# Bluetooth Document

This technical file will describe detail knowledge about Bluetooth.

Issued by Yin Xiao Tao(yinxiaotao@innofidei.com /yinxts@gmail.com /cameronyin@eyou.com)
On 3 November 2009. (Beijing)

# Contents

# 1 Basic knowledge

## 1.1 Bluetooth-This name come from?

丹麦国王 Harald Blaatand Bluetooth 二世致力于协调丹麦与挪威两国能和平沟通，后来成为短距离无线传输的标准名词。

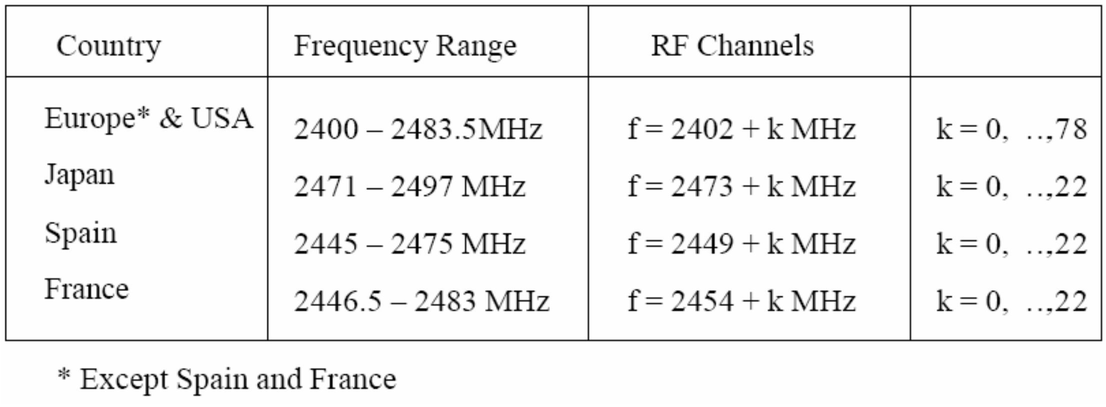
## 1.2 Bluetooth advantage features

Wireless, short distance, low power consult, cheap, support audio and video, many profiles.

## 1.3 RF

1. ISM band (license free) 2.4~2.4835GHZ, 79channels(each 1MHZ).

2. In order to comply with out-of-band regulations in each country, a guard band(2M) is used at the lower and upper band(3.5M) edge.



| Country | Frequency Range | RF Channels | |
|---|---|---|---|
| Europe* & USA | 2400 – 2483.5MHz | f = 2402 + k MHz | k = 0, …,78 |
| Japan | 2471 – 2497 MHz | f = 2473 + k MHz | k = 0, …,22 |
| Spain | 2445 – 2475 MHz | f = 2449 + k MHz | k = 0, …,22 |
| France | 2446.5 – 2483 MHz | f = 2454 + k MHz | k = 0, …,22 |

* Except Spain and France

3. Frequency Hopping

In order to avoid interference from other signals(microwave ovens, baby monitors, garage door openers, adjacent Bluetooth links…), the frequency will hopping in these 79channels.

- 1600 hops/s (nominal connection mode)(625us)

- 3200 hops/s (page and inquiry modes)
- Same hopping sequence for all devices in a piconet

4. Time Division Multiple Access(TDMA)
5. Auto Retransmission

FEC(1/3): retransmit three times if failed.
FEC(2/3):

## 1.4 Bluetooth version

| | |
|---|---|
| V1.2 | 721kB (1MB/S) |
| V2.0+EDR | 2.1M |
| V2.1+EDR | 3M |
| V3.0+HS | 24M (New Feature:1. broad cast channel, 2. 802.11hight speed) |

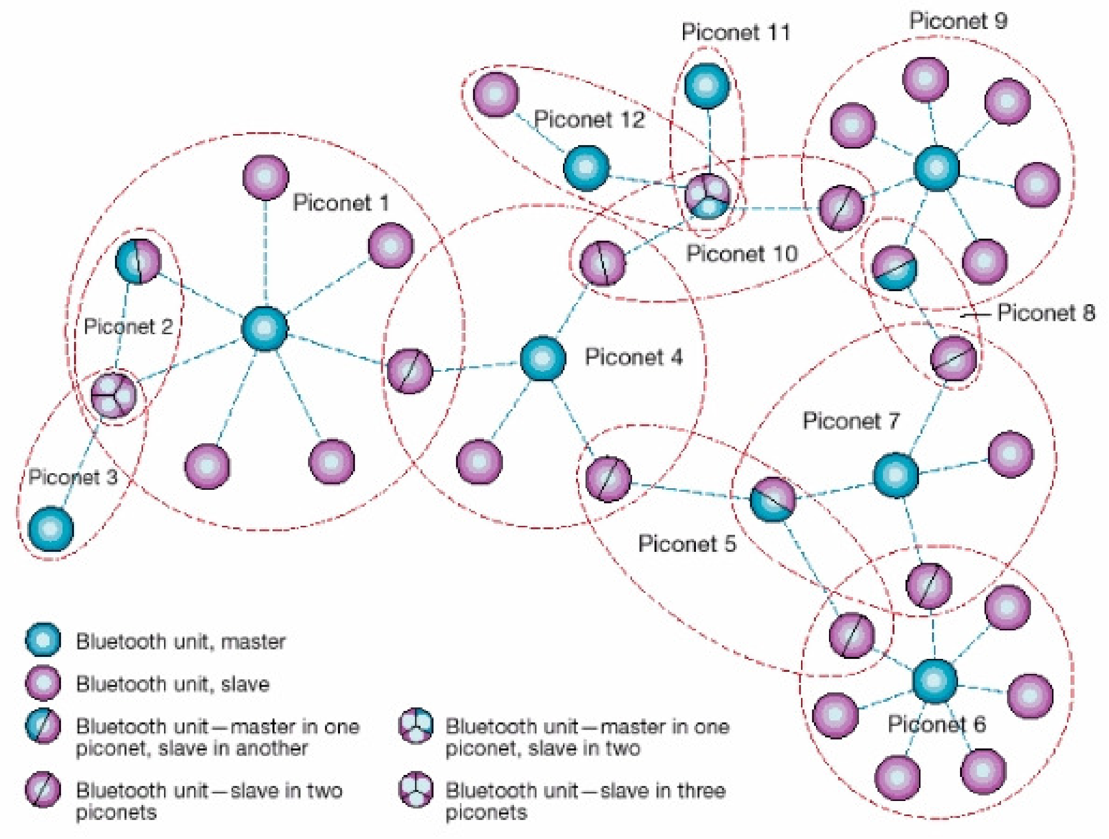
## 1.5 Bluetooth classes

| | |
|---|---|
| Class1: 100mv(20db) | 100 Meter |
| Class2: 2.5mv(4db) | 10 Meter |
| Class3: 1 mv(0db) | 1 Meter |

# 2 Base band(BB)

## 2.1 Piconet

Each piconet can have only one master, and up to seven slaves.


Piconet 11
Piconet 9
Piconet 12
Piconet 1
Piconet 10
Piconet 2
Piconet 8
Piconet 4
Piconet 7
Piconet 3
Piconet 5
Piconet 6
Bluetooth unit, master
Bluetooth unit, slave
Bluetooth unit—master in one piconet, slave in another
Bluetooth unit—master in one piconet, slave in two
Bluetooth unit—slave in two piconets
Bluetooth unit—slave in three piconets

## 2.2 Physical links

Asynchronous Connection-less(ACL)

- A master may have multiple ACL links to the slaves, but only one ACL link between any two devices
- Broadcast packets are ACL packets not addressed to any specific slaves

Synchronous Connection Oriented(SCO)

- A master can support up to three SCO links
- A slave can support up to three SCO links to same one master; and two SCO links to different master.
- SCO packets are never retransmitted

## 2.3 Bluetooth stack

We always heard ‘Bluetooth stack’, but we are confused. Bluetooth stack include protocols and profiles. The following is the architecture (figure 1).

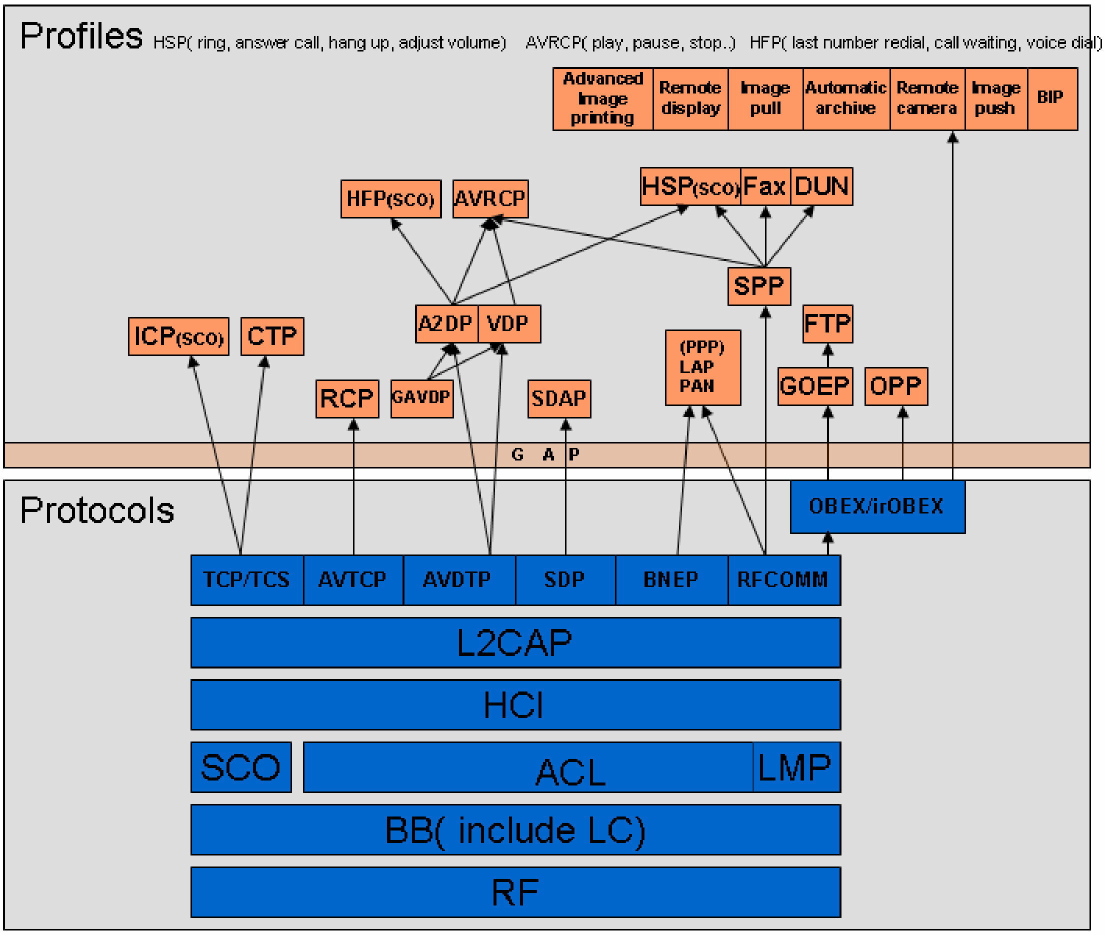


**Figure 1**

We only focus on the following stack map(not include RFCOMM upper level, and audio):

We only study RFCOMM L2CAP HCI SDP protocols.

