: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Total number of HCI SCO Data Packets that can be stored in the data buffers of the Host.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Host_Buffer_Size command succeeded.
0x01–0xFF	Host_Buffer_Size command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Host_Buffer_Size command has completed, a Command Complete event will be generated.

11.2.7.42 Host_Number_Of_Completed_Packets

Command	OCF	Command parameters	Return Parameters
HCI_Host_Number_Of_ Completed_Packets	0x0035	Number_Of_Handles, Connection_ Handle[i], Host_Num_Of_Completed_Packets [i]	

Description:

The Host_Number_Of_Completed_Packets command is used by the Host to indicate to the Host Controller the number of HCI Data Packets that have been completed for each Connection Handle since the previous Host_Number_Of_Completed_Packets command was sent to the Host Controller. This means that the corresponding buffer space has been freed in the Host. Based on this information, and the Host_Total_Num_ACL_Data_Packets and Host_Total_Num_SCO_Data_Packets command parameters of the Host_Buffer_Size command, the Host Controller can determine for which Connection Handles the following HCI Data Packets should be sent to the Host. The command should only be issued by the Host if

flow control in the direction from the Host Controller to the Host is on and there is at least one connection, or if the Host Controller is in local loopback mode. Otherwise, the command will be ignored by the Host Controller. While the Host has HCI Data Packets in its buffers, it must keep sending the Host_Number_Of_Completed_Packets command to the Host Controller at least periodically, until it finally reports that all buffer space in the Host used by ACL Data Packets has been freed. The rate with which this command is sent is manufacturer specific.

(The Set_Host_Controller_To_Host_Flow_Control command is used to turn flow control on or off.) If flow control from the Host controller to the Host is turned on, the Host_Buffer_Size command must after a power-on or a reset always be sent by the Host before the first Host_Number_Of_Completed_Packets command is sent.

Note that the Host_Number_Of_Completed_Packets command is a special command in the sense that no event is normally generated after the command has completed. The command may be sent at any time by the Host when there is at least one connection, or if the Host Controller is in local loopback mode independent of other commands. The normal flow control for commands is not used for the Host_Number_Of_Completed_Packets command.

Command Parameters:

Number_Of_Handles: *Size: 1 Byte*

Value	Parameter description
0xXX	The number of Connection Handles and Host_Num_Of_Completed_Packets parameters pairs contained in this command. Range: 0–255

Connection_Handle[i]: *Size: Number_Of_Handles*2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use)

Host_Num_Of_Completed_Packets [i]: *Size: Number_Of_Handles * 2 Bytes*

Value	Parameter description
N = 0xXXXX	The number of HCI Data Packets that have been completed for the associated Connection Handle since the previous time the event was returned. Range for *N*: 0x0000–0xFFFF

Return Parameters:

None.

Event(s) generated (unless masked away):

Normally, no event is generated after the Host_Number_Of_Completed_Packets command has completed. However, if the Host_Number_Of_Completed_Packets command contains one or more invalid parameters, the Host Controller will return a Command Complete event with a failure status indicating the Invalid HCI

Command Parameters error code. The Host may send the Host_Number_Of_Completed_Packets command at any time when there is at least one connection, or if the Host Controller is in local loopback mode. The normal flow control for commands is not used for this command.

11.2.7.43 Read_Link_Supervision_Timeout

Command	OCF	Command Parameters	Return parameters
HCI_Read_Link_Supervision_ Timeout	0x0036	Connection_Handle	Status, Connection_Handle, Link_Supervision_ Timeout

Description:

This command will read the value for the Link_Supervision_Timeout parameter for the device. The Link_Supervision_Timeout parameter is used by the master or slave Bluetooth device to monitor link loss. If, for any reason, no Baseband packets are received from that Connection Handle for a duration longer than the Link_Supervision_Timeout, the connection is disconnected. The same timeout value is used for both SCO and ACL connections for the device specified by the Connection Handle.

Note that the Connection_Handle used for this command shall be the ACL connection to the appropriate device. This command will set the Link_Supervision_Timeout values for other SCO Connection_Handle to that device.

Note that setting the Link_Supervision_Timeout to No Link_Supervision_Timeout (0x0000) will disable the Link_Supervision_Timeout check for the specified Connection Handle. This makes it unnecessary for the master of the piconet to unpark and then park each Bluetooth Device every ~40 s. By using the No Link_Supervision_Timeout setting, the scalability of the Park mode is not limited.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Specifies which Connection Handle's Link Supervision Timeout value is to be read. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Link_Supervision_Timeout command succeeded.
0x01–0xFF	Read_Link_Supervision_Timeout command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Specifies which Connection Handle's Link Supervision Timeout value was read. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Link_Supervision_Timeout: *Size: 2 Bytes*

Value	Parameter description
0x0000	No Link_Supervision_Timeout.
N = 0xXXXX	Measured in Number of Baseband slots Link_Supervision_Timeout = *N* * 0.625 ms (1 Baseband slot) Range for *N*: 0x0001–0xFFFF Time Range: 0.625ms–40.9 s

Event(s) generated (unless masked away):

When the Read_Link_Supervision_Timeout command has completed, a Command Complete event will be generated.

11.2.7.44 Write_Link_Supervision_Timeout

Command	OCF	Command parameters	Return parameters
HCI_Write_Link_Supervision_Timeout	0x0037	Connection_Handle, Link_Supervision_Timeout	Status, Connection_Handle

Description:

This command will write the value for the Link_Supervision_Timeout parameter for the device. The Link_Supervision_Timeout parameter is used by the master or slave Bluetooth device to monitor link loss. If, for any reason, no Baseband packets are received from that Connection_Handle for a duration longer than the Link_Supervision_Timeout, the connection is disconnected. The same timeout value is used for both SCO and ACL connections for the device specified by the Connection_Handle.

Note that the Connection_Handle used for this command shall be the ACL connection to the appropriate device. This command will set the Link_Supervision_Timeout values for other SCO Connection_Handle to that device.

Note that setting the Link_Supervision_Timeout parameter to No Link_Supervision_Timeout (0x0000) will disable the Link_Supervision_Timeout check for the specified Connection Handle. This makes it unnecessary for the master of the piconet to unpark and then park each Bluetooth Device every ~40 s. By using the No Link_Supervision_Timeout setting, the scalability of the Park mode is not limited.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Specifies which Connection Handle's Link Supervision Timeout value is to be written. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Link_Supervision_Timeout: *Size: 2 Bytes*

Value	Parameter description
0x0000	No Link_Supervision_Timeout.
N = 0xXXXX	Measured in Number of Baseband slots Link_Supervision_Timeout = *N* * 0.625 ms (1 Baseband slot) Range for *N*: 0x0001–0xFFFF Time Range: 0.625 ms–40.9 s **Default**: *N* = 0x7D00 Link_Supervision_Timeout = 20 s

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Link_Supervision_Timeout command succeeded.
0x01–0xFF	Write_Link_Supervision_Timeout command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Specifies which Connection Handle's Link Supervision Timeout value was written. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Event(s) generated (unless masked away):

When the Write_Link_Supervision_Timeout command has completed, a Command Complete event will be generated.

11.2.7.45 Read_Number_Of_Supported_IAC

Command	OCF	Command parameters	Return parameters
HCI_Read_Number_Of_Supported_IAC	0x0038	—	Status, Num_Support_IAC

Description:

This command will read the value for the number of IACs that the local Bluetooth device can simultaneous listen for during an Inquiry Scan. All Bluetooth devices are required to support at least one IAC, the General Inquiry Access Code (GIAC). Some Bluetooth devices support additional IACs.

Command Parameters:

None

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Number_Of_Supported_IAC command succeeded.
0x01–0xFF	Read_Number_Of_Supported_IAC command failed. See Table 109 for list of Error Codes.

Num_Support_IAC *Size: 1 Byte*

Value	Parameter description
0xXX	Specifies the number of Supported IAC that the local Bluetooth device can simultaneous listen for during an Inquiry Scan. Range: 0x01–0x40

Event(s) generated (unless masked away):

When the Read_Number_Of_Supported_IAC command has completed, a Command Complete event will be generated.

11.2.7.46 Read_Current_IAC_LAP

Command	OCF	Command parameters	Return parameters
HCI_Read_Current_IAC_LAP	0x0039	—	Status, Num_Current_IAC, IAC_LAP[i]

Description:

This command reads the LAP(s) used to create the IAC(s) that the local Bluetooth device is simultaneously scanning for during Inquiry Scans. All Bluetooth devices are required to support at least one IAC, the GIAC. Some Bluetooth devices support additional IACs.

Command Parameters:

None

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Current_IAC_LAP command succeeded.
0x01–0xFF	Read_Current_IAC_LAP command failed. See Table 109 for list of Error Codes.

Num_Current_IAC *Size: 1 Byte*

Value	Parameter description
0xXX	Specifies the number of IACs which are currently in use by the local Bluetooth device to simultaneously listen for during an Inquiry Scan. Range: 0x01–0x40

IAC_LAP[i] *Size: 3 Bytes * Num_Current_IAC*

Value	Parameter description
0xXXXXXX	LAPs used to create the IAC which is currently in use by the local Bluetooth device to simultaneously listen for during an Inquiry Scan. Range: 0x9E8B00–0x9E8B3F

Event(s) generated (unless masked away):

When the Read_Current_IAC_LAP command has completed, a Command Complete event will be generated.

11.2.7.47 Write_Current_IAC_LAP

Command	OCF	Command Parameters	Return parameters
HCI_Write_Current_IAC_LAP	0x003A	Num_Current_IAC, IAC_LAP[i]	Status

Description:

This command writes the LAP(s) used to create the IAC that the local Bluetooth device is simultaneously scanning for during Inquiry Scans. All Bluetooth devices are required to support at least one IAC, the GIAC. Some Bluetooth devices support additional IACs.

Note that this command writes over the current IACs used by the Bluetooth device. If the value of the Num_Current_IAC is more than the number of supported IACs, then only the first, X IAC (where X equals the number of supported IACs) will be stored without any error.

Command Parameters:

Num_Current_IAC *Size: 1 Byte*

Value	Parameter description
0xXX	Specifies the number of IACs which are currently in use by the local Bluetooth device to simultaneously listen for during an Inquiry Scan. Range: 0x01–0x40

IAC_LAP[i] *Size: 3 Bytes * Num_Current_IAC*

Value	Parameter description
0xXXXXXX	LAP(s) used to create IAC which is currently in use by the local Bluetooth device to simultaneously listen for during an Inquiry Scan. Range: 0x9E8B00–0x9E8B3F. The GIAC is the default IAC to be used. If additional IACs are supported, additional default IAC will be determined by the manufacturer.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Current_IAC_LAP command succeeded.
0x01–0xFF	Write_Current_IAC_LAP command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Current_IAC_LAP command has completed, a Command Complete event will be generated.

11.2.7.48 Read_Page_Scan_Period_Mode

Command	OCF	Command Parameters	Return parameters
HCI_Read_Page_Scan_Period_ Mode	0x003B	—	Status, Page_Scan_Period_Mode

Description:

This command is used to read the mandatory Page_Scan_Period_Mode of the local Bluetooth device. Every time an inquiry response message is sent, the Bluetooth device will start a timer (T_mandatory_pscan), the value of which is dependent on the Page_Scan_Period_Mode. As long as this timer has not expired, the Bluetooth device will use the Page_Scan_Period_Mode for all following page scans.

Note that the timer T_mandatory_pscan will be reset at each new inquiry response. For details, see Clause 8 (Keyword: SP-Mode, FHS-Packet, T_mandatory_pscan, Inquiry-Response).

After transmitting one or more inquiry response (FHS) packets as a result of the inquiry scan process, the local Bluetooth device is allowed to enter the page scan state using mandatory page scan mode regardless of the setting of the Scan_Enable parameter.

Command Parameters:

None

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Page_Scan_Period_Mode command succeeded.
0x01–0xFF	Read_Page_Scan_Period_Mode command failed. See Table 109 for list of Error Codes.

Page_Scan_Period_Mode: *Size: 1 Byte*

Value	Parameter description
0x00	P0
0x01	P1
0x02	P2
0x03–0xFF	Reserved.

Event(s) generated (unless masked away):

When the Read_Page_Scan_Period_Mode command has completed, a Command Complete event will be generated.

11.2.7.49 Write_Page_Scan_Period_Mode

Command	OCF	Command parameters	Return parameters
HCI_Write_Page_Scan_Period_Mode	0x003C	Page_Scan_Period_Mode	Status

Description:

This command is used to write the mandatory Page_Scan_Period_Mode of the local Bluetooth device. Every time an inquiry response message is sent, the Bluetooth device will start a timer (T_mandatory_pscan), the value of which is dependent on the Page_Scan_Period_Mode. As long as this timer has not expired, the Bluetooth device will use the Page_Scan_Period_Mode for all following page scans.
Note that the timer T_mandatory_pscan will be reset at each new inquiry response. For details, see Clause 8 (Keyword: SP-Mode, FHS-Packet, T_mandatory_pscan, Inquiry-Response).

After transmitting one or more inquiry response (FHS) packets as a result of the inquiry scan process, the local Bluetooth device is allowed to enter the page scan state using mandatory page scan mode regardless of the setting of the Scan_Enable parameter.

Command Parameters:

Page_Scan_Period_Mode: *Size: 1 Byte*

Value	Parameter description
0x00	P0. **Default.**
0x01	P1
0x02	P2
0x03–0xFF	Reserved.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Page_Scan_Period_Mode command succeeded.
0x01–0xFF	Write_Page_Scan_Period_Mode command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Page_Scan_Period_Mode command has completed, a Command Complete event will be generated.

11.2.7.50 Read_Page_Scan_Mode

Command	OCF	Command Parameters	Return parameters
HCI_Read_Page_Scan_Mode	0x003D	—	Status, Page_Scan_Mode

Description:

This command is used to read the default page scan mode of the local Bluetooth device. The Page_Scan_Mode parameter indicates the page scan mode that is used for default page scan. Currently one mandatory page scan mode and three optional page scan modes are defined. Following an inquiry response, if the Baseband timer T_mandatory_pscan has not expired, the mandatory page scan mode shall be applied. For details, see Clause 8 (Keyword: Page-Scan-Mode, FHS-Packet, T_mandatory_pscan).

Command Parameters:

None

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Page_Scan_Mode command succeeded.
0x01–0xFF	Read_Page_Scan_Mode command failed. See Table 109 for list of Error Codes.

Page_Scan_Mode: *Size: 1 Byte*

Value	Parameter description
0x00	Mandatory Page Scan Mode.
0x01	Optional Page Scan Mode I.
0x02	Optional Page Scan Mode II.
0x03	Optional Page Scan Mode III.
0x04–0xFF	Reserved.

Event(s) generated (unless masked away):

When the Read_Page_Scan_Mode command has completed, a Command Complete event will be generated.

11.2.7.51 Write_Page_Scan_Mode

Command	OCF	Command parameters	Return Parameters
HCI_Write_Page_Scan_Mode	0x003E	Page_Scan_Mode	Status

Description:

This command is used to write the default page scan mode of the local Bluetooth device. The Page_Scan_Mode parameter indicates the page scan mode that is used for the default page scan. Currently, one mandatory page scan mode and three optional page scan modes are defined. Following an inquiry response, if the Baseband timer T_mandatory_pscan has not expired, the mandatory page scan mode shall be applied. For details see Clause 8 (Keyword: Page-Scan-Mode, FHS-Packet, T_mandatory_pscan).

Command Parameters:

Page_Scan_Mode: *Size: 1 Byte*

Value	Parameter description
0x00	Mandatory Page Scan Mode. **Default**.
0x01	Optional Page Scan Mode I.
0x02	Optional Page Scan Mode II.
0x03	Optional Page Scan Mode III.
0x04–0xFF	Reserved.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Page_Scan_Mode command succeeded.
0x01–0xFF	Write_Page_Scan_Mode command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Page_Scan_Mode command has completed, a Command Complete event will be generated.

11.2.8 Informational parameters

The Informational Parameters are fixed by the manufacturer of the Bluetooth hardware. These parameters provide information about the Bluetooth device and the capabilities of the Host Controller, Link Manager, and Baseband. The host device cannot modify any of these parameters. For Informational Parameters Commands, the OGF is defined as 0x04.

Command	Command summary description
Read_Local_Version_Information	The Read_Local_Version_Information command will read the values for the version information for the local Bluetooth device.
Read_Local_Supported_Features	The Read_Local_Supported_Features command requests a list of the supported features for the local device.
Read_Buffer_Size	The Read_Buffer_Size command returns the size of the HCI buffers. These buffers are used by the Host Controller to buffer data that is to be transmitted.
Read_Country_Code	The Read_Country_Code command will read the value for the Country Code status parameter. The Country Code defines which range of frequency band of the ISM 2.4 GHz band will be used by the device.
Read_BD_ADDR	The Read_BD_ADDR command will read the value for the BD_ADDR parameter. The BD_ADDR is a 48-bit unique identifier for a Bluetooth device.

