ABSTRACT

Bluetooth is a new wireless technology standard developed by a consortium of telecommunication and software companies aimed at standardizing short range wireless communication the world over. This project aims to provide a flexible protocol stack, implementing the rules of communication and data transfer in the standard, which can be easily integrated into any Bluetooth enabled device that needs to make use of the unique features and services offered by Bluetooth.

This project deals with the design of the protocol layers that comprise the Bluetooth communication stack. Our project focuses on the design and testing of Bluetooth specific layers, namely the L2CAP, Baseband, Service Discovery Protocol layers as well as the RFCOMM layer used for serial port emulation. A basic description of the functionality of each layer and their role in Bluetooth based data transmission is as follows:-

Bluetooth Baseband:

The Baseband is the physical layer of the Bluetooth, managing physical channels and links apart from other services like error correction, data whitening, hop selection and Bluetooth security. It lies on top of the Bluetooth radio layer in the bluetooth stack. The baseband protocol is implemented as a Link Controller , which works with the link manager for carrying out link level routines like link connection and power control. The baseband also manages asynchronous and synchronous links, handles packets and does paging and inquiry to access and inquire Bluetooth devices in the area.

Logical Link Control and Adaptation Protocol (L2CAP):

L2CAP packets carry payloads which are carried to the upper layer protocols. L2CAP is layered over the Baseband Protocol and resides in the data link layer.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. The functional requirements for L2CAP include protocol multiplexing, segmentation and reassembly (SAR), and group management. L2CAP lies above the Baseband Protocol and interfaces with other communication protocols such as the Bluetooth Service Discovery Protocol (SDP), RFCOMM , and Telephony.

Service Discovery Protocol (SDP):

Using SDP, device information, services allowed and characteristics of the services are queried between Bluetooth enabled devices. The service discovery protocol (SDP) provides a means for applications to discover which services are available and to determine the characteristics of those available services.

RFCOMM

RFCOMM is a simple transport protocol, which provides emulation of RS232 serial ports over the L2CAP protocol.The protocol is based on the ETSI standard TS 07.10. Only a subset of the TS 07.10 standard is used and an RFCOMM - specific extension is added.

INTRODUCTION

Bluetooth is the code name for an alliance between mobile communications and mobile computing companies to develop a short-range communications standard allowing wireless data communications at ranges of about 10 meters to 100 meters. The technology allows users to make effortless, instant connections between a wide range of communication devices. Conceived by Ericsson, the Bluetooth technology is the result of the joint achievements of nine leading companies including Motorola, Nokia, Ericsson, IBM, Intel, Toshiba, 3Com, Lucent and Microsoft. Bluetooth will facilitate wireless Local Area Networking in which networks of different hand held mobile computing terminals can communicate and exchange data, even on the move when there is no line-of-sight between those devices.

The standard was developed by a group of electronics manufacturers that allows any sort of electronic equipment -- from computers and cell phones to keyboards and headphones -- to make its own connections, without wires, cables or any direct action from a user. The companies belonging to the Bluetooth Special Interest Group (SIG), and there are more than 1,000 of them, want to let Bluetooth's radio communications take the place of wires for connecting peripherals, telephones and computers. The hardware vendors, which include Siemens, Intel, Toshiba, Motorola and Ericsson, have developed a specification for a very small radio module to be built into computer, telephone and entertainment equipment.

It is mainly a cable-replacement technology. The standard has been developed for a small, cheap radio chip to be plugged into computers, printers, mobile phones, etc. A Bluetooth chip is designed to replace cables by taking the information normally carried by the cable, and transmitting it at a special frequency to a receiver Bluetooth chip, which will then give the information received to the computer, phone. It works by using short-range radio links, intended to replace the cable(s) connecting portable and/or fixed electronic devices. It is envisaged that it will allow for the replacement of the many propriety cables that connect one device to another with one universal radio link. Its key features are robustness, low complexity, low power and low cost. Designed to operate in noisy frequency environments, the Bluetooth radio uses a fast acknowledgement and frequency hopping scheme to make the link robust. Bluetooth radio modules operate in the unlicensed ISM band at 2.4GHz which has been set aside by international agreement for the use of industrial, scientific and medical devices (ISM, and avoid interference from other signals by hopping to a new frequency after transmitting or receiving a packet. Compared with other systems in the same frequency band, the Bluetooth radio hops faster and uses shorter packets.

The wireless technology inherent in Bluetooth revolutionizes the personal connectivity market by providing freedom from wired connections enabling links between mobile computers, mobile phones, portable handheld devices, and connectivity to the Internet. Interfacing, synchronization, exchange of data, you name it and Bluetooth has the ability to implement it.

Hardware that complies with the Bluetooth wireless specification ensures communication compatibility worldwide. As a low cost, low power solution with industry wide support, Bluetooth wireless technology allows the user to bring connectivity with himself. In fact given the market coverage provided by the members of the Bluetooth signatorium, a user owning a Bluetooth enabled device would be able to have connectivity almost anywhere in the world. The secret of this almost universal connectivity is the establishment of Bluetooth as an industry standard. This means integrating well tested technology with the power efficiency and low-cost of a compliant radio system. Furthermore it also requires having a group of industry leading promoter companies who drive the specification forward (members of the Bluetooth SIG). Bluetooth technology works because it has been developed as a cross industry solution that marries a vision of engineering innovation with an understanding of business and consumer expectations. Its continued existence and further development can be explained by the fact that Bluetooth wireless technology is supported by product and application development in a wide range of market segments, including software developers, silicon vendors, peripheral and camera manufacturers, mobile PC manufacturers and handheld device developers, consumer electronics manufacturers, car manufacturers, and test and measurement equipment manufacturers.

Because Bluetooth wireless technology is an open platform, all members of the Bluetooth SIG have permission to use Bluetooth wireless technology in their products and services. There are three levels of membership with unique benefits: Promoter, Associate and Adopter.

The SIG was founded in February 1998, and initially consisted of the five companies Ericsson, Intel, Toshiba, Nokia & IBM. Today more than 1800 companies have joined the SIG to work for an open standard for the Bluetooth concept.

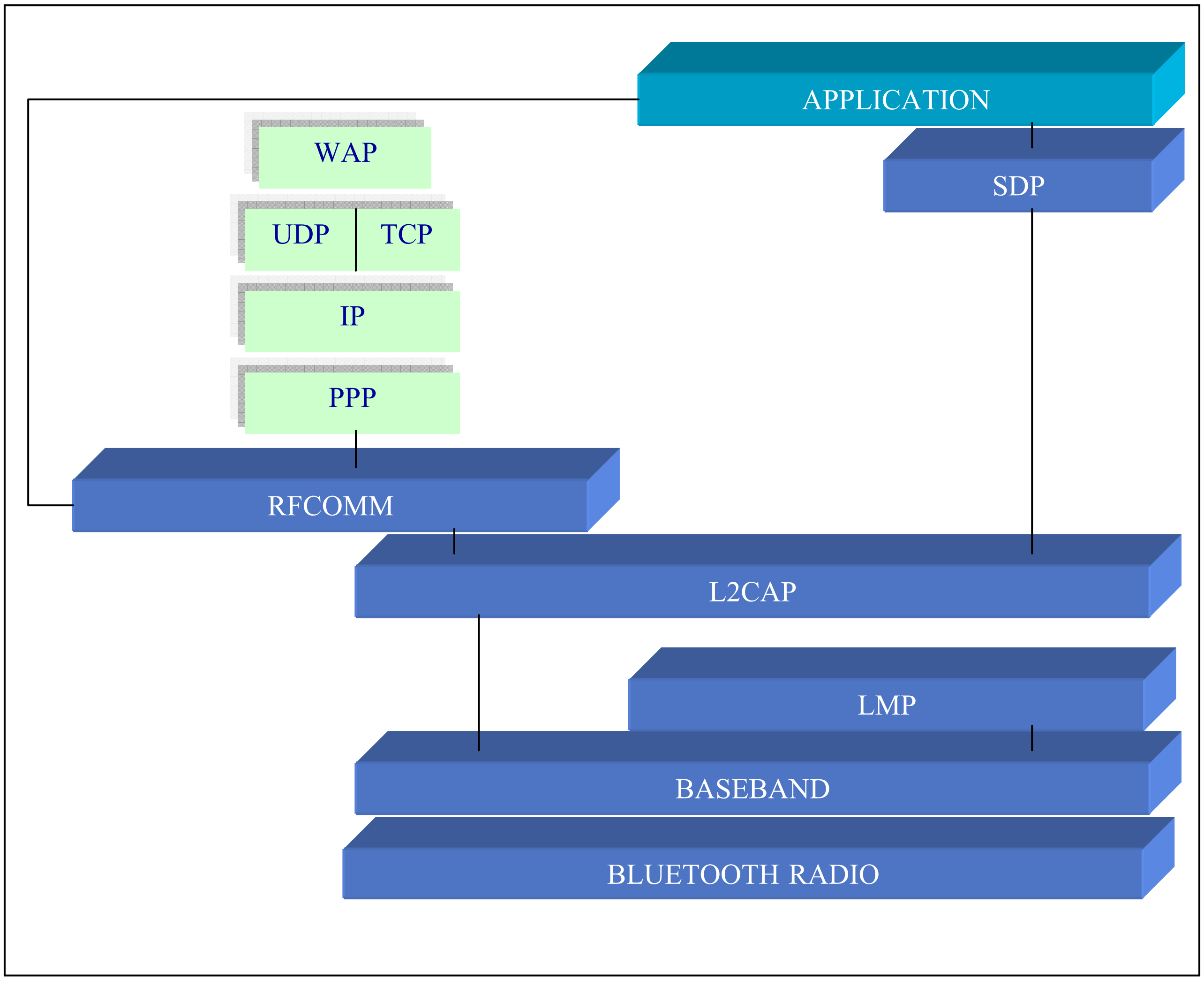
Today the SIG consists of 9 promoter members Motorola, Lucent, Toshiba, Lucent, Microsoft, 3Com, IBM, Intel, Nokia & Ericsson, and 1790 Adopter/Associate member companies. By signing a zero cost agreement, companies can join the SIG and qualify for a royalty-free license to build products based on the Bluetooth technology. To avoid different interpretations of the Bluetooth standard regarding how a specific type of application should be mapped to Bluetooth, the SIG has defined a number of user models and protocol profiles.

The name Bluetooth itself comes from a Danish Viking and King, Harald Blåtand (translated as Bluetooth in English), who lived in the latter part of the 10th century. Harald Blåtand united and controlled Denmark and Norway (hence the inspiration on the name: uniting devices through Bluetooth). He got his name from his very dark hair which was unusual for Vikings, Blåtand means dark complexion. However a more popular, (but less likely reason), was that Old Harald had a inclination towards eating Blueberries , so much so his teeth became stained with the colour, leaving Harald with a rather unique set of molars.

The complete Bluetooth stack is formed by the following seven constituent protocol layers:-

1. Baseband
2. Link Manager Protocol
3. Logical Link Control and Adaptation Protocol
4. Service Discovery Protocol
5. RFCOMM Protocol
6. Telephony Control Protocol
7. Adopted Protocols

The complete Bluetooth protocol stack has been designed to include the existing protocols as much as possible (like TCP, UDP, OBEX) as well as Bluetooth specific protocols like LMP and L2CAP. The protocol reuse ensures smooth interoperability between existing applications and hardware. The Link Manager Protocol is optional and can be bypassed, allowing direct communication between the Baseband and L2CAP layers. The upper portion of the stack, above the RFCOMM layer consists of some of the existing protocols specified above.


APPLICATION
WAP
SDP
UDP
TCP
IP
PPP
RFCOMM
L2CAP
LMP
BASEBAND
BLUETOOTH RADIO

CHAPTER 1

BLUETOOTH BASEBAND PROTOCOL

BASEBAND PROTOCOL

1.1 GENERAL DESCRIPTION

As specified before 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 Ms/s. A slotted channel is applied with a nominal slot length of 625 μs. For 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. The asynchronous channel has been implemented which supports maximum 723.2 kb/s asymmetric (and still up to 57.6 kb/s in the return direction), or 433.9 kb/s symmetric.

The baseband protocol is implemented as within a link controller which can also carry out other low level routines. These low level routines are used to interact with the radio layer which in the simulation is represented by the Local Network over which the communication is taking place.

The system provides a point-to-point connection (only two Bluetooth units involved), or a point-to-multipoint connection, see Figure 1.1. In the point-to-multipoint connection, the channel is shared among several Bluetooth units. Two or more units sharing the same channel form a ***picone***t. One Bluetooth unit acts as the master of the piconet, whereas the other unit(s) act 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 ***scatterne*****t**. 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.

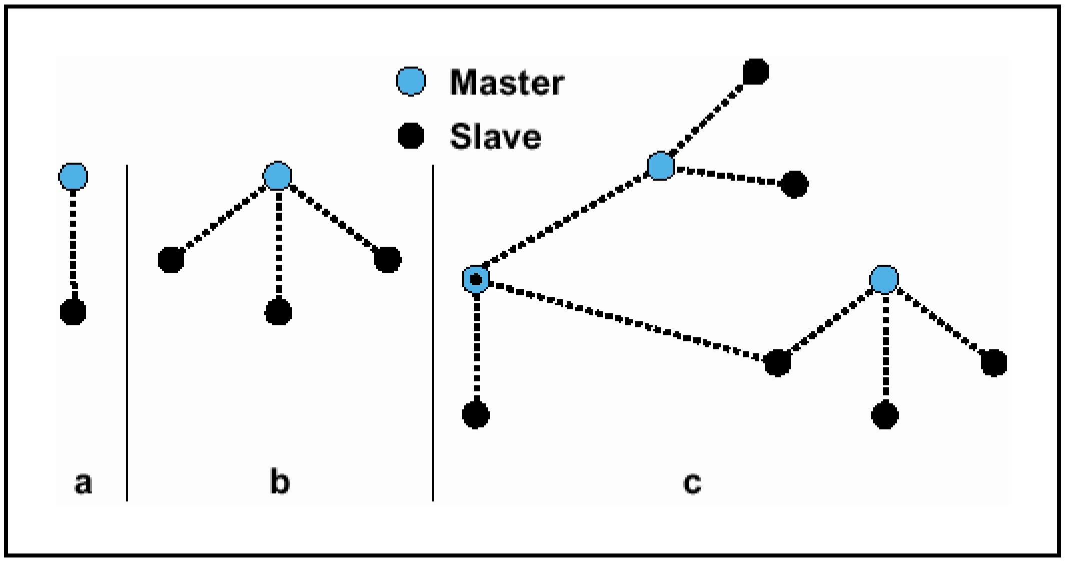


Figure 1.1 Piconets with a single slave operation (a), a multi-slave operation (b) and a scatternet operation (c).

1.2 PHYSICAL CHANNEL

1.2.1 DEFINITION

The channel is represented by a pseudo-random hopping sequence hopping through the 79 or 23 RF channels. The sequence is unique for the piconet and is determined by the Bluetooth device address of the master; the phase in the 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.

1.2.2 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} . In the time slots, master and slave can transmit packets.

A TDD scheme is used where master and slave alternatively transmit. See Figure 1.2 . The master starts its transmission in even-numbered time slots only, and the slave starts its transmission in odd-numbered time slots only. The packet start is 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 remains 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 multi-slot 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 multi-slot packet shall use the frequency as determined by the current Bluetooth clock value. Figure 1.3 illustrates the hop definition on single- and multi-slot 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.

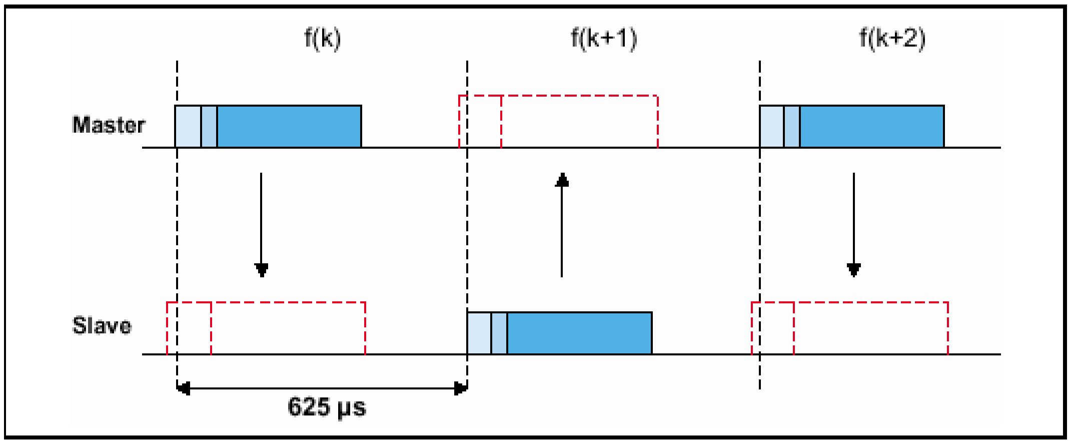


Figure 1.2 TDD and Timing

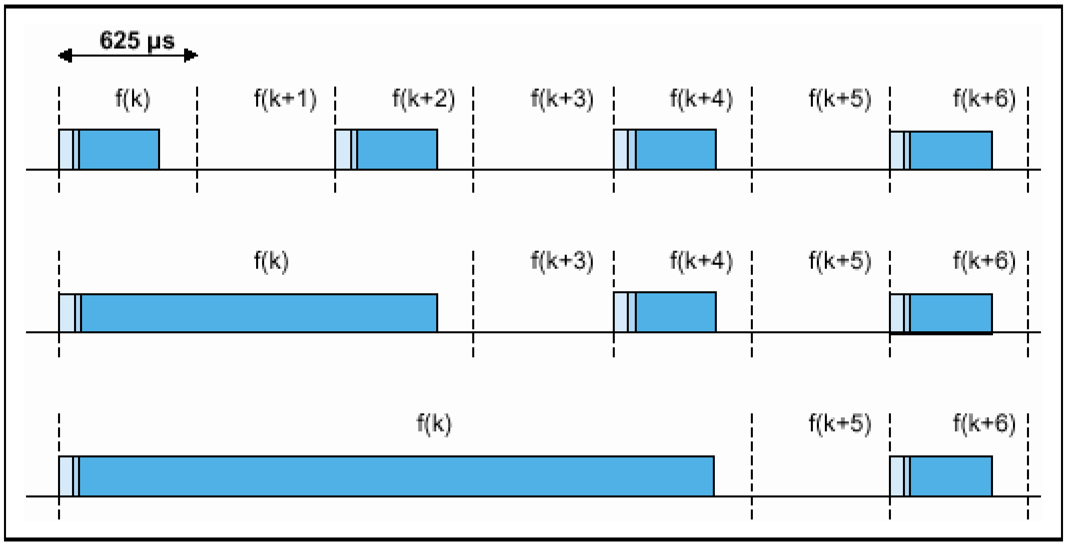


Figure 1.3 Multi-slot Packets

1.3 PHYSICAL LINKS

1.3.1 GENERAL

Between master and slave(s), the Asynchronous Connection-Less (ACL) link is established The ACL link is a point-to-multipoint link between the master and all the slaves participating on the piconet. The master can establish an ACL link on a per-slot basis to any slave, in another link.

1.3.2 ACO LINKS

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.

1.4 PACKETS

1.4.1 GENERAL DESCRIPTION

The bit ordering when defining packets and messages in the *Baseband Layer*, follows the *Little Endian forma*t, 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 pay-load header length, are transmitted with the LSB first. For instance, a 3-bit parameter X=3 is sent as over the air $b_0b_1b_2$

where 0 is sent first and 2 is sent last. The data on the piconet channel is conveyed in packets. The general packet format is shown in Figure 1.4. Each packet consists of 3 entities: the access code, the header, and the payload. In the figure, the number of bits per entity is indicated.

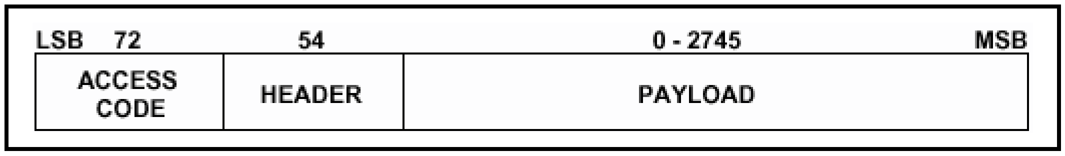


Figure 1.4 Standard Packet Format

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 of the access code˜ header, or of the access code˜ header˜ payload.

1.4.2 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.

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, as shown in Figure 1.5.

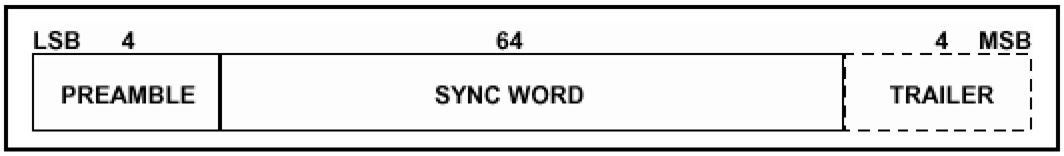


Figure 1.5 Access Code Format

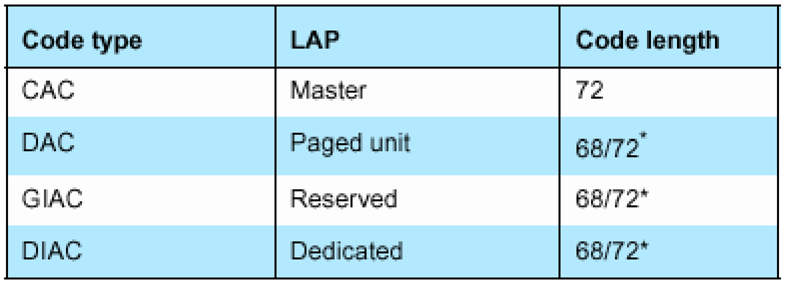
1.4.2.1 ACCESS CODE TYPES

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 **preambl**e, **sync wor**d, 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. A summary of the different access code types can be found In Table 1.1.



Code type	LAP	Code length
CAC	Master	72
DAC	Paged unit	68/72*
GIAC	Reserved	68/72*
DIAC	Dedicated	68/72*

Table 1.1 Summary of Access Code Types

1.4.2.2. PREAMBLE

The preamble is a fixed zero-one pattern of 4 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 1.6.

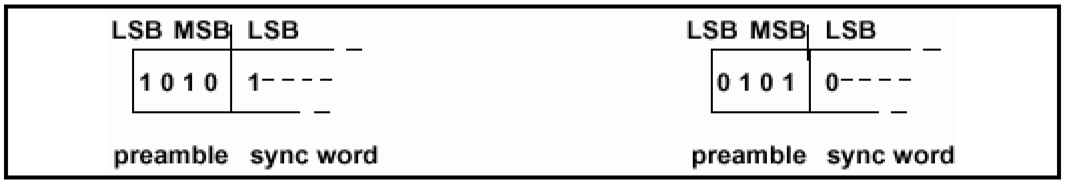


Figure 1.6 Preamble

1.4.2.3 SYNCH 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 good auto correlation properties of the sync word improve on the timing synchronization process.

1.4.2.4 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 sync word 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 1.7.

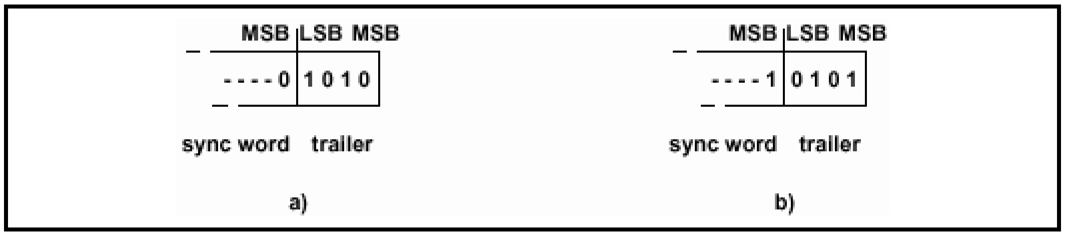


Figure 1.7: Trailer in CAC when MSB of sync word is 0 (a), and when MSB of sync word is 1 (b).

1.4.3 <u>PACKET HEADER</u>

The header contains link control (LC) information and consists of 5 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

The total header consists of 10 bits, see Figure 1.8 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 next.

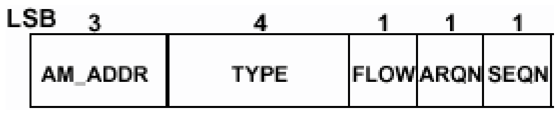


Figure 1.8: Header format

1.4.3.1 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. 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.

1.4.3.2 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 ACL link. Then it can be determined which type of 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. Each packet type is described in more detail further on.

1.4.3.3 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. Packets including only link control information (ID, POLL and NULL 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 NAK in response to this packet even if the packet passed the CRC check.

1.4.3.4 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. An unnumbered ARQ scheme which means that the ARQN relates to the latest received packet from the same source, is used.

1.4.3.5 SEQN

The SEQN bit provides a sequential numbering scheme to order the data packet stream. For each new transmitted packet 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. For broadcast packets, a modified sequencing method is used.

1.4.4 PACKET TYPES

The packets used on the piconet are related to the physical links they are used in. For each link, 12 different packet types can be defined. Four control packets are 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 1.2 summarizes the packets defined so far for the ACL link types.

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

Table 1.2: Packets defined for ACL link type

1.4.4.1 PACKET TYPES

There are five packets. In addition to the types listed in segment 1 of the previous table, there is the ID packet not listed. Each packet will now be examined in more detail.

1.4.4.1.1 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.

1.4.4.1.2 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.

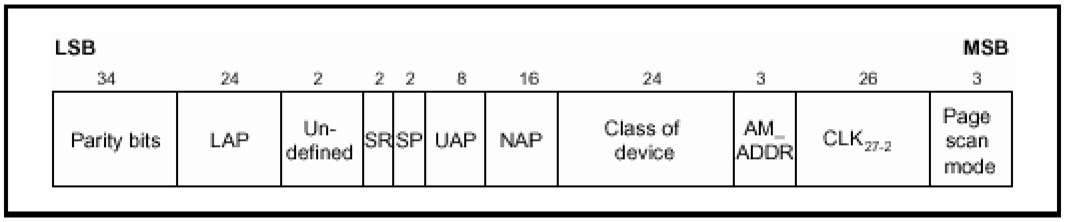
1.4.4.1.3 POLL packet

The POLL packet is very similar to the NULL packet. It does not have a pay-load 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.