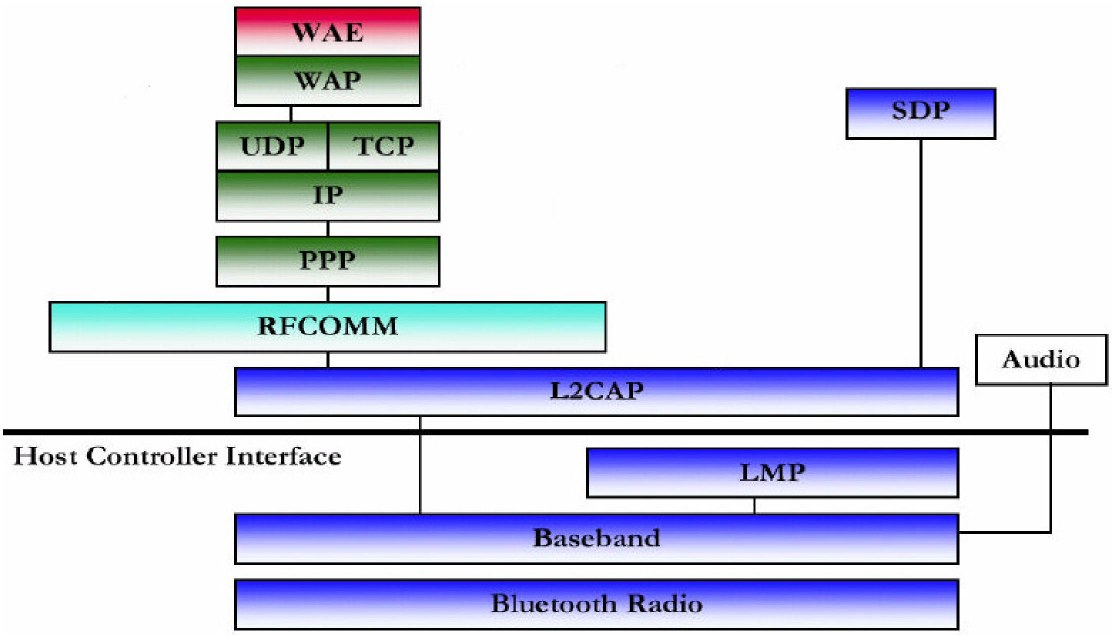


**Figure 2**

# 3 Package structure

## 3.1 Bluetooth Packets:

- Little Endian
- Each packet consists of 3 entities: Access code, Header, Payload

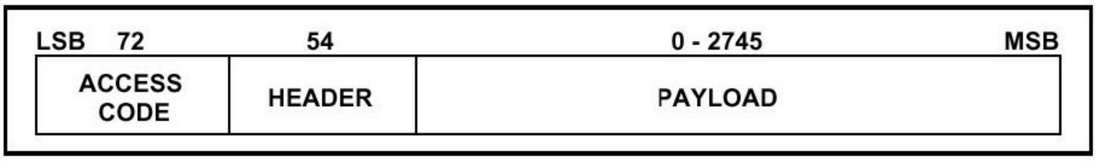


*Figure 4.1: Standard packet format.*

## 3.2 Detail information of each part

**3.2.1 ACCESS CODE [72bits(CAC, have trailer), 68bits]**

**3.2.1.1 Access code types**

Channel Access Code (CAC) — during connection

Device Access Code (DAC) — paging / paging scan

Inquiry Access Code (IAC)

—General Inquiry Access Code (GIAC).

Used by al devices during inquiry procedures. Fixed 0x9E8B33

—Dedicated Inquiry Access Code (DIAC)

For inquiry specific devices like printers and handsets. 0x9E8B00—0x9E8B3F.

Generic Access Profile only use LIAC(0x9E8B00)

**3.2.1.2 Access code format**

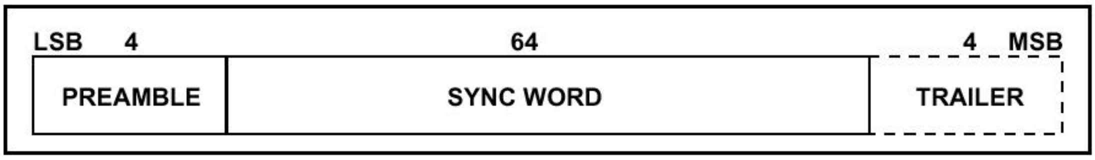


*Figure 4.2: Access code format*

**PREAMBLE**

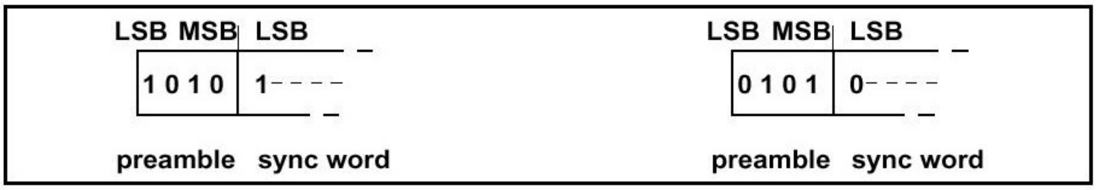


*Figure 4.3: Preamble*

The sequence 1010 or 0101, depending on whether the LSB of the following sync word is 1 or 0

**TRAILER**

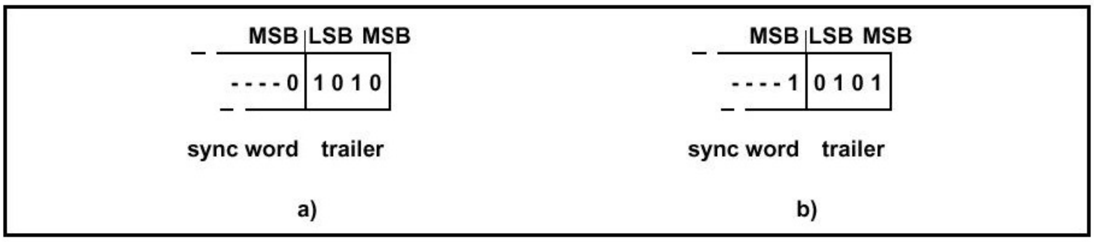


*Figure 4.4: Trailer in CAC when MSB of sync word is 0 (a), and when MSB of sync word is 1 (b).*

**SYNC WORD**

Used to hopping with connected BT device.

| | | |
|---|---|---|
| **BCH Parity Word (34bits)** | **LAP (24bits)** | **Sequence (6bits)** |

### 3.2.2 HEADER

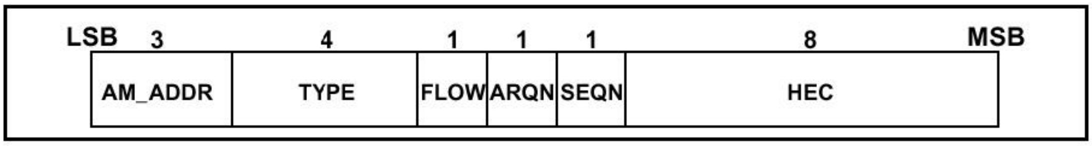


*Figure 4.5: Header format.*

Total 18 bits protected by a FEC code of 1/3, resulting in 54(18*3) bits.

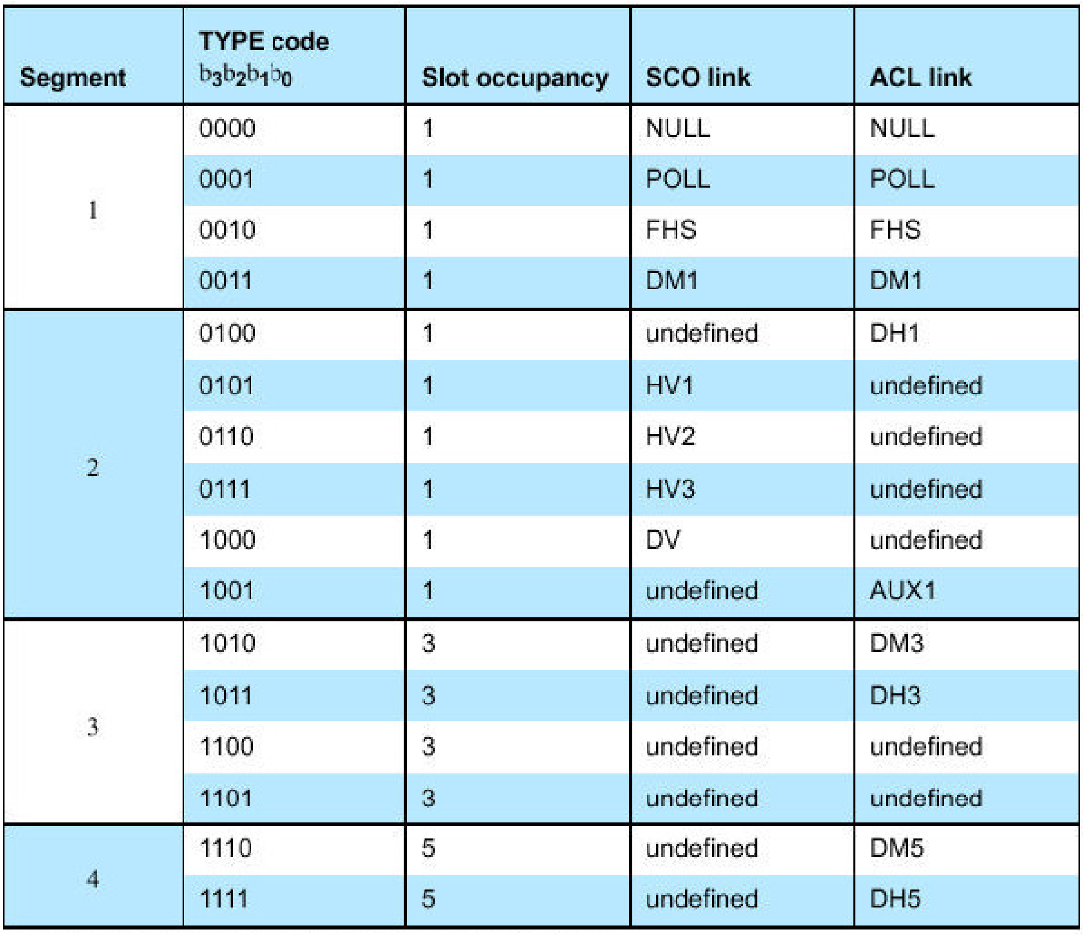
AM_ADDR

00:broadcast

01~07: distinguish the active members in the piconet

TYPE

SCO, ACL, {NULL(auto send inform the ARQN&FLOW), POLL, ID, FHS}.



| 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 |

*Table 4.2: Packets defined for SCO and ACL link types*

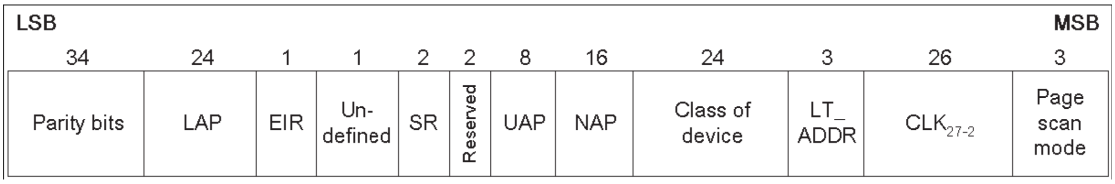
Common packet types

ID --- identity packet consists of the DAC or IAC, has a fixed length of 68bits.