11.2.8.1 Read_Local_Version_Information

Command	OCF	Command Parameters	Return parameters
HCI_Read_Local_Version_ Information	0x0001	—	Status, HCI Version, HCI Revision, LMP Version, Manufacturer_Name, LMP Subversion

Description:

This command will read the values for the version information for the local Bluetooth device. The version information consists of two parameters: the version and revision parameters.

The version parameter defines the major hardware version of the Bluetooth hardware. The version parameter only changes when new versions of the Bluetooth hardware are produced for new Bluetooth SIG specifications. The version parameter is controlled by the Bluetooth SIG, Inc.

The revision parameter should be controlled by the manufacturer and should be changed as needed. The Manufacturer_Name parameter indicates the manufacturer of the local Bluetooth module as specified by the LMP definition of this parameter. The subversion parameter should be controlled by the manufacturer and should be changed as needed. The subversion parameter defines the various revisions that each version of the Bluetooth hardware will go through as design processes change and errors are fixed. This allows the software to determine what Bluetooth hardware is being used, and to work around various bugs in the hardware if necessary.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Local_Version_Information command succeeded.
0x01–0xFF	Read_Local_Version_Information command failed. See Table 109 for list of Error Codes.

HCI_Version: *Size: 1 Byte*

Value	Parameter description
0x00	Bluetooth HCI Specification 1.0B.
0x01	Bluetooth HCI Specification 1.1.
0x02–0xfFF	Reserved for future use.

HCI_Revision: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Revision of the Current HCI in the Bluetooth hardware.

LMP_Version: *Size: 1 Byte*

Value	Parameter description
0xXX	Version of the Current LMP in the Bluetooth Hardware, see Table 68 in the Link Manager Protocol for assigned values (VersNr).

Manufacturer_Name: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Manufacturer Name of the Bluetooth Hardware, see Table 68 in the Link Manager Protocol for assigned values (CompId).

LMP_Subversion: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Subversion of the Current LMP in the Bluetooth Hardware, see Table 68 in the Link Manager Protocol for assigned values (SubVersNr).

Event(s) generated (unless masked away):

When the Read_Local_Version_Information command has completed, a Command Complete event will be generated.

11.2.8.2 Read_Local_Supported_Features

Command	OCF	Command Parameters	Return parameters
HCI_Read_Local_Supported_Features	0x0003	—	Status, LMP_Features

Description:

This command requests a list of the supported features for the local device. This command will return a list of the LMP features. For details, see Clause 9.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Local_Supported_Features command succeeded.
0x01–0xFF	Read_Local_Supported_Features command failed. See Table 109 for list of Error Codes.

LMP_Features: *Size: 8 Bytes*

Value	Parameter description
0xXXXXXXXX XXXXXXXX	Bit Mask List of LMP features. For details, see Clause 9.

Event(s) generated (unless masked away):

When the Read_Local_Supported_Features command has completed, a Command Complete event will be generated.

11.2.8.3 Read_Buffer_Size

Command	OCF	Command Parameters	Return parameters
HCI_Read_Buffer_Size	0x0005	—	Status, HC_ACL_Data_Packet_Length, HC_SCO_Data_Packet_Length, HC_ Total_Num_ACL_Data_Packets, HC_Total_Num_SCO_Data_Packets

Description:

The Read_Buffer_Size command is used to read the maximum size of the data portion of HCI ACL and SCO Data Packets sent from the Host to the Host Controller. The Host will segment the data to be transmitted from the Host to the Host Controller according to these sizes, so that the HCI Data Packets will contain data with up to these sizes. The Read_Buffer_Size command also returns the total number of HCI ACL and SCO Data Packets that can be stored in the data buffers of the Host Controller. The Read_Buffer_Size command must be issued by the Host before it sends any data to the Host Controller.

The HC_ACL_Data_Packet_Length return parameter will be used to determine the size of the L2CAP segments contained in ACL Data Packets, which are transferred from the Host to the Host Controller to be broken up into baseband packets by the Link Manager. The HC_SCO_Data_Packet_Length return parameter is used to determine the maximum size of HCI SCO Data Packets. Both the Host and the Host Controller must support command and event packets, where the data portion (excluding header) contained in the packets is 255 bytes in size. The HC_Total_Num_ACL_Data_Packets return parameter contains the total number of HCI ACL Data Packets that can be stored in the data buffers of the Host Controller. The Host will determine how the buffers are to be divided between different Connection Handles. The HC_Total_Num_SCO_Data_Packets return parameter gives the same information but for HCI SCO Data Packets.

Note that the HC_ACL_Data_Packet_Length and HC_SCO_Data_Packet_Length return parameters do not include the length of the HCI Data Packet header.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Buffer_Size command succeeded.
0x01–0xFF	Read_Buffer_Size command failed. See Table 109 for list of Error Codes.

HC_ACL_Data_Packet_Length: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Maximum length (in bytes) of the data portion of each HCI ACL Data Packet that the Host Controller is able to accept.

HC_SCO_Data_Packet_Length: *Size: 1 Byte*

Value	Parameter description
0xXX	Maximum length (in bytes) of the data portion of each HCI SCO Data Packet that the Host Controller is able to accept.

HC_Total_Num_ACL_Data_Packets: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Total number of HCI ACL Data Packets that can be stored in the data buffers of the Host Controller.

HC_Total_Num_SCO_Data_Packets: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Total number of HCI SCO Data Packets that can be stored in the data buffers of the Host Controller.

Event(s) generated (unless masked away):

When the Read_Buffer_Size command has completed, a Command Complete event will be generated.

11.2.8.4 Read_Country_Code

Command	OCF	Command parameters	Return parameters
HCI_Read_Country_Code	0x0007	—	Status, Country_Code

Description:

This command will read the value for the Country_Code return parameter. The Country_Code defines which range of frequency band of the ISM 2.4 GHz band will be used by the device. Each country has local regulatory bodies regulating which ISM 2.4 GHz frequency ranges can be used.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Country_Code command succeeded.
0x01–0xFF	Read_Country_Code command failed. See Table 109 for list of Error Codes.

Country_Code: *Size: 1 Byte*

Value	Parameter description
0x00	North America, Europe (except France), and Japan
0x01	France
0x04–FF	Reserved for future use.

Event(s) generated (unless masked away):

When the Read_Country_Code command has completed, a Command Complete event will be generated.

11.2.8.5 Read_BD_ADDR

Command	OCF	Command parameters	Return parameters
HCI_Read_BD_ADDR	0x0009	—	Status, BD_ADDR

Description:

This command will read the value for the BD_ADDR parameter. The BD_ADDR is a 48-bit unique identifier for a Bluetooth device. See Clause 8 for details of how BD_ADDR is used.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_BD_ADDR command succeeded.
0x01–0xFF	Read_BD_ADDR command failed. See Table 109 for list of Error Codes.

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device

Event(s) generated (unless masked away):

When the Read_BD_ADDR command has completed, a Command Complete event will be generated.

11.2.9 Status parameters

The Host Controller modifies all status parameters. These parameters provide information about the current state of the Host Controller, Link Manager, and Baseband. The host device cannot modify any of these parameters other than to reset certain specific parameters. For the Status and baseband, the OGF is defined as 0x05.

Command	Command summary description
Read_Failed_Contact_Counter	The Read_Failed_Contact_Counter will read the value for the Failed_Contact_Counter parameter for a particular connection to another device. The Failed_Contact_Counter records the number of consecutive incidents in which either the slave or master did not respond after the flush timeout had expired, and the L2CAP packet that was currently being transmitted was automatically “flushed”.
Reset_Failed_Contact_Counter	The Reset_Failed_Contact_Counter will reset the value for the Failed_Contact_Counter parameter for a particular connection to another device. The Failed_Contact_Counter records the number of consecutive incidents in which either the slave or master did not respond after the flush timeout had expired and the L2CAP packet that was currently being transmitted was automatically “flushed”.
Get_Link_Quality	The Get_Link_Quality command will read the value for the Link_Quality for the specified Connection Handle.
Read_RSSI	The Read_RSSI command will read the value for the Received Signal Strength Indication (RSSI) for a connection handle to another Bluetooth device.

11.2.9.1 Read_Failed_Contact_Counter

Command	OCF	Command parameters	Return parameters
HCI_Read_Failed_ Contact_Counter	0x0001	Connection_Handle	Status, Connection_Handle, Failed_Contact_Counter

Description:

This command will read the value for the Failed_Contact_Counter parameter for a particular connection to another device. The Connection_Handle shall be a Connection_Handle for an ACL connection. The Failed_Contact_Counter records the number of consecutive incidents in which either the slave or master did not respond after the flush timeout had expired, and the L2CAP packet that was currently being transmitted was automatically "flushed". When this occurs, the Failed_Contact_Counter is incremented by 1. The Failed_Contact_Counter for a connection is reset to zero on the following conditions:

1) When a new connection is established
2) When the Failed_Contact_Counter is > 0 and an L2CAP packet is acknowledged for that connection
3) When the Reset_Failed_Contact_Counter command has been issued

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	The Connection Handle for the Connection for which the Failed Contact Counter should be read. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Failed_Contact_Counter command succeeded.
0x01–0xFF	Read_Failed_Contact_Counter command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	The Connection Handle for the Connection for which the Failed Contact Counter has been read. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Failed_Contact_Counter: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Number of consecutive failed contacts for a connection corresponding to the connection handle.

Event(s) generated (unless masked away):

When the Read_Failed_Contact_Counter command has completed, a Command Complete event will be generated.

11.2.9.2 Reset_Failed_Contact_Counter

Command	OCF	Command parameters	Return parameters
HCI_Reset_Failed_ Contact_Counter	0x0002	Connection_Handle	Status, Connection_Handle

Description:

This command will reset the value for the Failed_Contact_Counter parameter for a particular connection to another device. The Connection_Handle shall be a Connection_Handle for an ACL connection. The Failed_Contact_Counter records the number of consecutive incidents in which either the slave or master did not respond after the flush timeout had expired, and the L2CAP packet that was currently being transmitted was automatically "flushed". When this occurs, the Failed_Contact_Counter is incremented by 1. The Failed_Contact_Counter for a connection is reset to zero on the following conditions:

1) When a new connection is established
2) When the Failed_Contact_Counter is > 0 and an L2CAP packet is acknowledged for that connection
3) When the Reset_Failed_Contact_Counter command has been issued

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	The Connection Handle for the Connection for which the Failed Contact Counter should be reset. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Reset_Failed_Contact_Counter command succeeded.
0x01–0xFF	Reset_Failed_Contact_Counter command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	The Connection Handle for the Connection for which the Failed Contact Counter has been reset. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Event(s) generated (unless masked away):

When the Reset_Failed_Contact_Counter command has completed, a Command Complete event will be generated.

11.2.9.3 Get_Link_Quality

Command	OCF	Command parameters	Return parameters
HCI_Get_Link_Quality	0x0003	Connection_Handle	Status, Connection_Handle, Link_Quality

Description:

This command will return the value for the Link_Quality for the specified Connection Handle. The Connection_Handle shall be a Connection_Handle for an ACL connection. This command will return a Link_Quality value from 0–255, which represents the quality of the link between two Bluetooth devices. The higher the value, the better the link quality is. Each Bluetooth module vendor will determine how to measure the link quality.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	The Connection Handle for the connection for which link quality parameters are to be read. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Get_Link_Quality command succeeded.
0x01–0xFF	Get_Link_Quality command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	The Connection Handle for the connection for which the link quality parameter has been read. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Link_Quality: *Size: 1 Byte*

Value	Parameter description
0xXX	The current quality of the Link connection between the local device and the remote device specified by the Connection Handle. Range: 0x00–0xFF The higher the value, the better the link quality is.

Event(s) generated (unless masked away):

When the Get_Link_Quality command has completed, a Command Complete event will be generated.

11.2.9.4 Read_RSSI

Command	OCF	Command parameters	Return parameters
HCI_Read_RSSI	0x0005	Connection_Handle	Status, Connection_Handle,RSSI

Description:

This command will read the value for the difference between the measured RSSI and the limits of the Golden Receive Power Range (see 7.4.7) for a connection handle to another Bluetooth device. The Connection_Handle shall be a Connection_Handle for an ACL connection. Any positive RSSI value returned by the Host Controller indicates how many dB the RSSI is above the upper limit, any negative value indicates how many dB the RSSI is below the lower limit. The value zero indicates that the RSSI is inside the Golden Receive Power Range.

Note how accurate the dB values will be depends on the Bluetooth hardware. The only requirements for the hardware are that the Bluetooth device is able to tell whether the RSSI is inside, above, or below the Golden Device Power Range.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	The Connection Handle for the Connection for which the RSSI is to be read. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_RSSI command succeeded.
0x01–0xFF	Read_RSSI command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	The Connection Handle for the Connection for which the RSSI has been read. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

RSSI: *Size: 1 Byte*

Value	Parameter description
N = 0xXX	Size: 1 Byte (signed integer) Range: -128 ≤ *N* ≤ 127 Units: dB

Event(s) generated (unless masked away):

When the Read_RSSI command has completed, a Command Complete event will be generated.

11.2.10 Testing commands

The Testing commands are used to provide the ability to test various functionalities of the Bluetooth hardware. These commands provide the ability to arrange various conditions for testing. For the Testing Commands, the OGF is defined as 0x06.

Command	Command summary description
Read_Loopback_Mode	The Read_Loopback_Mode will read the value for the setting of the Host Controllers Loopback Mode. The setting of the Loopback Mode will determine the path of information.
Write_Loopback_Mode	The Write_Loopback_Mode will write the value for the setting of the Host Controllers Loopback Mode. The setting of the Loopback Mode will determine the path of information.
Enable_Device_Under_Test_Mode	The Enable_Device_Under_Test_Mode command will allow the local Bluetooth module to enter test mode via LMP test commands. The Host issues this command when it wants the local device to be the DUT for the Testing scenarios as described in the Bluetooth Test Mode document.

11.2.10.1 Read_Loopback_Mode

Command	OCF	Command parameters	Return parameters
HCI_Read_Loopback_Mode	0x0001	—	Status, Loopback_Mode

Description:

This command will read the value for the setting of the Host Controller's Loopback Mode. The setting of the Loopback Mode will determine the path of information. In Nontesting Mode operation, the Loopback Mode is set to Nontesting Mode and the path of the information is as specified by the Bluetooth specifications. In Local Loopback Mode, every Data Packet (ACL and SCO) and Command Packet that is sent from the Host to the Host Controller is sent back with no modifications by the Host Controller, as shown in Figure 138.

When the Bluetooth Host Controller enters Local Loopback Mode, it shall respond with four Connection Complete events, one for an ACL channel and three for SCO channels, so that the Host gets connection handles to use when sending ACL and SCO data. When in Local Loopback Mode the Host Controller loops back commands and data to the Host. The Loopback Command event is used to loop back commands that the Host sends to the Host Controller.

There are some commands that are not looped back in Local Loopback Mode: Reset, Set_Host_Controller_To_Host_Flow_Control, Host_Buffer_Size, Host_Number_Of_Completed_Packets, Read_Buffer_Size, Read_Loopback_Mode and Write_Loopback_Mode.These commands should be executed in the way they are normally executed. The commands Reset and Write_Loopback_Mode can be used to exit local loopback mode. If Write_Loopback_Mode is used to exit Local Loopback Mode, four Disconnection Complete events should be sent to the Host, corresponding to the Connection Complete events that were sent when entering Local Loopback Mode. Furthermore, no connections are allowed in

Local Loopback mode. If there is a connection and there is an attempt to set the device to Local Loopback Mode, the attempt will be refused. When the device is in Local Loopback Mode, the Host Controller will refuse incoming connection attempts. This allows the Host Controller Transport Layer to be tested without any other variables.

If a device is set to Remote Loopback Mode, it will send back all data (ACL and SCO) that comes over the air, and it will only allow a maximum of one ACL connection and three SCO connections, and these should be all to the same remote device. If there already are connections to more than one remote device and there is an attempt to set the local device to Remote Loopback Mode, the attempt will be refused. See Figure 139, where the rightmost device is set to Remote Loopback Mode and the leftmost device is set to Nontesting Mode. This allows the Bluetooth Air link to be tested without any other variables.