1.4.4.1.4 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. The FHS packet covers a single time slot. Figure 1.9 illustrates the format and contents of the FHS payload. The payload consists of eleven fields. 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. This clock information is updated before each retransmission.

The retransmission of the FHS payload is thus somewhat different from the retransmission of ordinary data payloads where the same payload is used for each retransmission. 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.



LSB										MSB
34	24	2	2	2	8	16	24	3	26	3
Parity bits	LAP	Un-defined	SR	SP	UAP	NAP	Class of device	AM_ADDR	CLK_{27-2}	Page scan mode

Figure 1.9: Format 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.

LAP: This 24-bit field contains the lower address part of the unit that sends the FHS packet.

SR: This 2-bit field is the scan repetition field and indicates the interval between two consecutive page scan windows.

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.

UAP: This 8-bit field contains the upper address part of the unit that sends the FHS packet.

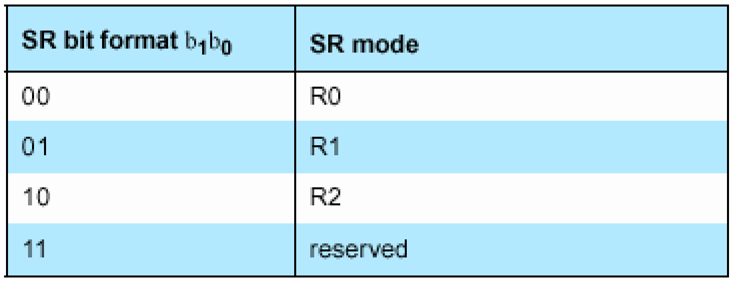
NAP: This 16-bit field contains the non-significant address part of the unit that sends the FHS packet (see also section on page for LAP, UAP, and NAP).

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 Bluetooth Assigned Numbers.

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.

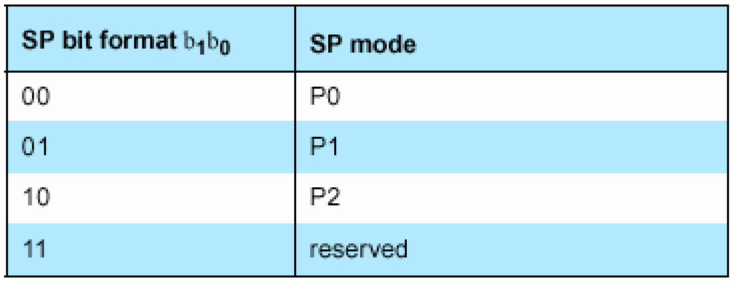
CLK 27-2: This 26-bit field contains the value of the native system clock of the unit that sends the FHS packet, sampled at the beginning of the transmission of the access code of this FHS packet. This clock value has a resolution of 1.25ms (two-slot interval). For each new transmission, this field is 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 1.5. Currently, the standard supports one mandatory scan mode and up to three optional scan modes.



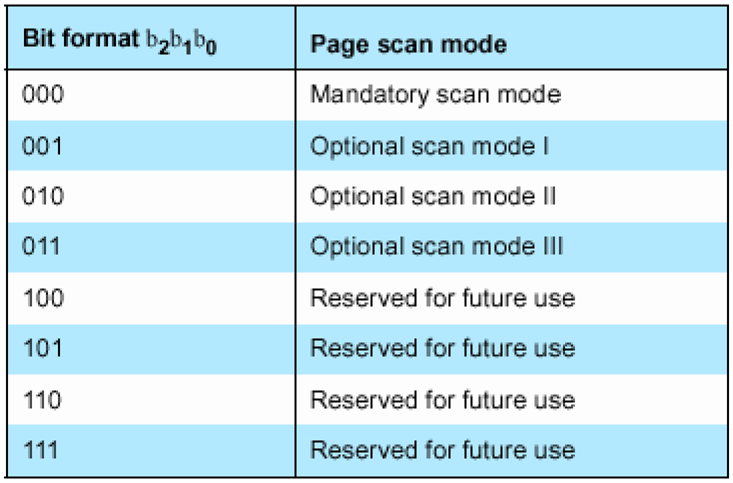
SR bit format b_1b_0	SR mode
00	R0
01	R1
10	R2
11	reserved

Table 1.3: Contents of SR field



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

Table 1.4: Contents of SP 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

Table 1.5: Contents of page scan mode field

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.

1.4.4.1.5 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 payload header in the DM1 packet is only 1 byte long. The length indicator in the payload header specifies the number of user bytes.

1.4.5 <u>PAYLOAD FORMAT</u>

The ACL packets only have a data field in their payload.

1.4.5.1 DATA FIELD

The data field consists of three segments: a payload header, a payload body, and possibly a CRC code.

1. Payload header

Only data fields have a payload header. The payload header is one or two 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-byte 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 the figures given below.

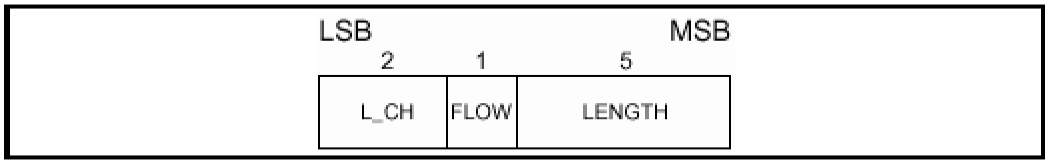


Figure 1.10: Payload header format for single-slot packets.

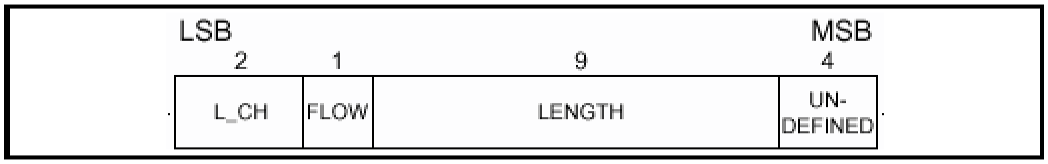


Figure 1.11: Payload header format for multi-slot packets.

The L_CH field is transmitted first, the length field last. In the table given below more details about the contents of the L_CH field are listed.

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

Table 1.6: Logical channel L_CH field contents

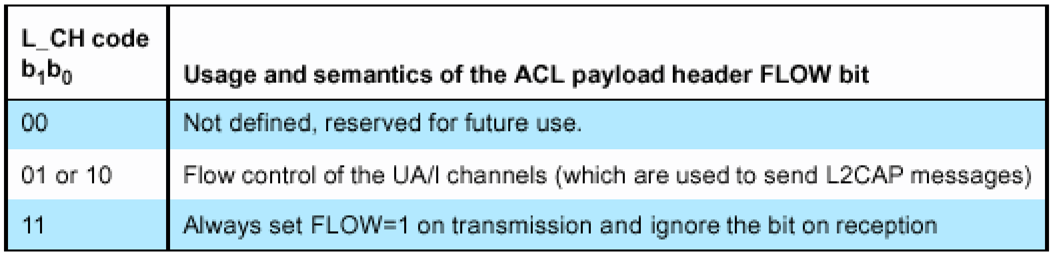
An L2CAP message can be fragmented into several packets. Code 10 is used for an L2CAP packet carrying the first fragment of such a message; code 01 is used for continuing fragments. If there is no fragmentation, code 10 is used for every packet. Code 11 is used for LMP messages. Code 00 is reserved for 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 must dedicate to this function. If the packets 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. 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. 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 pay-load flow bit has its own meaning for each logical channel (UA/I or LM. On the LM channel, no flow control is applied and the payload flow bit is always set at one.



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

Table 1.7: Use of payload header flow bit on the logical channels.

The length indicator indicates the number of bytes (i.e. 8-bit words) in the payload excluding the payload header; i.e. the payload body only. 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.

1.4.6 PACKET SUMMARY

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 1.8: Link control 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

Table 1.9: Data packet

1.5 <u>LOGICAL CHANNELS</u>

In the designed system, three logical channels are defined:

- LC control channel
- UA user channel
- UI user channel

The control channel LC is used at the link control level. The user channels UA, UI are used to carry asynchronous and isochronous respectively. The LC channel is carried in the packet header; all other channels are carried in the packet payload. The UA, and UI channels are indicated in the L_CH field in the payload header. The UA and UI channels are carried by the ACL link.

1.5.1 LC CHANNEL (LINK CONTROL)

The LC control 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.

1.5.2 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.

1.5.3 CHANNEL MAPPING

The LC channel is mapped onto the packet header. All other channels are mapped onto the payload. All channels are mapped on the ACL packets.

1.6 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 always starts 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 chapter 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 does not drift faster than 20 ppm relative to the ideal slot timing of 625 μs while the instantaneous timing does not deviate more than 1 μs from the average timing as per specifications.

1.6.1 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: it keeps an exact interval of Mx625 μs (where M is an even, positive integer larger than 0) between consecutive transmissions. 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. 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 is 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. The slave shall be able to receive the packets and adjust the RX timing as long as the timing mismatch remains within the 312μs uncertainty window. The master TX timing is strictly related to the master clock. The master shall keep an exact interval of Mx1250 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 Nx625 μs (where N is an odd, positive integer larger than 0). During the master RX cycle, the master will also use the 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 the next sections.

1.6.2 CONNECTION STATE

In the connection mode, the Bluetooth transceiver transmits and receives alternately. 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 multi-slot packets, several slots are covered by the same packet, and the hop frequency used in the first slot will be used throughout the transmission.

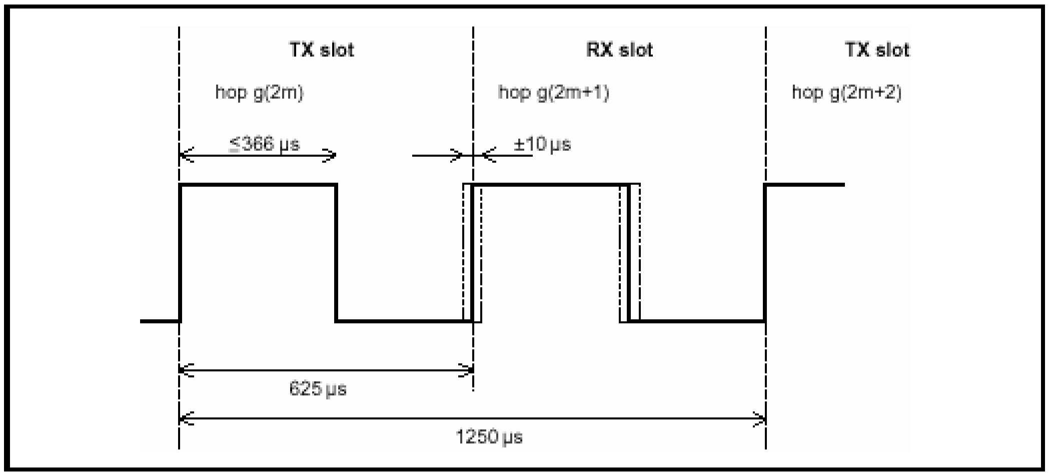


Figure 1.12: RX/TX cycle of Bluetooth master transceiver in normal mode for single-slot packets.

The master TX/RX timing is shown in the Figure In the figures shown here 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 Nx625 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 Mx125̃s 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 Nx62□s 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 takes the RX timing of the latest RX burst as reference.

1.6.3 RETURN FROM HOLD MODE

In the connection state, the Bluetooth unit can be placed in a **hold** mode. 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 caşe, the length of the search window in the slave unit may be increased from □̣̃ µs to a larger value X µs. 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 centered 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. 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.

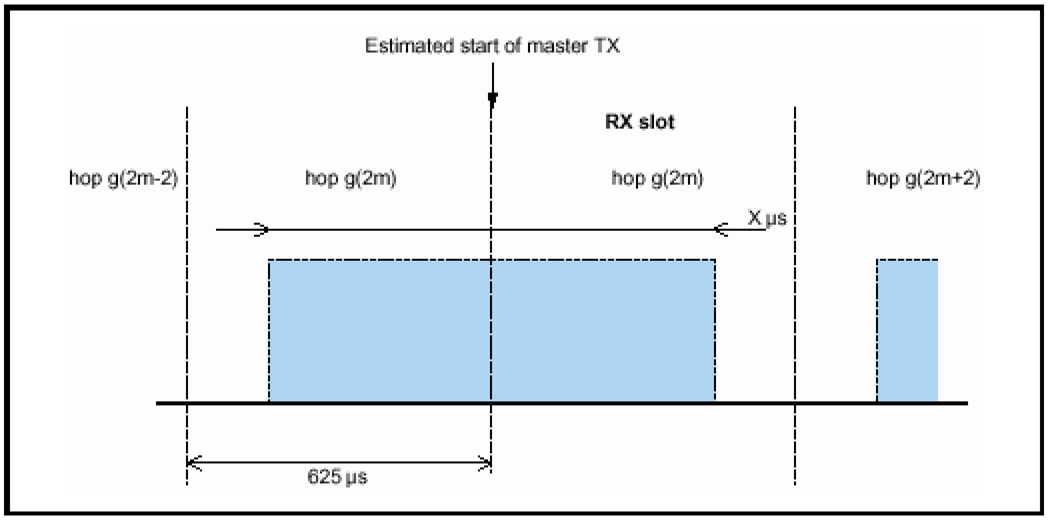


Figure 1.13: RX timing of slave returning from hold state.

1.6.4 PARK AND SNIFF MODES WAKE-UP

The **park** and **snif**f modes is 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 312µs to a larger value X µs as illustrated in Figure 1.13.

1.6.5 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 can be 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 1.14. 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 [illegible]µs uncertainty window.

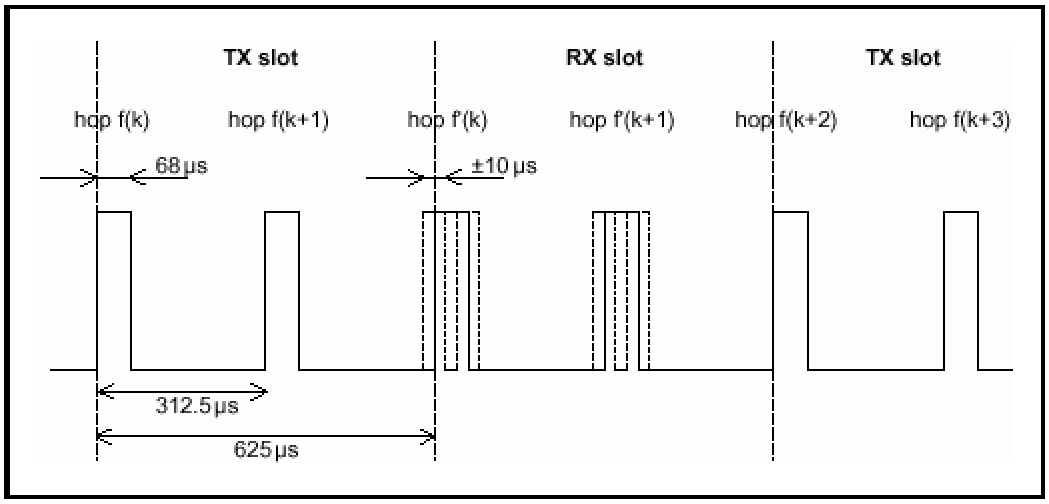


Figure 1.14: RX/TX cycle of Bluetooth transceiver in PAGE mode.

1.6.6 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. 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 the **FHS** message will depend on the timing of the page message the slave received. In figure 1.15, 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 1.16, the slave receives the paging message sent **secondly** in the 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.

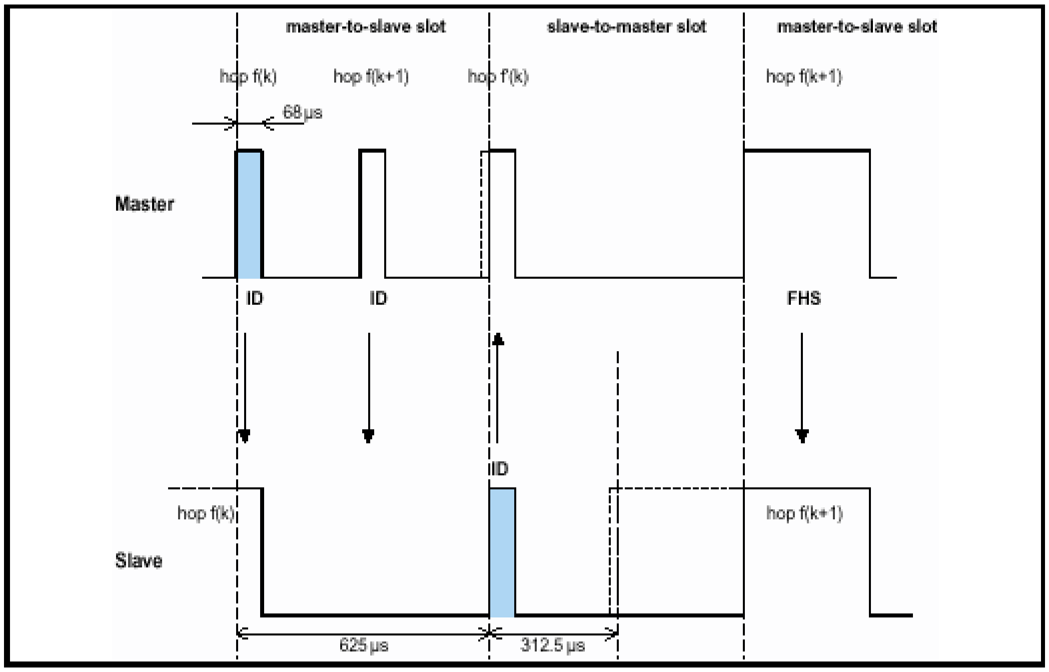


Figure 1.15: Timing of FHS packet on successful page in first half slot.

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). That is, the second response message that acknowledges the reception of the FHS packet is transmitted 625 µs after the start of the **FHS** packet.

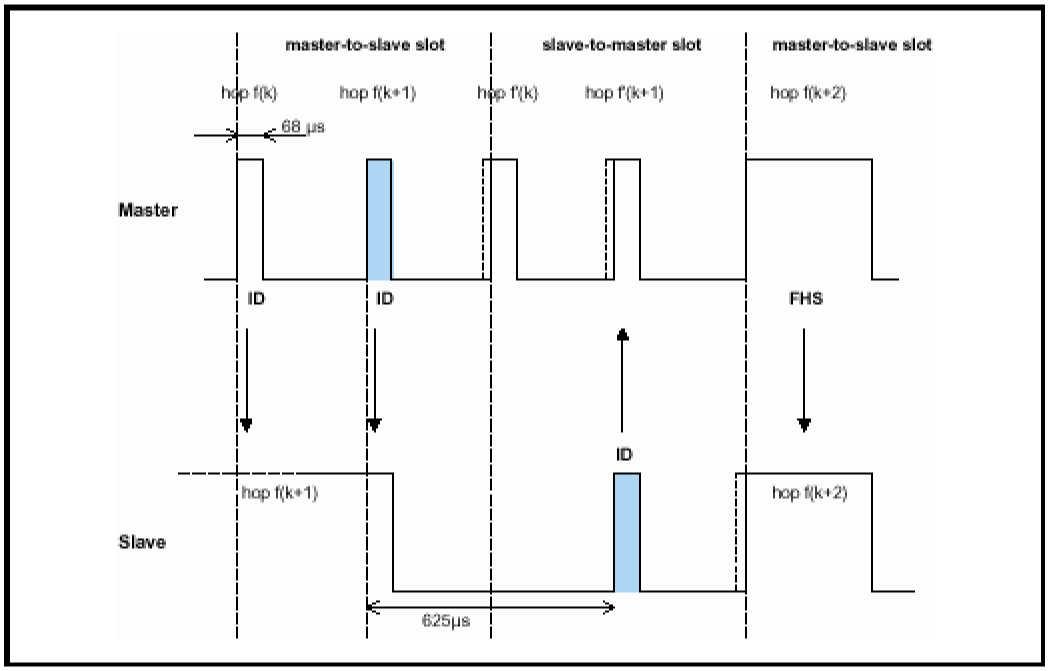


Figure 1.16: Timing of FHS packet on successful page in second half slot.

1.6.7 MULTI-SLAVE OPERATION

As was mentioned in the beginning of this chapter, 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 1.17. 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 found in the access window for access requests in the park mode.

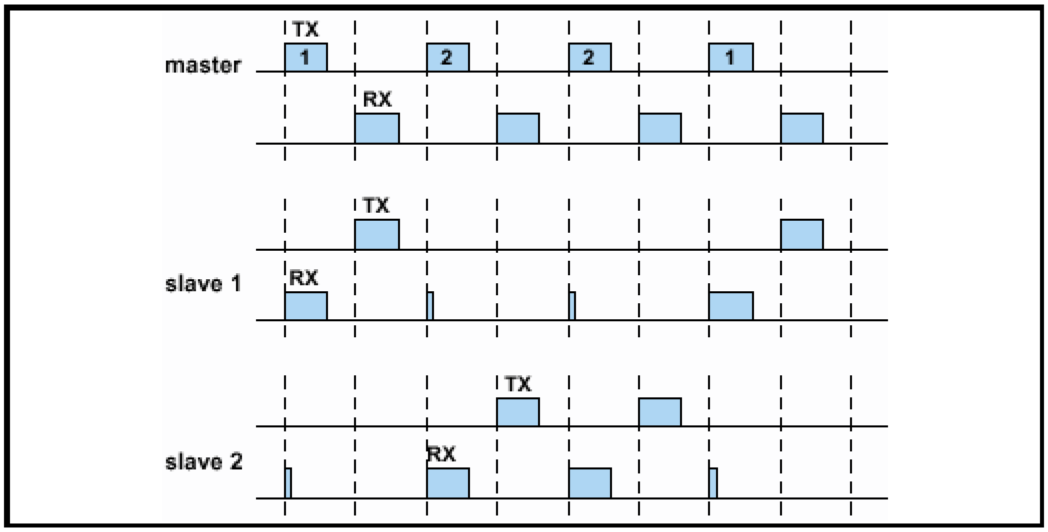


Figure 1.17: RX/TX timing in multi-slave configuration

1.7. CHANNEL CONTROL

1.7.1 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 scatter-net, is discussed. A special section is attributed to the Bluetooth clock which plays a major role in the FH synchronization.

1.7.2 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.

1.7.3 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 are updated regularly for accuracy. The clock determines critical periods and triggers the events in the Bluetooth receiver. Four periods are important in the Bluetooth system: 312.5 □□s, 625 □□s, 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 1.18. Master-to-slave transmission starts at the even-numbered slots when CLK 0 and CLK 1 are both zero.

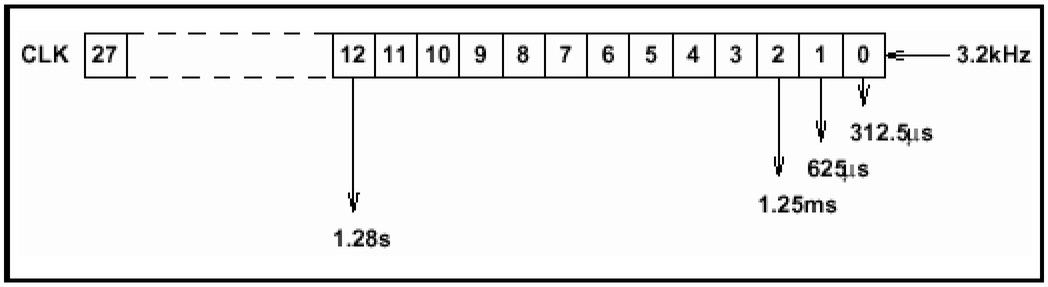


Figure 1.18: Bluetooth clock.

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 crystal oscillator with worst case accuracy of +/-20ppm. In the low power states, like **STANDBY, HOLD, PARK** and **SNIF**F, the native clock may be driven by a low power oscillator (LPO) with relaxed accuracy (+/-250ppm). 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 1.19. 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 1.20. 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.

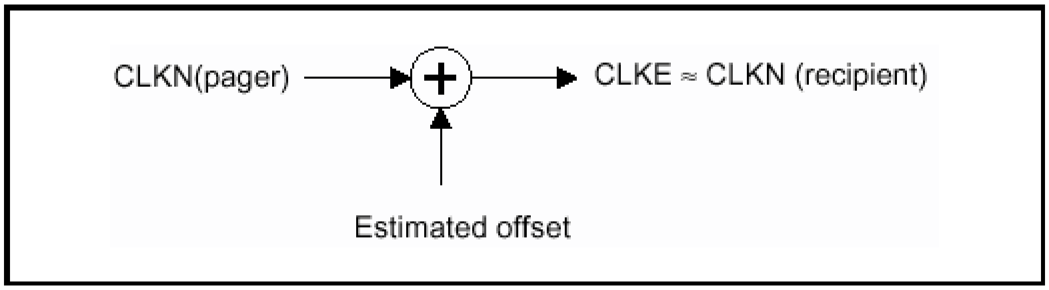


Figure 1.19: Derivation of CLKE

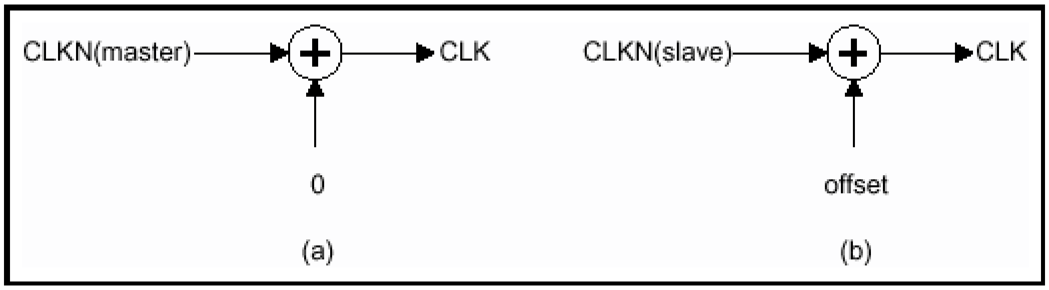


Figure 1.20: Derivation of CLK in master (a) and in slave (b).