NULL --- no payload, fixed length 126bits, used to return link info to the source regarding the success of the previous transmission(ARQN), or the status of the RX buffer(FLOW).

POLL --- used by the master in a piconet to check the presence of the slaves, must be respond, or the LC(link control protocol) connection will time out. Don't affect the ARQN and SEQN.

FHS --- Used in page master response, inquiry response and in role switch. Contains sender's LAP and real-time clock info.



LSB MSB

| 34 | 24 | 1 | 1 | 2 | 2 | 8 | 16 | 24 | 3 | 26 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Parity bits | LAP | EIR | Un-defined | SR | Reserved | UAP | NAP | Class of device | LT_ ADDR | $CLK_{27-2}$ | Page scan mode |

Format of FHS payload.

DM1 --- refer to Bluetooth core spec

SCO packet types

HV1, HV2, HV3, DV packet refer to Bluetooth core spec

ACL packet types

The ended number of the following types indicate that how many time slots will be used to send.

DM1 : carries data info only, coded with a rate 2/3 FEC which adds 5 parity bits to every 10bits of the payload data.

| header (8bits) | data (15bytes) | CRC (16bits) |
|---|---|---|

DH1 : similar to DM1, except that the payload isn't FEC encoded, can carry up to 30 info bytes(include 16bits CRC code)

DM3 : DM1 with an extended payload, 123 info bytes(2 bytes header) + 16bits CRC. Use three slots freq shouldn't hop before these three slots.

DH3 : similar to DM3, except no FEC encoded, 185 info bytes(include 2 bytes header)+16bits CRC code.

DM5 : is a DM1 packet with an extended payload. Up to 226 info bytes(include 2 bytes header)+16bits CRC code.

DH5 : similar to DM5, except no FEC, up to 341 info bytes(include 2bytes header)+16bits CRC.
AUX1 : similar to DH1, except no CRC code, up to 30 info bytes(include 1byte header)

**FLOW**

Used for flow control of packets over the ACL link. When the RX buffer in the recipient is full and is not emptied, FLOW=0 is returned to stop the transmission of data temporarily.
---only applicable to ACL packets
---packets containing only link control info(ID, POLL, NULL) or SCO can still be received

**ARQN**

ARQN=1(0)informs the sender of a successful(failed) transfer of payload data.
---positive acknowledge: ACK
---Negative acknowledge: NAK
---NAK is default acknowledge

**SEQN**

Provides a sequential number scheme to order the data packet stream.

**HEC**

Header Error Check is simply a CRC code (use UAP or all-zero encode).

### 3.2.3 Payload

| Payload header (8/16bits) | Payload (0-2712bits) | CRC (16bits) |
|---|---|---|

**Payload header**

| L_CH[2] | Flow[1] | Length[5or9] | Undef[0or4] |
|---|---|---|---|

L_CH:

--- 00 undefined

--- 01 Continue of an L2CAP message

--- 10 Start of an L2CAP message

--- 11 LMP message

Flow

Flow control at the L2CAP level, set by LM

**CRC**

All packets CRC use UAP calculate, except for FHS packets send in inquiry response state(use zero)

# 4 Protocols introduce

## 4.1 link control LC protocol [base band]

Carried in the LC channel (L_CH[2]==11?error), usually send by FHS packets, will finish paged phase.

LC status: Standby, Inquiry, Inquiry Scan, Page, Page Scan, Connection(Active, Hold, Sniff, Park??)

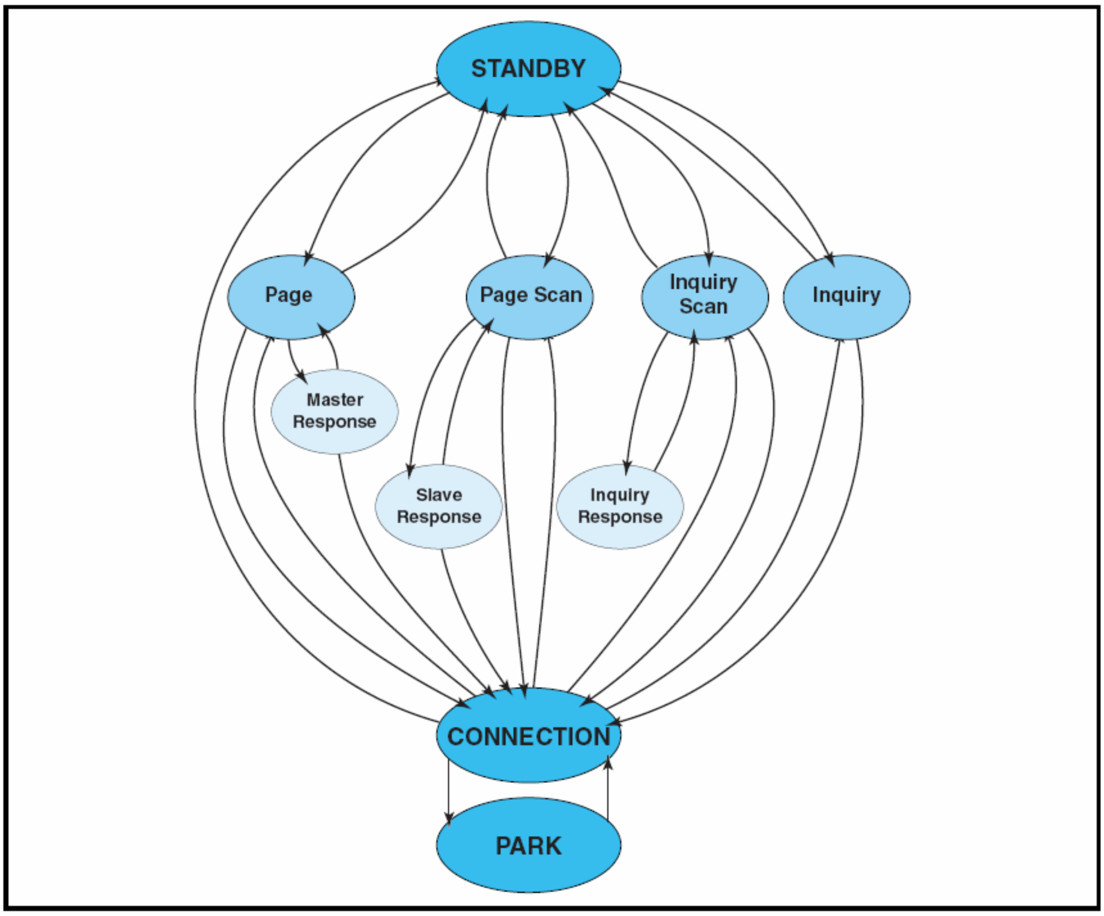


*Figure 8.1: State diagram of link controller.*

## 4.2 LMP protocol

LMP messages are used for link set-up, security and control, usually sent by DM1 packets(L_CH=11), higher priority than L2CAP, Not flow controlled FLOW bit=1.

- Attaching slaves to a piconet, and allocating their active member addresses
- Breaking connections to detach slaves from piconet
- Configuring the link including Master/Slave switches
- Establish ACL and SCO links
- Putting connections into low-power modes: Hold, Sniff, and Park
- Controlling test modes

### 4.2.1 Format of the LMP payload

| | | |
|---|---|---|
| TID | OpCode | content |

TID(1bit): transaction ID

0: if master initiated transaction

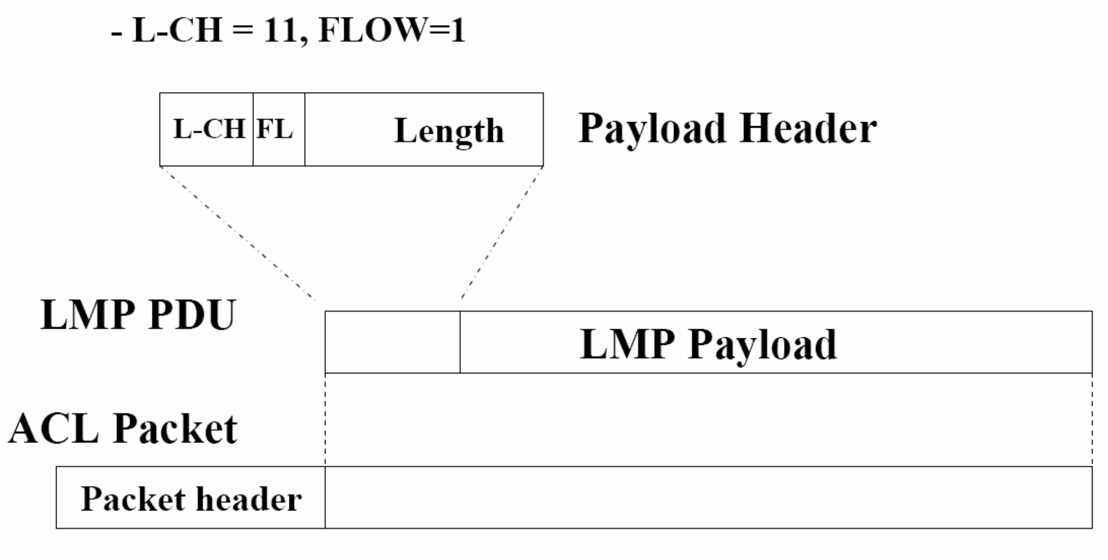
1: if slave initiated transaction

OpCode(7bit): Operation code

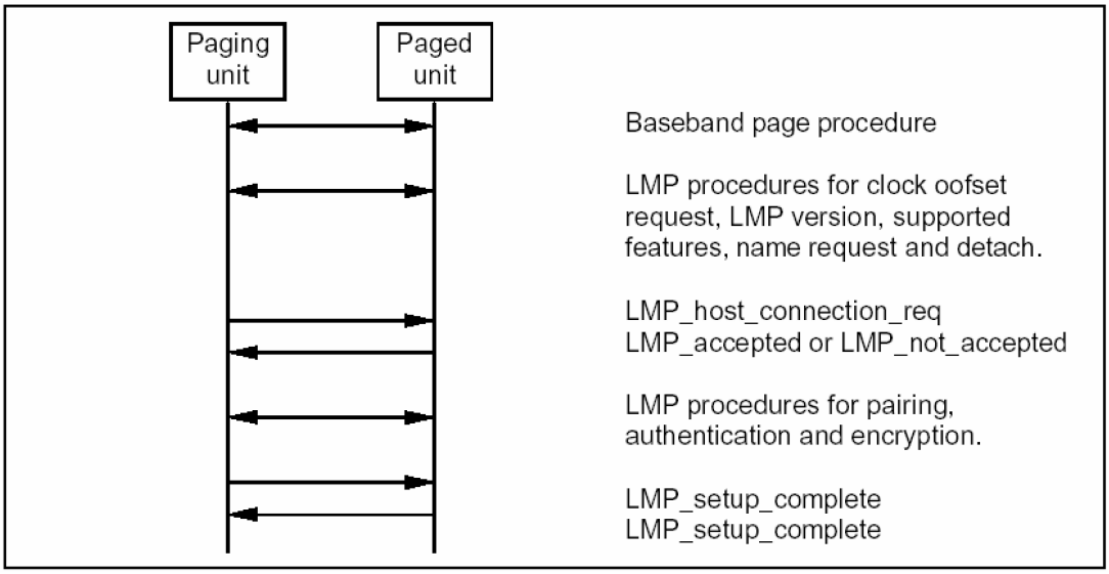
Identify the type of LMP message being sent

Content

Include one or more parameters, the length include in parameter.