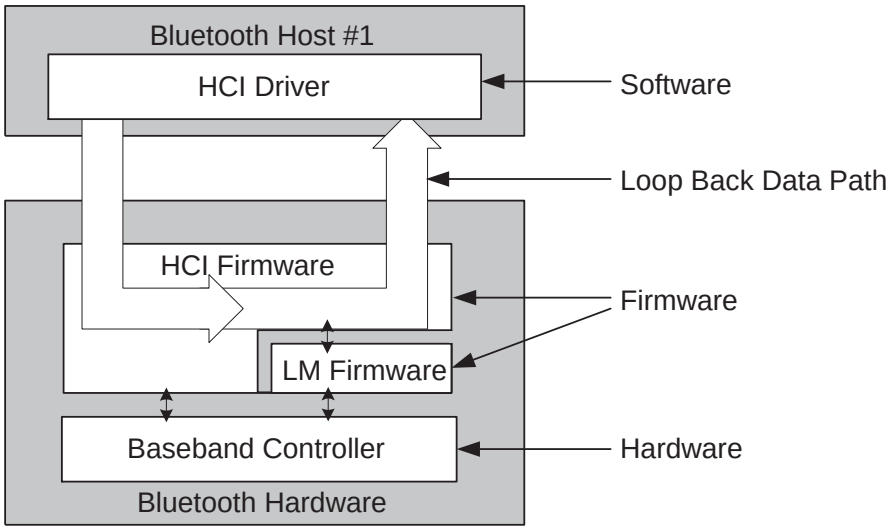


Figure 138—Local loopback mode

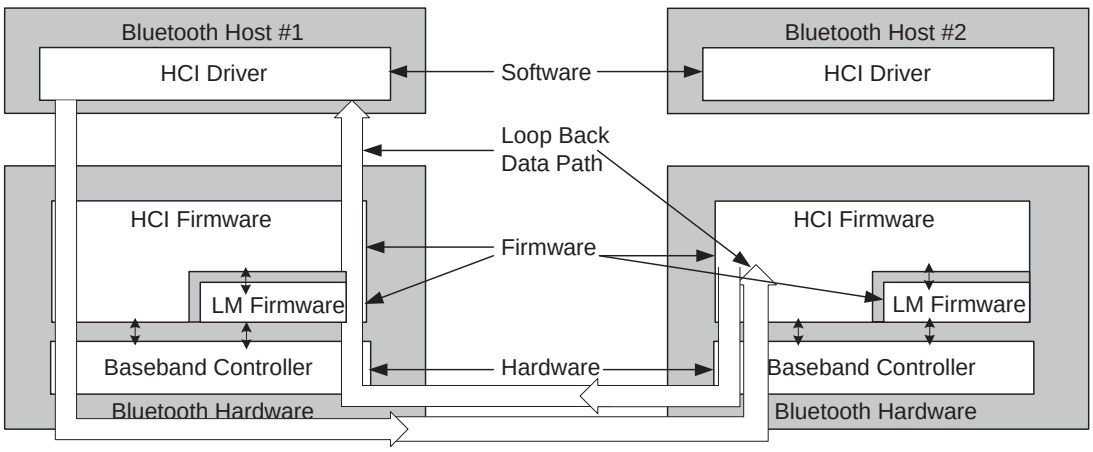


Figure 139—Remote loopback mode

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Loopback_Mode command succeeded.
0x01–0xFF	Read_Loopback_Mode command failed. See Table 109 for list of Error Codes.

Loopback_Mode: *Size: 1 Byte*

Value	Parameter description
0x00	No Loopback mode enabled. **Default.**
0x01	Enable Local Loopback.
0x02	Enable Remote Loopback.
0x03–0xFF	Reserved for future use.

Event(s) generated (unless masked away):

When the Read_Loopback_Mode command has completed, a Command Complete event will be generated.

11.2.10.2 Write_Loopback_Mode

Command	OCF	Command parameters	Return parameters
HCI_Write_Loopback_Mode	0x0002	Loopback_Mode	Status

Description:

This command will write the value for the setting of the Host Controller's Loopback Mode. The setting of the Loopback Mode will determine the path of information. In Nontesting Mode operation, the Loopback Mode is set to Nontesting Mode and the path of the information as specified by the Bluetooth specifications. In Local Loopback Mode, every Data Packet (ACL and SCO) and Command Packet that is sent from the Host to the Host Controller is sent back with no modifications by the Host Controller, as shown in Figure 140.

When the Bluetooth Host Controller enters Local Loopback Mode, it shall respond with four Connection Complete events, one for an ACL channel and three for SCO channels, so that the Host gets connection handles to use when sending ACL and SCO data. When in Local Loopback Mode, the Host Controller loops back commands and data to the Host. The Loopback Command event is used to loop back commands that the Host sends to the Host Controller.

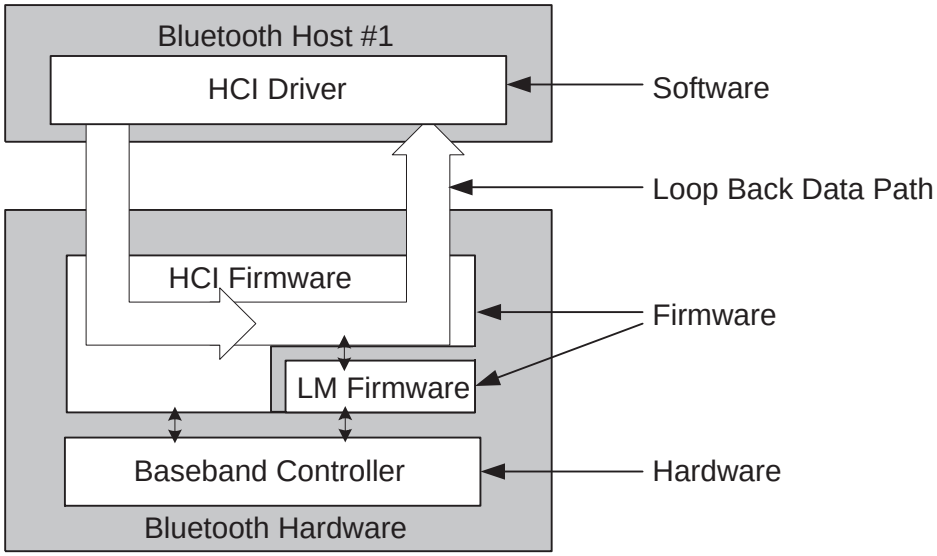


Figure 140—Local loopback mode

There are some commands that are not looped back in Local Loopback Mode: Reset, Set_Host_Controller_To_Host_Flow_Control, Host_Buffer_Size, Host_Number_Of_Completed_Packets, Read_Buffer_Size, Read_Loopback_Mode and Write_Loopback_Mode. These commands should be executed in the way they are normally executed. The commands Reset and Write_Loopback_Mode can be used to exit local loopback mode.

If Write_Loopback_Mode is used to exit Local Loopback Mode, four Disconnection Complete events should be sent to the Host corresponding to the Connection Complete events that were sent when entering Local Loopback Mode. Furthermore, no connections are allowed in Local Loopback mode. If there is a connection, and there is an attempt to set the device to Local Loopback Mode, the attempt will be refused. When the device is in Local Loopback Mode, the Host Controller will refuse incoming connection attempts. This allows the Host Controller Transport Layer to be tested without any other variables.

If a device is set to Remote Loopback Mode, it will send back all data (ACL and SCO) that comes over the air. It will only allow a maximum of one ACL connection and three SCO connections, and these should all be to the same remote device. If there already are connections to more than one remote device and there is an attempt to set the local device to Remote Loopback Mode, the attempt will be refused.

See Figure 141, where the rightmost device is set to Remote Loopback Mode and the leftmost device is set to Nontesting Mode. This allows the Bluetooth Air link to be tested without any other variables.

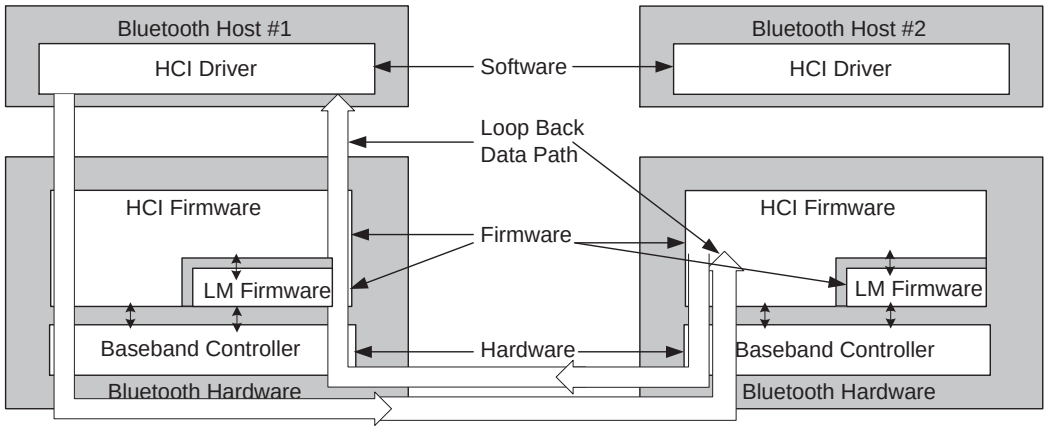


Figure 141—Remote loopback mode

Command Parameters:

Loopback_Mode: *Size: 1 Byte*

Value	Parameter description
0x00	No Loopback mode enabled. **Default.**
0x01	Enable Local Loopback.
0x02	Enable Remote Loopback.
0x03–0xFF	Reserved for future use.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Write_Loopback_Mode command succeeded.
0x01–0xFF	Write_Loopback_Mode command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Loopback_Mode command has completed, a Command Complete event will be generated.

11.2.10.3 Enable_Device_Under_Test_Mode

Command	OCF	Command Parameters	Return parameters
HCI_Enable_Device_Under_ Test_Mode	0x0003	—	Status

Description:

The Enable_Device_Under_Test_Mode command will allow the local Bluetooth module to enter test mode via LMP test commands. For details, see Clause 9. The Host issues this command when it wants the local device to be the DUT for the Testing scenarios as described in Annex E. When the Host Controller receives this command, it will complete the command with a Command Complete event. The Host Controller functions as normal until the remote tester issues the LMP test command to place the local device into Device Under Test mode. To disable and exit the Device Under Test Mode, the Host can issue the HCI_Reset command. This command prevents remote Bluetooth devices from causing the local Bluetooth device to enter test mode without first issuing this command.

Command Parameters:

None.

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Enter_Device_Under_Test_Mode command succeeded.
0x01–0xFF	Enter_Device_Under_Test_Mode command failed. See Table 109 for list of Error Codes.

Event(s) generated (unless masked away):

When the Enter_Device_Under_Test_Mode command has completed, a Command Complete event will be generated.

11.3 Events

11.3.1 Event

In addition to the events listed in Table 108, event code 0xFF is reserved for the event code used for vendor-specific debug events, and event code 0xFE is reserved for Bluetooth Logo Testing.

Table 108—List of supported events

Event	Event summary description
Inquiry complete event	The Inquiry Complete event indicates that the Inquiry is finished.
Inquiry result event	The Inquiry Result event indicates that a Bluetooth device or multiple Bluetooth devices have responded so far during the current Inquiry process.
Connection complete event	The Connection Complete event indicates to both of the Hosts forming the connection that a new connection has been established.
Connection Request event	The Connection Request event is used to indicate that a new incoming connection is trying to be established.
Disconnection Complete event	The Disconnection Complete event occurs when a connection has been terminated.
Authentication complete event	The Authentication Complete event occurs when authentication has been completed for the specified connection.
Remote name request complete event	The Remote Name Request Complete event is used to indicate a remote name request has been completed. The Remote_Name event parameter is a UTF-8 encoded string with up to 248 bytes in length.
Encryption change event	The Encryption Change event is used to indicate that the change in the encryption has been completed for the Connection Handle specified by the Connection_Handle event parameter.
Change connection link key complete event	The Change Connection Link Key Complete event is used to indicate that the change in the Link Key for the Connection Handle specified by the Connection_Handle event parameter had been completed.
Master link key complete event	The Master Link Key Complete event is used to indicate that the change in the temporary Link Key or in the semi-permanent link keys on the Bluetooth master side has been completed.
Read remote supported features complete event	The Read Remote Supported Features Complete event is used to indicate the completion of the process of the Link Manager obtaining the supported features of the remote Bluetooth device specified by the Connection_Handle event parameter.

Table 108—List of supported events *(continued)*

Event	Event summary description
Read remote version information complete event	The Read Remote Version Information Complete event is used to indicate the completion of the process of the Link Manager obtaining the version information of the remote Bluetooth device specified by the Connection_Handle event parameter.
QoS setup complete event	The QoS Setup Complete event is used to indicate the completion of the process of the Link Manager setting up QoS with the remote Bluetooth device specified by the Connection_Handle event parameter.
Command complete event	The Command Complete event is used by the Host Controller to pass the return status of a command and the other event parameters for each HCI Command.
Command status event	The Command Status event is used to indicate that the command described by the Command_Opcode parameter has been received and the Host Controller is currently performing the task for this command.
Hardware error event	The Error event is used to indicate some type of hardware failure for the Bluetooth device.
Flush occurred event	The Flush Occurred event is used to indicate that, for the specified Connection Handle, the current user data to be transmitted has been removed.
Role Change event	The Role Change event is used to indicate that the current Bluetooth role related to the particular connection has been changed.
Number of completed packets event	The Number Of Completed Packets event is used by the Host Controller to indicate to the Host how many HCI Data Packets have been completed for each Connection Handle since the previous Number Of Completed Packets event was sent.
Mode change event	The Mode Change event is used to indicate when the device associated with the Connection Handle changes between Active, Hold, Sniff and Park mode.
Return link keys event	The Return Link Keys event is used to return stored link keys after a Read_Stored_Link_Key command is used.
PIN code request event	The PIN Code Request event is used to indicate that a PIN code is required to create a new link key for a connection.
Link key request event	The Link Key Request event is used to indicate that a Link Key is required for the connection with the device specified in BD_ADDR.
Link key notification event	The Link Key Notification event is used to indicate to the Host that a new Link Key has been created for the connection with the device specified in BD_ADDR.

Table 108—List of supported events *(continued)*

Event	Event summary description
Loopback command event	The Loopback Command event is used to loop back most commands that the Host sends to the Host Controller.
Data buffer overflow event	The Data Buffer Overflow event is used to indicate that the Host Controller's data buffers have overflowed, because the Host has sent more packets than allowed.
Max slots change event	This event is used to notify the Host about the LMP_Max_Slots parameter when the value of this parameter changes.
Read clock offset complete event	The Read Clock Offset Complete event is used to indicate the completion of the process of the LM obtaining the Clock offset information.
Connection packet type changed event	The Connection Packet Type Changed event is used to indicate the completion of the process of the Link Manager changing the Packet Types used for the specified Connection_Handle.
QoS violation event	The QoS Violation event is used to indicate the Link Manager is unable to provide the current QoS requirement for the Connection Handle.
Page scan mode change event	The Page Scan Mode Change event indicates that the connected remote Bluetooth device with the specified Connection_Handle has successfully changed the Page_Scan_Mode.
Page scan repetition mode change event	The Page Scan Repetition Mode Change event indicates that the connected remote Bluetooth device with the specified Connection_Handle has successfully changed the Page_Scan_Repetition_Mode (SR).

11.3.2 Possible events

The events provide a method to return parameters and data associated for each event.

11.3.2.1 Inquiry complete event

Event	Event code	Event parameters
Inquiry Complete	0x01	Status

Description:

The Inquiry Complete event indicates that the Inquiry is finished. This event contains a status parameter, which is used to indicate if the Inquiry completed successfully or if the Inquiry was not completed.

Event Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Inquiry command completed successfully.
0x01–0xFF	Inquiry command failed. See Table 109 for list of Error Codes.

11.3.2.2 Inquiry result event

Event	Event code	Event parameters
Inquiry Result	0x02	Num_Responses, BD_ADDR[i], Page_Scan_Repetition_Mode[i], Page_Scan_Period_Mode[i], Page_Scan_Mode[i], Class_of_Device[i] Clock_Offset[i]

Description:

The Inquiry Result event indicates that a Bluetooth device or multiple Bluetooth devices have responded so far during the current Inquiry process. This event will be sent from the Host Controller to the Host as soon as an Inquiry Response from a remote device is received if the remote device supports only mandatory paging scheme. The Host Controller may queue these Inquiry Responses and send multiple Bluetooth devices information in one Inquiry Result event. The event can be used to return one or more Inquiry responses in one event. This event contains the BD_ADDR, Page_Scan_Repetition_Mode, Page_Scan_Period_Mode, Page_Scan_Mode, Clock_Offset, and the Class of Device for each Bluetooth device that responded to the latest inquiry.

Event Parameters:

Num_Responses: *Size: 1 Byte*

Value	Parameter description
0xXX	Number of responses from the Inquiry.

BD_ADDR[i]: *Size: 6 Bytes * Num_Responses*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR for each device that responded.

Page_Scan_Repetition_Mode[i]: *Size: 1 Byte* Num_Responses*

Value	Parameter description
0x00	R0
0x01	R1
0x02	R2
0x03–0xFF	Reserved

Page_Scan_Period_Mode[i]: *Size: 1 Byte* Num_Responses*

Value	Parameter description
0x00	P0
0x01	P1
0x02	P2
0x03–0xFF	Reserved

Page_Scan_Mode[i]: *Size: 1 Byte* Num_Responses*

Value	Parameter description
0x00	Mandatory Page Scan Mode.
0x01	Optional Page Scan Mode I.
0x02	Optional Page Scan Mode II.
0x03	Optional Page Scan Mode III.
0x04–0xFF	Reserved.

Class_of_Device[i]: *Size: 3 Bytes * Num_Responses*

Value	Parameter description
0xXXXXXX	Class of Device for the device.