1.7.4 OVERVIEW OF STATES

Figure 1.21 shows a state diagram illustrating the different states used in the Bluetooth link controller. There are two major states: **STANDBY** and **CONNECTIO**N; in addition, there are seven substates, **pag**e, **page sca**n, **inquir**y, **inquiry sca**n, **master respons**e, **slave respons**e, and **inquiry respons**e. 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 Blue-tooth 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).

1.7.5 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.

1.7.6 ACCESS PROCEDURES

1.7.6.1 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. In the current chapter, only the mandatory paging scheme is considered.

1.7.6.2 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 . 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. 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 16-12 of the unit's native clock (CLKN 15-12 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 sca**n, the unit will enter the **slave response** substate. 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 sca**n. 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 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.

The scan interval information is included in the SR field in the FHS packet.

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

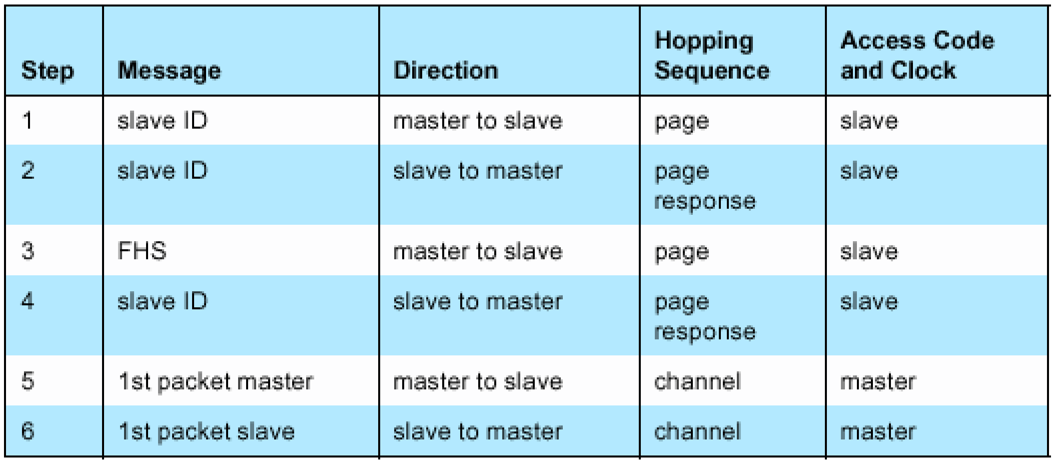
Table1.10: Relationship between scan interval, train repetition, and paging modes R0, R1 and R2.

1.7.6.3 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. 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 **page** substate can be entered from the **STANDBY** state or the **CONNEC-TION** 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.

1.7.6.4 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 1.10and in Figure 1.21 and Figure 1.22. 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.



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

Table 1.11: Initial messaging during start-up.

In step 1 the master unit is in **page** substate and the slave unit in the **page scan** substate. Assume in this step that the page message (slave's device access code) sent by the master reaches 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 mes-sage 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.

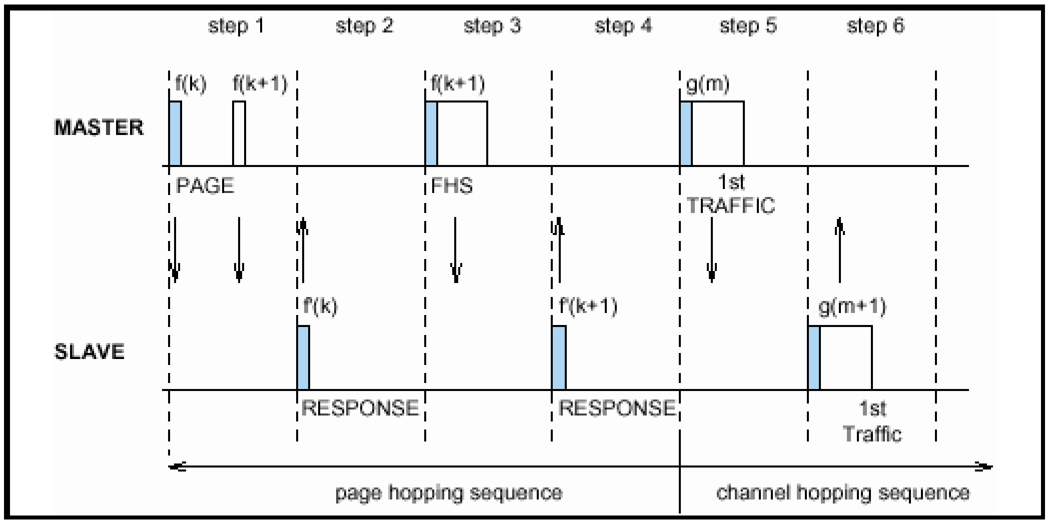


Figure 1.21: Messaging at initial connection when slave responds to first page message.

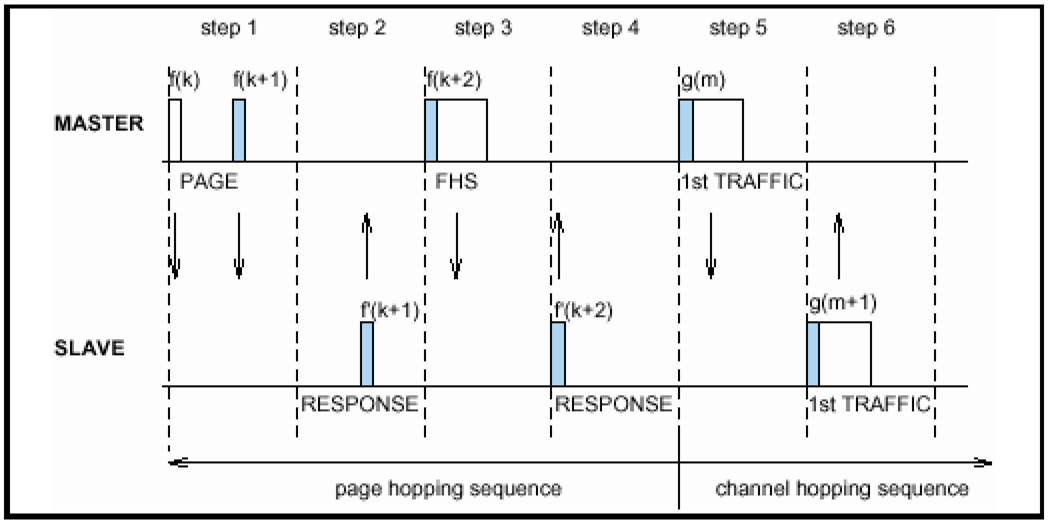


Figure 1.22: Messaging at initial connection when slave responds to second page message.

1.7.6.4.1 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 microseconds 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 16-12 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 microseconds after the start of the response message) and awaits the arrival of a **FHS** packet. Note that a **FHS** packet can already arrive 312.5 micros after the arrival of the page message as shown in Figure 1.23, and not after 625 micros as is usually the case in the RX/TX timing. 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 *pagerespT*O, the slave returns back to the **page scan** substate for one scan period. 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 a **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 off-set 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.

1.7.6.4.2 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 a **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 micros after the reception of the response packet like shown in Figure 1.23 and not 625 micros after the received packet as is usual in the RX/TX timing. 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, so 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 sequenc*e. The channel hopping sequence uses all 79 hop channels in a (pseudo) random fashion. 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. The master can now send its first traffic packet in a hop determined with the new (master) parameters. The first packet in this state is a POLL packet sent by the master. 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 *new-connectionTO* number of slots, the master and the slave will return to page and page scan substates, respectively.

1.7.7 INQUIRY PROCEDURES

1.7.7.1 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. 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 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 sections 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.

1.7.7.2 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 sequenc*e. These frequencies are determined by the general inquiry address. The phase is determined by the native clock of the unit carrying out the **inquiry sca**n; the phase changes every 1.28s.

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 sca**n. 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. The scan window, T w inquiry scan , shall be increased to increase the probability to respond to an inquiry message. 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.

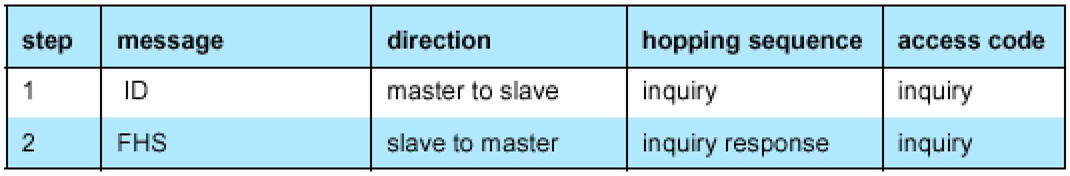
1.7.7.3 Inquiry

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. 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 must 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 must 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 *(inquiryT*O). 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.

1**.7.7.4 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 must 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 must 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 *inqre-spT*O, 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 responses 4 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. 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.



step	message	direction	hopping sequence	access code
1	ID	master to slave	inquiry	inquiry
2	FHS	slave to master	inquiry response	inquiry

Table 1.12: Messaging during inquiry routines.

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 per-forms 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.

SP mode	$T_{\text{mandatory pscan}}$
P0	$\geq$20s
P1	$\geq$ 40s
P2	$\geq$ 60s
Reserved	-

Table 1.13: Mandatory scan periods for P0, P1, P2 scan period modes.

1.7.8 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 sequenc*e. 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 **pag**e/**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 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.

1.7.8.1 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 non-addressed 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.

1.7.8.2 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* sniff. The slave starts listening at the sniff slots for N 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 sniff timeout or remaining of the receive slots, whichever is greater.

So, for N 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 sniff attempt =1 and N sniff timeout =0 cause the slave to listen only at the first sniff slot, irrespective of packets received from the master. Note that N 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* 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 mod *T* sniff = *D* sniff for initialization 1(CLK 27 ,CLK 26-1) mod *T* sniff = *D* sniff 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

1.7.8.3 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. 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.

1.7.8.4 Polling schemes

1.7.8.4.1 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 his 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.

1.7.8.5 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.

1.7.9 SCATTERNET

1.7.9.1 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.

1.7.9.2 Inter-piconet communications

Time multiplexing must 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.. In the four slots in between, one other piconet can be visited. Since the multiple piconets are not synchronized, guard time must be left to account for misalignment.

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.

1.7.9.3 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 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 in step1 through step 3 below. For the third example involving the transfer, new piconet parameters have to be communicated to each slave.

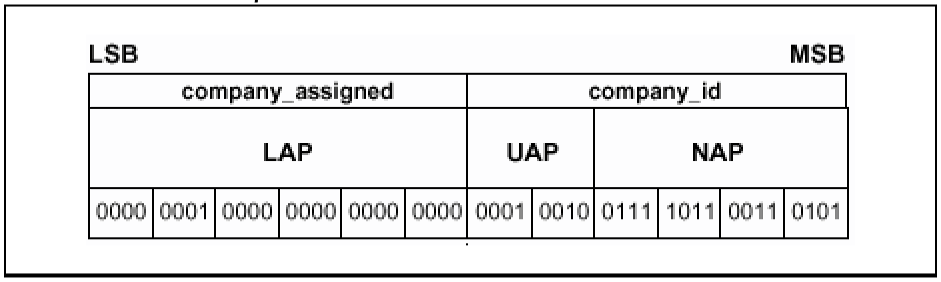
1.8 BLUETOOTH ADDRESSING

1.8.1 BLUETOOTH DEVICE ADDRESS (BD_ADDR)

Each Bluetooth transceiver is allocated a unique 48-bit Bluetooth device address (BD_ADDR). This address is derived from the IEEE802 standard. 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. The total address space obtained is 2 32 .
Figure 13.1: Format of BD_ADDR



LSB											MSB
company_assigned						company_id					
LAP						UAP		NAP			
0000	0001	0000	0000	0000	0000	0001	0010	0111	1011	0011	0101

Figure 1.26: Format of BD_ADDR

1.8.2 ACCESS CODES

In the Bluetooth system, 72-bit and 68-bit access codes are used for signaling purposes. Three different access codes are defined:-

• 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 sup-port 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.

1.8.3 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.

1.9 STATE DIAGRAM BASEBAND

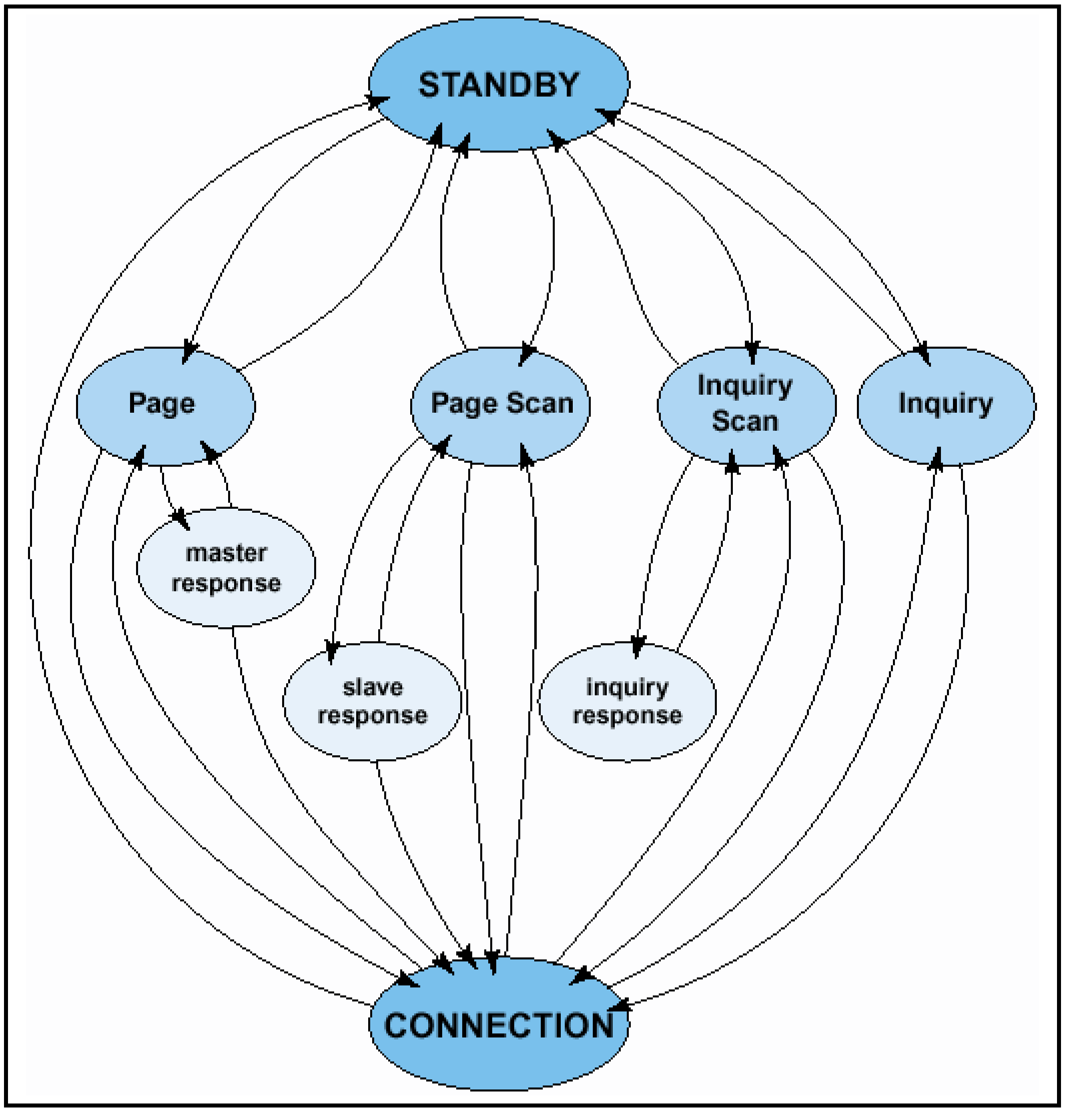
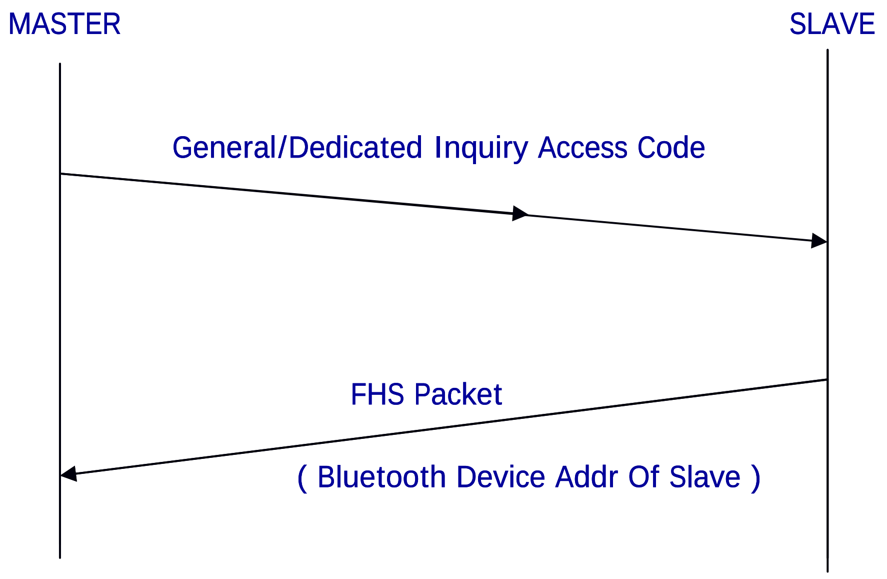


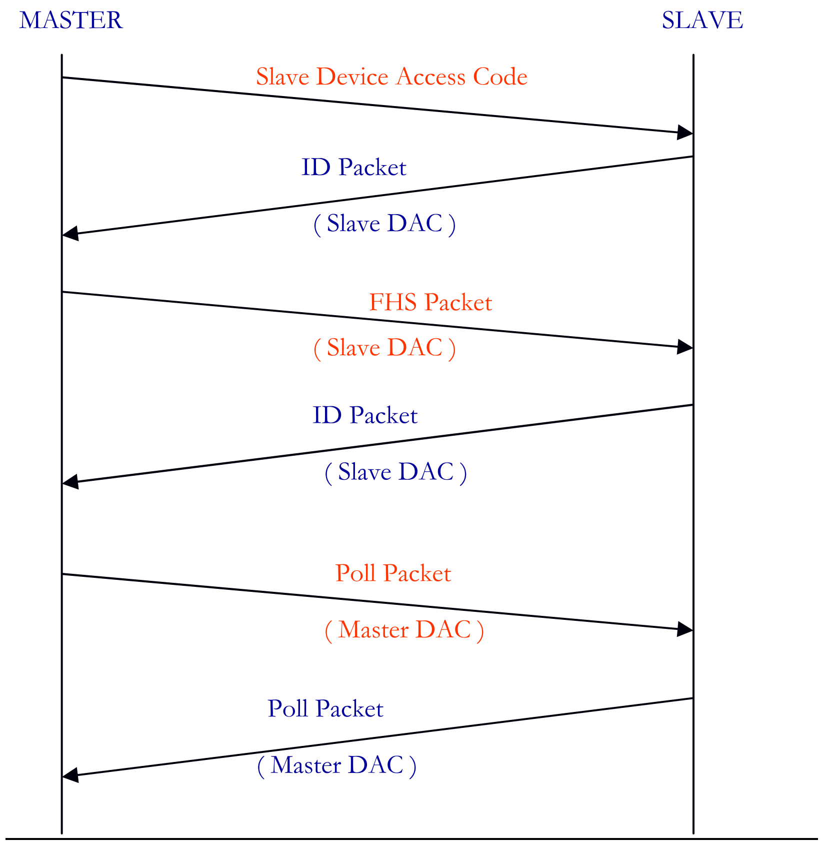
Figure 1.27: State diagram of Bluetooth link controller.

1.10 BASEBAND SEQUENCE DIAGRAMS

Sequence of inquiry:-


MASTER
SLAVE
General/Dedicated Inquiry Access Code
FHS Packet
(Bluetooth Device Addr Of Slave)

Sequence of Paging:-



CHAPTER 2

BLUETOOTH L2CAP PROTOCOL

L2CAP PROTOCOL

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. 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.

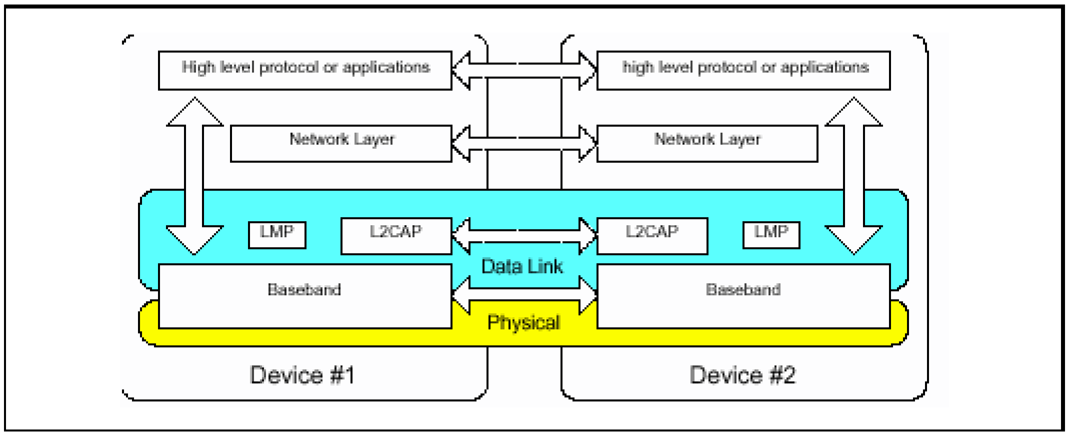


Figure 2.1: L2CAP within protocol layers

The format of the ACL payload header for the L2CAP Layer is shown here is shown below. Figure 2.2 displays the payload header used for single-slot packets and Figure 2.3 displays the header used in multi-slot 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 multi-slot packets.

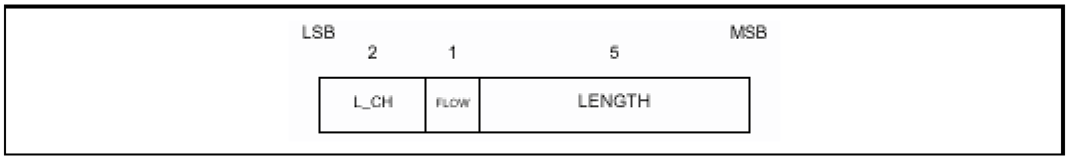


Figure 2.2: ACL Payload Header for single-slot packets

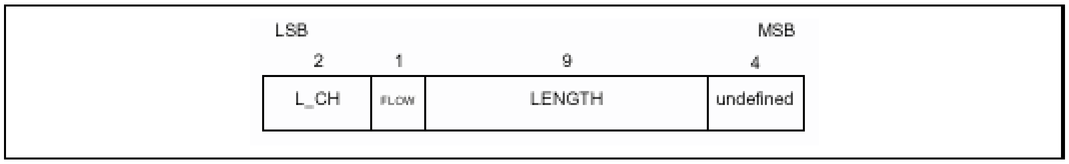


Figure 2.3: ACL Payload Header for multi-slot packets

The 2-bit logical channel (L_CH) field, defined in Table 1.1, distinguishes L2CAP packets. The remaining code is reserved for future use.

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

Table 2.1: Logical channel L_CH field contents

The FLOW bit in the ACL header is managed by the Link Controller (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.

2.1 L2CAP FUNCTIONAL REQUIREMENTS

The functional requirements for L2CAP include protocol multiplexing, segmentation and reassembly (SAR), and group management. Figure 1.4 illustrates how L2CAP fits into the Bluetooth Protocol Stack. L2CAP lies above the Base-band Protocol and interfaces with other communication protocols such as the Bluetooth Service Discovery Protocol (SDP) RFCOMM and Telephony Control. Packetized audio data, such as IPTelephony, may be sent using communication protocols running over L2CAP.

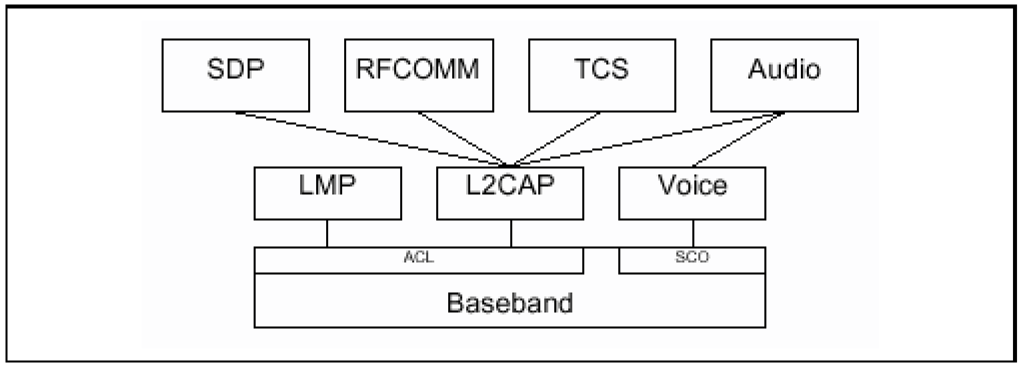


Figure 2.4: L2CAP in Bluetooth Protocol Architecture

Essential protocol requirements for L2CAP include simplicity and low over-head. L2CAP does not consume excessive power since that significantly sacrifices power efficiency achieved by the Bluetooth Radio. Memory requirements for protocol implementation are also kept to a minimum. The protocol complexity is acceptable to personal computers, PDAs, digital cellular phones, wireless headsets, joysticks and other wireless devices supported by Bluetooth so that maximum coverage of devices targeted for installation is achieved. Furthermore, the protocol is 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 therefore distinguishes between upper layer protocols such as the Service Discovery Protocol, RFCOMM , and Telephony Control .

• *Segmentation and Reassembly*

Compared to other wired physical media, the data packets defined by the Baseband Protocol are limited in size. Exporting a maximum transmission unit (MTU) associated with the largest Baseband payload 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. 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 quality of service (QoS) expected between two Blue-tooth 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.