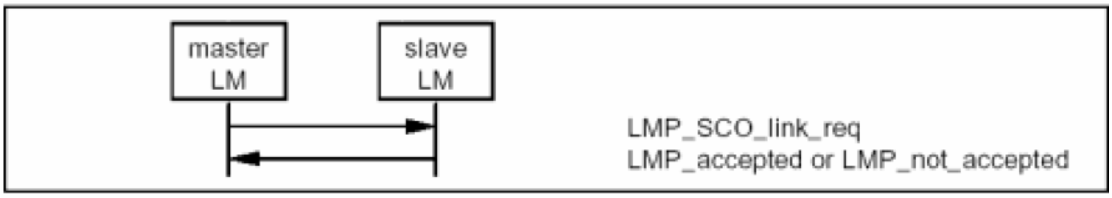
### 4.2.2 ACL link setup sequence


Paging unit
Paged unit
Baseband page procedure
LMP procedures for clock oofset request, LMP version, supported features, name request and detach.
LMP_host_connection_req
LMP_accepted or LMP_not_accepted
LMP procedures for pairing, authentication and encryption.
LMP_setup_complete
LMP_setup_complete

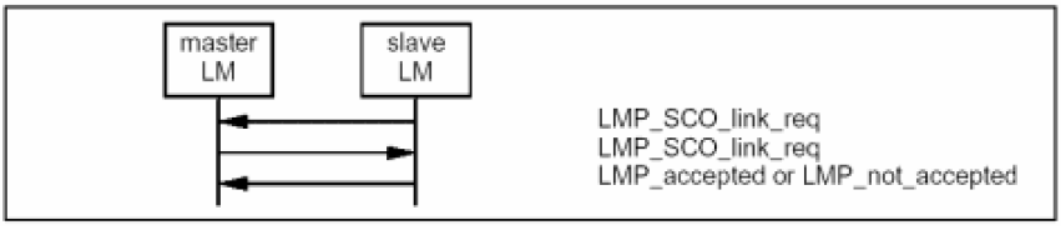
NOTE: authentication and pairing process refer to BT spec.

### 4.2.3 SCO Setup sequence

Once an ACL link has been established, either the Master or Slave can request a SCO link across the ACL link.



*Sequence 49: Master requests an SCO link.*



*Sequence 51: Master accepts slave's request for an SCO link.*

## 4.3 Logical Link Control and Adaptation Protocol (L2CAP)

Only used in ACL packet, the functions of this layer is:

- Protocol Multiplexing
- Segmentation and Reassembly (SAR)
- Group Management
- Quality of Service

### 4.3.1 Protocol Multiplexing

L2CAP must support protocol multiplexing(by defining logical channels) because the Baseband Protocol doesn't support any "type" field identifying the higher layer protocol (SDP, RFCOMM…).

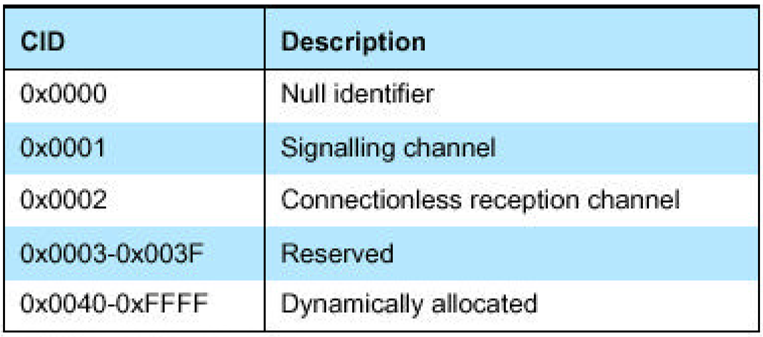
Logical channel types.

Signaling channel

Connection-oriented channel

Connectionless channel

Channel Identifier

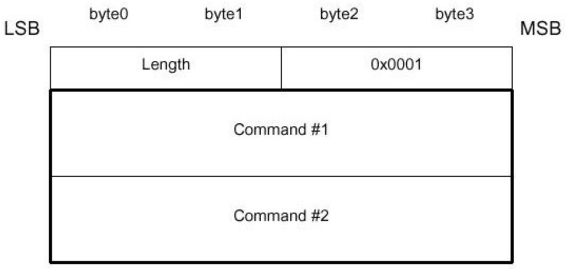


| CID | Description |
|---|---|
| 0x0000 | Null identifier |
| 0x0001 | Signalling channel |
| 0x0002 | Connectionless reception channel |
| 0x0003-0x003F | Reserved |
| 0x0040-0xFFFF | Dynamically allocated |

*Table 2.1: CID Definitions*

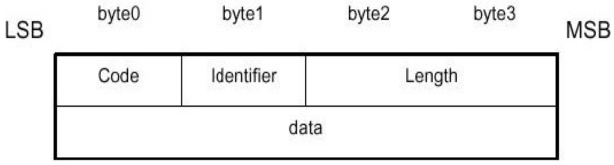
#### 4.3.1.1 Signalling channel

Data format



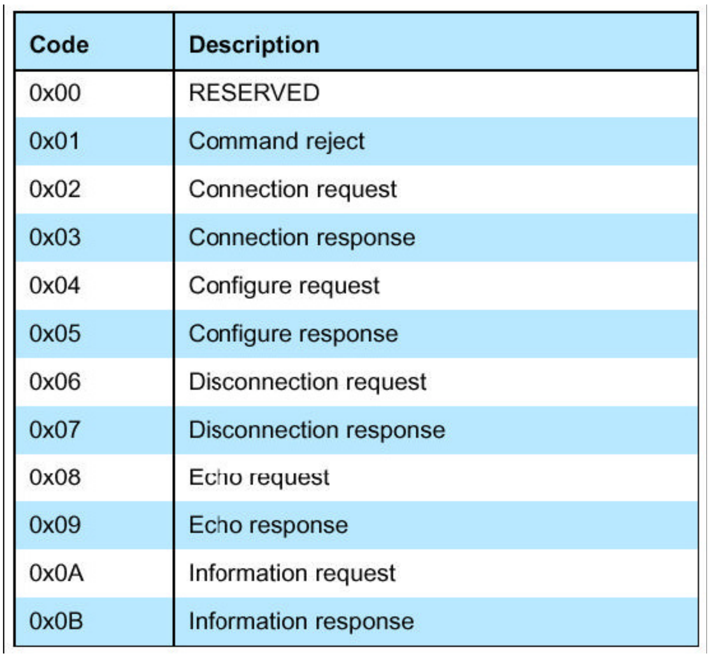
| LSB | byte0 | byte1 | byte2 | byte3 | MSB |
|---|---|---|---|---|---|
| | Length | | 0x0001 | | |
| | Command #1 | | | | |
| | Command #2 | | | | |

Command format



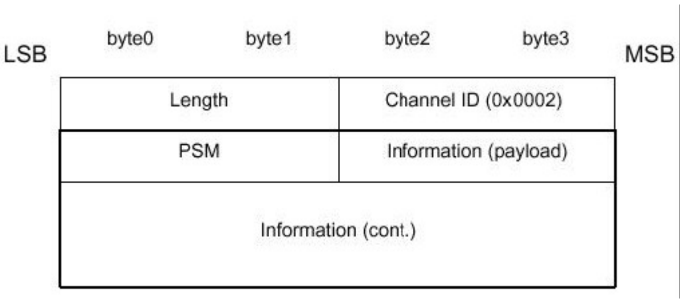
| LSB | byte0 | byte1 | byte2 | byte3 | MSB |
|---|---|---|---|---|---|
| | Code | Identifier | Length | | |
| | data | | | | |

Signal command code

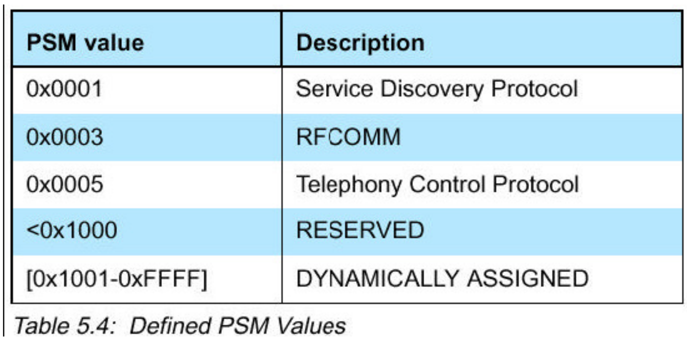


| Code | Description |
|---|---|
| 0x00 | RESERVED |
| 0x01 | Command reject |
| 0x02 | Connection request |
| 0x03 | Connection response |
| 0x04 | Configure request |
| 0x05 | Configure response |
| 0x06 | Disconnection request |
| 0x07 | Disconnection response |
| 0x08 | Echo request |
| 0x09 | Echo response |
| 0x0A | Information request |
| 0x0B | Information response |

### 4.3.1.2 Connectionless data channel



| LSB | byte0 | byte1 | byte2 | byte3 | MSB |
|---|---|---|---|---|---|
| | Length | | Channel ID (0x0002) | | |
| | PSM | | Information (payload) | | |
| | Information (cont.) | | | | |



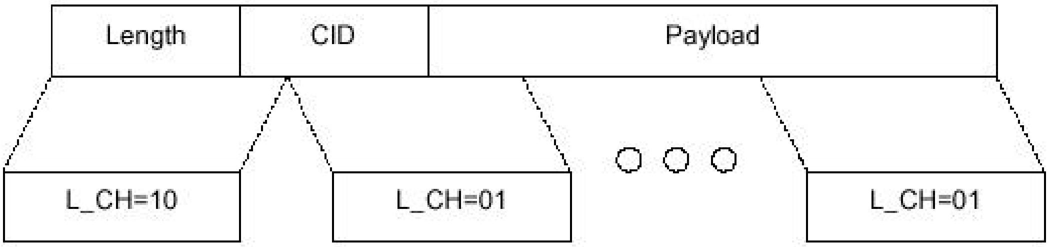
| PSM value | Description |
|---|---|
| 0x0001 | Service Discovery Protocol |
| 0x0003 | RFCOMM |
| 0x0005 | Telephony Control Protocol |
| <0x1000 | RESERVED |
| [0x1001-0xFFFF] | DYNAMICALLY ASSIGNED |