Clock_Offset[i]: *Size: 2 Bytes * Num_Responses*

Bit format	Parameter description
Bit 14–0	Bit 16-2 of CLKslave-CLKmaster.
Bit 15	Reserved

11.3.2.3 Connection complete event

Event	Event code	Event parameters
Connection Complete	0x03	Status, Connection_Handle, BD_ADDR, Link_Type, Encryption_Mode

Description:

The Connection Complete event indicates to both of the Hosts forming the connection that a new connection has been established. This event also indicates to the Host that issued the Create Connection, Add_SCO_Connection, or Accept_Connection_Request or Reject_Connection_Request command and then received a Command Status event, if the issued command failed or was successful.

Event Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Connection successfully completed.
0x01–0xFF	Connection failed to Complete. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle to be used to identify a connection between to Bluetooth devices. The Connection Handle is used as an identifier for transmitting and receiving voice or data. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the other connected Device forming the connection.

Link_Type: *Size: 1 Byte*

Value	Parameter description
0x00	SCO connection (Voice Channels).
0x01	ACL connection (Data Channels).
0x02–0xFF	Reserved for future use.

Encryption_Mode: *Size: 1 Byte*

Value	Parameter description
0x00	Encryption disabled.
0x01	Encryption only for point-to-point packets.
0x02	Encryption for both point-to-point and broadcast packets.
0x03–0xFF	Reserved.

11.3.2.4 Connection Request event

Event	Event code	Event parameters
Connection Request	0x04	BD_ADDR, Class_of_Device, Link_Type

Description:

The Connection Request event is used to indicate that a new incoming connection is trying to be established. The connection may either be accepted or rejected. If this event is masked away and there is an incoming connection attempt and the Host Controller is not set to auto-accept this connection attempt, the Host Controller will automatically refuse the connection attempt. When the Host receives this event, it should respond with either an Accept_Connection_Request or Reject_Connection_Request command before the timer Conn_Accept_Timeout expires.

Event Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the device that requests the connection.

Class_of_Device: *Size: 3 Bytes*

Value	Parameter description
0xXXXXXX	Class of Device for the device, that requests the connection.

Link_Type: *Size: 1 Byte*

Value	Parameter description
0x00	SCO connection requested (Voice Channels).
0x01	ACL connection requested (Data Channels).
0x02–0xFF	Reserved for future use.

11.3.2.5 Disconnection Complete event

Event	Event code	Event parameters
Disconnection Complete	0x05	Status, Connection_Handle, Reason

Description:

The Disconnection Complete event occurs when a connection is terminated. The status parameter indicates if the disconnection was successful or not. The reason parameter indicates the reason for the disconnection if the disconnection was successful. If the disconnection was not successful, the value of the reason parameter can be ignored by the Host. For example, this can be the case if the Host has issued the Disconnect command and there was a parameter error, or the command was not presently allowed, or a connection handle that didn't correspond to a connection was given.

Note that when a physical link fails, one Disconnection Complete event will be returned for each logical channel on the physical link with the corresponding Connection handle as a parameter.

Event Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Disconnection has occurred.
0x01–0xFF	Disconnection failed to complete. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle that was disconnected. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use)

Reason: *Size: 1 Byte*

Value	Parameter description
0xXX	Reason for disconnection. See Table 109 for list of Error Codes.

11.3.2.6 Authentication complete event

Event	Event code	Event parameters
Authentication Complete	0x06	Status, Connection_Handle

Description:

The Authentication Complete event occurs when authentication has been completed for the specified connection. The Connection_Handle shall be a Connection_Handle for an ACL connection.

Event Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Authentication Request successfully completed.
0x01–0xFF	Authentication Request failed to complete. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle for which Authentication has been performed. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

11.3.2.7 Remote name request complete event

Event	Event code	Event parameters
Remote Name Request Complete	0x07	Status, BD_ADDR, Remote_Name

Description:

The Remote Name Request Complete event is used to indicate that a remote name request has been completed. The Remote_Name event parameter is a UTF-8 encoded string with up to 248 bytes in length. The Remote_Name event parameter will be null-terminated (0x00) if the UTF-8 encoded string is less than 248 bytes. The BD_ADDR event parameter is used to identify which device the user-friendly name was obtained from.

Note that the Remote_Name Parameter is a string parameter. Endianess does therefore not apply to the Remote_Name Parameter. The first byte of the name is received first.

Event Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Remote_Name_Request command succeeded.
0x01–0xFF	Remote_Name_Request command failed. See Table 109 for list of Error Codes.

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR for the device whose name was requested.

Remote_Name: *Size: 248 Bytes*

Value	Parameter description
Name[248]	A UTF-8 encoded user-friendly descriptive name for the remote device. If the name contained in the parameter is shorter than 248 bytes, the end of the name is indicated by a NULL byte (0x00), and the following bytes (to fill up 248 bytes, which is the length of the parameter) do not have valid values.

11.3.2.8 Encryption change event

Event	Event code	Event parameters
Encryption Change	0x08	Status, Connection_Handle, Encryption_Enable

Description:

The Encryption Change event is used to indicate that the change in the encryption has been completed for the Connection Handle specified by the Connection_Handle event parameter. The Connection_Handle will be a Connection_Handle for an ACL connection. The Encryption_Enable event parameter specifies the new Encryption Enable for the Connection Handle specified by the Connection_Handle event parameter. This event will occur on both devices to notify both Hosts when Encryption has changed for the specified Connection Handle between two devices.

Event Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Encryption Change has occurred.
0x01–0xFF	Encryption Change failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle for which the link layer encryption has been enabled/disabled for all Connection Handles with the same Bluetooth device endpoint as the specified Connection Handle. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Encryption_Enable: *Size: 1 Byte*

Value	Parameter description
0x00	Link Level Encryption is OFF.
0x01	Link Level Encryption is ON.

11.3.2.9 Change connection link key complete event

Event	Event code	Event parameters
Change Connection Link Key Complete	0x09	Status, Connection_Handle

Description:

The Change Connection Link Key Complete event is used to indicate that the change in the Link Key for the Connection Handle specified by the Connection_Handle event parameter has been completed. The Connection_Handle will be a Connection_Handle for an ACL connection. The Change Connection Link Key Complete event is sent only to the Host that issued the Change_Connection_Link_Key command.

Event Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Change_Connection_Link_Key command succeeded.
0x01–0xFF	Change_Connection_Link_Key command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle which the Link Key has been change for all Connection Handles with the same Bluetooth device end point as the specified Connection Handle. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

11.3.2.10 Master link key complete event

Event	Event code	Event parameters
Master Link Key Complete	0x0A	Status, Connection_Handle, Key_Flag

Description:

The Master Link Key Complete event is used to indicate that the Link Key managed by the master of the piconet has been changed. The Connection_Handle will be a Connection_Handle for an ACL connection. The link key used for the connection will be the temporary link key of the master device or the semipermanent link key indicated by the Key_Flag. The Key_Flag event parameter is used to indicate which Link Key (temporary link key of the Master, or the semipermanent link keys) is now being used in the piconet.

Note that for a master, the change from a semipermanent Link Key to temporary Link Key will affect all Connection Handles related to the piconet. For a slave, this change affects only this particular connection

handle. A temporary link key shall be used when both broadcast and point-to-point traffic shall be encrypted.

Event Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Master_Link_Key command succeeded.
0x01–0xFF	Master_Link_Key command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle for which the Link Key has been changed for all devices in the same piconet. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Key_Flag: *Size: 1 Byte*

Value	Parameter description
0x00	Using Semipermanent Link Key.
0x01	Using Temporary Link Key.

11.3.2.11 Read remote supported features complete event

Event	Event code	Event parameters
Read Remote Supported Features Complete	0x0B	Status, Connection_Handle, LMP_Features

Description:

The Read Remote Supported Features Complete event is used to indicate the completion of the process of the Link Manager obtaining the supported features of the remote Bluetooth device specified by the Connection_Handle event parameter. The Connection_Handle will be a Connection_Handle for an ACL connection. The event parameters include a list of LMP features. For details, see Clause 9.

Event Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Remote_Supported_Features command succeeded.
0x01–0xFF	Read_Remote_Supported_Features command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle that is used for the Read_Remote_Supported_Features command. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

LMP_Features: *Size: 8 Bytes*

Value	Parameter description
0xXXXXXXXX XXXXXXXX	Bit Mask List of LMP features. See Clause 9.

11.3.2.12 Read remote version information complete event

Event	Event code	Event Parameters
Read Remote Version Information Complete	0x0C	Status, Connection_Handle, LMP_Version, Manufacturer_Name, LMP_Subversion

Description:

The Read Remote Version Information Complete event is used to indicate the completion of the process of the Link Manager obtaining the version information of the remote Bluetooth device specified by the Connection_Handle event parameter. The Connection_Handle will be a Connection_Handle for an ACL connection. The LMP_Version event parameter defines the major hardware version of the Bluetooth hardware. This event parameter only changes when new versions of the Bluetooth hardware are produced for new Bluetooth SIG specifications; it is controlled by the Bluetooth SIG, Inc. The Manufacturer_Name event parameter indicates the manufacturer of the remote Bluetooth module. The LMP_Subversion event parameter should be controlled by the manufacturer and should be changed as needed. The LMP_Subversion event parameter defines the various revisions that each version of the Bluetooth hardware will go through as design processes change and errors are fixed. This allows the software to determine what Bluetooth hardware is being used and, if necessary, to work around various bugs in the hardware.

Event Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Remote_Version_Information command succeeded.
0x01–0xFF	Read_Remote_Version_Information command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle that is used for the Read_Remote_Version_Information command. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

LMP_Version: *Size: 1 Byte*

Value	Parameter description
0xXX	Version of the Current LMP in the remote Bluetooth Hardware, see Table 68 in the Link Manager Protocol for assigned values (VersNr).

Manufacturer_Name: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Manufacturer Name of the remote Bluetooth Hardware, see Table 68 in the Link Manager Protocol for assigned values (CompId).

LMP_Subversion: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Subversion of the Current LMP in the remote Bluetooth Hardware, see Table 68 in the Link Manager Protocol for assigned values (SubVersNr).

11.3.2.13 QoS setup complete event

Event	Event code	Event parameters
QoS Setup Complete	0x0D	Status, Connection_Handle, Flags, Service_Type, Token_Rate, Peak_Bandwidth, Latency, Delay_Variation

Description:

The QoS Setup Complete event is used to indicate the completion of the process of the Link Manager setting up QoS with the remote Bluetooth device specified by the Connection_Handle event parameter. The Connection_Handle will be a Connection_Handle for an ACL connection. For more detail, see Clause 9.

Event Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	QoS_Setup command succeeded.
0x01–0xFF	QoS_Setup command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle that is used for the QoS_Setup command. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Flags: *Size: 1 Byte*

Value	Parameter description
0x00–0xFF	Reserved for future use.

Service_Type: *Size: 1 Byte*

Value	Parameter description
0x00	No Traffic Available.
0x01	Best Effort Available.
0x02	Guaranteed Available.
0x03–0xFF	Reserved for future use.

Token_Rate: *Size: 4 Bytes*

Value	Parameter description
0xXXXXXXXX	Available Token Rate, in bytes per second.

Peak_Bandwidth: *Size: 4 Bytes*

Value	Parameter description
0xXXXXXXXX	Available Peak Bandwidth, in bytes per second.

Latency: *Size: 4 Bytes*

Value	Parameter description
0xXXXXXXXX	Available Latency, in microseconds.

Delay_Variation: *Size: 4 Bytes*

Value	Parameter description
0xXXXXXXXX	Available Delay Variation, in microseconds.

11.3.2.14 Command complete event

Event	Event code	Event parameters
Command Complete	0x0E	Num_HCI_Command_Packets, Command_Opcode, Return_Parameters

Description:

The Command Complete event is used by the Host Controller for most commands to transmit return status of a command and the other event parameters that are specified for the issued HCI command.

The Num_HCI_Command_Packets event parameter allows the Host Controller to indicate the number of HCI command packets the Host can send to the Host Controller. If the Host Controller requires the Host to stop sending commands, the Num_HCI_Command_Packets event parameter will be set to zero. To indicate to the Host that the Host Controller is ready to receive HCI command packets, the Host Controller generates a Command Complete event with the Command_Opcode 0x0000, and the Num_HCI_Command_Packets event parameter is set to 1 or more. Command_Opcode, 0x0000 is a NOP (No OPeration), and can be used to change the number of outstanding HCI command packets that the Host can send before waiting. See each command for the parameters that are returned by this event.

Event Parameters:

Num_HCI_Command_Packets: *Size: 1 Byte*

Value	Parameter description
0xXX	The Number of HCI command packets that are allowed to be sent to the Host Controller from the Host. Range for *N*: 0–255

Command_Opcode: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Opcode of the command which caused this event.

Return_Parameter(s): *Size: Depends on Command*

Value	Parameter description
0xXX	This is the return parameter(s) for the command specified in the Command_Opcode event parameter. See each command's definition for the list of return parameters associated with that command.

11.3.2.15 Command status event

Event	Event code	Event parameters
Command Status	0x0F	Status, Num_HCI_Command_Packets, Command_Opcode

Description:

The Command Status event is used to indicate that the command described by the Command_Opcode parameter has been received and that the Host Controller is currently performing the task for this command. This event is needed to provide mechanisms for asynchronous operation, which makes it possible to prevent the Host from waiting for a command to finish. If the command can not begin to execute (a parameter error may have occurred, or the command may currently not be allowed), the Status event parameter will contain the corresponding error code, and no complete event will follow since the command was not started. The Num_HCI_Command_Packets event parameter allows the Host Controller to indicate the number of HCI command packets the Host can send to the Host Controller. If the Host Controller requires the Host to stop sending commands, the Num_HCI_Command_Packets event parameter will be set to zero. To indicate to the Host that the Host Controller is ready to receive HCI command packets, the Host Controller generates a Command Status event with Status 0x00 and Command_Opcode 0x0000, and the Num_HCI_Command_Packets event parameter is set to 1 or more. Command_Opcode, 0x0000 is a NOP (No OPeration) and can be used to change the number of outstanding HCI command packets that the Host can send before waiting.

Event Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Command currently in pending.
0x01–0xFF	Command failed. See Table 109 for list of Error Codes.

Num_HCI_Command_Packets: *Size: 1 Byte*

Value	Parameter description
0xXX	The Number of HCI command packets that are allowed to be sent to the Host Controller from the Host. Range for *N*: 0–255

Command_Opcode: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Opcode of the command that caused this event and is pending completion.

11.3.2.16 Hardware error event

Event	Event code	Event parameters
Hardware Error	0x10	Hardware_Code

Description:

The Hardware Error event is used to indicate some type of hardware failure for the Bluetooth device. This event is used to notify the Host that a hardware failure has occurred in the Bluetooth module.

Event Parameters:

Hardware_Code: *Size: 1 Byte*

Value	Parameter description
0xXX	These Hardware_Codes will be implementation-specific, and will be assigned to indicate various hardware problems.

11.3.2.17 Flush occurred event

Event	Event code	Event parameters
Flush Occurred	0x11	Connection_Handle

Description:

The Flush Occurred event is used to indicate that, for the specified Connection Handle, the current user data to be transmitted has been removed. The Connection_Handle will be a Connection_Handle for an ACL connection. This could result from the flush command, or be due to the automatic flush. Multiple blocks of an L2CAP packet could have been pending in the Host Controller. If one baseband packet part of an L2CAP packet is flushed, then the rest of the HCI data packets for the L2CAP packet shall also be flushed.

Event Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle which was flushed. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

11.3.2.18 Role Change event

Event	Event code	Event parameters
Role Change	0x12	Status, BD_ADDR, New_Role

Description:

The Role Change event is used to indicate that the current Bluetooth role related to the particular connection has changed. This event only occurs when both the remote and local Bluetooth devices have completed their role change for the Bluetooth device associated with the BD_ADDR event parameter. This event allows both affected Hosts to be notified when the Role has been changed.

Event Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Role change has occurred.
0x01–0xFF	Role change failed. See Table 109 for list of Error Codes.

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device for which a role change has completed.

New_Role: *Size: 1 Byte*

Value	Parameter description
0x00	Currently the Master for specified BD_ADDR.
0x01	Currently the Slave for specified BD_ADDR.

11.3.2.19 Number of completed packets event

Event	Event code	Event parameters
Number Of Completed Packets	0x13	Number_of_Handles, Connection_Handle[i], HC_Num_Of_Completed_Packets[i]

Description:

The Number Of Completed Packets event is used by the Host Controller to indicate to the Host how many HCI Data Packets have been completed (transmitted or flushed) for each Connection Handle since the previous Number Of Completed Packets event was sent to the Host. This means that the corresponding buffer space has been freed in the Host Controller. Based on this information, and the HC_Total_Num_ACL_Data_Packets and HC_Total_Num_SCO_Data_Packets return parameter of the Read_Buffer_Size command, the Host can determine for which Connection Handles the following HCI Data Packets should be sent to the Host Controller. The Number Of Completed Packets event must not be sent before the corresponding Connection Complete event. While the Host Controller has HCI data packets in its buffer, it must keep sending the Number Of Completed Packets event to the Host at least periodically, until it finally reports that all the pending ACL Data Packets have been transmitted or flushed. The rate with which this event is sent is manufacturer specific.