2.2 ASSUMPTIONS

The protocol design makes the following assumptions:

1. The ACL link between two units is set up using the Link Manager Protocol. The Baseband provides orderly delivery of data packets, although there might be individual packet corruption and duplicates. No more than 1 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 bidirectional. 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. 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.

2.3 GENERAL OPERATION

The Logical Link Control and Adaptation Protocol (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*.

2.3.1 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 must 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 2.2 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 the table below). 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.

2.3.2 OPERATION BETWEEN 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 channels within an L2CAP entity is

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

Table 2.2: CID Definitions

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.

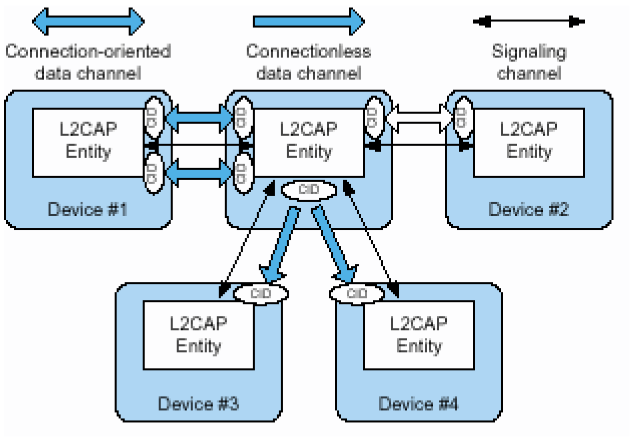


Figure 2.5: Channels between devices

Table 2.3 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. the state machine associated with each connection-oriented channel.

Channel Type	Local CID	Remote CID
Connection-oriented	Dynamically allocated	Dynamically allocated
Connectionless data	Dynamically allocated	0x0002 (fixed)
Signalling	0x0001 (fixed)	0x0001 (fixed)

Table 2.3: Types of Channel Identifiers

2.3.3 OPERATION BETWEEN LAYERS

L2CAP implementations should follow the general architecture described below. 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. The L2Cap events are passed to the upper and lower layers in the form of constant integers identifying each specific event.

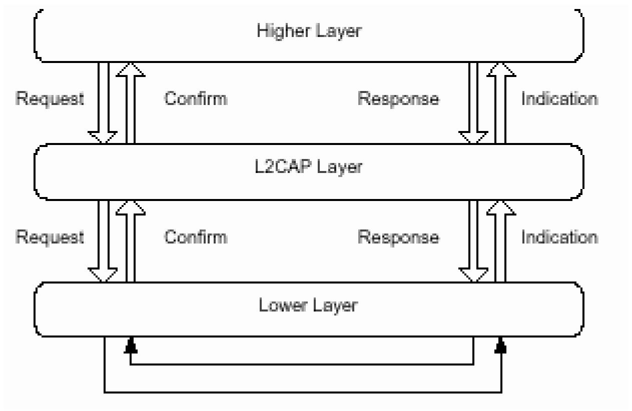


Figure 2.6: L2CAP Architecture

2.4 <u>SEGMENTATION AND REASSEMBLY</u>

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. On the receiving side, an L2CAP implementation receives ’chunks’ and reassembles those chunks into L2CAP packets using information from the packet header.

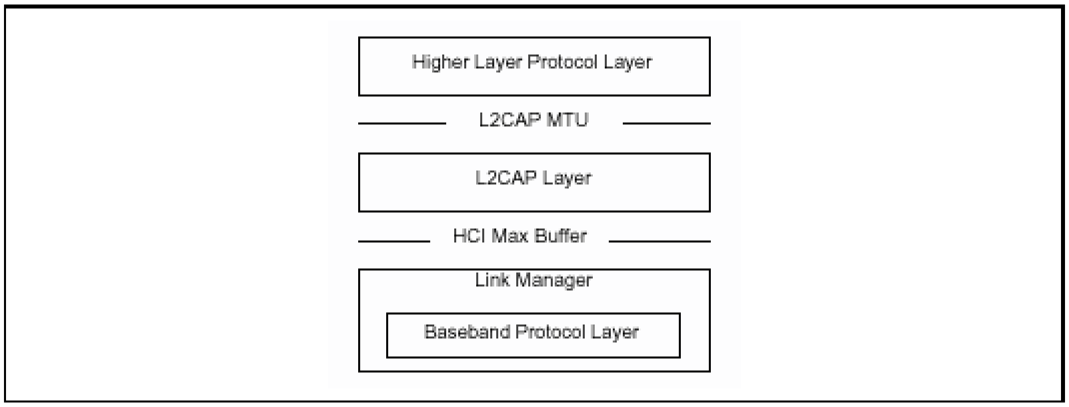


Figure 2.7: L2CAP SAR Variables

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.

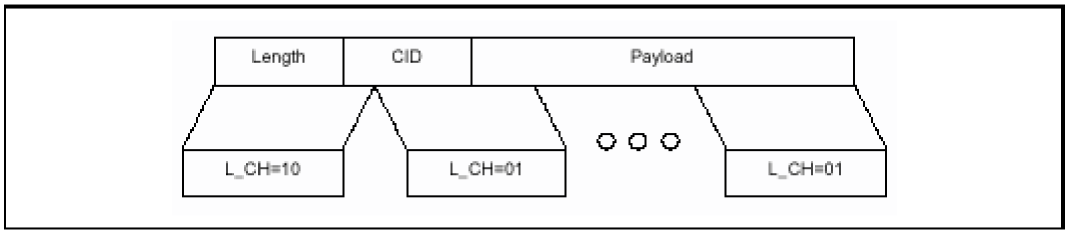


Figure 2.8: L2CAP segmentation

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 must be passed through to the Baseband before any other L2CAP packet destined to the same unit may be sent.

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 must use the length field in the header of L2CAP packets. If channel reliability is not needed, packets with improper lengths may be silently discarded. For reliable channels, L2CAP implementations must indicate to the upper layer that the channel has become unreliable. Reliable channels are defined by having an infinite flush timeout value. The figure illustrates segmentation and reassembly in the case of a single large PDU 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.

2.5 STATE MACHINE

This section describes the L2CAP connection-oriented channel state machine. The section 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.

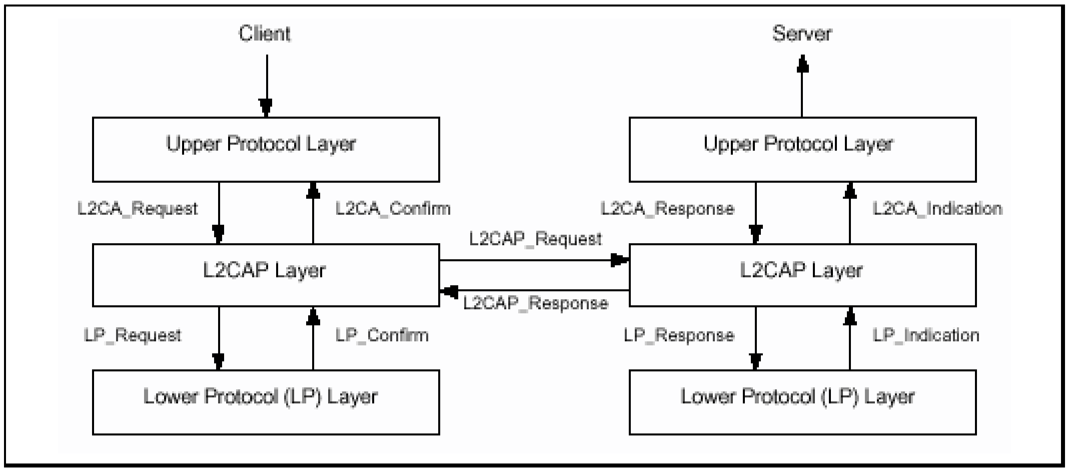


Figure 2.9: L2CAP Layer Interactions

Figure 2.7 illustrates the events and actions performed by the 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. 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.

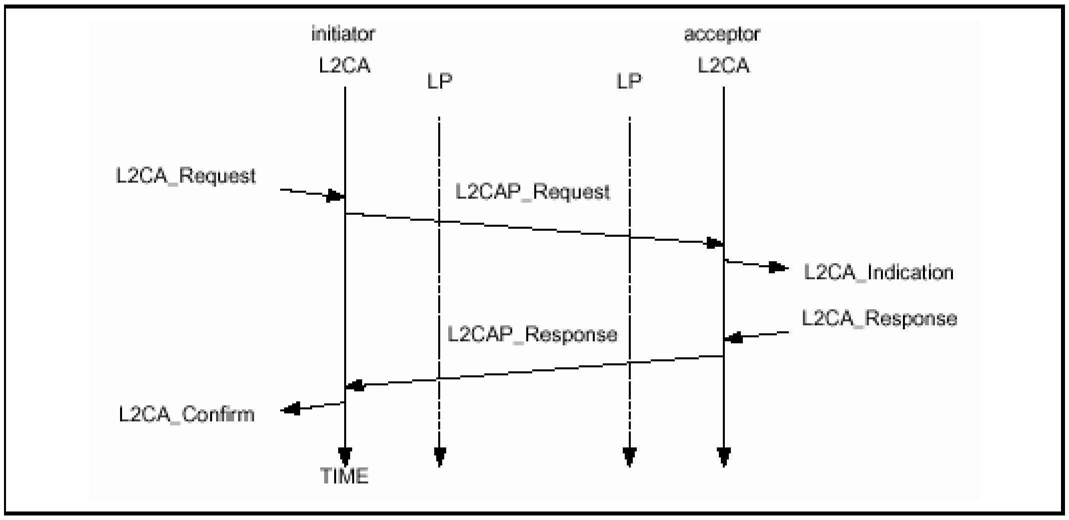


Figure 2.10:MSC of Layer Interactions

Figure 2.8 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 the to initiator. The result is passed back to the initiator's upper protocol using a Confirm message.

2.5.1 EVENTS

Events are all incoming messages to the L2CA 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.

2.5.1.1 Lower-Layer Protocol (LP) to L2CAP events

• *LP_ConnectCfm*
Confirms the request 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 to establish a lower layer (Baseband) connection failed. This could be because the device could not be contacted, refused the request, or the LMP authenti-cation 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 to 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 for a given quality of service.

• *LP_QoSCfmNeg*
Confirms the failure of the request for a given quality of service.

• *LP_QoSViolationInd*
Indicates the lower protocol has detected a violation of the QoS agreement specified in the previous *LP_QoSReq*

2.5.1.2 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. 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 signaling events are exchanged directly between the L2CAP peer entities

• *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 must 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.

2.5.1.3 L2CAP to L2CAP Data events

• *L2CAP_Data*
A Data packet has been received.

2.5.1.4 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 .

• *L2CA_ConnectRspNeg*
Negative response (rejection) from upper layer to the indication of a connection request from a remote device.

• *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

• *L2CA_ConfigRspNeg*

A negative response from upper layer to the indication of a (re) configuration request .

• *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 . 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.

2.5.1.5 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 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 must 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 signaling 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 minimum initial value is 1 second and the maximum initial value is 60 seconds. One RTX timer MUST 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 seconds.
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 minimum initial value is 60 seconds and the maximum initial value is 300 seconds. Similar to RTX, there MUST be at least one ERTX timer for each outstanding request that received a Pending response. There should be at most one (RTX or ERTX) associated with each outstanding request. The maximum elapsed time between the initial start of this timer and the initiation of channel termination (if no response is received) is 300 seconds. When terminating the channel, it is not necessary to send a L2CAP DisconnectReq and enter disconnection state. Channels should be transitioned directly to the Closed state.

2.5.2 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.

2.5.2.1 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 must be sent to the lower protocol to establish the physical connection. Since no more than a single

ACL link between two devices can exist additional L2CAP channels between these two devices must 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.

• *LP_ConnectRspNeg*
A negative response denying the previous connection indication request .

2.5.2.2 L2CAP to L2CAP Signalling actions

These actions define the events having the same names identified in the first section except the actions refer to the transmission, rather than reception, of these messages.

2.5.2.3 L2CAP to L2CAP Data actions

This section is the counterpart of 2.4.1.3. Data transmission is the action performed here.

2.5.2.4 L2CAP to Upper Layer actions

• *L2CA_ConnectInd*
Indicates a Connection Request has been received from a remote device.

• *L2CA_ConnectCfm*
Confirms that a Connection Request has been accepted (see
following the receipt of a Connection message from the remote device.

• *L2CA_ConnectCfmNeg*

Negative confirmation (failure) of a Connection Request. An RTX timer expiration 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 following the receipt of a Configuration Response from the remote device.

• *L2CA_ConfigCfmNeg*
Negative confirmation (failure) of a Configuration Request. An RTX timer expiration 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.

• *L2CA_DisconnectCfm*
Confirms that a Disconnect Request has been processed by the remote device following the receipt of a Disconnection Response from the remote device. An RTX timer expiration 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 occurs once sending a L2CA_DisconnectInd.

• *L2CA_QoSViolationInd*
Indicates that the quality of service agreement has been violated.

2.5.3 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 (Baseban*d)* 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 must be accepted until the remote channel endpoint has entered the CONFIG state.

In the CONFIG state, both sides must issue L2CAP_ConfigReq messages if only defaults are being used, a null message should be sent. If a large amount of parameters need to be negotiated, multiple messages may be sent to avoid any MTU limitations and negotiate incrementally. 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.

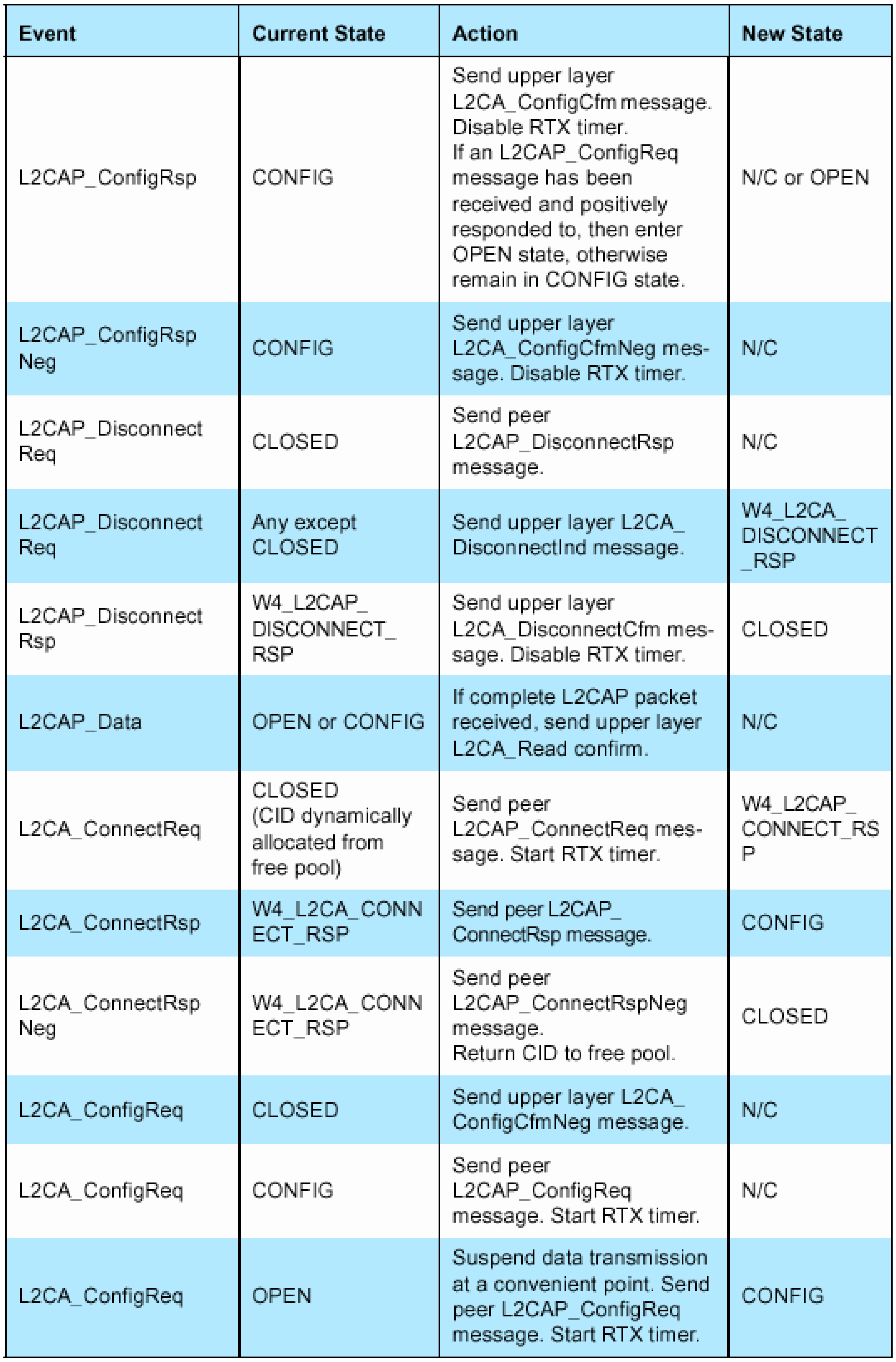
• *OPEN*
In this state, the connection has been established and configured, and data flow may proceed.

• *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.

2.5.4 MAPPING EVENTS TO ACTIONS

The Table 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.



Event	Current State	Action	New State
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 mes-sage. 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 mes-sage. 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 mes-sage. Start RTX timer.	W4_L2CAP_ CONNECT_RS P
L2CA_ConnectRsp	W4_L2CA_CONN ECT_RSP	Send peer L2CAP_ ConnectRsp message.	CONFIG
L2CA_ConnectRsp Neg	W4_L2CA_CONN ECT_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 2.4: L2CAP Channel State Machine

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 peer 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 resend 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 resend Request.	N/C or CLOSED

Table 2.4: L2CAP Channel State Machine

An example state diagram and sequence diagram illustrating the flow of events and actions within the L2CAP Layer moving it from one state to another are given at the end of the chapter.

2.6 DATA PACKET FORMAT

L2CAP is packet-based but follows a communication model based on *channel*s. A channel represents a data flow between L2CAP entities in remote devices. Channels may be connection-oriented or connectionless. All packet4 fields use Little Endian byte order.

2.6.1 CONNECTION-ORIENTED CHANNEL

Figure 2.10 illustrates the format of the L2CAP packet (also referred to as the L2CAP PDU) within a connection-oriented channel.

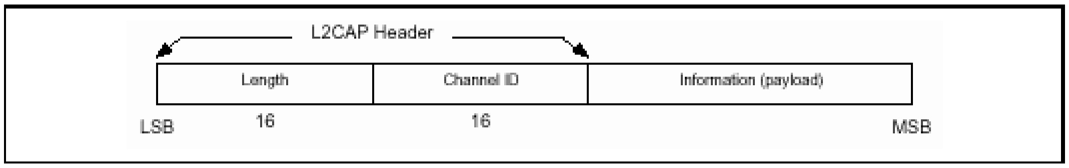


Figure 2.11: L2CAP Packet (field sizes in bits)

The fields shown are:

• *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. The minimum supported MTU for the signalling packet (MTU sig) is 48 bytes.

2.7 <u>SIGNALLING</u>

This section 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 2.11 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 must 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.

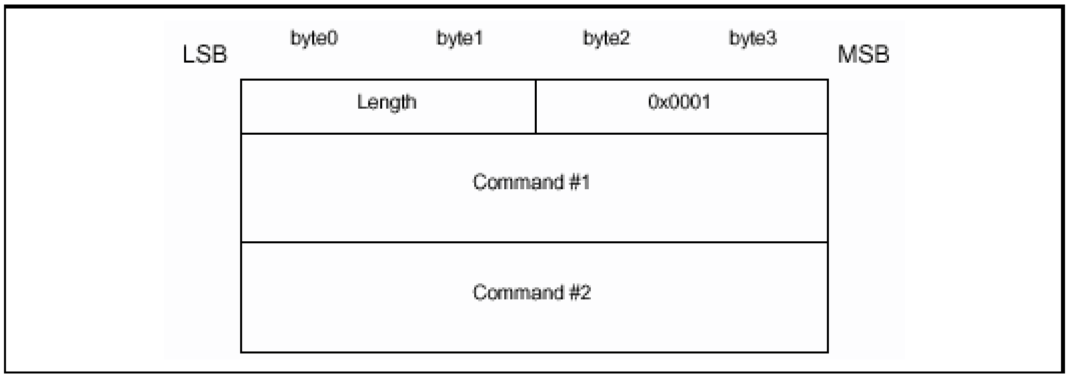


Figure.2.12: Signalling Command Packet Format

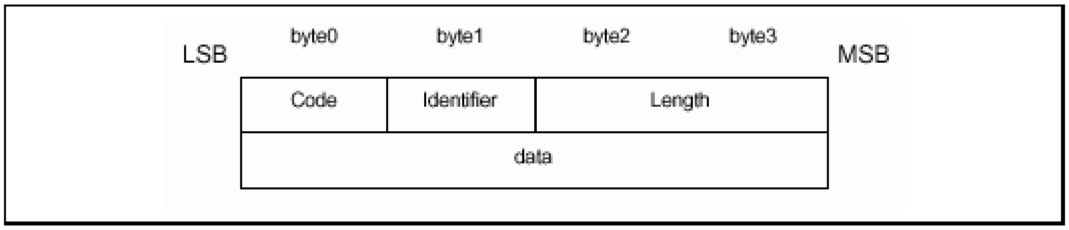
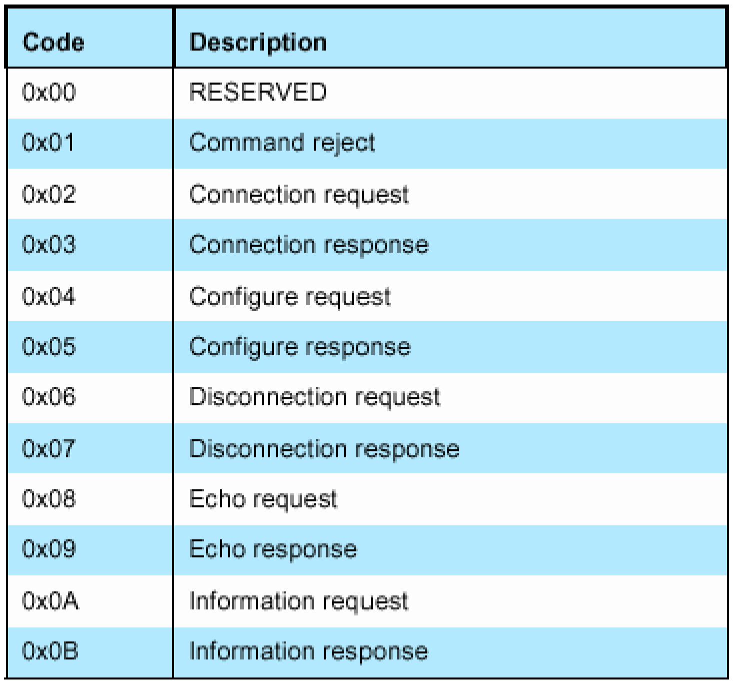


Figure 2.13: Command format

The fields shown are:

• *Code: 1 octet*

The Code field is one octet long and identifies the type of command. When a packet is received with an unknown Code field, a Command Reject packet is sent in response. Table 2.5 lists the codes used. All codes are specified with the most significant bit in the left-most position



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

Table 2.5: Signalling Command Codes

• *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 must be used for each original command. Identifiers should not be recycled until a period of 360 seconds 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 re-sent. 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.

2.7.1 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. The format of the packet is displayed in the adjoining figure. The Identifier should match the Identifier of the packet containing the unidentified code field. Implementations must always send these packets in response to unidentified signalling packets. Command Reject packets should not be sent in response to an identified Response packet. When multiple commands are included in an L2CAP packet and the packet exceeds the 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. The Code field determines the format of the Data field.

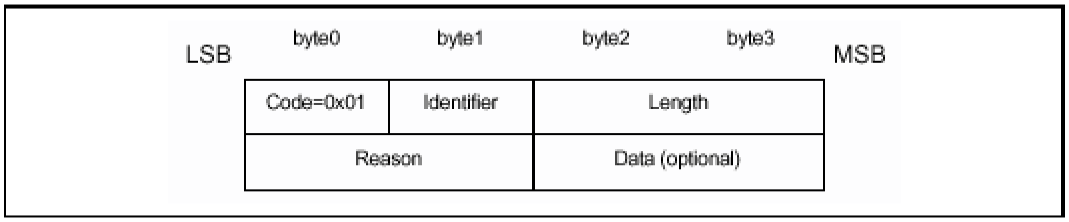
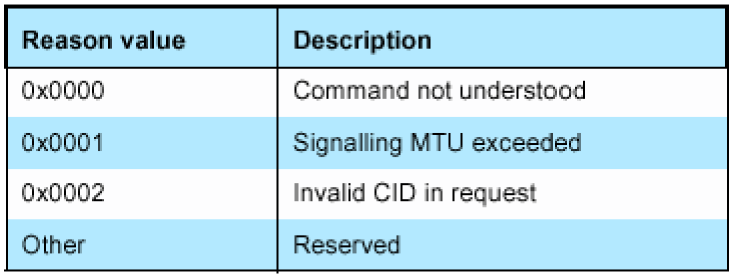


Figure 2.14: Command Reject Packet

• *Length = 0x0002 or more octets*
• *Reason: 2 octets*
The Reason field describes why the Request packet was rejected.



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

Table 2.6: Reason Code Descriptions

• *Data: 0 or more octets*
The length and content of the Data field depends on the Reason code. If the Reason code is 0x0000, “Command not understood”, no Data field is used. If the Reason code is 0x0001, “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 contains 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.

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

Table 2.7: Reason Data values

2.7.2 CONNECTION REQUEST (CODE 0x02)

Connection request packets are sent to create a channel between two devices. The channel connection must be established before configuration may begin. Figure 2.15 illustrates a Connection Request packet.