*Table 5.4: Defined PSM Values*

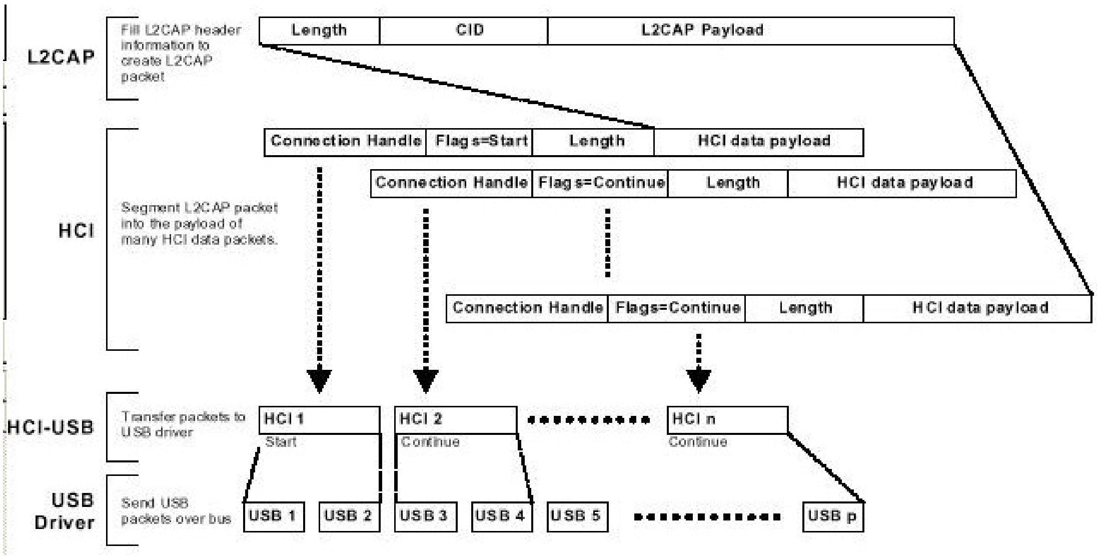
### 4.3.2 Segmentation and Reassembly

Large L2CAP packets must be segmented into multiple smaller Baseband packets.

Largest Baseband payload is 341bytes, MTU of L2CAP is 64Kbytes.

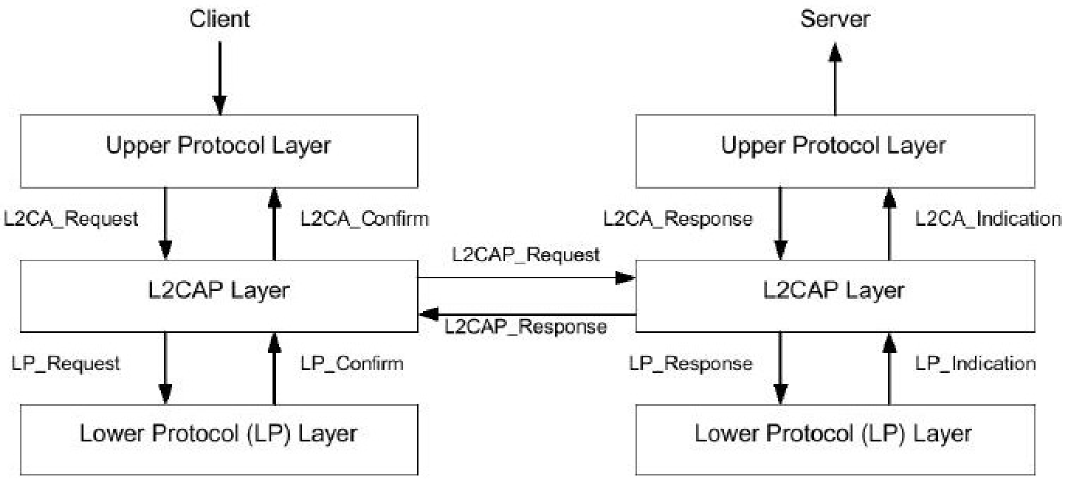


SAR Service in a unit with an HCI



### 4.3.3 L2CAP Interaction process

The process always like: request(C)->Indication(D)->Response(D)->Confirm(C)



### 4.3.4 Channel Operational State machine

CLOSED – channel not connected

W4_L2CAP_CONNECT_RSP – a L2CAP_ConnectReq has been sent, now waiting for L2CAP_ConnectRsp message

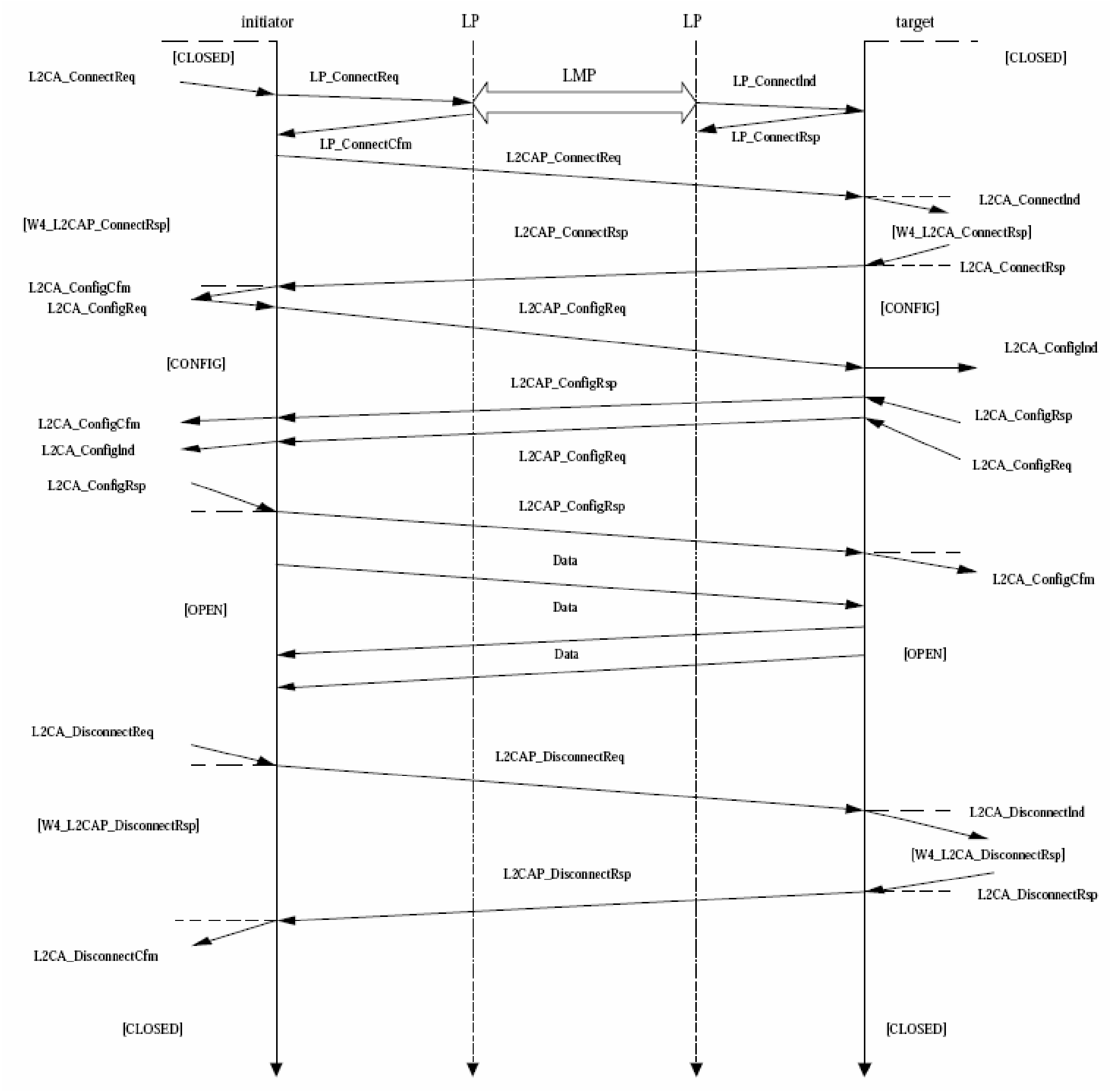
W4_L2CA_CONNECT_RSP – a L2CAP_ConnectReq has been received, and L2CAP_ConnectInd has been sent to the upper layer.

CONFIG – the connection has been established, negotiation is on going.

OPEN – connection established and configured, data can be sent.

W4_L2CAP_DISCONNECT_RSP – a L2CAP_DisconnectReq has been sent, waiting resp

W4_L2CA_DISCONNECT_RSP – a L2CAP_DisconnectReq has been received, L2CAP_DisconnectInd has been sent to the upper layer.



## 4.4 HCI TRANSPORT introduce

HCI transport protocol include: BlueCore Serial Protocol(BCSP)、H4/UART、H3/RS232、USB.
BCSP support error check and re-transmit mechanism, used in CSR NOT usb modules(include PCMCIA&CF card).

BlueZ support BCSP and H4.

BCCMD is a data format defined by CSR, used to send command from host to CSR module by either BCSP, H4, ….

## 4.5 HCI driver over UART

Function:

a) Packet up-level data, and send to BT module by UART;

b) Phase received data, and call upper protocol to phase.

Packet type

| HCI packet type | HCI packet indicator |
|---|---|
| HCI Command Packet | 0x01 |
| HCI ACL Data Packet | 0x02 |
| HCI SCO Data Packet | 0x03 |
| HCI Event Packet | 0x04 |
| Error Message Packet* | 0x05 |
| Negotiation Packet* | 0x06 |