Note that Number Of Completed Packets events will not report on SCO connection handles if SCO Flow Control is disabled. (See Read/Write_SCO_Flow_Control_Enable in 11.2.7.38 and 11.2.7.39.)

Event Parameters:

Number_of_Handles: *Size: 1 Byte*

Value	Parameter description
0xXX	The number of Connection Handles and Num_HCI_Data_Packets parameters pairs contained in this event. Range: 0–255

Connection_Handle[i]: *Size: Number_of_Handles * 2 Bytes(12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

HC_Num_Of_Completed_Packets [i]: *Size: Number_of_Handles * 2 Bytes*

Value	Parameter description
0xXXXX	The number of HCI Data Packets that have been completed (transmitted or flushed) for the associated Connection Handle since the previous time the event was returned. Range for *N*: 0x0000–0xFFFF

11.3.2.20 Mode change event

Event	Event code	Event parameters
Mode Change	0x14	Status, Connection_Handle, Current_Mode, Interval

Description:

The Mode Change event is used to indicate when the device associated with the Connection Handle changes between Active, Hold, Sniff and Park mode. The Connection_Handle will be a Connection_Handle for an ACL connection. The Connection_Handle event parameter is used to indicate which connection the Mode Change event is for. The Current_Mode event parameter is used to indicate which state the connection is currently in. The Interval parameter is used to specify a time amount specific to each state. Each Host Controller that is associated with the Connection Handle which has changed Modes will send the Mode Change event to its Host.

Event Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	A Mode Change has occurred.
0x01–0xFF	Hold_Mode, Sniff_Mode, Exit_Sniff_Mode, Park_Mode, or Exit_Park_Mode command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes(12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Current_Mode: *Size: 1 Byte*

Value	Parameter description
0x00	Active Mode.
0x01	Hold Mode.
0x02	Sniff Mode.
0x03	Park Mode.
0x04-0xFF	Reserved for future use.

Interval: *Size: 2 Bytes*

Value	Parameter description
0xXXXX	Hold: Number of Baseband slots to wait in Hold Mode. Hold Interval = *N* * 0.625 ms (1 Baseband slot) Range for *N*: 0x0000–0xFFFF Time Range: 0-40.9 s Sniff: Number of Baseband slots between sniff intervals. Time between sniff intervals = 0.625 msec (1 Baseband slot) Range for *N*: 0x0000–0xFFFF Time Range: 0–40.9 s Park: Number of Baseband slots between consecutive beacons. Interval Length = *N* * 0.625 ms (1 Baseband slot) Range for *N*: 0x0000–0xFFFF Time Range: 0-40.9 s

11.3.2.21 Return link keys event

Event	Event code	Event parameters
Return Link Keys	0x15	Num_Keys, BD_ADDR [i], Link_Key[i]

Description:

The Return Link Keys event is used by the Host Controller to send the Host one or more stored Link Keys. Zero or more instances of this event will occur after the Read_Stored_Link_Key command. When there are no link keys stored, no Return Link Keys events will be returned. When there are link keys stored, the number of link keys returned in each Return Link Keys event is implementation specific.

Event Parameters:

Num_Keys: *Size: 1 Byte*

Value	Parameter description
0xXX	Number of Link Keys contained in this event. Range: 0x01–0x0B

BD_ADDR [i]: *Size: 6 Bytes * Num_Keys*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR for the associated Link Key.

Link_Key[i]: *Size:16 Bytes * Num_Keys*

Value	Parameter description
0xXXXXXXXXXX XXXXXXXXXX XXXXXXXXXX	Link Key for the associated BD_ADDR.

11.3.2.22 PIN code request event

Event	Event code	Event parameters
PIN Code Request	0x16	BD_ADDR

Description:

The PIN Code Request event is used to indicate that a PIN code is required to create a new link key. The Host shall respond using either the PIN Code Request Reply or the PIN Code Request Negative Reply command, depending on whether the Host can provide the Host Controller with a PIN code or not.

Note that if the PIN Code Request event is masked away, then the Host Controller will assume that the Host has no PIN Code.

When the Host Controller generates a PIN Code Request event in order for the local Link Manager to respond to the request from the remote Link Manager (as a result of a Create_Connection or Authentication_Requested command from the remote Host), the local Host shall respond with either a PIN_Code_Request_Reply or PIN_Code_Request_Negative_Reply command before the remote Link Manager detects LMP response timeout. (See Clause 9.)

Event Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device which a new link key is being created for.

11.3.2.23 Link key request event

Event	Event code	Event parameters
Link Key Request	0x17	BD_ADDR

Description:

The Link Key Request event is used to indicate that a Link Key is required for the connection with the device specified in BD_ADDR. If the Host has the requested stored Link Key, then the Host will pass the requested Key to the Host Controller using the Link_Key_Request_Reply Command. If the Host does not have the requested stored Link Key, then the Host will use the Link_Key_Request_Negative_Reply Command to indicate to the Host Controller that the Host does not have the requested key.

Note that if the Link Key Request event is masked away, then the Host Controller will assume that the Host has no additional link keys.

When the Host Controller generates a Link Key Request event in order for the local Link Manager to respond to the request from the remote Link Manager (as a result of a Create_Connection or Authentication_Requested command from the remote Host), the local Host shall respond with either a Link_Key_Request_Reply or Link_Key_Request_Negative_Reply command before the remote Link Manager detects LMP response timeout. (See Clause 9.)

Event Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device for which a stored link key is being requested.

11.3.2.24 Link key notification event

Event	Event code	Event parameters
Link Key Notification	0x18	BD_ADDR, Link_Key,Key_Type

Description:

The Link Key Notification event is used to indicate to the Host that a new Link Key has been created for the connection with the device specified in BD_ADDR. The Host can save this new Link Key in its own storage for future use. Also, the Host can decide to store the Link Key in the Host Controller's Link Key Storage by using the Write_Stored_Link_Key command. The Key_Type event parameter informs the Host about which key type (combination key, local unit key or remote unit key) that has been used during pairing. If pairing with unit key is not supported, the Host can, for instance, discard the key or disconnect the link.

Event Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the Device for which the new link key has been generated.

Link_Key: *Size: 16 Bytes*

Value	Parameter description
0xXXXXXXXXXX XXXXXXXXXX XXXXXXXXXX	Link Key for the associated BD_ADDR.

Key_Type: *Size: 1 Bytes*

Value	Parameter description
0x00	Combination Key.
0x01	Local Unit Key.
0x02	Remote Unit Key.
0x03–0xFF	Reserved.

11.3.2.25 Loopback command event

Event	Event code	Event parameters
Loopback Command	0x19	HCI_Command_Packet

Description:

When in Local Loopback mode, the Host Controller loops back commands and data to the Host. The Loopback Command event is used to loop back all commands that the Host sends to the Host Controller with some exceptions. See 11.2.10.1 for a description of which commands that are not looped back. The HCI_Command_Packet event parameter contains the entire HCI Command Packet including the header.

Note that the event packet is limited to a maximum of 255 bytes in the payload; since an HCI Command Packet has 3 bytes of header data, only the first 252 bytes of the command parameters will be returned.

Event Parameters:

HCI_Command_Packet: *Size: Depends on Command*

Value	Parameter description
0xXXXXXX	HCI Command Packet, including header.

11.3.2.26 Data buffer overflow event

Event	Event code	Event parameters
Data Buffer Overflow	0x1A	Link_Type

Description:

This event is used to indicate that the Host Controller's data buffers have been overflowed. This can occur if the Host has sent more packets than allowed. The Link_Type parameter is used to indicate that the overflow was caused by ACL or SCO data.

Event Parameters:

Link_Type: *Size: 1 Byte*

Value	Parameter description
0x00	SCO Buffer Overflow (Voice Channels).
0x01	ACL Buffer Overflow (Data Channels).
0x02–0xFF	Reserved for future use.

11.3.2.27 Max slots change event

Event	Event code	Event parameters
Max Slots Change	0x1B	Connection_Handle, LMP_Max_Slots

Description:

This event is used to notify the Host about the LMP_Max_Slots parameter when the value of this parameter changes. It will be sent each time the maximum allowed length, in number of slots, for baseband packets transmitted by the local device, changes. The Connection_Handle will be a Connection_Handle for an ACL connection.

Event Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

LMP_Max_Slots: *Size: 1 byte*

Value	Parameter description
0x01, 0x03, 0x05	Maximum number of slots allowed to use for baseband packets, see 9.3.22 and 9.5.1.

11.3.2.28 Read clock offset complete event

Event	Event code	Event parameters
Read Clock Offset Complete	0x1C	Status, Connection_Handle, Clock_Offset

Description:

The Read Clock Offset Complete event is used to indicate the completion of the process of the Link Manager obtaining the Clock Offset information of the Bluetooth device specified by the Connection_Handle event parameter. The Connection_Handle will be a Connection_Handle for an ACL connection.

Event Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Read_Clock_Offset command succeeded.
0x01–0xFF	Read_Clock_Offset command failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 bits meaningful)*

Value	Parameter description
0xXXXX	Specifies which Connection Handle's Clock Offset parameter is returned. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Clock_Offset: *Size: 2 Bytes*

Bit format	Parameter description
Bit 14–0	Bit 16–2 of CLKslave-CLKmaster.
Bit 15	Reserved.

11.3.2.29 Connection packet type changed event

Event	Event code	Event parameters
Connection Packet Type Changed	0x1D	Status, Connection_Handle, Packet_Type

Description:

The Connection Packet Type Changed event is used to indicate that the process has completed of the Link Manager changing which packet types can be used for the connection. This allows current connections to be dynamically modified to support different types of user data. The Packet_Type event parameter specifies

which packet types the Link Manager can use for the connection identified by the Connection_Handle event parameter for sending L2CAP data or voice. The Packet_Type event parameter does not decide which packet types the LM is allowed to use for sending LMP PDUs.

Event Parameters:

Status: *Size: 1 Byte*

Value	Parameter description
0x00	Connection Packet Type changed successfully.
0x01–0xFF	Connection Packet Type Changed failed. See Table 109 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

Packet_Type: *Size: 2 Bytes*

For ACL_Link_Type

Value	Parameter description
0x0001	Reserved for future use.
0x0002	Reserved for future use.
0x0004	Reserved for future use.
0x0008	DM1
0x0010	DH1
0x0020	Reserved for future use.
0x0040	Reserved for future use.
0x0080	Reserved for future use.
0x0100	Reserved for future use.
0x0200	Reserved for future use.
0x0400	DM3
0x0800	DH3
0x1000	Reserved for future use.
0x2000	Reserved for future use.
0x4000	DM5
0x8000	DH5

For SCO_Link_Type

Value	Parameter description
0x0001	Reserved for future use.
0x0002	Reserved for future use.
0x0004	Reserved for future use.
0x0008	Reserved for future use.
0x0010	Reserved for future use.
0x0020	HV1
0x0040	HV2
0x0080	HV3
0x0100	Reserved for future use.
0x0200	Reserved for future use.
0x0400	Reserved for future use.
0x0800	Reserved for future use.
0x1000	Reserved for future use.
0x2000	Reserved for future use.
0x4000	Reserved for future use.
0x8000	Reserved for future use.

11.3.2.30 QoS violation event

Event	Event code	Event parameters
QoS Violation	0x1E	Connection_Handle

Description:

The QoS Violation event is used to indicate the Link Manager is unable to provide the current QoS requirement for the Connection Handle. This event indicates that the Link Manager is unable to provide one or more of the agreed QoS parameters. The Host chooses what action should be done. The Host can reissue QoS_Setup command to renegotiate the QoS setting for Connection Handle. The Connection_Handle will be a Connection_Handle for an ACL connection.

Event Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter description
0xXXXX	Connection Handle that the LM is unable to provide the current QoS requested for. Range: 0x0000–0x0EFF (0x0F00–0x0FFF Reserved for future use).

11.3.2.31 Page scan mode change event

Event	Event code	Event parameters
Page Scan Mode Change	0x1F	BD_ADDR, Page_Scan_Mode

Description:

The Page Scan Mode Change event indicates that the remote Bluetooth device with the specified BD_ADDR has successfully changed the Page_Scan_Mode.

Event Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the remote device.

Page_Scan_Mode: *Size: 1 Byte*

Value	Parameter description
0x00	Mandatory Page Scan Mode.
0x01	Optional Page Scan Mode I.
0x02	Optional Page Scan Mode II.
0x03	Optional Page Scan Mode III.
0x04–0xFF	Reserved.

11.3.2.32 Page scan repetition mode change event

Event	Event code	Event parameters
Page Scan Repetition Mode Change	0x20	BD_ADDR, Page_Scan_Repetition_Mode

Description:

The Page Scan Repetition Mode Change event indicates that the remote Bluetooth device with the specified BD_ADDR has successfully changed the Page_Scan_Repetition_Mode (SR).

Event Parameters:

BD_ADDR: *Size: 6 Bytes*

Value	Parameter description
0xXXXXXXXXXXXX	BD_ADDR of the remote device.

Page_Scan_Repetition_Mode: *Size: 1 Byte*

Value	Parameter description
0x00	R0
0x01	R1
0x02	R2
0x03–0xFF	Reserved.

11.4 List of error codes

11.4.1 List of error codes

Table 109 lists the various possible error codes. When a command fails, Error codes are returned to indicate the reason for the error. Error codes have a size of one byte, and the possible range of failure codes is 0x01-0xFF. Subclause 11.4.2 provides an error code usage description for each error code.

Table 109—List of possible error codes

Error Code	Description
0x01	Unknown HCI Command.
0x02	No Connection.
0x03	Hardware Failure.
0x04	Page Timeout.
0x05	Authentication Failure.
0x06	Key Missing.
0x07	Memory Full.
0x08	Connection Timeout.
0x09	Max Number Of Connections.
0x0A	Max Number Of SCO Connections To A Device.
0x0B	ACL connection already exists.
0x0C	Command Disallowed.
0x0D	Host Rejected due to limited resources.
0x0E	Host Rejected due to security reasons.
0x0F	Host Rejected due to remote device is only a personal device.
0x10	Host Timeout.
0x11	Unsupported Feature or Parameter Value.
0x12	Invalid HCI Command Parameters.
0x13	Other End Terminated Connection: User Ended Connection.
0x14	Other End Terminated Connection: Low Resources.
0x15	Other End Terminated Connection: About to Power Off.
0x16	Connection Terminated by Local Host.
0x17	Repeated Attempts.
0x18	Pairing Not Allowed.
0x19	Unknown LMP PDU.

Table 109—List of possible error codes *(continued)*

Error Code	Description
0x1A	Unsupported Remote Feature.
0x1B	SCO Offset Rejected.
0x1C	SCO Interval Rejected.
0x1D	SCO Air Mode Rejected.
0x1E	Invalid LMP Parameters.
0x1F	Unspecified Error.
0x20	Unsupported LMP Parameter Value.
0x21	Role Change Not Allowed
0x22	LMP Response Timeout
0x23	LMP Error Transaction Collision
0x24	LMP PDU Not Allowed
0x25	Encryption Mode Not Acceptable
0x26	Unit Key Used
0x27	QoS is Not Supported
0x28	Instant Passed
0x29	Pairing with Unit Key Not Supported
0x2A-0xFF	Reserved for future use.

11.4.2 HCI error code usage descriptions

The purpose of this subclause is to give descriptions of how the error codes specified in 11.4.1 should be used. It is beyond the scope of this standard to give detailed descriptions of all situations where error codes can be used, especially as this may also, in certain cases, be implementation-dependent. However, some error codes that are to be used only in very special cases are described in more detail than other, more general, error codes.

The following error codes are only used in LMP messages, and are therefore not described in this subclause:

— Unknown LMP PDU (0x19)

— SCO Offset Rejected (0x1B)

— SCO Interval Rejected (0x1C)

— SCO Air Mode Rejected (0x1D)

— Invalid LMP Parameters (0x1E)

Some of the following error code descriptions describe as implementation-dependent whether the error should be returned using a Command Status event or the event associated with the issued command

(following a Command Status event with Status = 0x00). In these cases, the command can not start executing because of the error, and it is therefore recommended to use the Command Status event. The reason for this suggested course of action is that it is not possible to use the Command Status event in all software architectures.

11.4.3 Unknown HCI Command (0x01)

The “Unknown HCI Command” error code is returned by the Host Controller in the Status parameter in a Command Complete event or a Command Status event when the Host Controller receives an HCI Command Packet with an Opcode that it does not recognize. The OpCode given might not correspond to any of the OpCodes specified in this standard, or any vendor-specific OpCodes, or the command may not have been implemented. If a Command Complete event is returned, the Status parameter is the only parameter contained in the Return_Parameters event parameter. Which of the two events is used is implementation-dependent.

11.4.4 No Connection (0x02)

The “No Connection” error code is returned by the Host Controller in the Status parameter in an event when the Host has issued a command that requires an existing connection and there is currently no connection corresponding to the specified Connection Handle or BD Address. If the issued command is a command for which a Command Complete event should be returned, the event containing the error code is a Command Complete event. Otherwise, the event containing the error code is a Command Status event or the event associated with the issued command (following a Command Status event with Status = 0x00), depending on the implementation.