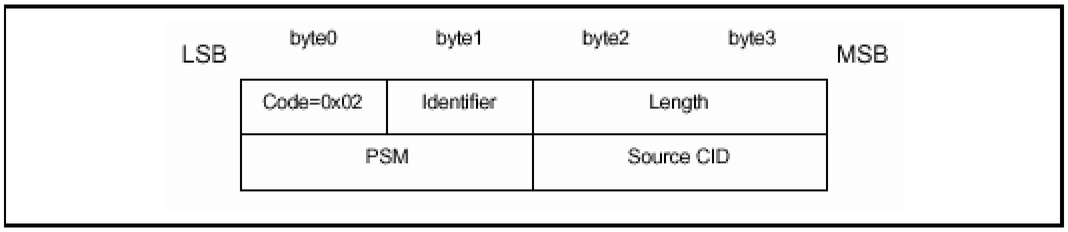


Figure 2.15: Connection Request Packet

• *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 must be ODD, that is, the least significant bit of the least significant octet must be ’1’. Also, all PSM values must 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 Service Discovery Protocol (SDP). The dynamically assigned values may be used to support multiple implementations of a particular protocol.

PSM value	Description
0x0001	Service Discovery Protocol
0x0003	RFCOMM
0x0005	Telephony Control Protocol
<0x1000	RESERVED

Table 2.8: Defined PSM Values

• *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 to the sender of the request must be send to this CID. In this section, the Source CID represents the channel endpoint on the device sending the request and receiving the response.

2.7.3 CONNECTION RESPONSE (CODE 0x03)

When a unit receives a Connection Request packet, it must send a Connection Response packet. The format of the connection response packet is shown in

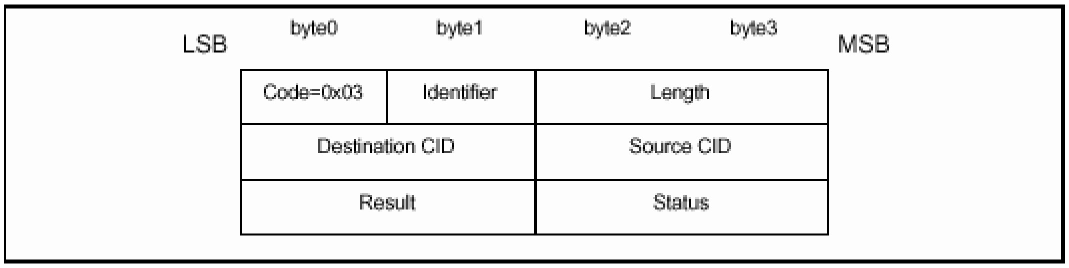


Figure 2.16: Connection Response Packet

• *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 chan-nel 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 non-zero value indicates the connection request failed or is pending. A logical channel is established on the receipt of a successful result. If the result field is not zero. The DCID and SCID fields should be ignored when the result field indicates the connection was refused.

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.

Table 2.9: Result values

• *Status: 2 octets*
Only defined for Result = Pending. Indicates the status of the connection.

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

Table 2.10: Status values

2.7.4 CONFIGURATION REQUEST (CODE 0x04)

Configuration Request packets are sent to establish an initial logical link transmission contract between two L2CAP entities and also to re-negotiate this contract whenever appropriate. During a re-negotiation session, all data traffic on the channel is 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. If an L2CAP entity receives a Configuration Request while it is waiting for a response it does not block sending the Configuration Response, otherwise the configuration process may result in deadlock. If no parameters need to be negotiated, no options need to be inserted and the C-bit is cleared. L2CAP entities in remote devices 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. Event if all default values are acceptable, a Configuration Request packet with no options is sent. Since most of the values are implicitly accepted they are infact the default values for each parameter 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 is sent in the direction opposite the one used for the original ConfigurationRequest. The amount of time (or messages) spent on arbitrating the channel parameters before terminating the negotiation is minimal due to acceptance of default values.

The figure defines the format of the Configuration Request packet.

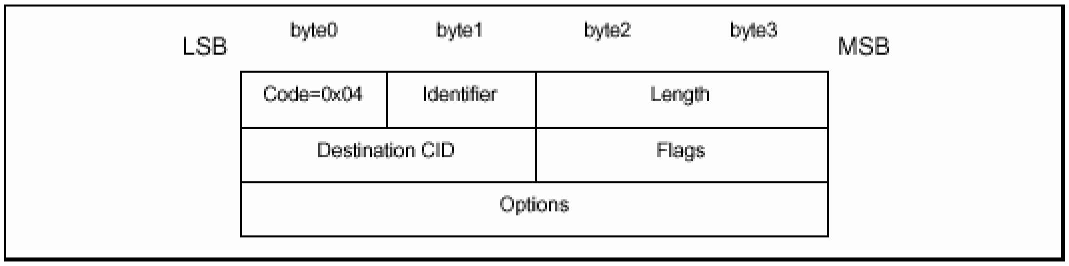


Figure 2.17: Configuration Request Packet

• *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*
The Figure displays the two-octet Flags field. Note the most significant bit is shown on the left.

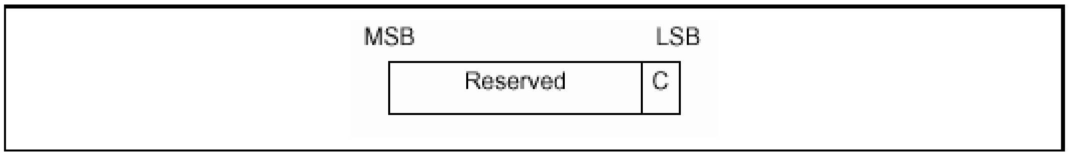


Figure 2.18: Configuration Request Flags field format

Of the C - continuation flag. When all configuration options cannot fit into the receiver's MTU sig , the are passed in multiple configuration command packets. If all options fit into the receiver's MTU, then the continuation bit is not used. Each Configuration Request contains an integral number of options. Each Request is tagged with a different Identifier and 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 replies 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 is treated as the Configuration Request event in the channel state machine. When used in the Configuration Response, the continuation flag must 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 sends 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 must succeed for the configuration transaction to succeed. Other flags are reserved and are therefore cleared. These are ignored by the L2Cap Layer.

Configuration Options

The list of the parameters and their values to be negotiated. 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.

2.7.5 CONFIGURE RESPONSE (CODE 0X05)

Configure Response packets are sent in reply to Configuration Request packets except when the error condition id is covered by a Command Reject response. Each configuration parameter value (if any is present) in a Configuration Response reflects an 'adjustment' to a configuration parameter value that has been sent (or, in case of default values, implied) in the corresponding Configuration Request. The options sent in the Response depend on the value in the Result field.

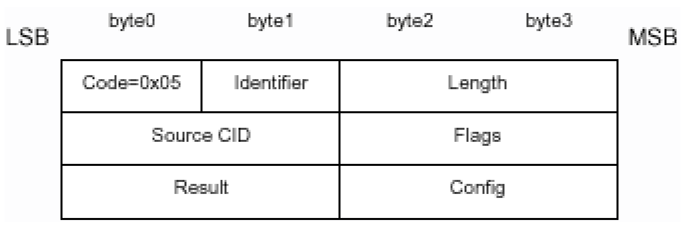


Figure 2.19: Configuration Response Packet

• *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 checks 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*
The Figure displays the two-octet Flags field. Note the most significant bit is shown on the left.

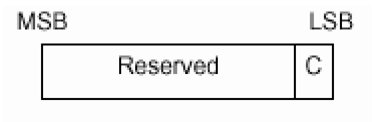


Figure 2.20: Configuration Response Flags field format

C – 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 are cleared. L2CAP implementations ignore these bits.

• *Result: 2 octets*
The Result field indicates whether or not the Request was acceptable.

Result	Description
0x0000	Success
0x0001	Failure – unacceptable parameters
0x0002	Failure – rejected (no reason provided)
0x0003	Failure – unknown options
Other	RESERVED

Table 2.11: Configuration Response Result codes

• *Configuration Options*
This field contains the list of parameters being negotiated.

2.7.6 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. The figure displays a disconnection packet request. The receiver must 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.

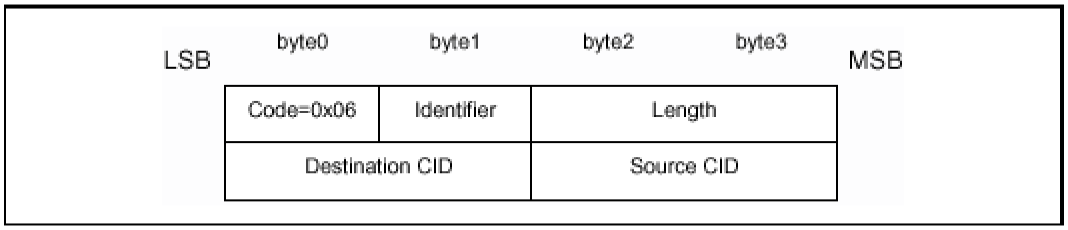


Figure 2.21: Disconnection Request Packet

• *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 matches those of the channel to be disconnected. If the DCID is not recognized by the receiver of this message, a CommandReject message with 'invalid CID' result code is sent in response. If the receivers finds a DCID match but the SCID fails to find the same match, the request is silently discarded.

2.7.7 DISCONNECTION RESPONSE (CODE 0x07)

Disconnection responses are sent in response to each disconnection request.

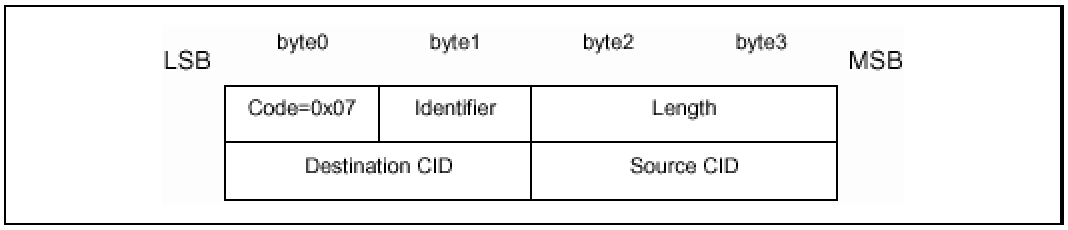


Figure 2.22 : Disconnection Response Packet

• *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 match those of the corresponding disconnection request command. If the CIDs do not match, the response is silently discarded at the receiver.

2.7.8 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 respond to well-formed Echo Request packets with an Echo Response packet. The Data field is optional and implementation-dependent. L2CAP entities ignore the contents of this field.

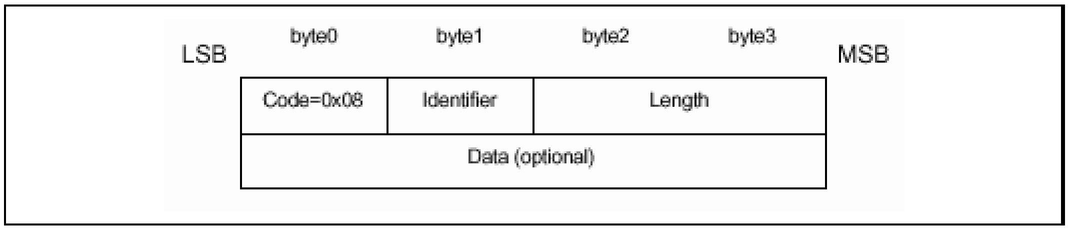


Figure 2.23: Echo Request Packet

2.7.9 ECHO RESPONSE (CODE 0x09)

Echo responses are sent upon receiving Echo Request packets. The identifier in the response matches 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.

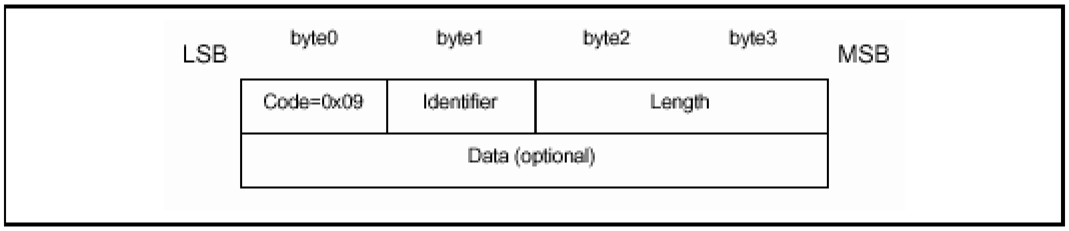


Figure 2.24: Echo Response Packet

2.7.10 INFORMATION REQUEST (CODE 0X0A)

Information requests are used to solicit implementation-specific information from a remote L2CAP entity. L2CAP entities respond to well-formed Information Request packets with an Information Response packet.

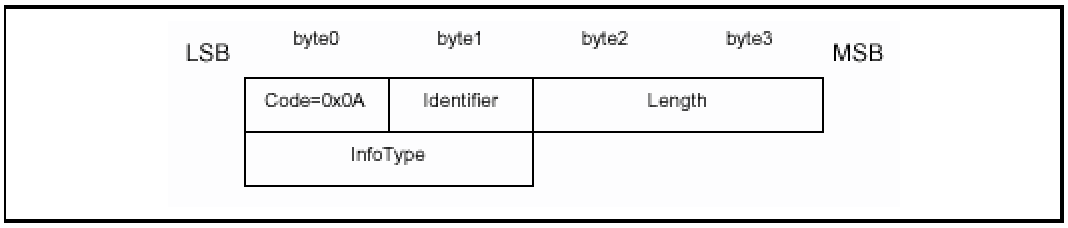


Figure 2.25: Information Request Packet

• *Length = 0x0002 octets*

• *InfoType: 2 octets*
The InfoType defines the type of implementation-specific information being solicited.

Value	Description
0x0001	Connectionless MTU
Other	Reserved

Table 2.11: InfoType definitions

2.7.11 INFORMATION RESPONSE (CODE 0X0B)

Information responses are sent upon receiving Information Request packets. The identifier in the response matches 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.

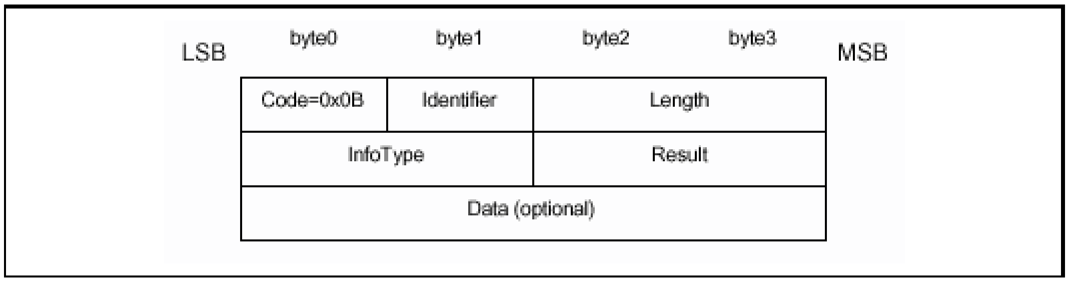


Figure 2.26: Information Response Packet

• *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 2.12. If result is "Not supported", no data is returned.

Value	Description
0x0000	Success
0x0001	Not supported
Other	Reserved

Table 2.12: Information Response Result values

• *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.

InfoType	Data	Data Length (in octets)
0x0001	Connectionless MTU	2

Table 2.13: Information Response Data fields

2.8 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 comprising an option type, an option length, and one or more option data fields. Figure 2.27 illustrates the format of an option.

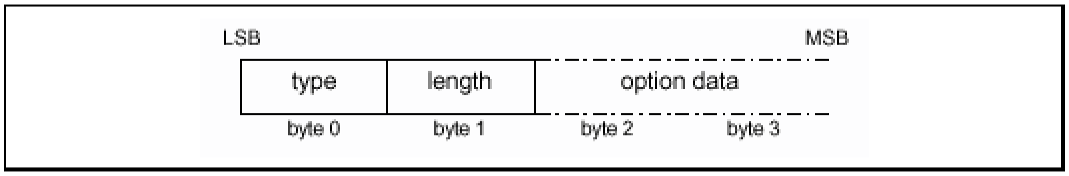


Figure 2.27: Configuration option format

• *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 below.

0 - option must 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.

2.8.1 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. 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, The device will stop configuration negotiation and try send data according to the default values otherwise the request is denied and connection establishment is attempted with another device. 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.

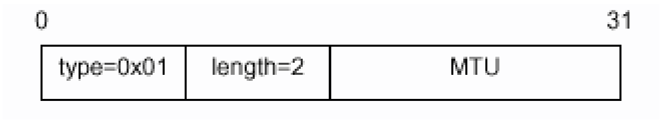


Figure 2.28: MTU Option Format

• Maximum Transmission Unit (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. The value is 672 bytes.

2.8.2 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.

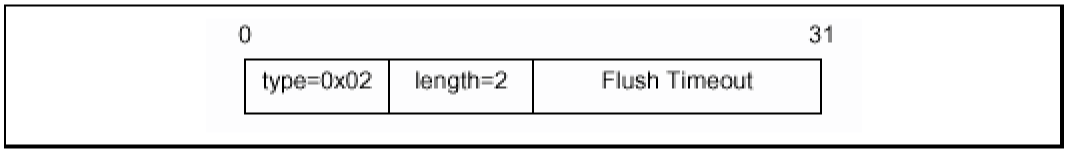


Figure 2.29: Flush Timeout

• *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 continues 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.

2.8.3 CONFIGURATION PROCESS

Negotiating the channel parameters involves three steps:

1. Informing the remote side of the non-default 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.

2.8.3.1 Request Path

The Request Path negotiates the incoming MTU, flush timeout, and outgoing flowspec. Table 2.13 defines the configuration options that may be placed in the Configuration Request message and their semantics.

Parameter	Description
MTU	Incoming MTU information
FlushTO	Outgoing flush timeout
OutFlow	Outgoing flow information.

Table 2.13: Parameters allowed in Request

2.8.3.2 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'soutgoing flowspec). If a request-oriented parameter is not present in the Request message (reverts to default value), the remote side may negotiate for a non-default value by including the proposed value in a negative Response message.

Parameter	Description
MTU	Outgoing MTU information
FlushTO	Incoming flush timeout
InFlow	Incoming flow information

Table 2.14: Parameters allowed in Response

2.8.3.3 Configuration State Machine

The configuration state machine shown below depicts two paths. Before leaving the CONFIG state and moving into the OPEN state, both paths must 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.

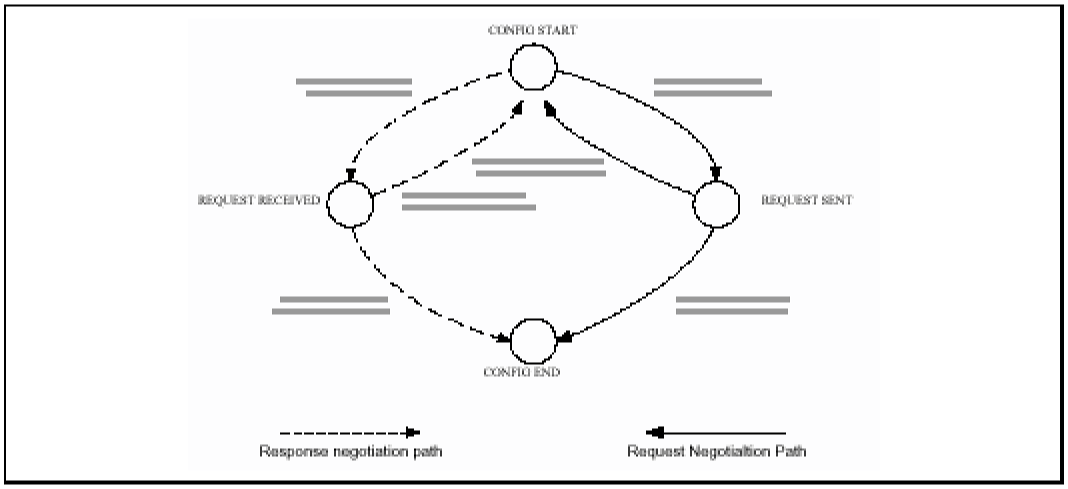
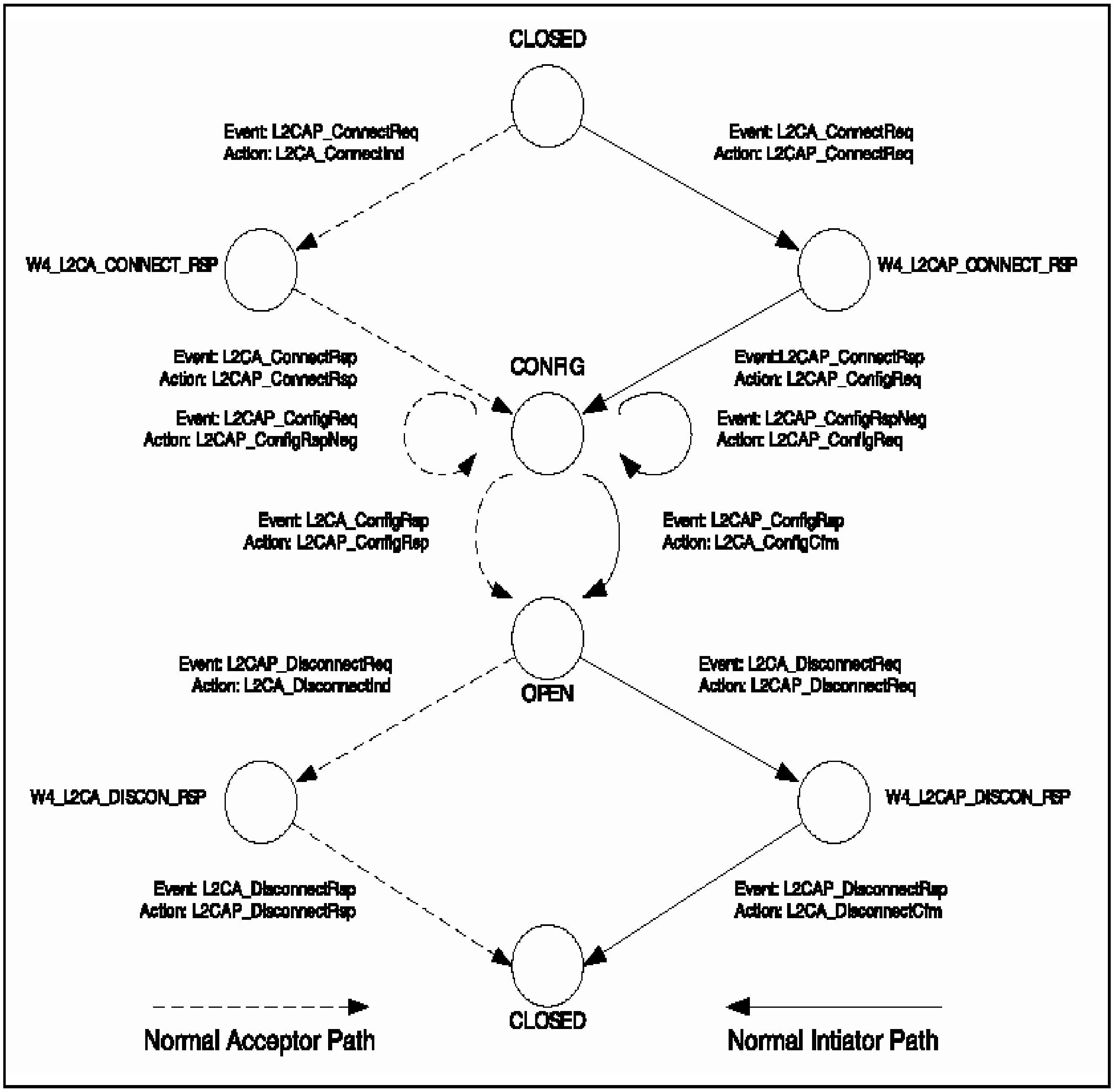
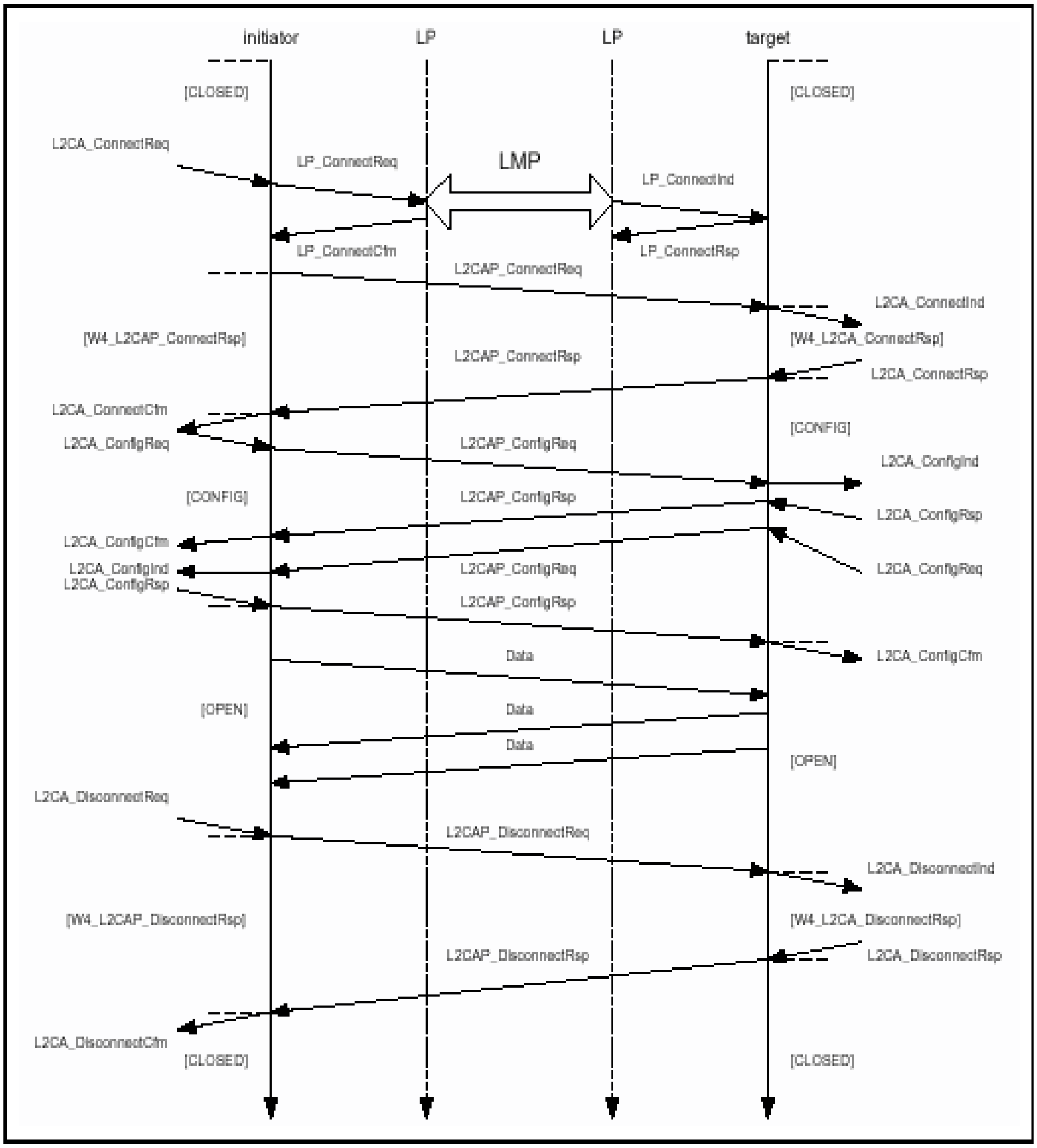


Figure 2.30: Configuration State Machine

2.9 SAMPLE STATE DIAGRAM



2.10 SEQUENCE OF EVENTS IN L2CAP INTERACTION



CHAPTER 3

SERVICE DICOVERY PROTOCOL

SERVICE DICOVERY PROTOCOL

3.1 INTRODUCTION

3.1.1 GENERAL DESCRIPTION

The service discovery protocol (SDP) provides a means for applications to discover which services are available and to determine the characteristics of those available services.