*Table 2.1: HCI RS232 Packet Header*

Command packets used by the host to control the module

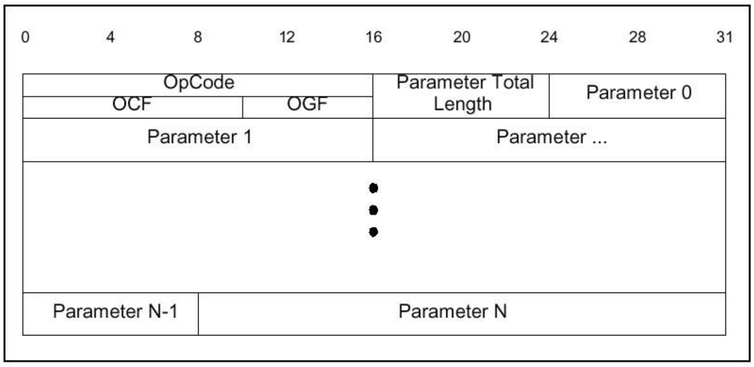


*Figure 4.1: HCI Command Packet*

Event packets used by the module to inform the host

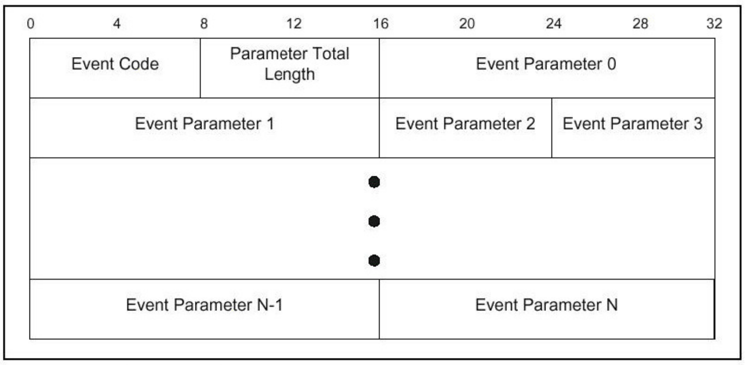


*Figure 4.2: HCI Event Packet*

Data packets to pass voice and data between host and module

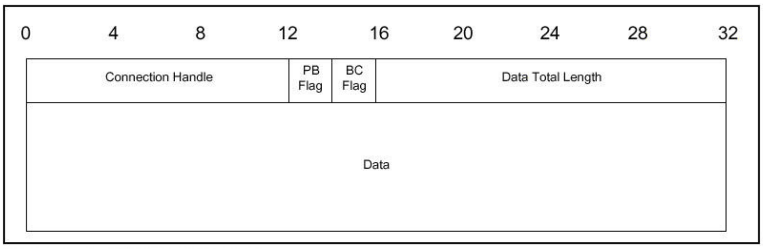


*Figure 4.3: HCI ACL Data Packet*

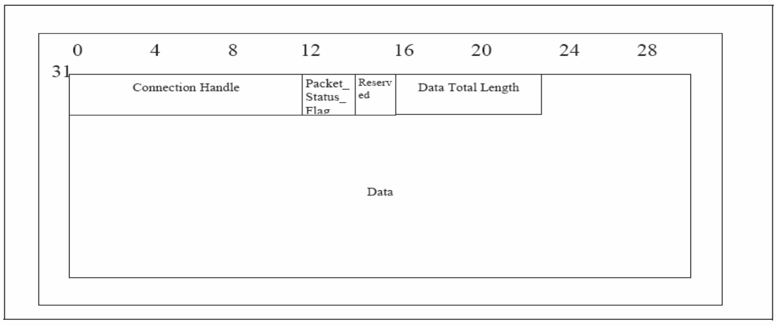


*Figure 5.3: HCI Synchronous Data Packet*

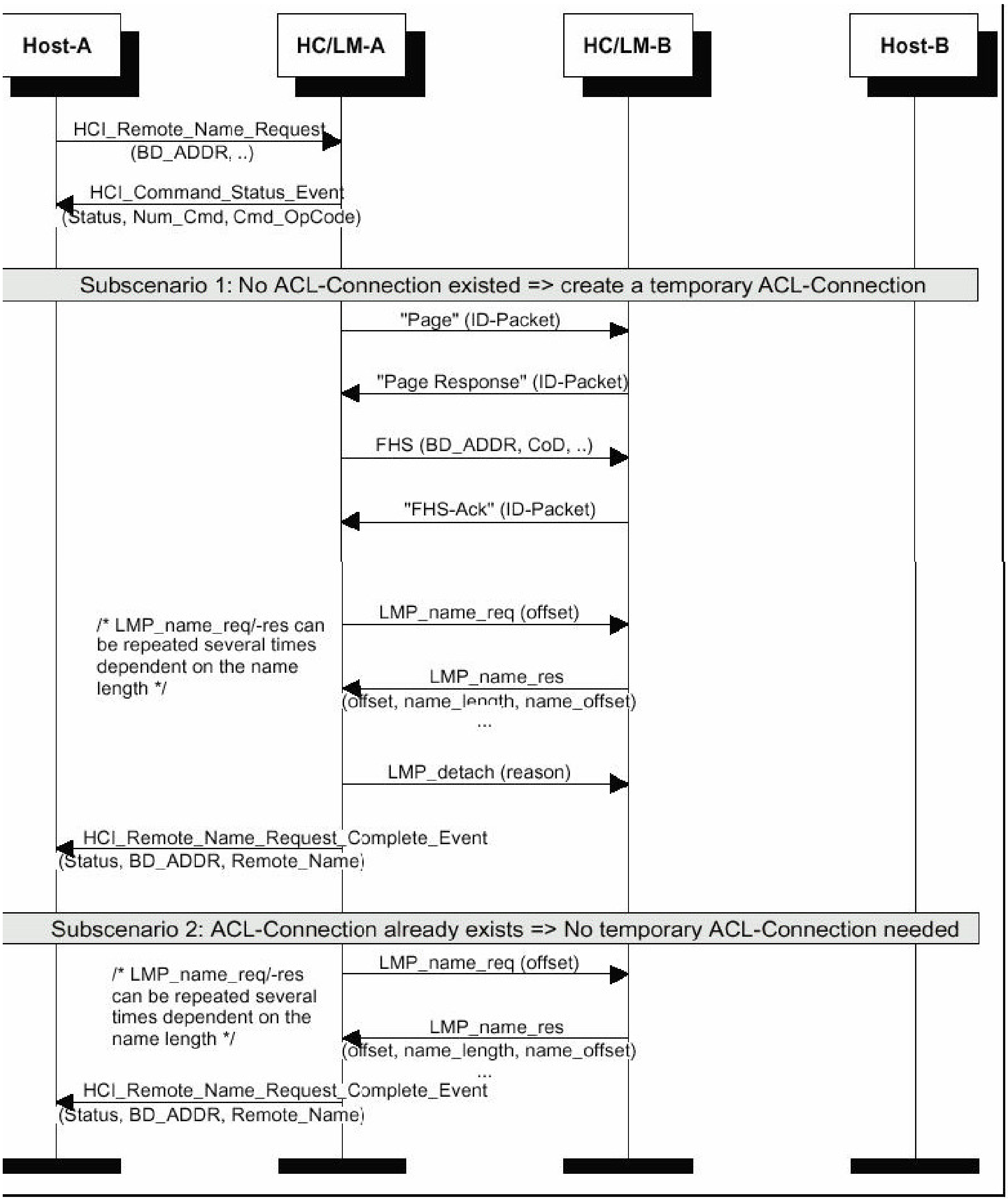
### 4.5.1 HCI commands of sub functions

1) Remote name request

A) Command and result

```
$ hcitool name    00:06:6e:19:1b:77
$ CSR - bc4
```

B) Chart:


Host-A
HC/LM-A
HC/LM-B
Host-B
HCI_Remote_Name_Request (BD_ADDR, ..)
HCI_Command_Status_Event (Status, Num_Cmd, Cmd_OpCode)
Subscenario 1: No ACL-Connection existed => create a temporary ACL-Connection
"Page" (ID-Packet)
"Page Response" (ID-Packet)
FHS (BD_ADDR, CoD, ..)
"FHS-Ack" (ID-Packet)
/* LMP_name_req/-res can be repeated several times dependent on the name length */
LMP_name_req (offset)
LMP_name_res (offset, name_length, name_offset)
...
LMP_detach (reason)
HCI_Remote_Name_Request_Complete_Event (Status, BD_ADDR, Remote_Name)
Subscenario 2: ACL-Connection already exists => No temporary ACL-Connection needed
/* LMP_name_req/-res can be repeated several times dependent on the name length */
LMP_name_req (offset)
LMP_name_res (offset, name_length, name_offset)
...
HCI_Remote_Name_Request_Complete_Event (Status, BD_ADDR, Remote_Name)