11.4.5 Hardware Failure (0x03)

The “Hardware Failure” error code is returned by the Host Controller in the Status parameter in an event when the Host has issued a command and this command cannot be executed because of a hardware failure. If the issued command is a command for which a Command Complete event should be returned, the event containing the error code is a Command Complete event. Otherwise, the event containing the error code is a Command Status event or the event associated with the issued command (following a Command Status event with Status=0x00) depending on the implementation.

11.4.6 Page Timeout (0x04)

The “Page Timeout” error code is returned by the Host Controller in the Status parameter of the Connection Complete event when the Host has issued a Create_Connection command and the specified device to connect to does not respond to a page at baseband level before the page timer expires (a page timeout occurs). The error code can also be returned in the Status parameter of a Remote Name Request Complete event when the Host has issued a Remote_Name_Request command and a temporary connection needs to be established but a page timeout occurs. (The page timeout is set using the Write_Page_Timeout command.)

11.4.7 Authentication Failure (0x05)

The “Authentication Failure” error code is returned by the Host Controller in the Status parameter in a Connection Complete event or Authentication Complete event when pairing or authentication fails due to incorrect results in the pairing/authentication calculations (because of an incorrect PIN code or link key).

"The “Authentication Failure” error code can also be used as a value for the Reason parameter in the Disconnect command (as a reason code). The error code will then be sent over the air so that it is returned in the Reason parameter of a Disconnection Complete event on the remote side. In the Disconnection Complete event following a Command Status event (where Status = 0x00) on the local side on which the

Disconnect command has been issued, the Reason parameter will, however, contain the reason code "Connection Terminated By Local Host".

11.4.8 Key Missing (0x06)

The "Key Missing" error code is returned by the Host Controller in the Status parameter in a Connection Complete event or Authentication Complete event when pairing fails because of missing PIN code(s).

11.4.9 Memory Full (0x07)

The "Memory Full" error code is returned by the Host Controller in the Status parameter in a Command Complete event when the Host has issued a command that requires the Host Controller to store new parameters and the Host Controller does not have memory capacity for this. This may be the case after the Set_Event_Filter command has been issued. Note that for the Write_Stored_Link_Key command, no error is returned when the Host Controller cannot store any more link keys. The Host Controller stores as many link keys as there is free memory to store in, and the Host is notified of how many link keys were successfully stored.

11.4.10 Connection Timeout (0x08)

Note that this error code is used to indicate a reason for disconnection. It is normally returned in the Reason parameter of a Disconnection Complete event. It is, therefore, called reason code in the following description.

The "Connection Timeout" reason code is sent by the Host Controller in an event when the link supervision timer (see Annex F) expires, and the link, therefore, is considered to be lost. The link supervision timeout is set using Write_Link_Supervision_Timeout. The event that returns this reason code will most often be a Disconnection Complete event (in the Reason parameter). The event will be returned on both sides of the connection, where one Disconnection Complete event will be sent from the Host Controller to the Host for each Connection Handle that exists for the physical link to the other device.

(It is possible for a link loss to be detected during connection setup, in which case, the reason code would be returned in a Connection Complete event.)

11.4.11 Max Number Of Connections (0x09)

The "Max Number Of Connections" error code is returned by the Host Controller in the Status parameter of a Command Status event, a Connection Complete event, or a Remote Name Request Complete event when the Bluetooth module cannot establish any more connections. It is implementation specific whether the error is returned in a Command Status event or the event following the Command Status event (where Status = 0x00 in the Command Status event). The reason for this error may be hardware or firmware limitations. Before the error is returned, the Host has issued a Create_Connection, Add_SCO_Connection, or Remote_Name_Request command. The error can be returned in a Remote Name Request Complete event when a temporary connection needs to be established to request the name.

11.4.12 Max Number Of SCO Connections To A Device (0x0A)

The "Max Number Of SCO Connections To A Device" error code is returned by the Host Controller in the Status parameter of a Command Status event or a Connection Complete event (following a Command Status event with Status = 0x00) when the maximum number of SCO connections to a device has been reached. Which of the two events that is used depends on the implementation. The device is a device that has been specified in a previously issued Add_SCO_Connection command.

11.4.13 ACL Connection Already Exists (0x0B)

The “ACL Connection Already Exists” error code is returned by the Host Controller in the Status parameter of a Command Status event or a Connection Complete event (following a Command Status event with Status = 0x00) when there already is one ACL connection to a device and the Host tries to establish another one using Create_Connection. Which of the two events that is used depends on the implementation.

11.4.14 Command Disallowed (0x0C)

The “Command Disallowed” error code is returned by the Host Controller in the Status parameter in a ComNSmand Complete event or a Command Status event when the Host Controller is in a state where it is only prepared to accept commands with certain OpCodes and the HCI Command Packet received does not contain any of these OpCodes. The Command Complete event should be used if the issued command is a command for which a Command Complete event should be returned. Otherwise, the Command Status event should be used. The Host Controller is not required to use the “Unknown HCI Command” error code, since this may require unnecessary processing of the received (and currently not allowed) OpCode. When to use the “Command Disallowed” error code is mainly implementation-dependent. Certain implementations may, for example, only accept the appropriate HCI response commands after the Connection Request, Link Key Request, or PIN Code Request events.

Note that the Reset command should always be allowed.

11.4.15 Host Rejected due to … (0x0D–0x0F)

Note that these error codes are used to indicate a reason for rejecting an incoming connection. They are therefore called reason codes in the following description.

When a Connection Request event has been received by the Host and the Host rejects the incoming connection by issuing the Reject_Connection_Request command, one of these reason codes is used as value for the Reason parameter. The issued reason code will be returned in the Status parameter of the Connection Complete event that will follow the Command Status event (with Status = 0x00) returned by the Host Controller after the Reject_Connection_Request command has been issued. The reason code issued in the Reason parameter of the Reject_Connection_Request command will also be sent over the air, so that it is returned in a Connection Complete event on the initiating side. Before this, the initiating side has issued a Create_Connection command or Add_SCO_Connection command and has received a Command Status event (with Status = 0x00).

11.4.16 Host Timeout (0x10)

Note that this error code is used to indicate a reason for rejecting an incoming connection. It is, therefore, called reason code in the following description.

Assume that a Connection Request event has been received by the Host and that the Host does not issue the Accept_Connection_Request or Reject_Connection_Request command before the connection accept timer expires (the connection accept timeout is set using Write_Connection_Accept_Timeout). In this case, the “Host Timeout” reason code will be sent by the Host Controller in the Status parameter of a Connection Complete event. The reason code will also be sent over the air, so that it is returned in a Connection Complete event on the initiating side. The initiating side has before this issued a Create_Connection or Add_SCO_Connection command and has received a Command Status event (with Status = 0x00).

11.4.17 Unsupported Feature or Parameter Value (0x11)

The “Unsupported Feature or Parameter Value” error code is returned by the Host Controller in the Status parameter in an event when the Host Controller has received a command where one or more parameters have values that are not supported by the hardware (the parameters are, however, within the allowed parameter range specified in this standard). If the issued command is a command for which a Command Complete event should be returned, the event containing the error code is a Command Complete event. Otherwise, the event containing the error code is a Command Status event or the event associated with the issued command (following a Command Status event with Status=0x00), depending on the implementation.

11.4.18 Invalid HCI Command Parameters (0x12)

The “Invalid HCI Command Parameters” error code is returned by the Host Controller in the Status parameter of an event when the total parameter length (or the value of one or more parameters in a received command) does not conform to what is specified in this standard.

The error code can also be returned if a parameter value is currently not allowed although it is inside the allowed range for the parameter. One such case is when a command requires a Connection Handle for an ACL connection but the Host has given a Connection Handle for an SCO connection as a parameter instead. Another case is when a link key, a PIN code or a reply to an incoming connection has been requested by the Host Controller by using an event, but the Host replies using a response command with a BD_ADDR for which no request has been made.

If the issued command is a command for which a Command Complete event should be returned, the event containing the error code is a Command Complete event. Otherwise, the event containing the error code is a Command Status event or the event associated with the issued command (following a Command Status event with Status = 0x00), depending on the implementation.

11.4.19 Other End Terminated Connection: … (0x13-0x15)

Note that these error codes are used to indicate a reason for disconnecting a connection. They are, therefore, called reason codes in the following description.

When the Host issues the Disconnect command, one of these reason codes is used as value for the reason parameter. The “Connection Terminated By Local Host” reason code will then be returned in the Reason parameter of the Disconnection Complete event that will follow the Command Status event (with Status = 0x00) that is returned by the Host Controller after the Disconnect command has been issued. The reason code issued in the Reason parameter of the Disconnect command will also be sent over the air, so that it is returned in the Reason parameter of a Disconnection Complete event on the remote side.

11.4.20 Connection Terminated By Local Host (0x16)

This error code is called a reason code, since it is returned in the Reason parameter of a Disconnection Complete event. See description in 11.4.19.

11.4.21 Repeated Attempts (0x17)

The “Repeated Attempts” error code is returned by the Host Controller in the Status parameter in a Connection Complete event or Authentication Complete event when a device does not allow authentication or pairing because too little time has elapsed since an unsuccessful authentication or pairing attempt. See Clause 9 for a description of how repeated attempts work.

11.4.22 Pairing Not Allowed (0x18)

The “Pairing Not Allowed” error code is returned by the Host Controller in the Status parameter in a Connection Complete event or Authentication Complete event when a device for some reason does not allow pairing. An example may be a PSTN adapter that only allows pairing during a certain time window after a button has been pressed on the adapter.

11.4.23 Unsupported Remote Feature (0x1A)

The “Unsupported Remote Feature” error code is returned by the Host Controller in the Status parameter of the event associated with the issued command when a remote device that has been specified in the command parameters does not support the feature associated with the issued command. The “Unsupported Remote Feature” error code can also be used as a value for the Reason parameter in the Disconnect command (as a reason code). The error code will then be sent over the air so that it is returned in the Reason parameter of a Disconnection Complete event on the remote side. In the Disconnection Complete event following a Command Status event (where Status = 0x00) on the local side on which the Disconnect command has been issued, the Reason parameter will, however, contain the reason code “Connection Terminated By Local Host”. (The “Unsupported Remote Feature” error code is called “Unsupported LMP Feature” in the LMP specification; see Clause 9.)

11.4.24 Unspecified error (0x1F)

The “Unspecified error” error code is used when no other error code specified in this document is appropriate to use.

11.4.25 Unsupported LMP Parameter Value (0x20)

The “Unsupported LMP Parameter Value” error code is returned by the Host Controller in the Status parameter of the event associated with the issued command when a remote device that has been specified in the command parameters sent back an LMP message containing the LMP error code 0x20, “Unsupported parameter values” (see Clause 9).

11.4.26 Role Change Not Allowed (0x21)

The “Role Change Not Allowed” error code is returned by the Host Controller in the Status parameter in a Connection Complete event or Role Change event when role change is not allowed. If the local Host issues the Switch_Role command and the remote device rejects the role change, the error code will be returned in a Role Change event. If a connection fails because a device accepts an incoming ACL connection with a request for role change and the role change is rejected by the initiating device, the error code will be returned in a Connection Complete event on both sides.

11.4.27 LMP Response Timeout (0x22)

The “LMP Response Timeout” error code is returned by the Host Controller in the Status parameter in a Command Complete event or an event associated with the issued command following a Command Status event with Status=0x00, when the remote device does not respond to the LMP PDUs from the local device as a result of the issued command within LMP response timeout (see Clause 9).

11.4.28 LMP Error Transaction Collision (0X23)

The “LMP Error Transaction Collision” error code is returned by the Host Controller in the Status parameter of the event associated with the issued command when a remote device that has been specified in the command parameters sends back an LMP message containing the LMP error code 0x23, “LMP Error Transaction Collision” (see Clause 9).

11.4.29 LMP PDU Not Allowed (0X24)

The “LMP PDU Not Allowed” error code is returned by the Host Controller in the Status parameter of the event associated with the issued command when a remote device that has been specified in the command parameters sends back an LMP message containing the LMP error code 0x24, “PDU Not Allowed” (see Clause 9).

11.4.30 Encryption Mode Not Acceptable (0X25)

The “Encryption Mode Not Acceptable” error code is returned by the Host Controller in the Status parameter of a Connection Complete event or an Encryption Change event when no agreement can be reached on which encryption mode to use.

11.4.31 Unit Key Used (0X26)

The “Unit Key Used” error code is returned by the Host Controller in the Status parameter of a Command Status event or a Change Connection Link Key Complete event (following a Command Status event with Status = 0x00) if the link key cannot be changed because it is a unit key.

11.4.32 QoS Is Not Supported (0X27)

The “QoS Is Not Supported” error code is returned by the Host Controller in the Status parameter of a Command Status event or a QoS Setup Complete event (following a Command Status event with Status = 0x00) if the requested QoS is not supported.

11.4.33 Instant Passed (0X28)

The “Instant Passed” error code is returned by the Host Controller in the Status parameter of a Role Change event if a role switch cannot be performed because the instant at which the role switch shall start is in the past.

11.4.34 Pairing With Unit Key Not Supported (0X29)

Note that this error code can be used to indicate a reason for disconnecting a connection. It is, therefore, called reason code in the following description.

When the Host issues the Disconnect command, this reason code can be used as a value for the Reason parameter (if the Key_Type parameter in a Link Key Notification event indicates that unit key is used and this is not supported). The reason code will then be returned in the Reason parameter of the Disconnection Complete event that will follow the Command Status event (with Status = 0x00) that is returned by the Host Controller after the Disconnect command has been issued. The reason code issued in the Reason parameter of the Disconnect command will also be sent over the air, so that it is returned in the Reason parameter of a Disconnection Complete event on the remote side.

12. Service access point interfaces and primitives

12.1 IEEE 802 interfaces

Figure 142 indicates the relationship of the Bluetooth protocol stack to this clause. This clause describes the functions, features, protocol, services, and SAP interfaces between the MAC and LLC sublayers within the data link layer of the ISO/IEC 8802 LAN Protocol (IEEE 802.2). The LLC sublayer constitutes the top sublayer in the data link layer (see Figure 143) and is common to the various medium access methods that are defined and supported by the ISO/IEC 8802 activity.

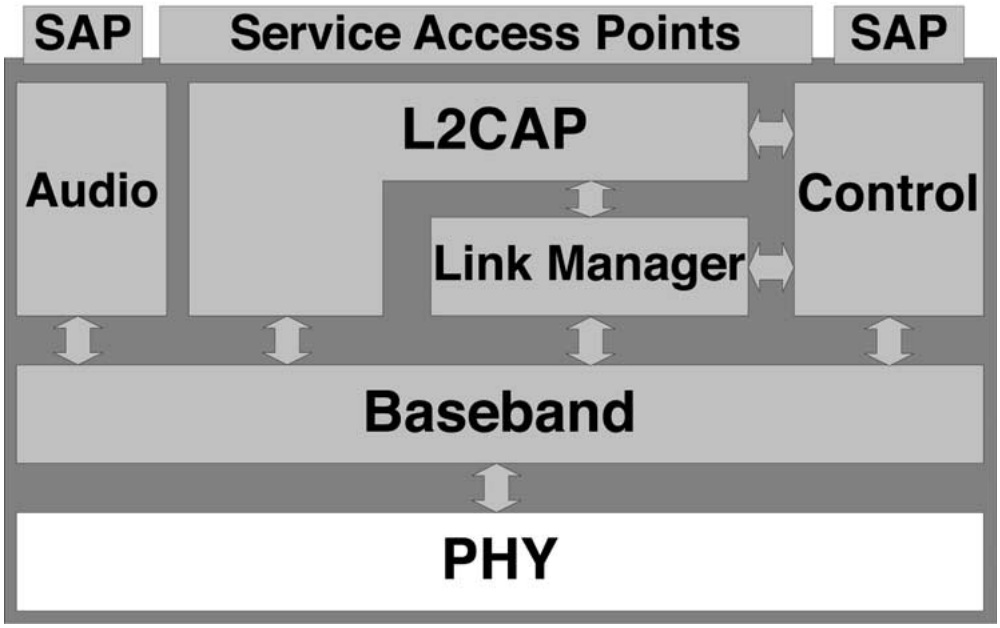


Figure 142—SAP relationships

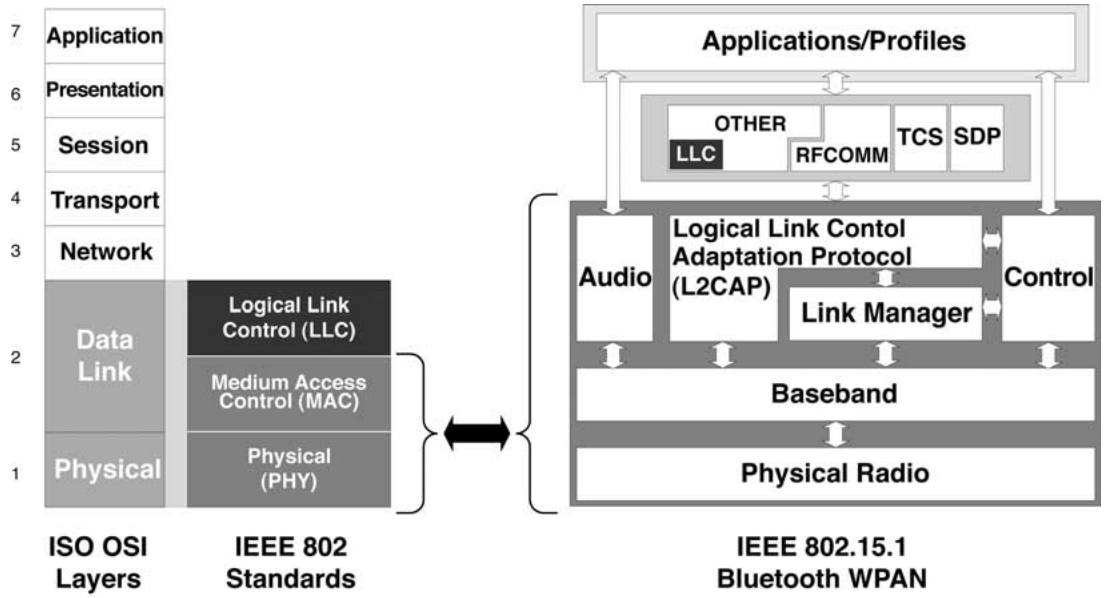


Figure 143—OSI and Bluetooth protocols

The following concepts are discussed in this clause:

a) **Data link layer:** The conceptual layer of control or processing logic existing in the hierarchical structure of a station that is responsible for maintaining control of the data link. The data link layer functions provide an interface between the station higher-layer logic and the data link. These functions include address/control field interpretation, channel access and command, PDU/response, PDU generation, sending, and interpretation.