3.1.2 MOTIVATION

Service Discovery in the Bluetooth environment, where the set of services that are available changes dynamically based on the RF proximity of devices in motion, is qualitatively different from service discovery in traditional network-based environments. The service discovery protocol defined in this specification is intended to address the unique characteristics of the Bluetooth environment.

3.1.3 CAPABILITIES

The following capabilities are present in the Service Discovery Protocol.

1. SDP provides the ability for clients to search for needed services based on specific attributes of those services.

2. SDP permits services to be discovered based on the class of service.

3. SDP enables browsing of services without a priori knowledge of the specific characteristics of those services.

4. SDP provides the means for the discovery of new services that become available when devices enter RF proximity with a client device as well as when a new service is made available on a device that is in RF proximity with the client device.

5. SDP provides a mechanism for determining when a service becomes unavailable when devices leave RF proximity with a client device as well as when a service is made unavailable on a device that is in RF proximity with the client device.

6. SDP provides for services, classes of services, and attributes of services to be uniquely identified.

7. SDP allows a client on one device to discover a service on another device without consulting a third device.

8. SDP is suitable for use on devices of limited complexity.

9. SDP provides a mechanism to incrementally discover information about the services provided by a device. This is intended to minimize the quantity of data that must be exchanged in order to determine that a particular service is not needed by a client.

10.SDP supports the caching of service discovery information by intermediary agents to improve the speed or efficiency of the discovery process.

11.SDP is transport independent.

12.SDP functions while using L2CAP as its transport protocol.

13.SDP permits the discovery and use of services that provide access to other service discovery protocols.

14.SDP supports the creation and definition of new services without requiring registration with a central authority.

3.1.4 CONVENTIONS

3.1.4.1 Bit And Byte Ordering Conventions

When multiple bit fields are contained in a single byte and represented in a drawing in this specification, the more significant (high-order) bits are shown toward the left and less significant (low-order) bits toward the right. Multiple-byte fields are drawn with the more significant bytes toward the left and the less significant bytes toward the right. Multiple-byte fields are transferred in network byte order.

3.2 <u>OVERVIEW</u>

3.2.1 SDP CLIENT-SERVER INTERACTION

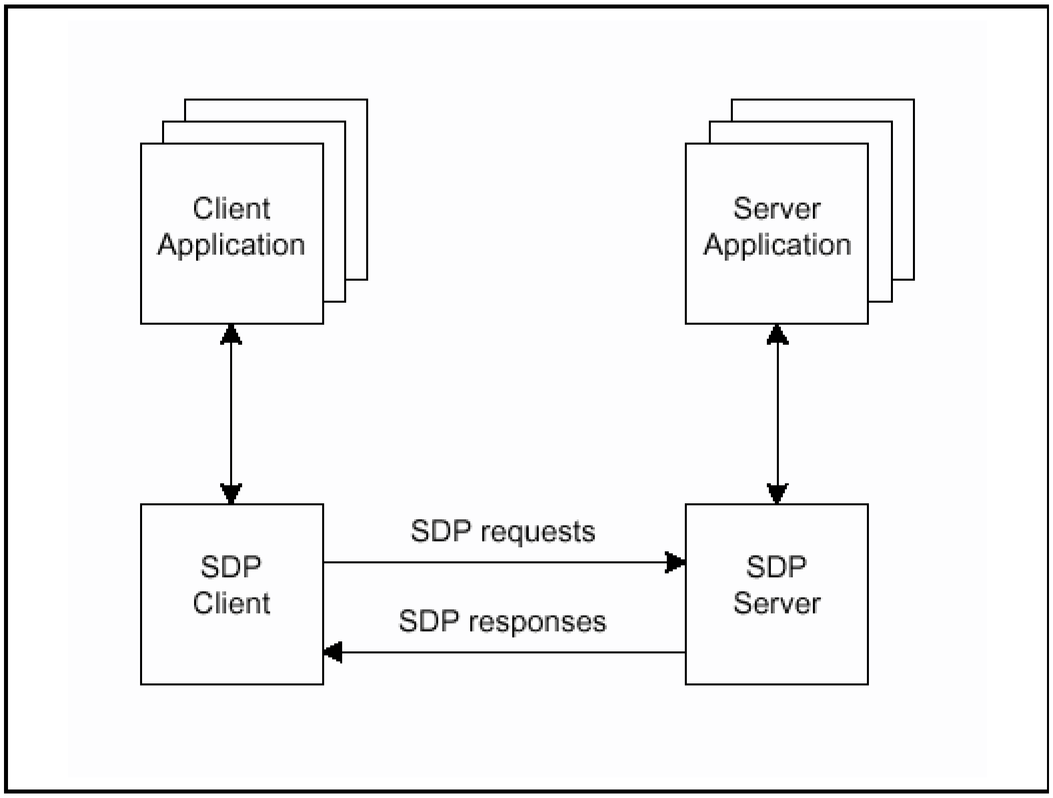


Figure 3.1

The service discovery mechanism provides the means for client applications to discover the existence of services provided by server applications as well as the attributes of those services. The attributes of a service include the type or class of service offered and the mechanism or protocol information needed to utilize the service. As far as the Service Discovery Protocol (SDP) is concerned, the configuration shown in Figure 1 may be simplified to that shown in Figure 2.

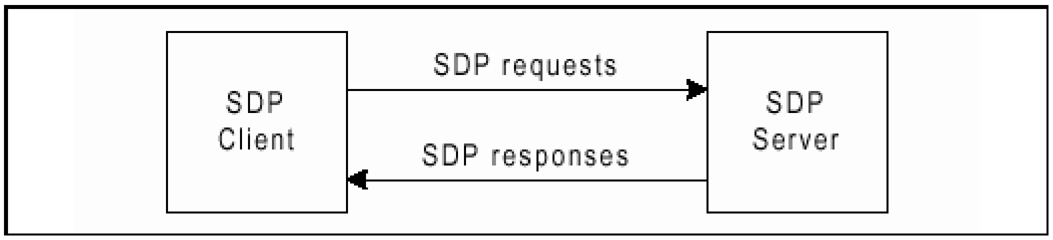


Figure 3.2:

SDP involves communication between an SDP server and an SDP client. The server maintains a list of service records that describe the characteristics of services associated with the server. Each service record contains information about a single service. A client may retrieve information from a service record maintained by the SDP server by issuing an SDP request. If the client, or an application associated with the client, decides to use a service, it must open a separate connection to the service provider in order to utilize the service. SDP provides a mechanism for discovering services and their attributes (including associated service access protocols), but it does not provide a mechanism for utilizing those services (such as delivering the service access protocols). There is a maximum of one SDP server per Bluetooth device. (If a Bluetooth device acts only as a client, it needs no SDP server.) A single Bluetooth device may function both as an SDP server and as an SDP client. If multiple applications on a device provide services, an SDP server may act on behalf of those service providers to handle requests for information about the services that they provide. Similarly, multiple client applications may utilize an SDP client to query servers on behalf of the client applications. The set of SDP servers that are available to an SDP client can change dynamically based on the RF proximity of the servers to the client. When a server becomes available, a potential client must be notified by a means other than SDP so that the client can use SDP to query the server about its services. Similarly, when a server leaves proximity or becomes unavailable for any reason, there is no explicit notification via the service discovery protocol. However,
the client may use SDP to poll the server and may infer that the server is not available if it no longer responds to requests.

3.2.2 SERVICE RECORD

A service is any entity that can provide information, perform an action, or control a resource on behalf of another entity. A service may be implemented as software, hardware, or a combination of hardware and software. All of the information about a service that is maintained by an SDP server is contained within a single service record. The service record consists entirely of a list of service attributes.

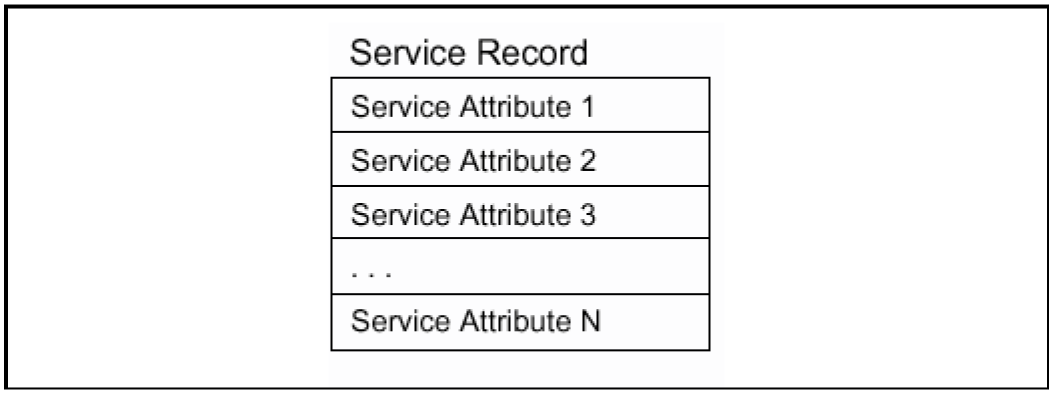


Figure 3.3: Service Record

A service record handle is a 32-bit number that uniquely identifies each service record within an SDP server. It is important to note that, in general, each handle is unique only within each SDP server. If SDP server S1 and SDP server S2 both contain identical service records (representing the same service), the service record handles used to reference these identical service records are completely independent. The handle used to reference the service on S1 will be meaningless if presented to S2. The service discovery protocol does not provide a mechanism for notifying clients when service records are added to or removed from an SDP server. While an L2CAP (Logical Link Control and Adaptation Protocol) connection is established to a server, a service record handle acquired from the server will remain valid unless the service record it represents is removed. If a service is removed from the server, further requests to the server (during the L2CAP connection in which the service record handle was acquired) using the service's (now stale) record handle will result in an error response indicating an invalid service record handle. An SDP server must ensure that no service record handle values are reused while an L2CAP connection remains established. The service record handles remain valid across successive L2CAP connections while the ServiceDatabaseState attribute value remains unchanged. There is one service record handle whose meaning is consistent across all SDP servers. This service record handle has the value 0x00000000 and is a handle to the service record that represents the SDP server itself. This service record contains attributes for the SDP server and the protocol it supports. For example, one of its attributes is the list of SDP protocol versions supported by the server. Service record handle values 0x00000001-0x0000FFFF are reserved.

3.2.3 SERVICE ATTRIBUTE

Each service attribute describes a single characteristic of a service. Some examples of service attributes are:

ServiceClassIDList	Identifies the type of service represented by a service record. In other words, the list of classes of which the service is an instance
ServiceID	Uniquely identifies a specific instance of a service
ProtocolDescriptorList	Specifies the protocol stack(s) that may be used to utilize a service
ProviderName	The textual name of the individual or organization that provides a service
IconURL	Specifies a URL that refers to an icon image that may be used to represent a service
ServiceName	A text string containing a human-readable name for the service
ServiceDescription	A text string describing the service

Service providers can also define their own service attributes. A service attribute consists of two components: an attribute ID and an attribute value.

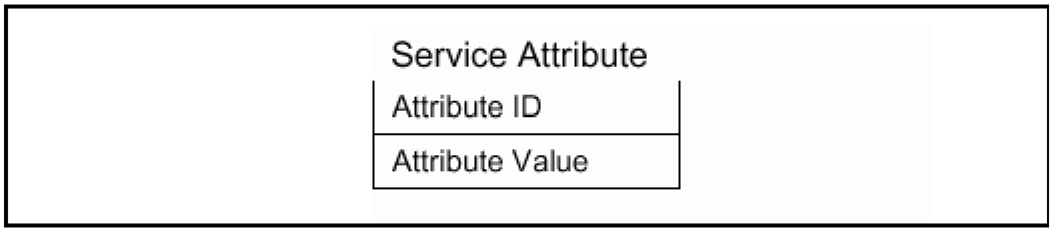


Figure 3.4: Service Attribute

3.2.4 ATTRIBUTE ID

An attribute ID is a 16-bit unsigned integer that distinguishes each service attribute from other service attributes within a service record. The attribute ID also identifies the semantics of the associated attribute value. A service class definition specifies each of the attribute IDs for a service class and assigns a meaning to the attribute value associated with each attribute ID. For example, assume that service class C specifies that the attribute value associated with attribute ID 12345 is a text string containing the date the service was created. Assume further that service A is an instance of service class C. If service A’s service record contains a service attribute with an attribute ID of 12345, the attribute value must be a text string containing the date that service A was created. However, services that are not instances of service class C may assign a different meaning to attribute ID 12345. All services belonging to a given service class assign the same meaning to each particular attribute ID. In the Service Discovery Protocol, an attribute ID is often represented as a data element.

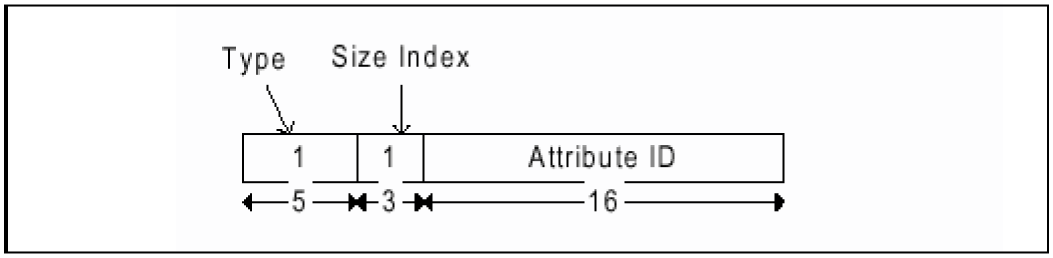


Figure 3.5:

3.2.5 ATTRIBUTE VALUE

The attribute value is a variable length field whose meaning is determined by the attribute ID associated with it and by the service class of the service record in which the attribute is contained. In the Service Discovery Protocol, an attribute value is represented as a data element. Generally, any type of data element is permitted as an attribute value, subject to the constraints specified in the service class definition that assigns an attribute ID to the attribute and assigns a meaning to the attribute value.

3.2.6 SERVICE CLASS

Each service is an instance of a service class. The service class definition provides the definitions of all attributes contained in service records that represent instances of that class. Each attribute definition specifies the numeric value of the attribute ID, the intended use of the attribute value, and the format of the attribute value. A service record contains attributes that are specific to a service class as well as universal attributes that are common to all services. Each service class is also assigned a unique identifier. This service class identifier is contained in the attribute value for the ServiceClassIDList attribute, and is represented as a UUID Since the format and meanings of many attributes in a service record are dependent on the service class of the service record, the ServiceClassIDList attribute is very important. Its value is examined or verified before any class-specific attributes are used. Since all of the attributes in a service record conform to all of the service's classes, the service class identifiers contained in the ServiceClassIDList attribute are related. Typically, each service class is a subclass of another class whose identifier is contained in the list. A service subclass definition differs from its superclass in that the subclass contains additional attribute definitions that are specific to the subclass. The service class identifiers in the ServiceClassIDList attribute are listed in order from the most specific class to the most general class. When a new service class is defined that is a subclass of an existing service class, the new service class retains all of the attributes defined in its super-class. Additional attributes will be defined that are specific to the new service class. In other words, the mechanism for adding new attributes to some of the instances of an existing service class is to create a new service class that is a subclass of the existing service class.

3.2.7 SEARCHING FOR SERVICES

Once an SDP client has a service record handle, it may easily request the values of specific attributes, but how does a client initially acquire a service record handle for the desired service records? The Service Search transaction allows a client to retrieve the service record handles for particular service records based on the values of attributes contained within those service records. The capability search for service records based on the values of arbitrary attributes is not provided. Rather, the capability is provided to search only for attributes whose values are Universally Unique Identifiers 1 (UUIDs). Important attributes of services that can be used to search for a service are represented as UUIDs.

3.2.7.1 UUID

A UUID is a universally unique identifier that is guaranteed to be unique across all space and all time. UUIDs can be independently created in a distributed fashion. No central registry of assigned UUIDs is required. A UUID is a 128-bit value.

To reduce the burden of storing and transferring 128-bit UUID values, a range of UUID values has been pre-allocated for assignment to often-used, registered purposes. The first UUID in this pre-allocated range is known as the Bluetooth Base UUID and has the value 00000000-0000-1000-8000-00805F9B34FB, from the Bluetooth Assigned Numbers document. UUID values in the pre-allocated range have aliases that are represented as 16-bit or 32-bit values. These aliases are often called 16-bit and 32-bit UUIDs, but it is important to note that each actually represents a 128-bit UUID value.
The full 128-bit value of a 16-bit or 32-bit UUID may be computed by a simple arithmetic operation. 128_bit_value = 16_bit_value * 2 96 + Bluetooth_Base_UUID 128_bit_value = 32_bit_value * 2 96 +

Bluetooth_Base_UUID A 16-bit UUID may be converted to 32-bit UUID format by zero-extending the 16-bit value to 32-bits. An equivalent method is to add the 16-bit UUID value to a zero-valued 32-bit UUID. Note that two 16-bit UUIDs may be compared directly, as may two 32-bit UUIDs or two 128-bit UUIDs. If two UUIDs of differing sizes are to be compared, the shorter UUID must be converted to the longer UUID format before comparison.

3.2.7.2 Service Search Patterns

A service search pattern is a list of UUIDs used to locate matching service records. A service search pattern is said to match a service record if each and every UUID in the service search pattern is contained within any of the service record’s attribute values. The UUIDs need not be contained within any specific attributes or in any particular order within the service record. The service search pattern matches if the UUIDs it contains constitute a subset of the UUIDs in the service record’s attribute values. The only time a service search pattern does not match a service record is if the service search pattern contains at least one UUID that is not contained within the service record’s attribute values. Note also that a valid service search pattern must contain at least one UUID.

3.2.8 BROWSING FOR SERVICES

Normally, a client searches for services based on some desired characteristic(s) (represented by a UUID) of the services. However, there are times when it is desirable to discover which types of services are described by an SDP server’s service records without any a priori information about the services. This process of looking for any offered services is termed browsing. In SDP, the mechanism for browsing for services is based on an attribute shared by all service classes. This attribute is called the BrowseGroupList attribute. The value of this attribute contains a list of UUIDs. Each UUID represents a browse group with which a service may be associated for the purpose of browsing. When a client desires to browse an SDP server’s services, it creates a service search pattern containing the UUID that represents the root browse group. All services that may be browsed at the top level are made members of the root browse group by having the root browse group’s UUID as a value within the BrowseGroupList attribute. Normally, if an SDP server has relatively few services, all of its services will be placed in the root browse group. However, the services offered by an SDP server may be organized in a browse group hierarchy, by defining additional browse groups below the root browse group. Each of these additional browse groups is described by a service record with a service class of BrowseGroupDescriptor. A browse group descriptor service record defines a new browse group by means of its Group ID attribute. In order for a service contained in one of these newly defined browse groups to be browseable, the browse group descriptor service record that defines the new browse group must in turn be browseable. The hierarchy of browseable services that is provided by the use of browse group descriptor service records allows the services contained in an SDP server to be incrementally browsed and is particularly useful when the SDP server contains many service records.

3.3 DATA REPRESENTATION

Attribute values can contain information of various types with arbitrary complexity; thus enabling an attribute list to be generally useful across a wide variety of service classes and environments. SDP defines a simple mechanism to describe the data contained within an
attribute value. The primitive construct used is the data element.

3.3.1 DATA ELEMENT

A data element is a typed data representation. It consists of two fields: a header field and a data field. The header field, in turn, is composed of two parts: a type descriptor and a size descriptor. The data is a sequence of bytes whose length is specified in the size descriptor and whose meaning is (partially) specified by the type descriptor.

3.3.2 DATA ELEMENTTYPE DESCRIPTOR

A data element type is represented as a 5-bit type descriptor. The type descriptor is contained in the most significant (high-order) 5 bits of the first byte of the data element header. The following types have been defined.

Type Descriptor Value	Valid Size Descriptor Values	Type Description
0	0	Nil, the null type
1	0, 1, 2, 3, 4	Unsigned Integer
2	0, 1, 2, 3, 4	Signed twos-complement integer
3	1, 2, 4	UUID, a universally unique identifier
4	5, 6, 7	Text string
5	0	Boolean
6	5, 6, 7	Data element sequence, a data element whose data field is a sequence of data elements
7	5, 6, 7	Data element alternative, data element whose data field is a sequence of data elements from which one data element is to be selected.
8	5, 6, 7	URL, a uniform resource locator
9-31		Reserved

Table 3.1

3.3.3 DATA ELEMENT SIZE DESCRIPTOR

The data element size descriptor is represented as a 3-bit size index followed by 0, 8, 16, or 32 bits. The size index is contained in the least significant (low-order) 3 bits of the first byte of the data element header. The size index is encoded as follows.

Size Index	Additional bits	Data Size
0	0	1 byte. Exception: if the data element type is nil, the data size is 0 bytes.
1	0	2 bytes
2	0	4 bytes
3	0	8 bytes
4	0	16 bytes
5	8	The data size is contained in the additional 8 bits, which are interpreted as an unsigned integer.
6	16	The data size is contained in the additional 16 bits, which are interpreted as an unsigned integer.
7	32	The data size is contained in the additional 32 bits, which are interpreted as an unsigned integer.

Table 3.2:

3.4 PROTOCOL DESCRIPTION

SDP is a simple protocol with minimal requirements on the underlying transport. It can function over a reliable packet transport (or even unreliable, if the client implements timeouts and repeats requests as necessary). SDP uses a request/response model where each transaction consists of one request protocol data unit (PDU) and one response PDU. In the case where SDP is used with the Bluetooth L2CAP transport protocol, only one SDP request PDU per connection to a given SDP server may be outstanding at a given instant. In other words, a client must receive a response to each request before issuing another request on the same L2CAP connection. Limiting SDP to sending one unacknowledged request PDU provides a simple form of flow control.

3.4.1 TRANSFER BYTE ORDER

The service discovery protocol transfers multiple-byte fields in standard net-work byte order (Big Endian), with more significant (high-order) bytes being transferred before less-significant (low-order) bytes.

3.4.2 PROTOCOL DATA UNIT FORMAT

Every SDP PDU consists of a PDU header followed by PDU-specific parameters. The header contains three fields: a PDU ID, a Transaction ID, and a ParameterLength. Each of these header fields is described here. Parameters may include a continuation state parameter, described below; PDU-specific parameters for each PDU type are described later in separate PDU descriptions.