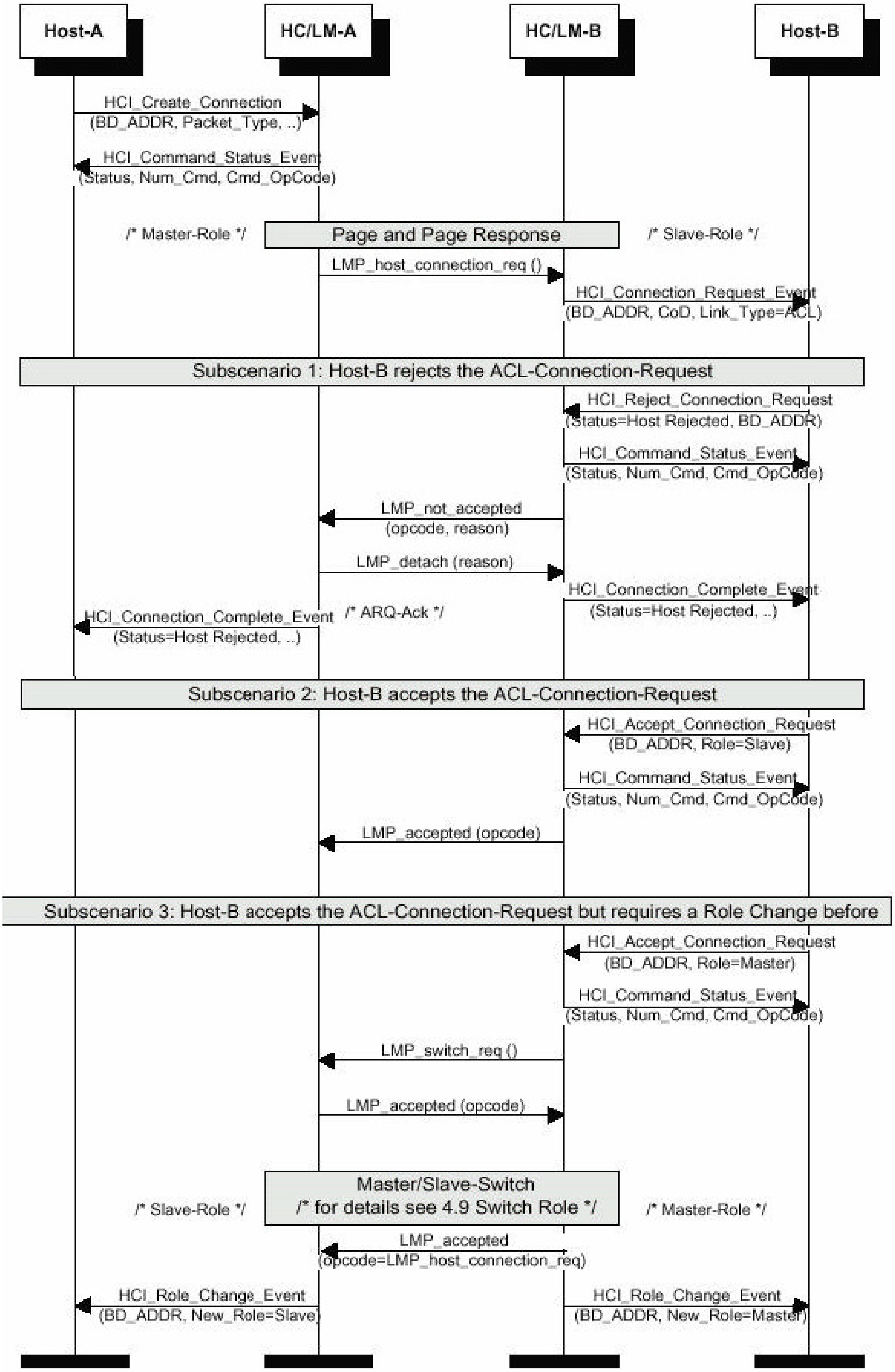
C) Raw data

```
hcidump -x -R
< 01 19 04 0A 77 1B 19 6E 06 00 02 00 00 00 [HCI command: remote name request]
> 04 0F 04 00 01 19 04                        [Event: command status]
> 04 07 FF 00 77 1B 19 6E 06 00 43 53 52 20 2D 20 62 63 34 00 [Event:rem nam req copt]
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

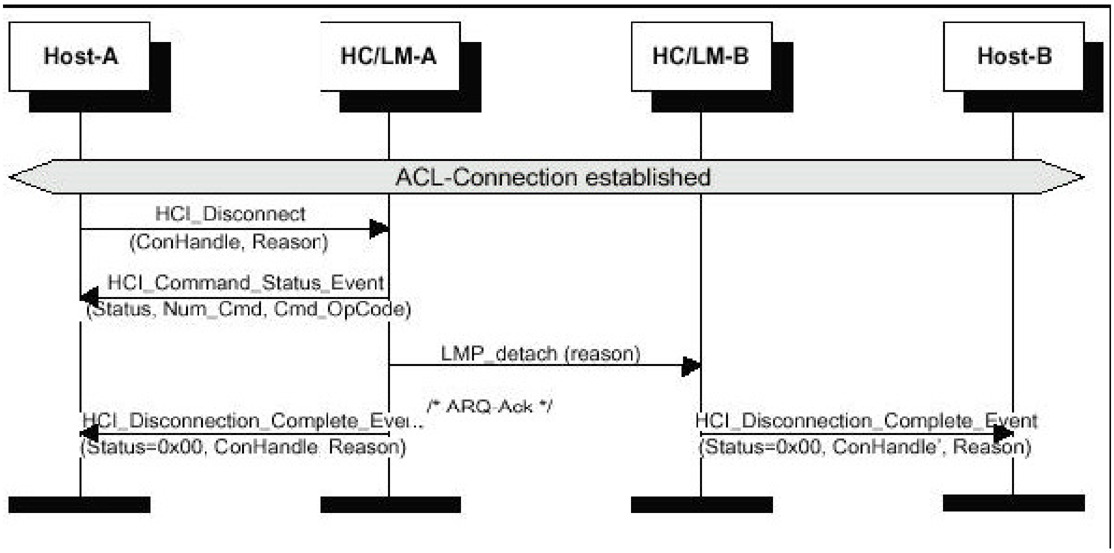
2) HCI create connection

Chart:


Host-A
HC/LM-A
HC/LM-B
Host-B
HCI_Create_Connection
(BD_ADDR, Packet_Type, ..)
HCI_Command_Status_Event
(Status, Num_Cmd, Cmd_OpCode)
/* Master-Role */
Page and Page Response
/* Slave-Role */
LMP_host_connection_req ()
HCI_Connection_Request_Event
(BD_ADDR, CoD, Link_Type=ACL)
Subscenario 1: Host-B rejects the ACL-Connection-Request
HCI_Reject_Connection_Request
(Status=Host Rejected, BD_ADDR)
HCI_Command_Status_Event
(Status, Num_Cmd, Cmd_OpCode)
LMP_not_accepted
(opcode, reason)
LMP_detach (reason)
HCI_Connection_Complete_Event
(Status=Host Rejected, ..)
HCI_Connection_Complete_Event
(Status=Host Rejected, ..)
/* ARQ-Ack */
Subscenario 2: Host-B accepts the ACL-Connection-Request
HCI_Accept_Connection_Request
(BD_ADDR, Role=Slave)
HCI_Command_Status_Event
(Status, Num_Cmd, Cmd_OpCode)
LMP_accepted (opcode)
Subscenario 3: Host-B accepts the ACL-Connection-Request but requires a Role Change before
HCI_Accept_Connection_Request
(BD_ADDR, Role=Master)
HCI_Command_Status_Event
(Status, Num_Cmd, Cmd_OpCode)
LMP_switch_req ()
LMP_accepted (opcode)
Master/Slave-Switch
/* for details see 4.9 Switch Role */
/* Slave-Role */
/* Master-Role */
LMP_accepted
(opcode=LMP_host_connection_req)
HCI_Role_Change_Event
(BD_ADDR, New_Role=Slave)
HCI_Role_Change_Event
(BD_ADDR, New_Role=Master)

3) Disconnect

Chart:


Host-A
HC/LM-A
HC/LM-B
Host-B
ACL-Connection established
HCI_Disconnect
(ConHandle, Reason)
HCI_Command_Status_Event
(Status, Num_Cmd, Cmd_OpCode)
LMP_detach (reason)
/* ARQ-Ack */
HCI_Disconnection_Complete_Event
(Status=0x00, ConHandle Reason)
HCI_Disconnection_Complete_Event
(Status=0x00, ConHandle', Reason)

## 4.6 RFCOMM protocol

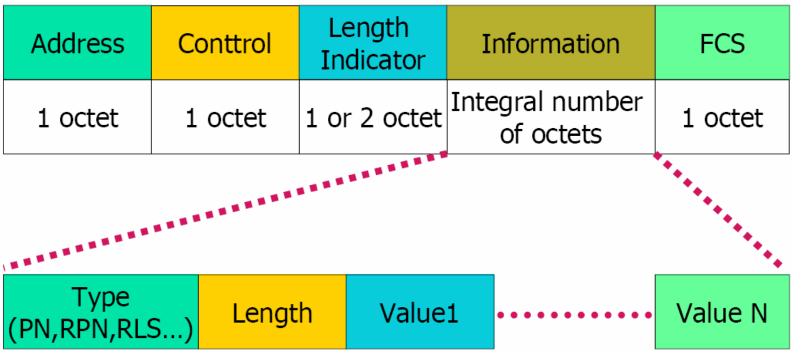
### 4.6.1 Features

1. Byte order: LSB → MSB, reading from left to right.
2. RFCOMM protocol provides emulation of serial ports over the L2CAP protocol, supports up to 60 simultaneous connections between two BT device.
   - A Data Link Connection Identifier(DLCI) identifies an ongoing connection between a client and a server application.
   - The DLCI is represented by 6 bits, but usually use range 2…61.(0 is channel; 1,62,63are reserved)
   - Server application on initiating device use DLCI:3, 5, 7 … 61;
     Server application on non-initiating device use DLCI:2, 4, 6 … 60.

### 4.6.2 Protocol introduce

RFCOMM is a subset of TS07.10, with a different Address field.

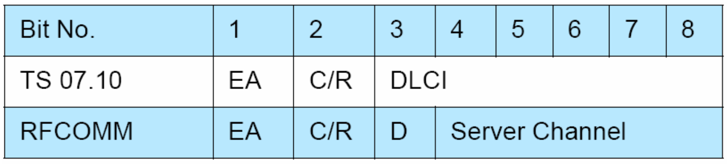
**4.6.2.1 Frame structure**



| Address | Conttrol | Length Indicator | Information | FCS |
|---|---|---|---|---|
| 1 octet | 1 octet | 1 or 2 octet | Integral number of octets | 1 octet |

| Type (PN,RPN,RLS...) | Length | Value1 | ………… | Value N |
|---|---|---|---|---|

1. Address Field:



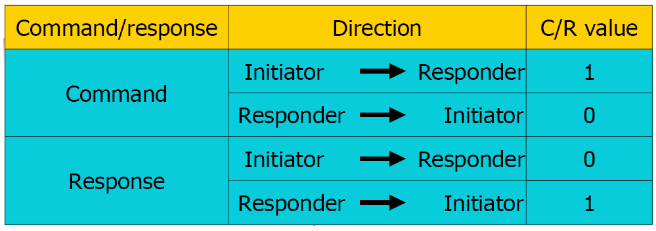
| Bit No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| TS 07.10 | EA | C/R | DLCI | | | | | |
| RFCOMM | EA | C/R | D | Server Channel | | | | |

*Table 5.2: The format of the Address Field*

EA: Set 1 = this octet is the last octet of the address filed

Set 0 = another octet of the address filed follows.

C/R:



| Command/response | Direction | C/R value |
|---|---|---|
| Command | Initiator → Responder | 1 |
| | Responder → Initiator | 0 |
| Response | Initiator → Responder | 0 |
| | Responder → Initiator | 1 |

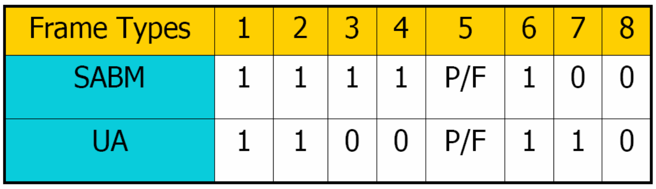
D: Set 1 (Initiator)

Set 0 (Responder)

Server Channel:

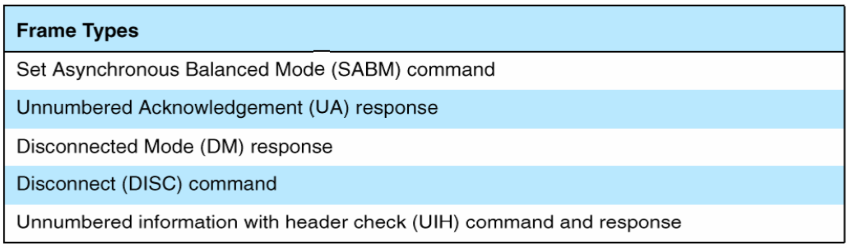
All commands send in DLCI0

2. Control Field



| Frame Types | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| SABM | 1 | 1 | 1 | 1 | P/F | 1 | 0 | 0 |
| UA | 1 | 1 | 0 | 0 | P/F | 1 | 1 | 0 |

Frame types



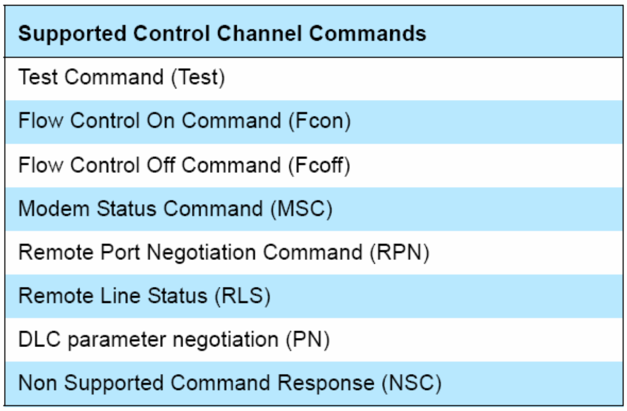
| **Frame Types** |
|---|
| Set Asynchronous Balanced Mode (SABM) command |
| Unnumbered Acknowledgement (UA) response |
| Disconnected Mode (DM) response |
| Disconnect (DISC) command |
| Unnumbered information with header check (UIH) command and response |

*Table 4.1: Supported frar ˎ types in RFCOMM*

- SABM, DISC, UA, DM used to create and release DLCI channel, data send on DLCI0
- Command/Response use UIH frame, data send on DLCI0
- User data use UIH frame, and send on DLCIx (created before)

P/F: refer to rfcomm protocol spec

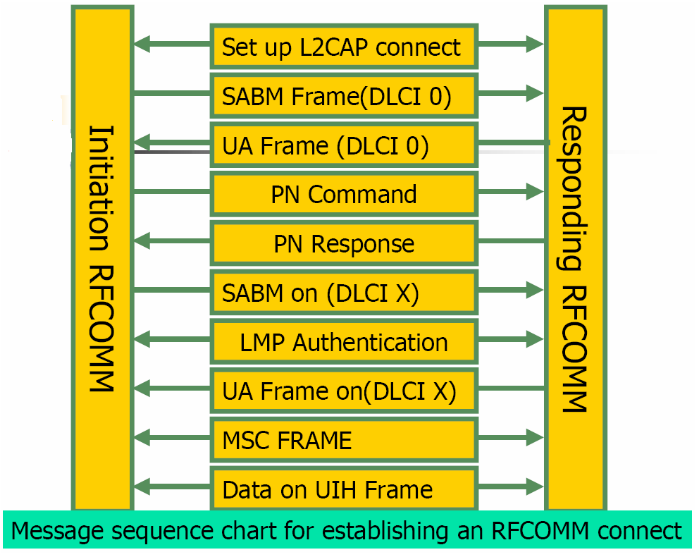
Commands (UIH frame on DLCI0):



| Supported Control Channel Commands |
|---|
| Test Command (Test) |
| Flow Control On Command (Fcon) |
| Flow Control Off Command (Fcoff) |
| Modem Status Command (MSC) |
| Remote Port Negotiation Command (RPN) |
| Remote Line Status (RLS) |
| DLC parameter negotiation (PN) |
| Non Supported Command Response (NSC) |

### 4.6.2.2 Connection set up and close down process

1. Set up process: SDP find RFCOMM Server channel number→Establish and L2CAP channel→Start the RFCOMM multiplexes[Establish control channel(DLCI0); Establish Data channel(DLCIx).



Message sequence chart for establishing an RFCOMM connect

2. Close down process

Closing the multiplexes by sending a DISC command frame on DLCIx, if it's the last link, the RFCOMM session will be release too.

3．Raw Data example [refer to printed paper(hand write note)]