b) **LLC sublayer:** The part of a data station that supports the logical link control functions of one or more logical links. The LLC generates command PDUs and response PDUs for sending and interprets received command PDUs and response PDUs. Specific responsibilities assigned to an LLC include the following:

   1) Initiation of control signal interchange

   2) Organization of data flow

   3) Interpretation of received command PDUs and generation of appropriate response PDUs

   4) Actions regarding error control and error recovery functions in the LLC sublayer

c) **MAC sublayer:** The part of a data station that supports the MAC functions that reside just below the LLC sublayer. The MAC procedures include framing/deframing data units, performing error checking, and acquiring the right to use the underlying physical medium.

d) **LLC sublayer service specifications to the mac sublayer:** The service specifications to the MAC sublayer provide a description of the services the LLC sublayer requires of the MAC sublayer. These services are defined to be independent of the form of the medium access methodology and the nature of the medium itself. All the above service specifications are given in the form of primitives that represent in an abstract way the logical exchange of information and control between the LLC sublayer and the identified service function (MAC sublayer). They do not specify or constrain the implementation of entities or interfaces.

e) **Type of operation:** One type of data link control (DLC) operation. “Type 1” data link control operation provides a data-link-connectionless-mode service across a data link with minimum protocol complexity. This type of operation may be useful when higher layers provide any essential recovery and sequencing services so that these layers do not need replicating in the data link layer. In addition, this type of operation may prove useful in applications where it is not essential to guarantee the delivery of every data link layer data unit. This type of service is described in this section in terms of logical data links.

   With Type 1 operation, PDUs shall be exchanged between LLCs without the need for the establishment of a data link connection. In the LLC sublayer, these PDUs shall not be acknowledged, and there shall not be any flow control or error recovery in Type 1 operation.

f) **Class of operation:** One class of LLC operation. Class I provides data-link-connectionless-mode service only.

   Class I LLCs shall support Type 1 operation only. Class I service shall be applicable to individual, group, global, and null DSAP addressing and applications requiring no data link layer (DLL) acknowledgment or flow control procedures.

   The basic protocols described here are peer protocols for use in multistation, multiaccess environments. Because of the multistation, multiaccess environment, it shall be possible for a station to be involved in a multiplicity of peer protocol data exchanges with a multiplicity of different stations over a multiplicity of different logical data links and/or data link connections that are carried by a single PHY over a single physical medium. Each unique to–from pairing at the data link layer shall define a separate logical data link or data link connection with separate logical parameters and variables. Except where noted, the procedures described shall relate to each data link layer logical data link or data link connection separately and independently of any other logical data link or data link connection that may exist at the stations involved.

12.1.1 LLC sublayer service specifications (General)

This subclause covers the services required of, or by, the MAC sublayer. In general, the services of a layer (or sublayer) are the capabilities it offers a user in the next higher layer (or sublayer). To provide its service, a layer (or sublayer) builds its functions on the services it requires from the next lower layer (or sublayer). Figure 144 illustrates this notion of service hierarchy and shows the relationship of the two correspondent *N* users and their associated *N* layer (or sublayer) peer protocol entities.

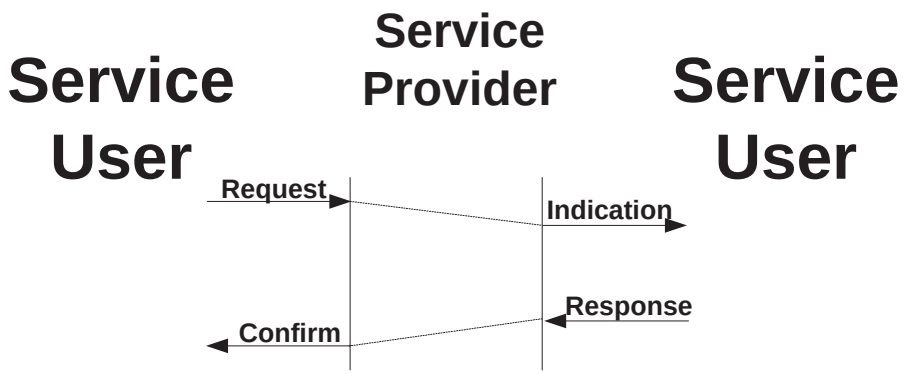


Figure 144—Service primitives

Services are specified by describing the information flow between the *N* user and the *N* layer (or sublayer). This information flow is modeled by discrete, instantaneous events that characterize the provision of a service. Each event consists of passing a service primitive from one layer (or sublayer) to the other through an *N* layer (or sublayer) service access point associated with an *N* user. Service primitives convey the information required in providing a particular service. These service primitives are an abstraction in that they specify only the service provided rather than the means by which that service is provided. This definition of service is independent of any particular interface implementation.

Services are specified by describing the service primitives and parameters that characterize each service. A service may have one or more related primitives that constitute the activity that is related to that service. Each service primitive may have zero or more parameters that convey the information required to provide that service.

Primitives are of four generic types as follows:

a) **REQUEST:** The request primitive is passed from the *N* user to the *N* layer (or sublayer) to request that a service be initiated.

b) **INDICATION:** The indication primitive is passed from the *N* layer (or sublayer) to the *N* user to indicate an internal *N* layer (or sublayer) event that is significant to the *N* user. This event may be logically related to a remote service request or may be caused by an event internal to the *N* layer (or sublayer).

c) **RESPONSE:** The response primitive is passed from the *N* user to the *N*-layer (or sublayer) to complete a procedure previously invoked by an indication primitive.

d) **CONFIRM:** The confirm primitive is passed from the *N* layer (or sublayer) to the *N*-user to convey the results of one or more associated previous service request(s).

Possible relationships among primitive types are illustrated by the time-sequence diagrams shown in Figure 145. The figure also indicates the logical relationship of the primitive types. Primitive types that occur earlier in time and are connected by dotted lines in the diagrams are the logical antecedents of subsequent primitive types.

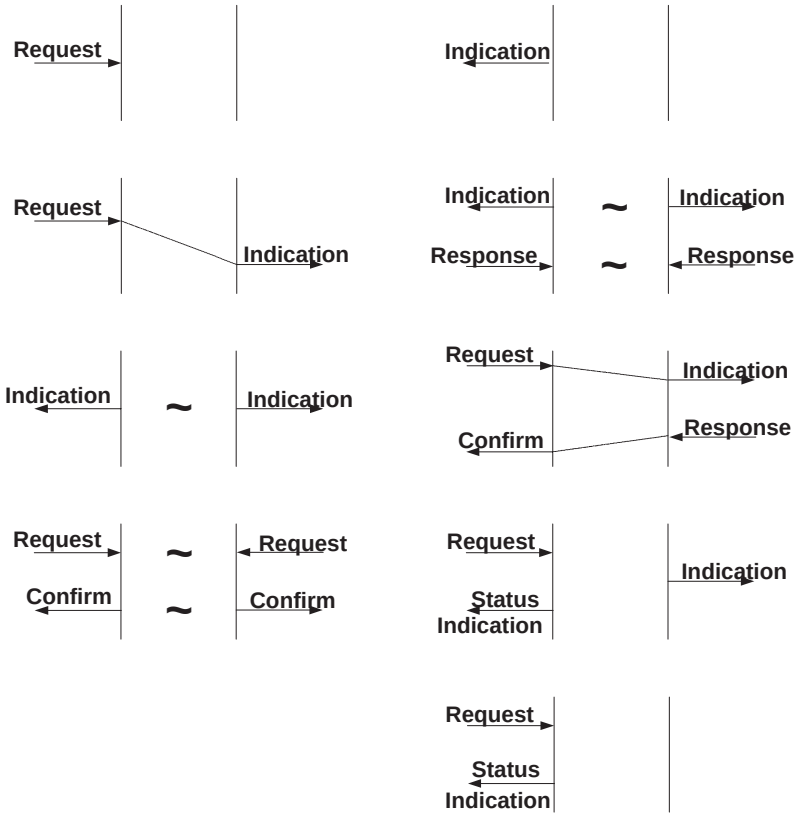


Figure 145—Sequence diagrams

12.2 LLC sublayer/MAC sublayer interface service specification

This subclause specifies the services required of the MAC sublayer by the LLC sublayer to allow the local LLC sublayer entity to exchange LLC data units with peer LLC sublayer entities. The services are described in an abstract way and do not imply any particular implementation or any exposed interface.

NOTE—Work is in progress to produce a single-service specification common to all the MAC sublayers. When this is available, it will be referenced in this Standard instead of the current MAC service text.

The interactions described are as follows:

— MA-UNITDATA request

— MA-UNITDATA indication

— MA-UNITDATA-STATUS indication

12.2.1 MA-UNITDATA request

12.2.1.1 Function

This primitive requests the transfer of an MSDU from a local LLC sublayer entity to a single peer LLC sublayer entity or multiple peer LLC sublayer entities in the case of group addresses.

12.2.1.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

MA-UNITDATA request (

source_address,
destination_address,
routing_information,
data,
priority,
service_class
)

The source_address parameter shall specify an individual MAC sublayer entity address. The destination_address parameter shall specify either an individual or a group MAC sublayer entity address. Together they shall contain sufficient information to create the source address (SA) and destination address (DA) fields that are appended to the LLC service data unit (SDU) by the local MAC sublayer entity as well as any physical layer address information (e.g., transmit frequency in broadband applications). The routing_information parameter specifies the route desired for the data unit transfer (a null value indicates that source routing is not to be used). The data parameter specifies the MAC service data unit to be transmitted by the MAC sublayer entity, which includes the DSAP, SSAP, C, and information (if present) fields as specified in Clause 3 of ISO/IEC 8802-2:1998(E), as well as sufficient information for the MAC sublayer entity to determine the length of the data unit. The priority parameter specifies the priority desired for the data unit transfer. The service_class parameter specifies the class of service desired for the data unit transfer.

12.2.1.3 When generated

This primitive is generated by the LLC sublayer entity to request that an MSDU be transferred to a peer LLC sublayer entity or entities. This can occur as a result of a request from higher layers of protocol or from an LSDU generated internally in the LLC sublayer, such as that required by Type 2 operation.

12.2.1.4 Effect on receipt

The receipt of this primitive shall cause the MAC sublayer entity to append all MAC-specified fields, including DA, SA, and any fields that are unique to the particular medium access method, and pass the properly formatted frame to the lower layers of protocol for transfer to the peer MAC sublayer entity or entities for subsequent transfer to the associated LLC sublayer(s).

12.2.1.5 Additional comments

A possible logical sequence of primitives associated with successful MAC service data unit transfer is illustrated in Figure 145.

12.2.2 MA-UNITDATA indication

12.2.2.1 Function

This primitive defines the transfer of an MSDU from the MAC sublayer entity to the LLC sublayer entity or entities in the case of group addresses. In the absence of errors, the contents of the data parameter are logically complete and unchanged relative to the data parameter in the associated MA-UNITDATA request primitive.

12.2.2.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

MA-UNITDATA indication (

source_address,
destination_address,
routing_information,
data,
reception_status,
priority,
service_class
)

The source_address parameter shall specify an individual address as specified by the SA field of the incoming frame. The destination_address parameter shall be either an individual or a group address as specified by the DA field of the incoming frame. The routing_information parameter specifies the route used for the data unit transfer (for MAC sublayers that do not support source routing, this field will be set to null). The data parameter specifies the MAC service data unit as received by the local MAC entity. The reception_status parameter indicates the success or failure of the incoming frame. The priority parameter specifies the priority desired for the data unit transfer. The service_class parameter specifies the class of service desired for the data unit transfer.

12.2.2.3 When generated

The MA-UNITDATA indication primitive is passed from the MAC sublayer entity to the LLC sublayer entity or entities to indicate the arrival of a frame at the local MAC sublayer entity. Frames are reported only if, at the MAC sublayer, they are validly formatted and received without error and their destination address designates the local MAC sublayer entity.

12.2.2.4 Effect on receipt

The effect on receipt of this primitive by the LLC sublayer is dependent on the validity and content of the frame.

12.2.2.5 Additional comments

If the local MAC sublayer entity is designated by the destination_address parameter of an MA-UNITDATA request primitive, the indication primitive will also be invoked by the MAC sublayer entity to the local LLC sublayer entity. This full-duplex characteristic of the MAC sublayer may be due to unique functionality within the MAC sublayer or full-duplex characteristics of the lower layers (e.g., all frames transmitted to the broadcast address will invoke MA-UNITDATA indication primitives at all stations in the network, including the station that generated the request).

12.2.3 MA-UNITDATA-STATUS indication

12.2.3.1 Function

This primitive has local significance and shall provide the LLC sublayer with status information for a previous associated MA-UNITDATA request primitive.

12.2.3.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

MA-UNITDATA-STATUS indication (

source_address,
destination_address,
transmission_status,
provided_priority,
provided_service_class
)

The source_address parameter shall be an individual MAC sublayer entity address as specified in the associated MA-UNITDATA request primitive. The destination_address parameter shall be either an individual or a group MAC sublayer entity address as specified in the associated MA-UNITDATA request primitive. The Transmission_Status parameter is used to pass status information back to the local requesting LLC sublayer entity. The types of status that can be associated with this primitive are dependent on the particular imple-mentation as well as the type of MAC sublayer that is used (e.g., “excessive collisions” may be a status returned by a CSMA/CD MAC sublayer entity). The provided_priority parameter specifies the priority that was used for the associated data unit transfer. The provided_service_class parameter specifies the class of service provided for the data unit transfer.

12.2.3.3 When generated

The MA-UNITDATA-STATUS indication primitive is passed from the MAC sublayer entity to the LLC sublayer to indicate the status of the service provided for a previous associated MA-UNITDATA request primitive.

12.2.3.4 Effect on receipt

The effect on receipt of this primitive by the LLC sublayer is dependent on the type of operation employed by the LLC sublayer entity.

12.2.3.5 Additional comments

It is assumed that sufficient information is available to the LLC sublayer entity to associate the status with the appropriate request.

12.3 Bluetooth interfaces

Figure 146 illustrates the events and actions performed by an implementation of the L2CAP layer. Client and server SAPs simply represent the initiator of the request and the acceptor of the request, respectively. An application-level client would both initiate and accept requests. The naming convention is as follows. The interface between two layers (vertical interface) uses the prefix of the lower layer offering the service to the higher layer, e.g., L2CA. The interface between two entities of the same layer (horizontal interface) uses the prefix of the protocol (adding a P to the layer identification), e.g., L2CAP. Events coming from above are called requests (Req), and the corresponding replies are called confirms (Cfm). Events coming from below are called indications (Ind), and the corresponding replies are called responses (Rsp). Responses that require further processing are called pending (Pnd). The notation for confirms and responses assumes positive replies. Negative replies are denoted by a “Neg” suffix such as L2CAP_ConnectCfmNeg.