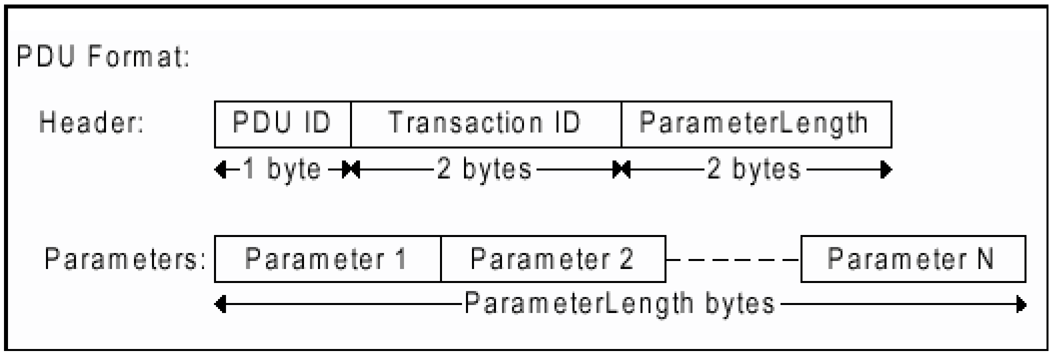
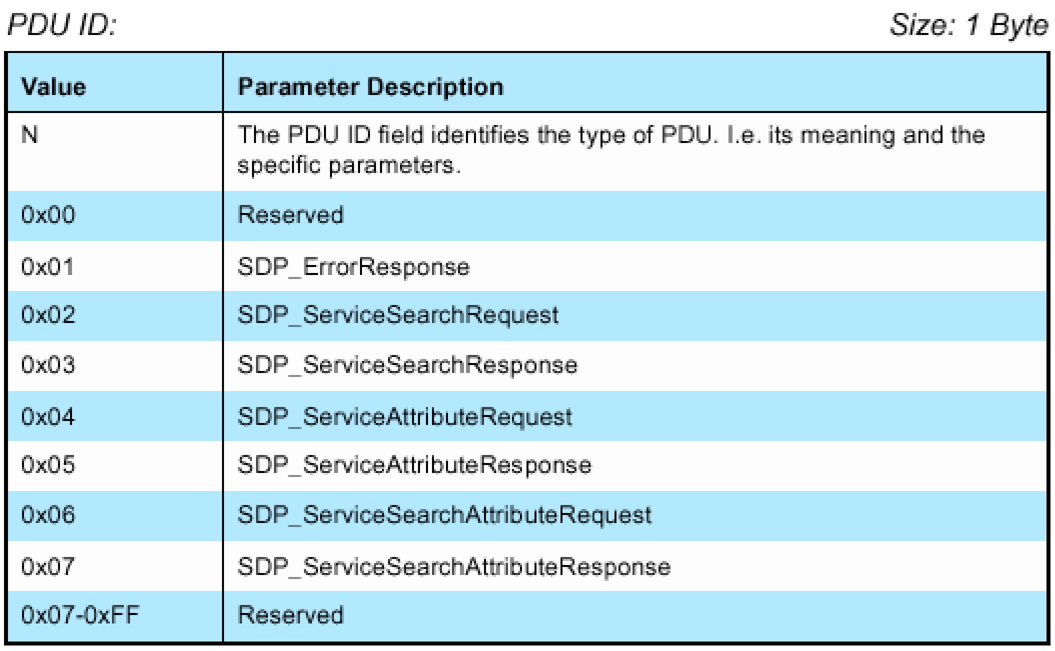
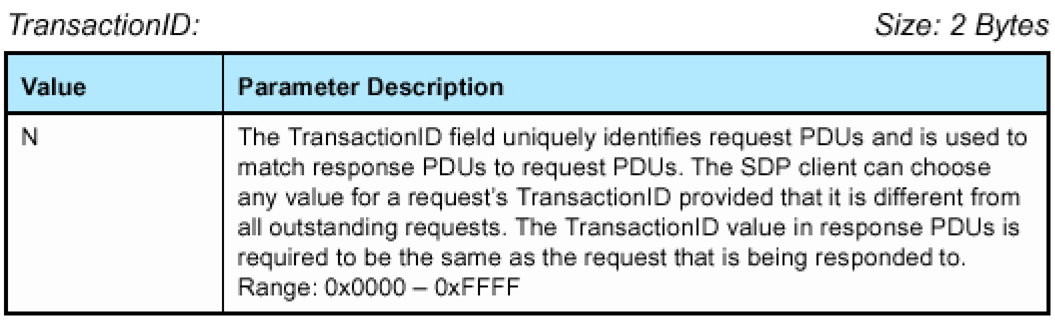


Figure 3.6:



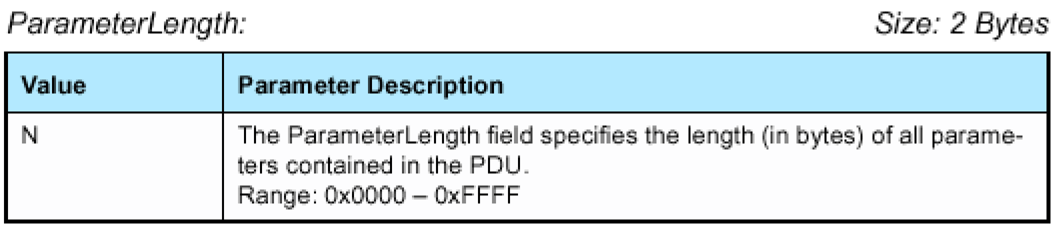
PDU ID: *Size: 1 Byte*

Value	Parameter Description
N	The PDU ID field identifies the type of PDU. I.e. its meaning and the specific parameters.
0x00	Reserved
0x01	SDP_ErrorResponse
0x02	SDP_ServiceSearchRequest
0x03	SDP_ServiceSearchResponse
0x04	SDP_ServiceAttributeRequest
0x05	SDP_ServiceAttributeResponse
0x06	SDP_ServiceSearchAttributeRequest
0x07	SDP_ServiceSearchAttributeResponse
0x07-0xFF	Reserved



TransactionID: *Size: 2 Bytes*

Value	Parameter Description
N	The TransactionID field uniquely identifies request PDUs and is used to match response PDUs to request PDUs. The SDP client can choose any value for a request's TransactionID provided that it is different from all outstanding requests. The TransactionID value in response PDUs is required to be the same as the request that is being responded to. Range: 0x0000 – 0xFFFF



ParameterLength: *Size: 2 Bytes*

Value	Parameter Description
N	The ParameterLength field specifies the length (in bytes) of all parameters contained in the PDU. Range: 0x0000 – 0xFFFF

3.4.3 PARTIAL RESPONSES AND CONTINUATION STATE

Some SDP requests may require responses that are larger than can fit in a single response PDU. In this case, the SDP server will generate a partial response along with a continuation state parameter. The continuation state parameter can be supplied by the client in a subsequent request to retrieve the next portion of the complete response. The continuation state parameter is a variable length field whose first byte contains the number of additional bytes of continuation information in the field. The format of the continuation information is not standardized among SDP servers. Each continuation state parameter is meaningful only to the SDP server that generated it.

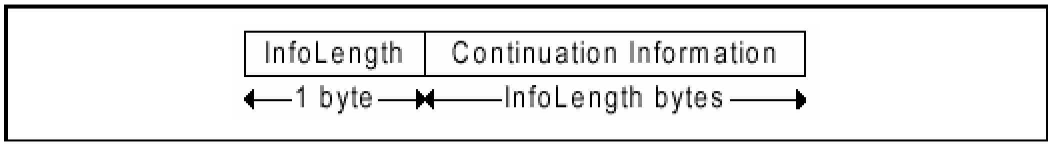


Figure 3.7: Continuation State Format

After a client receives a partial response and the accompanying continuation state parameter, it can re-issue the original request (with a new transaction ID) and include the continuation state in the new request indicating to the server that the remainder of the original response is desired. The maximum allowable value of the InfoLength field is 16 (0x10). An SDP server can split a response at any arbitrary boundary when it generates a partial response. The SDP server may select the boundary based on the contents of the reply, but is not required to do so. After a client receives a partial response and the accompanying continuation state parameter, it can re-issue the original request (with a new transaction ID) and include the continuation state in the new request indicating to the server that the remainder of the original response is desired. The maximum allowable value of the InfoLength field is 16 (0x10). Note that an SDP server can split a response at any arbitrary boundary when it generates a partial response. The SDP server may select the boundary based on the contents of the reply, but is not required to do so.

3.4.4 ERROR HANDLING

Each transaction consists of a request and a response PDU. Generally, each type of request PDU has a corresponding type of response PDU. However, if the server determines that a request is improperly formatted or for any reason the server cannot respond with the appropriate PDU type, it will respond with an SDP_ErrorResponse PDU.

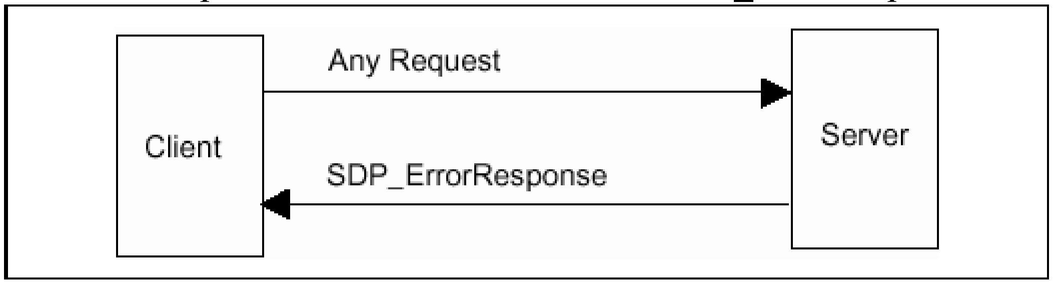


Figure 3.8:

3.4.4.1 SDP_ErrorResponse PDU

PDU Type	PDU ID	Parameters
SDP_ErrorResponse	0x01	ErrorCode, ErrorInfo

Description:

The SDP server generates this PDU type in response to an improperly formatted request PDU or when the SDP server, for whatever reason, cannot generate an appropriate response PDU.

PDU Parameters:

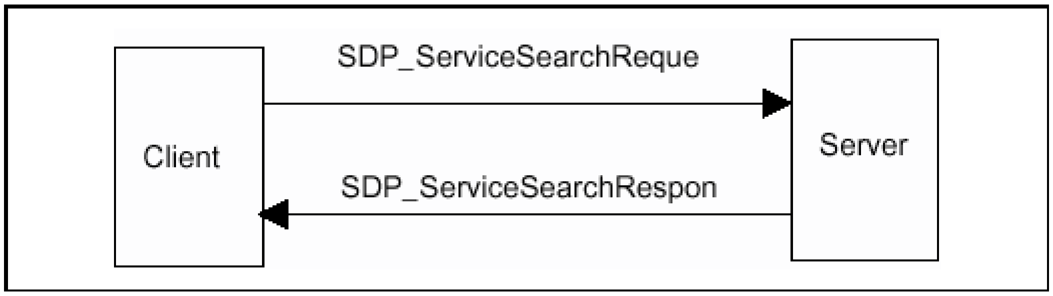
ErrorCode: *Size: 2 Bytes*

Value	Parameter Description
N	The ErrorCode identifies the reason that an SDP_ErrorResponse PDU was generated.
0x0000	Reserved
0x0001	Invalid/unsupported SDP version
0x0002	Invalid Service Record Handle
0x0003	Invalid request syntax
0x0004	Invalid PDU Size
0x0005	Invalid Continuation State
0x0006	Insufficient Resources to satisfy Request
0x0007-0xFFFF	Reserved

ErrorInfo: *Size: N Bytes*

Value	Parameter Description
Error-specific	ErrorInfo is an ErrorCode-specific parameter. Its interpretation depends on the ErrorCode parameter. The currently defined ErrorCode values do not specify the format of an ErrorInfo field.

3.4.5 SERVICESEARCH TRANSACTION



3.4.5.1 SDP_ServiceSearchRequest PDU

PDU Type	PDU ID	Parameters
SDP_ServiceSearchRequest	0x02	ServiceSearchPattern, MaximumServiceRecordCount, ContinuationState

Description:

The SDP client generates an SDP_ServiceSearchRequest to locate service records that match the service search pattern given as the first parameter of the PDU. Upon receipt of this request, the SDP server will examine its service record data base and return an SDP_ServiceSearchResponse containing the service record handles of service records that match the given service search pattern. No mechanism is provided to request information for all service records.

PDU Parameters:

ServiceSearchPattern: *Size: Varies*

Value	Parameter Description
Data Element Sequence	The ServiceSearchPattern is a data element sequence where each element in the sequence is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in the sequence is 12*. The list of UUIDs constitutes a service search pattern.

*. The value of 12 has been selected as a compromise between the scope of a service search and the size of a search request PDU. It is not expected that more than 12 UUIDs will be useful in a service search pattern.

MaximumServiceRecordCount: *Size: 2 Bytes*

Value	Parameter Description
N	MaximumServiceRecordCount is a 16-bit count specifying the maximum number of service record handles to be returned in the response(s) to this request. The SDP server should not return more handles than this value specifies. If more than N service records match the request, the SDP server determines which matching service record handles to return in the response(s). Range: 0x0001-0xFFFF

ContinuationState: *Size: 1 to 17 Bytes*

Value	**Parameter Description**
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation state information that were returned in a previous response from the server. N is required to be less than or equal to 16. If no continuation state is to be provided in the request, N is set to 0.

3.4.5.2 SDP_ServiceSearchResponse PDU

PDU Type	**PDU ID**	**Parameters**
SDP_ServiceSearchResponse	0x03	TotalServiceRecordCount, CurrentServiceRecordCount, ServiceRecordHandleList, ContinuationState

Description:

The SDP server generates an SDP_ServiceSearchResponse upon receipt of a valid SDP_ServiceSearchRequest. The response contains a list of service record handles for service records that match the service search pattern given in the request. Note that if a partial response is generated, it must contain an integral number of complete service record handles; a service record handle value may not be split across multiple PDUs.

PDU Parameters:

TotalServiceRecordCount: *Size: 2 Bytes*

Value	Parameter Description
N	The TotalServiceRecordCount is an integer containing the number of service records that match the requested service search pattern. If no service records match the requested service search pattern, this parameter is set to 0. N should never be larger than the MaximumServiceRecordCount value specified in the SDP_ServiceSearchRequest. When multiple partial responses are used, each partial response contains the same value for TotalServiceRecordCount. Range: 0x0000-0xFFFF

CurrentServiceRecordCount: *Size: 2 Bytes*

Value	Parameter Description
N	The CurrentServiceRecordCount is an integer indicating the number of service record handles that are contained in the next parameter. If no service records match the requested service search pattern, this parameter is set to 0. N should never be larger than the TotalServiceRecordCount value specified in the current response. Range: 0x0000-0xFFFF

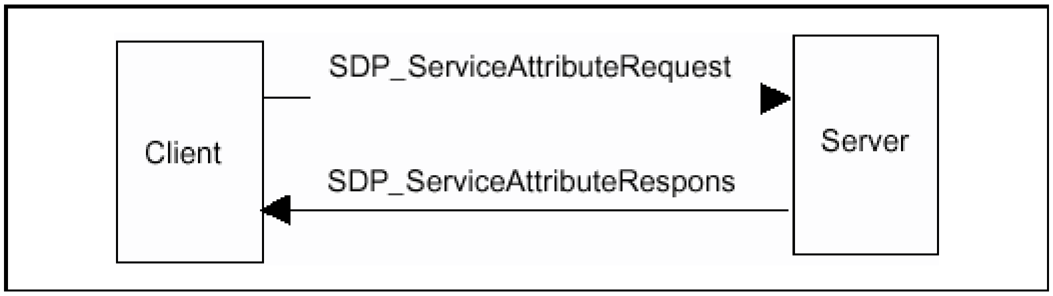
ServiceRecordHandleList: *Size: (CurrentServiceRecordCount*4) Bytes*

Value	Parameter Description
List of 32-bit handles	The ServiceRecordHandleList contains a list of service record handles. The number of handles in the list is given in the CurrentServiceRecordCount parameter. Each of the handles in the list refers to a service record that matches the requested service search pattern. Note that this list of service record handles does not have the format of a data element. It contains no header fields, only the 32-bit service record handles.

ContinuationState: *Size: 1 to 17 Bytes*

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation information. If the current response is complete, this parameter consists of a single byte with the value 0. If a partial response is contained in the PDU, the ContinuationState parameter may be supplied in a subsequent request to retrieve the remainder of the response.

3.4.6 SERVICEATTRIBUTE TRANSACTION



3.4.6.1 SDP_ServiceAttributeRequest PDU

PDU Type	PDU ID	Parameters
SDP_ServiceAttributeRequest	0x04	ServiceRecordHandle, MaximumAttributeByteCount, AttributeIDList, ContinuationState

Description:

The SDP client generates an SDP_ServiceAttributeRequest to retrieve specified attribute values from a specific service record. The service record handle of the desired service record and a list of desired attribute IDs to be retrieved from that service record are supplied as parameters.

Command Parameters:

ServiceRecordHandle: *Size: 4 Bytes*

Value	Parameter Description
32-bit handle	The ServiceRecordHandle parameter specifies the service record from which attribute values are to be retrieved. The handle is obtained via a previous SDP_ServiceSearch transaction.

MaximumAttributeByteCount: *Size: 2 Bytes*

Value	Parameter Description
N	MaximumAttributeByteCount specifies the maximum number of bytes of attribute data to be returned in the response to this request. The SDP server should not return more than N bytes of attribute data in the response PDU. If the requested attributes require more than N bytes, the SDP server determines how to segment the list. In this case the client may request each successive segment by issuing a request containing the continuation state that was returned in the previous response PDU. Note that in the case where multiple response PDUs are needed to return the attribute data, MaximumAttributeByteCount specifies the maximum size of the portion of the attribute data contained in each response PDU. Range: 0x0007-0xFFFF

AttributeIDList: *Size: Varies*

Value	Parameter Description
Data Element Sequence	The AttributeIDList is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. Each attribute ID is encoded as a 16-bit unsigned integer data element. Each attribute ID range is encoded as a 32-bit unsigned integer data element, where the high order 16 bits are interpreted as the beginning attribute ID of the range and the low order 16 bits are interpreted as the ending attribute ID of the range. The attribute IDs contained in the AttributeIDList must be listed in ascending order without duplication of any attribute ID values. Note that all attributes may be requested by specifying a range of 0x0000-0xFFFF.

ContinuationState: *Size: 1 to 17 Bytes*

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation state information that were returned in a previous response from the server. N is required to be less than or equal to 16. If no continuation state is to be provided in the request, N is set to 0.

3.4.6.2 SDP_ServiceAttributeResponse PDU

PDU Type	PDU ID	Parameters
SDP_ServiceAttributeResponse	0x05	AttributeListByteCount, AttributeList, ContinuationState

Description:

The SDP server generates an SDP_ServiceAttributeResponse upon receipt of a valid SDP_ServiceAttributeRequest. The response contains a list of attributes (both attribute ID and attribute value) from the requested service record.

PDU Parameters:

AttributeListByteCount: *Size: 2 Bytes*

Value	Parameter Description
N	The AttributeListByteCount contains a count of the number of bytes in the AttributeList parameter. N must never be larger than the Maximu-mAttributeByteCount value specified in the SDP_ServiceAttributeRequest. Range: 0x0002-0xFFFF

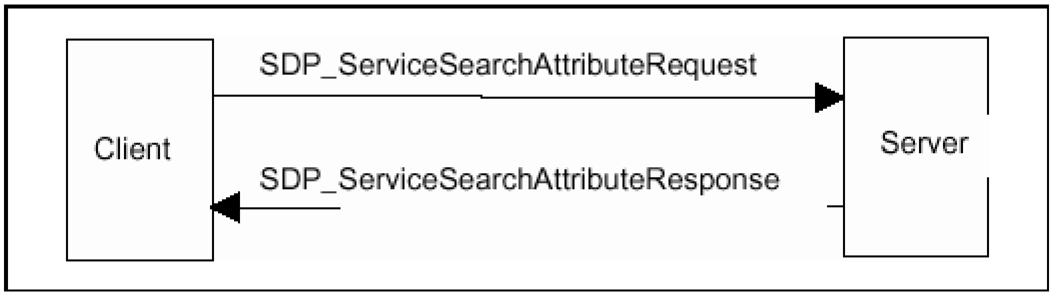
AttributeList: *Size: AttributeListByteCount*

Value	Parameter Description
Data Element Sequence	The AttributeList is a data element sequence containing attribute IDs and attribute values. The first element in the sequence contains the attribute ID of the first attribute to be returned. The second element in the sequence contains the corresponding attribute value. Successive pairs of elements in the list contain additional attribute ID and value pairs. Only attributes that have non-null values within the service record and whose attribute IDs were specified in the SDP_ServiceAttributeRequest are contained in the AttributeList. Neither an attribute ID nor an attribute value is placed in the AttributeList for attributes in the service record that have no value. The attributes are listed in ascending order of attribute ID value.

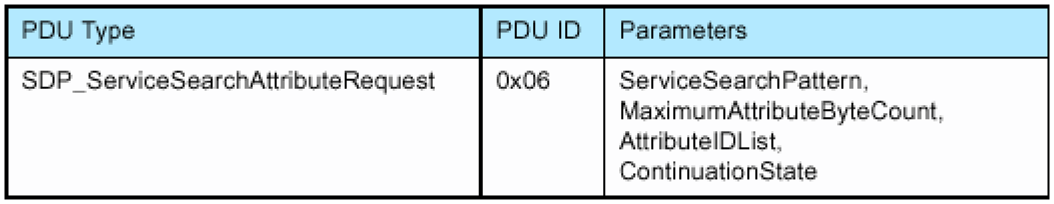
ContinuationState: *Size: 1 to 17 Bytes*

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continua-tion information. If the current response is complete, this parameter consists of a single byte with the value 0. If a partial response is given, the ContinuationState parameter may be supplied in a subsequent request to retrieve the remainder of the response.

3.4.7 SERVICESEARCHATTRIBUTE TRANSACTION


SDP_ServiceSearchAttributeRequest
Client
Server
SDP_ServiceSearchAttributeResponse

3.4.7.1 SDP_ServiceSearchAttributeRequest PDU



PDU Type	PDU ID	Parameters
SDP_ServiceSearchAttributeRequest	0x06	ServiceSearchPattern, MaximumAttributeByteCount, AttributeIDList, ContinuationState

Description:

The SDP_ServiceSearchAttributeRequest transaction combines the capabilities of the SDP_ServiceSearchRequest and the SDP_ServiceAttributeRequest into a single request. As parameters, it contains both a service search pattern and a list of attributes to be retrieved from service records that match the service search pattern. The SDP_ServiceSearchAttributeRequest and its response are more complex and may require more bytes than separate SDP_ServiceSearch and SDP_ServiceAttribute transactions. However, using SDP_ServiceSearchAttributeRequest may reduce the total number of SDP transactions, particularly when retrieving multiple service records. The service record handle for each service record is contained in the ServiceRecordHandle attribute of that service and may be requested along with other attributes.

PDU Parameters:

ServiceSearchPattern: *Size: Varies*

Value	Parameter Description
Data Element Sequence	The ServiceSearchPattern is a data element sequence where each element in the sequence is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in the sequence is 12[*]. The list of UUIDs constitutes a service search pattern.

*. The value of 12 has been selected as a compromise between the scope of a service search and the size of a search request PDU. It is not expected that more than 12 UUIDs will be useful in a service search pattern.

MaximumAttributeByteCount: *Size: 2 Bytes*

Value	Parameter Description
N	MaximumAttributeByteCount specifies the maximum number of bytes of attribute data to be returned in the response to this request. The SDP server should not return more than N bytes of attribute data in the response PDU. If the requested attributes require more than N bytes, the SDP server determines how to segment the list. In this case the client may request each successive segment by issuing a request containing the continuation state that was returned in the previous response PDU. Note that in the case where multiple response PDUs are needed to return the attribute data, MaximumAttributeByteCount specifies the maximum size of the portion of the attribute data contained in each response PDU. Range: 0x0007-0xFFFF

AttributeIDList: *Size: Varies*

Value	Parameter Description
Data Element Sequence	The AttributeIDList is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. Each attribute ID is encoded as a 16-bit unsigned integer data element. Each attribute ID range is encoded as a 32-bit unsigned integer data element, where the high order 16 bits are interpreted as the beginning attribute ID of the range and the low order 16 bits are interpreted as the ending attribute ID of the range. The attribute IDs contained in the AttributeIDList must be listed in ascending order without duplication of any attribute ID values. Note that all attributes may be requested by specifying a range of 0x0000-0xFFFF.

ContinuationState: *Size: 1 to 17 Bytes*

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation state information that were returned in a previous response from the server. N is required to be less than or equal to 16. If no continuation state is to be provided in the request, N is set to 0.

3.4.7.2 SDP_ServiceSearchAttributeResponse PDU

PDU Type	PDU ID	Parameters
SDP_ServiceSearchAttributeResponse	0x07	AttributeListsByteCount, AttributeLists, ContinuationState

Description:

The SDP server generates an SDP_ServiceSearchAttributeResponse upon receipt of a valid SDP_ServiceSearchAttributeRequest. The response contains a list of attributes (both attribute ID and attribute value) from the service records that match the requested service search pattern.

PDU Parameters:

AttributeListsByteCount: *Size: 2 Bytes*

Value	Parameter Description
N	The AttributeListsByteCount contains a count of the number of bytes in the AttributeLists parameter. N must never be larger than the MaximumAttributeByteCount value specified in the SDP_ServiceSearchAttributeRequest. Range: 0x0002-0xFFFF

AttributeLists: *Size: Varies*

Value	Parameter Description
Data Element Sequence	The AttributeLists is a data element sequence where each element in turn is a data element sequence representing an attribute list. Each attribute list contains attribute IDs and attribute values from one service record. The first element in each attribute list contains the attribute ID of the first attribute to be returned for that service record. The second element in each attribute list contains the corresponding attribute value. Successive pairs of elements in each attribute list contain additional attribute ID and value pairs. Only attributes that have non-null values within the service record and whose attribute IDs were specified in the SDP_ServiceSearchAttributeRequest are contained in the AttributeLists. Neither an attribute ID nor attribute value is placed in AttributeLists for attributes in the service record that have no value. Within each attribute list, the attributes are listed in ascending order of attribute ID value.

ContinuationState: *Size: 1 to 17 Bytes*

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation information. If the current response is complete, this parameter consists of a single byte with the value 0. If a partial response is given, the ContinuationState parameter may be supplied in a subsequent request to retrieve the remainder of the response.

3.5 SERVICE ATTRIBUTE DEFINITIONS

3.5.1 UNIVERSAL ATTRIBUTE DEFINITIONS

Universal attributes are those service attributes whose definitions are common to all service records. Note that this does not mean that every service record must contain values for all of these service attributes. However, if a service record has a service attribute with an attribute ID allocated to a universal attribute, the attribute value must conform to the universal attribute's definition.

Only two attributes are required to exist in every service record instance. They are the ServiceRecordHandle (attribute ID 0x0000) and the ServiceClassIDList (attribute ID 0x0001). All other service attributes are optional within a service record.

3.5.1.1 ServiceRecordHandle Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceRecordHandle	0x0000	32-bit unsigned integer

Description:

A service record handle is a 32-bit number that uniquely identifies each service record within an SDP server. It is important to note that, in general, each handle is unique only within each SDP server. If SDP server S1 and SDP server S2 both contain identical service records (representing the same service), the service record handles used to reference these identical service records are completely independent. The handle used to reference the service on S1 will, in general, be meaningless if presented to S2.

3.5.1.2 ServiceClassIDList Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceClassIDList	0x0001	Data Element Sequence

Description:

The ServiceClassIDList attribute consists of a data element sequence in which each data element is a UUID representing the service classes that a given service record conforms to. The UUIDs are listed in order from the most specific class to the most general class. The ServiceClassIDList must contain at least one service class UUID.

3.5.1.3 ServiceName Attribute

Attribute Name	Attribute ID Offset	Attribute Value Type
ServiceName	0x0000	String

Description:

The ServiceName attribute is a string containing the name of the service represented by a service record. It should be brief and suitable for display with an Icon representing the service. The offset 0x0000 must be added to the attribute ID base (contained in the LanguageBaseAttributeIDList attribute) in order to compute the attribute ID for this attribute.