```
$ sudo /usr/sbin/hcidump -x -R
   HCI sniffer - Bluetooth packet analyzer ver 1.42
   device: hci0 snap_len: 1028 filter: 0xffffffff
   < 01 05 04 0D 77 1B 19 6E 06 00 18 CC 02 00 00 00 01 [create connection]
   > 04 0F 04 00 01 05 04 [command status]
   > 04 03 0B 00 2E 00 77 1B 19 6E 06 00 01 00 [connection completed]
   < 02 2E 20 0A 00 06 00 01 00 0A 01 02 00 02 00 [info request]
   < 01 1B 04 02 2E 00 [read remote supported feature]
   > 04 13 05 01 2E 00 01 00 [number of completed packets]
   > 04 20 07 77 1B 19 6E 06 00 01 [page scan repeat mode change event (bd addr remote dev,
                       mode)]
   > 04 0F 04 00 00 1B 04 [command status]
   > 04 1B 03 2E 00 05 [max slots change]
   > 04 0F 04 00 01 00 00 [command status]
   < 01 0D 08 04 2E 00 0F 00 [write link policy]
   > 04 0B 0B 00 2E 00 FF FF 8F FE 9B F9 00 80 [read remote supported feature complete event]
   > 04 0E 06 01 0D 08 00 2E 00 [command complete evnet]
   < 01 19 04 0A 77 1B 19 6E 06 00 02 00 00 00 [remote name request]
   > 04 0F 04 00 01 19 04 [command status]
   > 04 07 FF 00 77 1B 19 6E 06 00 43 53 52 20 2D 20 62 63 34 00 [remote name request
   complete event]
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
   < 02 2E 20 0C 00 08 00 01 00 02 02 04 00 03 00 40 00 [connect request]
   > 04 13 05 01 2E 00 01 00 [number of completed packets]
   > 02 2E 20 10 00 0C 00 01 00 03 02 08 00 40 00 40 00 00 00 00
     00 [connection response]
   < 02 2E 20 10 00 0C 00 01 00 04 03 08 00 40 00 00 00 01 02 F5
     03 [configuration request]
   > 02 2E 20 0E 00 0A 00 01 00 05 03 06 00 40 00 00 00 00 00 [configure response]
   > 02 2E 20 10 00 0C 00 01 00 04 01 08 00 40 00 00 00 01 02 A0
     02 [configure request]
   < 02 2E 20 12 00 0E 00 01 00 05 01 0A 00 40 00 00 00 00 00 01
     02 A0 02 [configure response]
   < 02 2E 20 08 00 04 00 40 00 03 3F 01 1C [SABM(dlci=0)]
   > 04 13 05 01 2E 00 01 00 [number of completed packets]
   > 02 2E 20 08 00 04 00 40 00 03 73 01 D7 [UA]
   < 02 2E 20 12 00 0E 00 40 00 03 EF 15 83 11 02 F0 07 00 9B 02
     00 07 70 [UIH(test, dlci=0)]
   > 02 2E 20 12 00 0E 00 40 00 01 EF 15 81 11 02 00 07 00 9B 02
     00 00 AA [UIH(test, response, dlci=0)]
   < 02 2E 20 08 00 04 00 40 00 0B 3F 01 59 [SABM(dlci=2)]
   > 02 2E 20 08 00 04 00 40 00 0B 73 01 92 [UA(dlci=2)]
   < 02 2E 20 0C 00 08 00 40 00 03 EF 09 E3 05 0B 8D 70 [UIH(MSC, cr=1, dlci=0)]
```

> 04 13 05 01 2E 00 01 00 [number of completed packets]

> 02 2E 20 0C 00 08 00 40 00 03 EF 09 E3 05 09 8D 70 [UIH(MSC, response, cr=1, dlci=0]

< 02 2E 20 0C 00 08 00 40 00 03 EF 09 E1 05 0B 8D 70 [UIH(MSC, cr=0, dlci=0)]

> 02 2E 20 0C 00 08 00 40 00 01 EF 09 E1 05 0B 8D AA [UIH(MSC, cr=0, dlci=0)]

< 02 2E 20 0E 00 0A 00 40 00 0B EF 0D 68 65 6C 6C 6F 21 9A [send “hello!” to device]

## 4.7 SDP protocol

Carried out by L2CAP connection.

### 4.7.1 General work flow

Step1: Service search

client send a request → server

← return all the supported service record handles

Step2: Service attribute

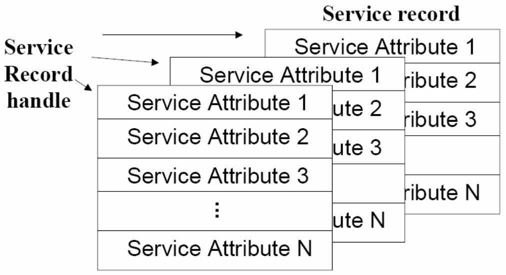
Client request the service attributes use the record handle, got in step1 → server

←return attributes

Step3: Create connect by some protocol by the information got in step2, and then use this protocol transmit data.

4.7.2 Service attribute example

The server stored it’s services as a database, like following :



# 5 PC example C code with libbluetooth

apt-get install libbluetooth1-dev bluez-utils

## 5.1 Rfcomm connection

### 5.1.1 Rfcomm server

```
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <bluetooth/bluetooth.h>
#include <bluetooth/rfcomm.h>
void baswap(bdaddr_t *dst, const bdaddr_t *src)
{
    register unsigned char *d = (unsigned char *) dst;
    register const unsigned char *s = (const unsigned char *) src;
    register int i;
    for (i = 0; i < 6; i++)
        d[i] = s[5-i];
}
int str2ba(const char *str, bdaddr_t *ba)
{
    uint8_t b[6];
    const char *ptr = str;
    int i;
    for (i = 0; i < 6; i++) {
       b[i] = (uint8_t) strtol(ptr, NULL, 16);
       if (i != 5 && !(ptr = strchr(ptr, ':')))
       ptr = ":00:00:00:00:00";
       ptr++;
     }
     baswap(ba, (bdaddr_t *) b);
     return 0;
}
int ba2str(const bdaddr_t *ba, char *str)
{
    uint8_t b[6];
    baswap((bdaddr_t *) b, ba);
    return sprintf(str, "%2.2X:%2.2X:%2.2X:%2.2X:%2.2X:%2.2X",
                  b[0], b[1], b[2], b[3], b[4], b[5]);
}
int main(int argc, char **argv)
{
    struct sockaddr_rc loc_addr = { 0 }, rem_addr = { 0 };
```

```
        char buf[1024] = { 0 };
        int s, client, bytes_read;
        int opt = sizeof(rem_addr);
        // allocate socket
        s = socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);
        // bind socket to port 1 of the first available
        // local bluetooth adapter
        loc_addr.rc_family = AF_BLUETOOTH;
        loc_addr.rc_bdaddr = *BDADDR_ANY;
        loc_addr.rc_channel = (uint8_t) 1;
        bind(s, (struct sockaddr *)&loc_addr, sizeof(loc_addr));
        // put socket into listening mode
        listen(s, 1);
        // accept one connection
        client = accept(s, (struct sockaddr *)&rem_addr, &opt);
        ba2str( &rem_addr.rc_bdaddr, buf );
        fprintf(stderr, "accepted connection from %s\n", buf);
        memset(buf, 0, sizeof(buf));
        // read data from the client
        bytes_read = read(client, buf, sizeof(buf));
        if( bytes_read > 0 ) {
                printf("received [%s]\n", buf);
        }
        // close connection
        close(client);
        close(s);
        return 0;
}
```

### 5.1.2 Rfcomm client

```
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <bluetooth/bluetooth.h>
#include <bluetooth/rfcomm.h>
int main(int argc, char **argv)
{
        struct sockaddr_rc addr = { 0 };
        int s, status;
        char dest[18] = "01:23:45:67:89:AB";
        // allocate a socket
        s = socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);
        // set the connection parameters (who to connect to)
        addr.rc_family = AF_BLUETOOTH;
```

```
    addr.rc_channel = (uint8_t) 1;
    str2ba( dest, &addr.rc_bdaddr );
    // connect to server
    status = connect(s, (struct sockaddr *)&addr, sizeof(addr));
    // send a message
    if( status == 0 ) {
        status = write(s, "hello!", 6);
    }
    if( status < 0 ) perror("uh oh");
    close(s);
    return 0;
}
```

## 5.2 Other L2CAP sockets and SDP ... refer to “BTBook.pdf”

# 6 Linux command introduce

If need input link-key we should download some software apt-get install bluez…libbluetooth… , I forgot which packets needed to install.

1. Hcitools scan/inq/
2. hcidump –x –R
3. rfcomm  
   band 0 xx:xx:xx:xx:xx:xx  
   connect 0 xx:xx:xx:xx:xx:xx 1  
   release 0 [0→/dev/rfcomm0]
4. sdptool service  
   sdptool browse xx:xx:xx:xx:xx:xx

# 7 Terms and Abbreviations

LM --- Link-level and medium access management

LC --- Packet-level access control

FEC --- Forward error correcting

LAP --- BD address [47:32](NAP[15:0]), used to initialize the encryption engine stream LFSR.
BD address[31:24](UAP[7:0]), used to initialize the HEC and CRC for freq hopping.
BD address[23:0](LAP[23:0]), used by sync word generation and freq hopping.

FHS --- frequency hop synchronization

OCF --- OpCode Command Field

OGF --- OpCode Group Field

# 8 Change version

| Version | Data | Description | Author |
|---|---|---|---|
| 1.0 | 3 November 2009 | Original version | YinXiaoTao |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 9 Reference

http://www.bluetooth.com/Bluetooth/Technology/Works/

http://www.bluetooth.com/Bluetooth/Technology/Building/Specifications/

Axis openbt source code

http://hccc.ee.ccu.edu.tw/courses/bt/

WOSP project (porting openbt to ecos)

Bluez tools and source code

http://www.bluetooth.com/Bluetooth/Technology/Building/Specifications/

Bluetooth core spec [message sequence chart v2.1+edr P851]