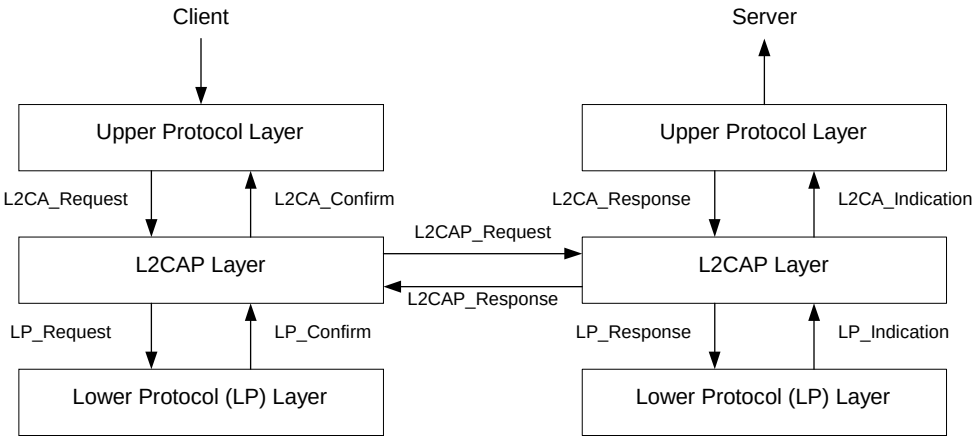


Figure 146—L2CAP actions and events

While requests for an action always result in a corresponding confirmation (for the successful or unsuccessful satisfaction of the action), indications do not always result in corresponding responses. This is especially true if the indications are informative about locally triggered events, e.g., seeing the LP_QoSViolationInd or L2CA_TimeOutInd.

12.3.1 Message Sequence chart of layer interactions

Figure 147 uses a message sequence chart to illustrate the normal sequence of events. The two outer vertical lines represent the L2CA interface on the initiator (the device issuing a request) and the acceptor (the device responding to the initiator's request). Request commands at the L2CA interface result in Requests defined by the protocol. When the protocol communicates the request to the acceptor, the remote L2CA entity presents the upper protocol with an Indication. When the acceptor's upper protocol responds, the response is packaged by the protocol and communicated back the to initiator. The result is passed back to the initiator's upper protocol using a confirm message.

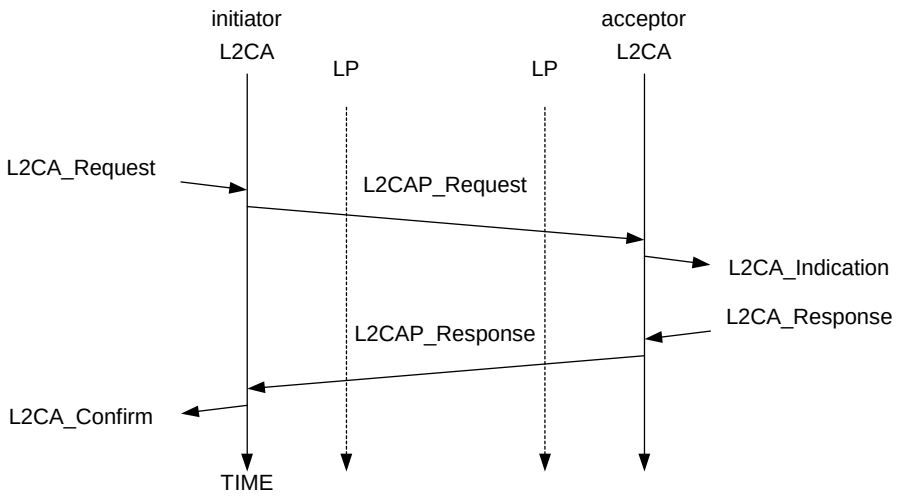


Figure 147—L2CA interaction

12.3.2 Relationship of Bluetooth protocol entities to IEEE 802 constructs

Figure 148 shows a mapping of the IEEE 802 protocol concepts to the Bluetooth components described in this Standard. The PHY is all of the Radio portion and part of the Baseband. The MAC contains the L2CAP and the rest of the Baseband. The Link Manager, Link Manager Protocol, and HCI functions form the MAC Management function. The PHY Management functions, like synchonization and generation of various frequency hopping sequences generation, are incorporated within the Baseband.

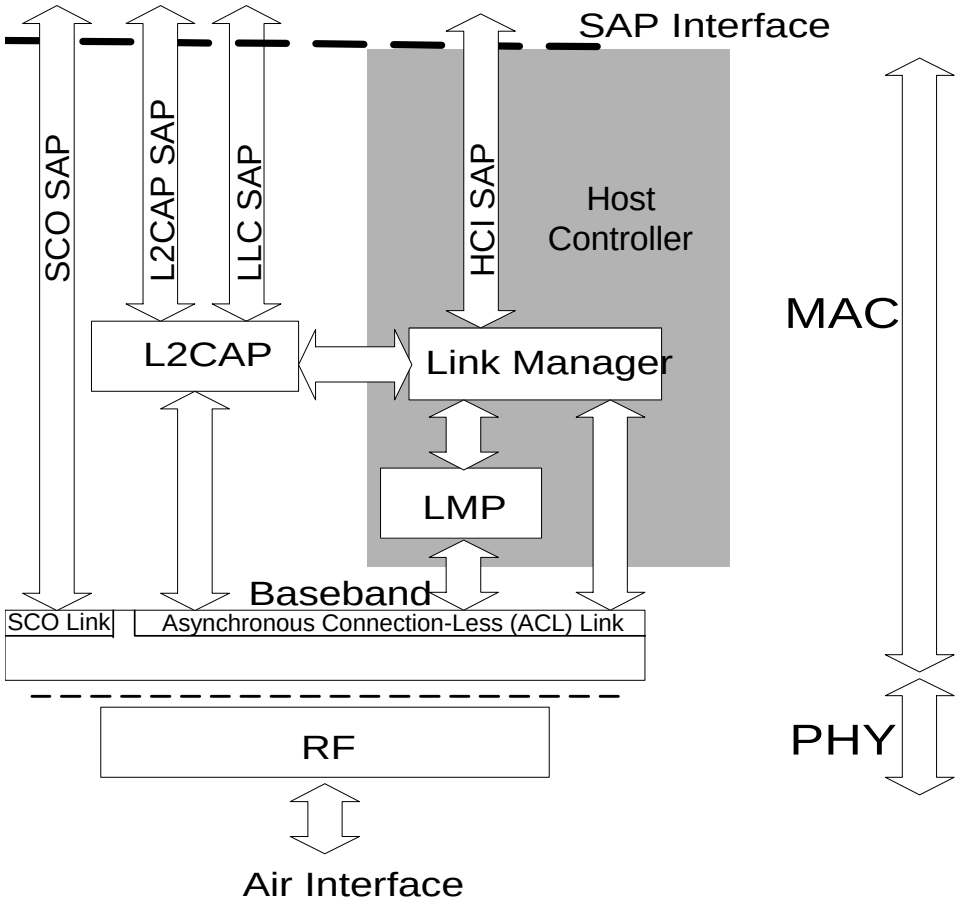


Figure 148—Bluetooth protocol entities mapped onto IEEE 802 constructs

12.3.2.1 Client SAP interface (Upper-layer) to L2CAP request event primitives (10.3.1.4):

L2CA_ConnectReq (10.3.1.4 and 10.7.3)
L2CA_ConfigReq (10.3.1.4 and 10.7.4)
L2CA_DisconnectReq (10.3.1.4 and 10.7.6)
L2CA_DataWrite (10.3.1.4 and 10.7.7)
L2CA_DataRead (10.7.9 and 10.7.8)
L2CA_GroupCreate (10.7.9)
L2CA_GroupClose (10.7.10)
L2CA_GroupAddMember (10.7.11)
L2CA_GroupRemoveMember (10.7.12)
L2CA_GroupMembership (10.7.13)
L2CA_Ping (10.7.14)
L2CA_InfoReq (implied by L2CA_GetInfo primitive 10.7.15)
L2CA_DisableCLT (10.7.16)
L2CA_EnableCLT (10.7.17)

12.3.2.2 Server SAP interface (Upper-layer) to L2CAP response events primitives (10.3.1.4):

L2CA_ConnectRsp (10.3.1.4 and 10.7.3)
L2CA_ConnectRspNeg (10.3.1.4 implied by response parameter of the L2CA_ConnectRsp primitive)
L2CA_ConnectPnd (10.3.2.4 implied by response parameter of the L2CA_ConnectRsp primitive)
L2CA_ConfigRsp (10.3.1.4 and 10.7.5)
L2CA_ConfigRspNeg (10.3.1.4 implied by result parameter of the L2CA_ConfigRsp primitive)
L2CA_DisconnectRsp (10.3.1.4 implied by result parameter of the L2CA_Disconnect_Req primitive)
L2CA_DataWriteRsp (implied by result parameter of the L2CA_DataWrite primitive)
L2CA_DataReadRsp (implied by result parameter of the L2CA_DataRead primitive)
L2CA_GroupCloseRsp (implied by result parameter of the L2CA_GroupClose primitive)
L2CA_GroupAddMemberRsp (implied by result parameter of L2CA_GroupAddMember primitive)
L2CA_GroupRemoveMemberRsp (implied by result parameter of L2CA_GroupRemoveMember primitive)
L2CA_GroupMembershipRsp (implied by result parameter of L2CA_GroupMembership primitive)
L2CA_EchoRsp (implied by result parameter of L2CA_Ping primitive)
L2CA_InfoRsp (implied by L2CA_GetInfo primitive 10.7.15)
L2CA_DisableCLTRsp (implied by result parameter of L2CA_DisableCLT primitive)
L2CA_EnableCLTRsp (implied by result parameter of L2CA_EnableCLT primitive)

12.3.2.3 L2CAP to client SAP interface (Upper-layer) action primitives (10.3.2.4):

L2CA_ConnectCfm (10.3.2.4)
L2CA_ConnectCfmNeg (10.3.2.4)
L2CA_ConfigCfm (10.3.2.4)
L2CA_ConfigCfmNeg (10.3.2.4)
L2CA_DisconnectCfm (10.3.2.4)

12.3.2.4 L2CAP to server SAP interface (Upper-layer) event indication primitives (10.7.1):

L2CA_ConnectInd (10.3.2.4)
L2CA_ConfigInd (10.3.2.4)
L2CA_DisconnectInd (10.3.2.4)
L2CA_QosViolationInd (10.3.2.4)
L2CA_TimeOutInd (10.3.2.4)

12.3.2.5 HCI SAP primitives:

HCI_Inquiry (11.2.5.1)
HCI_Inquiry_Cancel (11.2.5.2)
HCI_Periodic_Inquiry_Mode (11.2.5.3)
HCI_Exit_Periodic_Inquiry_Mode (11.2.5.4)
HCI_Create_Connection (11.2.5.5)
HCI_Disconnect (11.2.5.6)
HCI_Add_SCO_Connection (11.2.5.7)
HCI_Accept_Connection_Request (11.2.5.8)
HCI_Reject_Connection_Request (11.2.5.9)
HCI_Link_Key_Request_Reply (11.2.5.10)
HCI_Link_Key_Request_Negative_Reply (11.2.5.11)
HCI_PIN_Code_Request_Reply (11.2.5.12)
HCI_PIN_Code_Request_Negative_Reply (11.2.5.13)
HCI_Change_Connection_Packet_Type (11.2.5.14)
HCI_Authentication_Requested (11.2.5.15)
HCI_Set_Connection_Encryption (11.2.5.16)
HCI_Change_Connection_Link_Key (11.2.5.17)
HCI_Master_Link_Key (11.2.5.18)
HCI_Remote_Name_Request (11.2.5.19)
HCI_Read_Remote_Supported_Features (11.2.5.20)
HCI_Read_Remote_Version_Information (11.2.5.21)
HCI_Read_Clock_Offest (11.2.5.22)
HCI_Hold_Mode (11.2.6.1)
HCI_Sniff_Mode (11.2.6.2)
HCI_Exit_Sniff_Mode (11.2.6.3)

HCI_Park_Mode (11.2.6.4)
HCI_Exit_Park_Mode (11.2.6.5)
HCI_Qos_Setup (11.2.6.6)
HCI_Role_Discovery (11.2.6.7)
HCI_Switch_Role (11.2.6.8)
HCI_Read_Link_Policy_Settings (11.2.6.9)
HCI_Write_Link_Policy_Settings (11.2.6.10)
HCI_Set_Event_Mask (11.2.7.1)
HCI_Reset (11.2.7.2)
HCI_Set_Event_Filter (11.2.7.3)
HCI_Flush (11.2.7.4)
HCI_Read_PIN_Type (11.2.7.5)
HCI_Write_PIN_Type (11.2.7.6)
HCI_Create_New_Unit_Key (11.2.7.7)
HCI_Read_Stored_Link_Key (11.2.7.8)
HCI_Write_Stored_Link_Key (11.2.7.9)
HCI_Delete_Stored_Link_Key (11.2.7.10)
HCI_Change_Local_Name (11.2.7.11)
HCI_Read_Local_Name (11.2.7.12)
HCI_Read_Connection_Accept_Timeout (11.2.7.13)
HCI_Write_Connection_Accept_Timeout (11.2.7.14)
HCI_Read_Page_Timeout (11.2.7.15)
HCI_Write_Page_Timeout (11.2.7.16)
HCI_Read_Scan_Enable (11.2.7.17)
HCI_Write_Scan_Enable (11.2.7.18)
HCI_Read_Page_Scan_Activity (11.2.7.19)
HCI_Write_Page_Scan_Activity (11.2.7.20)
HCI_Read_Inquiry_Scan_Activity (11.2.7.21)
HCI_Write_Inquiry_Scan_Activity (11.2.7.22)
HCI_Read_Authentication_Enable (11.2.7.23)
HCI_Write_Authentication_Enable (11.2.7.24)
HCI_Read_Encryption_Mode (11.2.7.25)
HCI_Write_Encryption_Mode (11.2.7.26)
HCI_Read_Class_of_Device (11.2.7.27)
HCI_Write_Class_of_Device (11.2.7.28)
HCI_Read_Voice_Setting (11.2.7.29)
HCI_Write_Voice_Setting (11.2.7.30)
HCI_Read_Automatic_Flush_Timeout (11.2.7.31)
HCI_Write_Automatic_Flush_Timeout (11.2.7.32)
HCI_Read_Num_Broadcast_Retransmissions (11.2.7.33)
HCI_Write_Num_Broadcast_Retransmissions (11.2.7.34)
HCI_Read_Hold_Mode_Activity (11.2.7.35)
HCI_Write_Hold_Mode_Activity (11.2.7.36)
HCI_Read_Transmit_Power_level (11.2.7.37)
HCI_Set_Host_Controller_To_Host_flow_Control (11.2.7.40)
HCI_Host_Buffer_Size (11.2.7.41)
HCI_Host_Number_Of_Completed_Packets (11.2.7.42)
HCI_Read_Link_Supervision_Timeout (11.2.7.43)
HCI_Write_Link_Supervision_Timeout (11.2.7.44)
HCI_Read_Number_Of_Supported_IAC (11.2.7.45)
HCI_Read_Current_IAC_LAP (11.2.7.46)
HCI_Write_Current_IAC_LAP (11.2.7.47)
HCI_Read_Page_Scan_Period_Mode (11.2.7.48)
HCI_Write_Page_Scan_Period_Mode (11.2.7.49)
HCI_Read_Page_Scan_Mode (11.2.7.50)
HCI_Write_Page_Scan_Mode (11.2.7.51)
HCI_Read_Local_Version_Information (11.2.8.1)
HCI_Read_Local_Supported_Features (11.2.8.2)
HCI_Read_Buffer_Size (11.2.8.3)
HCI_Read_Country_Code (11.2.8.4)
HCI_Read_BD_ADDR (11.2.8.5)

12.3.2.6 SCO SAP primitives:

SCO Protocol Data Unit Indications
SCO Protocol Data Unit Requests

12.3.2.7 Possible HCI events (11.3.1):

Inquiry Complete Event (11.3.2.1)
Inquiry Result Event (11.3.2.2)
Connection Complete Event (11.3.2.3)
Connection Request Event (11.3.2.4)
Disconnection Complete Event (11.3.2.5)
Authentication Complete Event (11.3.2.6)
Remote Name Request Complete Event (11.3.2.7)
Encryption Change Event (11.3.2.8)
Change Connection Link Key Complete Event (11.3.2.9)
Master Link Key Complete Event (11.3.2.10)
Read Remote Supported Features Complete Event (11.3.2.11)
Read Remote Version Information Complete Event (11.3.2.12)
QoS Setup Complete Event (11.3.2.13)
Command Complete Event (11.3.2.14)
Command Status Event (11.3.2.15)
Hardware Error Event (11.3.2.16)
Flush Occurred Event (11.3.2.17)
Role Change Event (11.3.2.18)
Number of Completed Packets Event (11.3.2.19)
Mode Change Event (11.3.2.20)
Return_Link_Keys Event (11.3.2.21)
PIN Code Request Event (11.3.2.22)
Link Key Request Event (11.3.2.23)
Link Key Notification Event (11.3.2.24)
Loopback Command Event (11.3.2.25)
Data Buffer Overflow Event (11.3.2.26)
Max Slots Change Event (11.3.2.27)
Read Clock Offset Complete Event (11.3.2.28)
Connection Packet Type Changed Event (11.3.2.29)
QoS Violation Event (11.3.2.30)
Page Scan Mode Change Event (11.3.2.31)
Page Scan Repetition Mode Change Event (11.3.2.32)