3.5.1.4 ServiceDescription Attribute

Attribute Name	Attribute ID Offset	Attribute Value Type
ServiceDescription	0x0001	String

Description:

This attribute is a string containing a brief description of the service. It should be less than 200 characters in length. The offset 0x0001 must be added to the attribute ID base (contained in the LanguageBaseAttributeIDList attribute) in order to compute the attribute ID for this attribute.

3.5.1.5 ProviderName Attribute

Attribute Name	Attribute ID Offset	Attribute Value Type
ServiceDescription	0x0001	String

Description:

This attribute is a string containing the name of the person or organization providing the service. The offset 0x0002 must be added to the attribute ID base (contained in the LanguageBaseAttributeIDList attribute) in order to compute the attribute ID for this attribute.

Bluetooth Service Discovery Protocol (SDP) addresses service discovery specifically for the Bluetooth environment. It is optimized for the highly dynamic nature of Bluetooth communications. SDP focuses primarily on discovering services available from or through Bluetooth devices. SDP does not define methods for accessing services; once services are discovered with SDP, they can be accessed in various ways, depending upon the service. This might include the use of other service discovery and access mechanisms such as those mentioned above; SDP provides a means for other protocols to be used along with SDP in those environments where this can be beneficial. While SDP can coexist with other service discovery protocols, it does not require them. In Bluetooth environments, services can be discovered using SDP and can be accessed using other protocols defined by Bluetooth.

CHAPTER 4

BLUETOOTH RFCOMM PROTOCOL

4.1 INTRODUCTION

The RFCOMM protocol provides emulation of serial ports over the L2CAP protocol. The protocol is based on the ETSI standard TS 07.10.

4.1.1 OVERVIEW

RFCOMM is a simple transport protocol, with additional provisions for emulating the 9 circuits of RS-232 (EIATIA-232-E) serial ports. The RFCOMM protocol supports up to 60 simultaneous connections between two Bluetooth devices.

4.1.2 DEVICE TYPES

For the purposes of RFCOMM, a complete communication path involves two applications running on different devices (the communication endpoints) with a communication segment between them. Figure 4.1hows the complete communication path. (In this context, the term *application* may mean other things than end-user application; e.g. higher layer protocols or other services acting on behalf of end-user applications.)

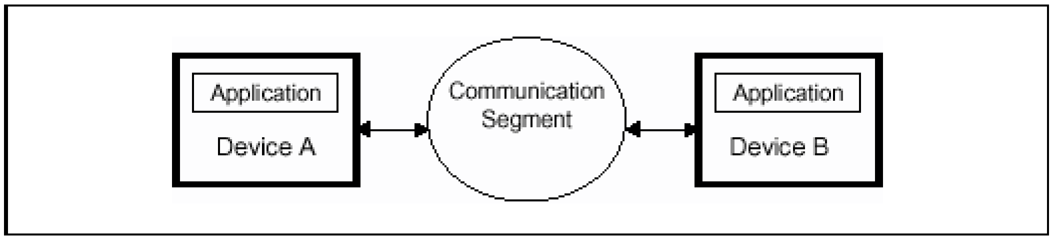


Figure 4.1: RFCOMM Communication Segment

RFCOMM is intended to cover applications that make use of the serial ports of the devices in which they reside. In the simple configuration, the communication segment is a Bluetooth link from one device to another (direct connect),see Figure 4.2. Where the communication segment is another network, Blue-tooth wireless technology is used for the path between the device and a net-work connection device like a modem. RFCOMM is only concerned with the connection between the devices in the direct connect case, or between the device and a modem in the network case. RFCOMM can support other configurations, such as modules that communicate via Bluetooth wireless technology on one side and provide a wired interface on the other side. These devices are not really modems but offer a similar service. They are therefore not explicitly discussed here. Basically two device types exist that the RFCOMM accommodates.

Type 1

Devices are communication end points such as computers and printers.

Type 2

Devices are those that are part of the communication segment; e.g. modems. Though RFCOMM does not make a distinction between these two device types in the protocol, accommodating both types of devices impacts the RFCOMM protocol.

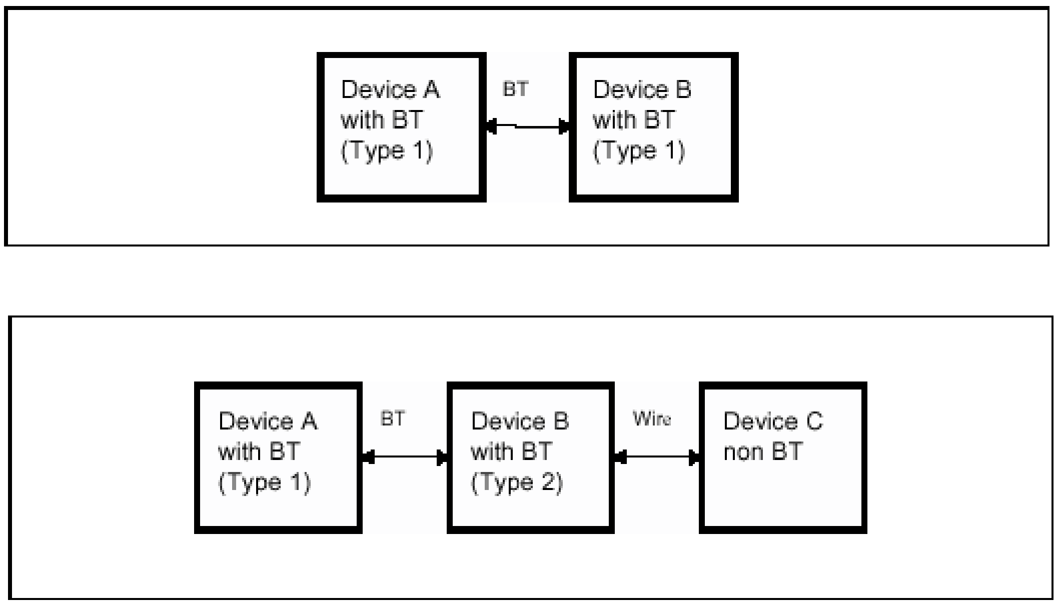

Device A with BT (Type 1)
BT
Device B with BT (Type 1)
Device A with BT (Type 1)
BT
Device B with BT (Type 2)
Wire
Device C non BT

Figure 4.2 RFCOMM interaction in Bluetooth devices

The information transferred between two RFCOMM entities has been defined to support both type 1 and type 2 devices. Some information is only needed by type 2 devices while other information is intended to be used by both. In the protocol, no distinction is made between type 1 and type 2. It is therefore up to the RFCOMM implementers to determine if the information passed in the RFCOMM protocol is of use to the implementation. Since the device is not aware of the type of the other device in the communication path, each must pass on all available information specified by the protocol.

4.1.3 BYTE ORDERING

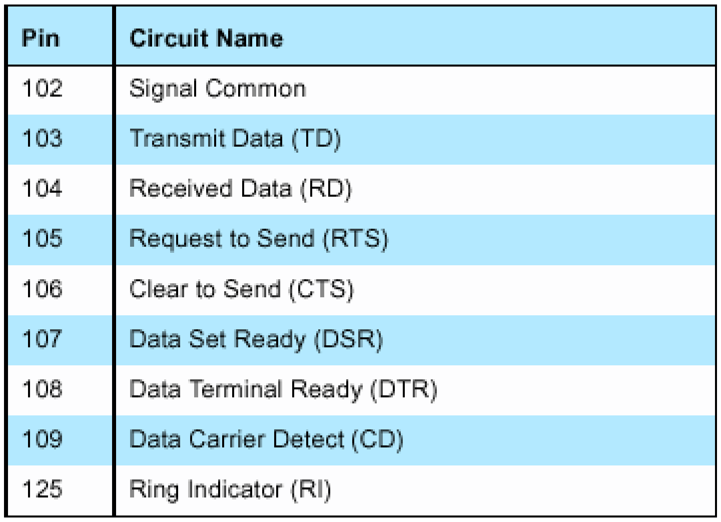
This implementation uses the same byte ordering as the TS 07.10 specification; i.e. all binary numbers are in Least Significant Bit to Most Significant Bit order, reading from left to right.

4.2 RFCOMM SERVICE OVERVIEW

RFCOMM emulates RS-232 (EIATIA-232-E) serial ports. The emulation includes transfer of the state of the non-data circuits. RFCOMM has a built-in scheme for null modem emulation. In the event that a baud rate is set for a particular port through the RFCOMM service interface, that will not affect the actual data throughput in RFCOMM; i.e. RFCOMM does not incur artificial rate limitation or pacing. However, if either device is a type 2 device (relays data onto other media), or if data pacing is done above the RFCOMM service interface in either or both ends, actual throughput will, on an average, reflect the baud rate setting. RFCOMM supports emulation of multiple serial ports between two devices and also emulation of serial ports between multiple devices.

4.2.1 RS-232 CONTROL SIGNALS

RFCOMM emulates the 9 circuits of an RS-232 interface. The circuits are listed below.



Pin	Circuit Name
102	Signal Common
103	Transmit Data (TD)
104	Received Data (RD)
105	Request to Send (RTS)
106	Clear to Send (CTS)
107	Data Set Ready (DSR)
108	Data Terminal Ready (DTR)
109	Data Carrier Detect (CD)
125	Ring Indicator (RI)

Table 4.1: Emulated RS-232 circuits in RFCOMM

4.2.2 MULTIPLE EMULATED SERIAL PORTS

4.2.2.1 Multiple Emulated Serial Ports between two Devices

Two Bluetooth devices using RFCOMM in their communication may open multiple emulated serial ports. RFCOMM supports up to 60 open emulated ports. 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 its usable value range is 2…61; in TS 07.10, DLCI 0 is the dedicated control channel, DLCI 1 is unusable due to the concept of Server Channels, and DLCI 62-63 is reserved. The DLCI is unique within one RFCOMM session between two devices. To account for the fact that both client and server applications may reside on both sides of an RFCOMM session, with clients on either side making connections independent of each other, the DLCI value space is divided between the two communicating devices using the concept of RFCOMM server channels.

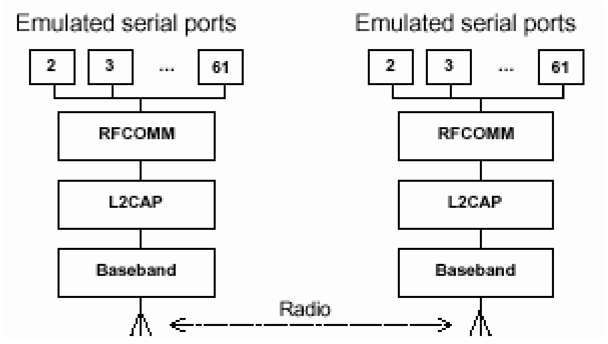


Figure 4.3: Multiple Emulated Serial Ports.

4.2.2.2 Multiple Emulated Serial Ports and Multiple Bluetooth Devices

If a Bluetooth device supports multiple emulated serial ports and the connections are allowed to have endpoints in different Bluetooth devices, then the RFCOMM entity can run multiple TS 07.10 multiplexer sessions. Each multiplexer session in the figure uses its own L2CAP channel ID (CID).

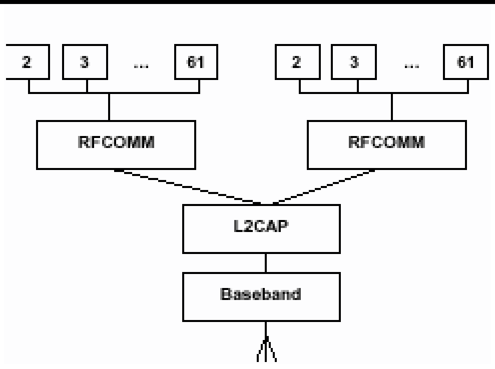


Figure 4.4: Emulating serial ports coming from two Bluetooth devices.

4.3 SERVICE INTERFACE DESCRIPTION

RFCOMM is intended to define a protocol that is used to emulate serial ports. Hence RFCOMM is part of a port driver which includes a serial port emulation entity.

4.3.1 SERVICE DEFINITION MODEL

The figure below shows a model of how RFCOMM fits into a typical system. This figure represents the RFCOMM reference model.

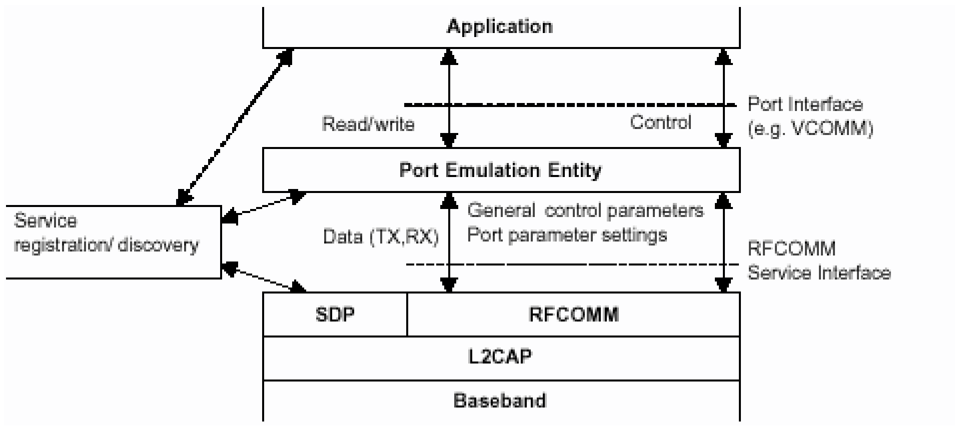


Figure 4.5: RFCOMM reference model

The elements of the RFCOMM reference model are described below.

Element	Description
Application	Applications that utilize a serial port communication interface
Port Emulation Entity	The port emulation entity maps a system-specific communication interface (API) to the RFCOMM services. The port emulation entity plus RFCOMM make up a port driver
RFCOMM	Provides a transparent data stream and control channel over an L2CAP channel. Multiplexes multiple emulated serial ports
Service Registration/ Discovery	Server applications register here on local device, and it provides services for client applications to discover how to reach server applications on other devices
L2CAP	Protocol multiplexing, SAR
Baseband	Baseband protocols defined by Bluetooth Specification

4.4 INTERACTION WITH OTHER ENTITIES

4.4.1 PORT EMULATION AND PORT PROXY ENTITIES

This section defines how the RFCOMM protocol should be used to emulate serial ports. Figure 4.6 shows the two device types that the RFCOMM protocol supports.

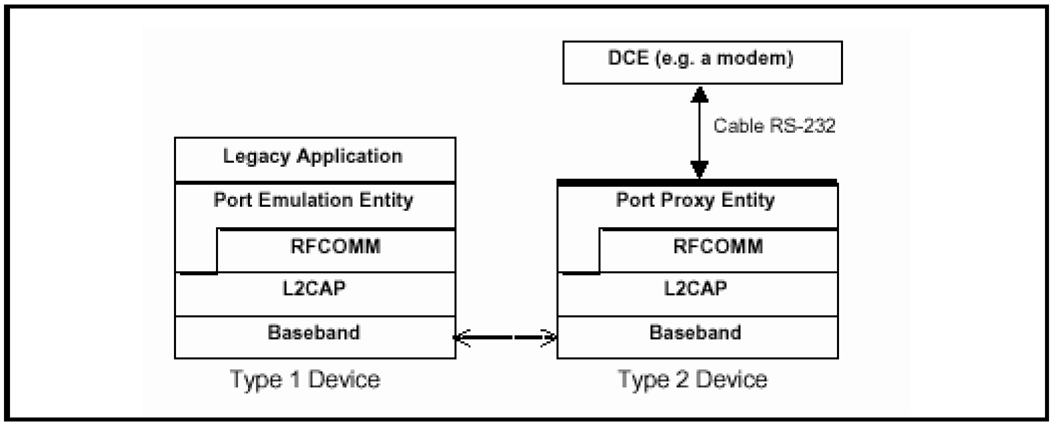


Figure 4.6: The RFCOMM communication model

Type 1 devices are communication endpoints such as computers and printers.
Type 2 devices are part of a communication segment; e.g. modems.

4.4.1.1 Port Emulation Entity

The port emulation entity maps a system specific communication interface (API) to the RFCOMM services.

4.4.1.2 Port Proxy Entity

The port proxy entity relays data from RFCOMM to an external RS-232 inter-face linked to a DCE. The communications parameters of the RS-232 interface are set according to received RPN commands.

4.4.2 SERVICE REGISTRATION AND DISCOVERY

Registration of individual applications or services, along with the information needed to reach those (i.e. the RFCOMM Server Channel) is the responsibility of each application respectively (or possibly a Bluetooth configuration application acting on behalf of legacy applications not directly aware of Bluetooth). Below is description of developing service records for a given service or profile using RFCOMM. It illustrates the inclusion of the ServiceClassList with a single service class, a ProtocolDescriptor List with two protocols (although there may be more protocols on top of RFCOMM). One other universal attribute namely (ServiceName) is also used in the process. For each service running on top of RFCOMM, appropriate SDP-defined universal attributes and/or service-specific attributes will apply. The attributes that a client application needs (at a minimum) to connect to a service on top of RFCOMM are the ServiceClassIDList and the ProtocolDescriptorList (corresponding to the shaded rows in the table below).

Item	Definition	Type/Size	Value	Attribute ID
ServiceClassIDList			Note1	0x0001
ServiceClass0	Note5	UUID/32-bit	Note1	
ProtocolDescriptorList				0x0004
Protocol0	L2CAP	UUID/32-bit	L2CAP /Note1	
Protocol1	RFCOMM	UUID/32-bit	RFCOMM /Note1	
ProtocolSpecificParameter0	Server Channel	Uint8	N = server channel #	
[other protocols]		UUID/32-bit	Note1	
[other protocol-specific parameters]	Note3	Note3	Note3	
ServiceName	Displayable text name	DataElement/String	'Example service'	Note2
[other universal attributes as appropriate for this service]	Note4	Note4	Note4	Note4
[service-specific attributes]	Note3	Note3	Note3	Note3

4.4.3 Reliability

RFCOMM uses the services of L2CAP to establish L2CAP channels to RFCOMM entities on other devices. An L2CAP channel is used for the RFCOMM/TS 07.10 multiplexer session.

RFCOMM requires L2CAP to provide channels with maximum reliability, to ensure that all frames are delivered in order, and without duplicates. Should an L2CAP channel fail to provide this, RFCOMM will expect a link loss notification, which should be handled by RFCOMM.

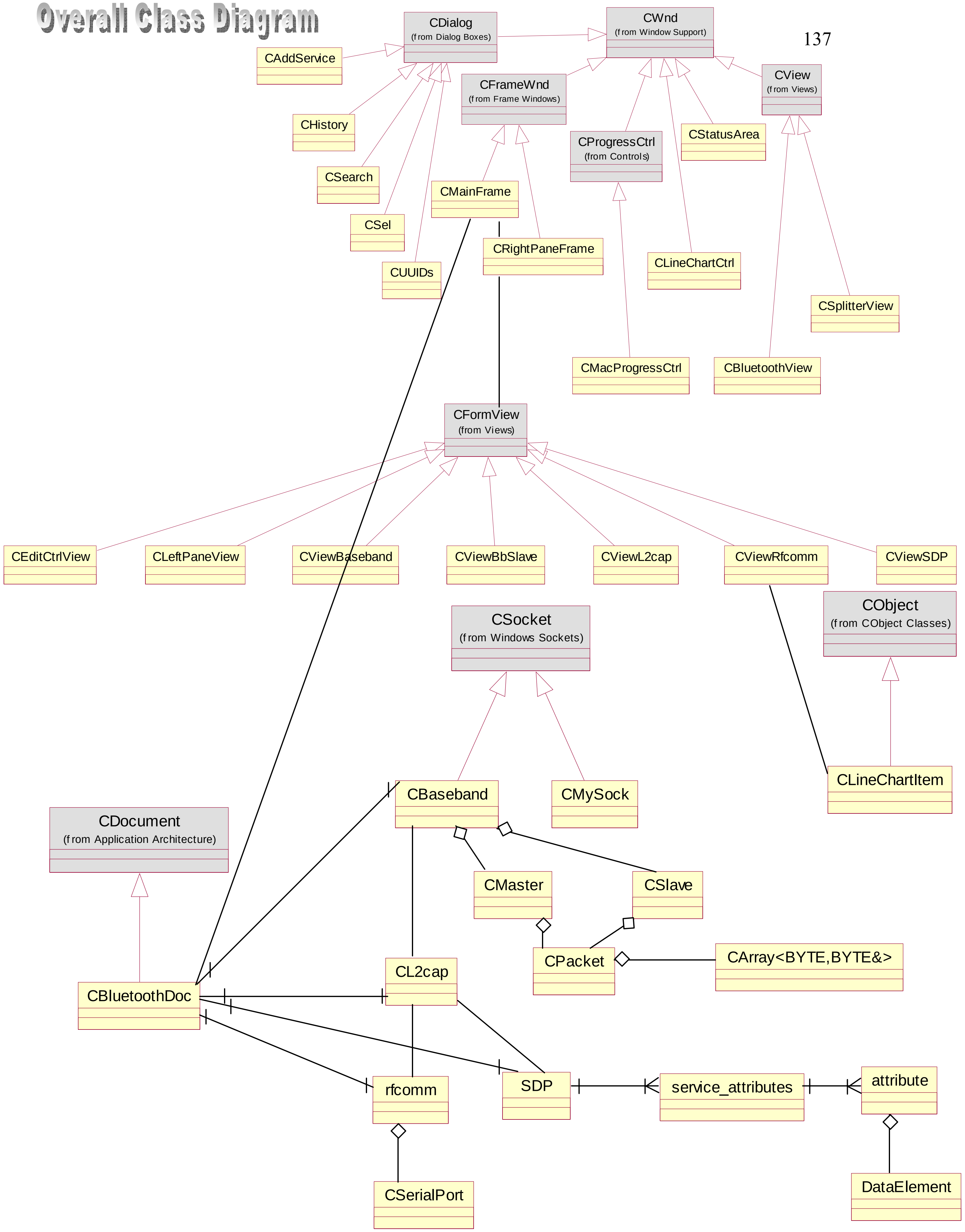
For the purposes of RFCOMM, a complete communication path involves two applications running on different devices (the communication endpoints) with a communication segment between them.

CHAPTER 5

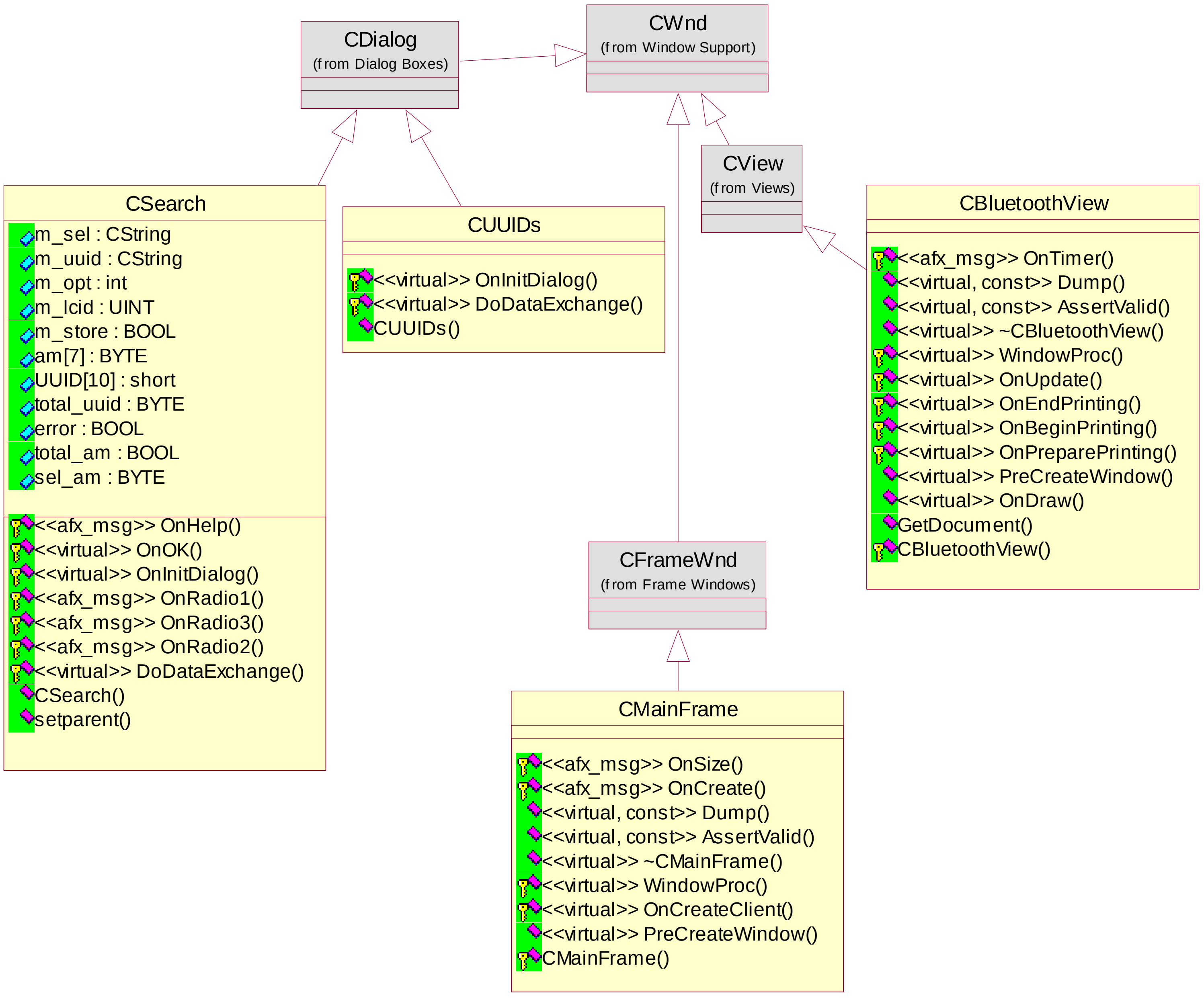
CLASS DIAGRAMS

DETAILED STRUCTURE
OF
SOFTWARE IMPLMENTATION

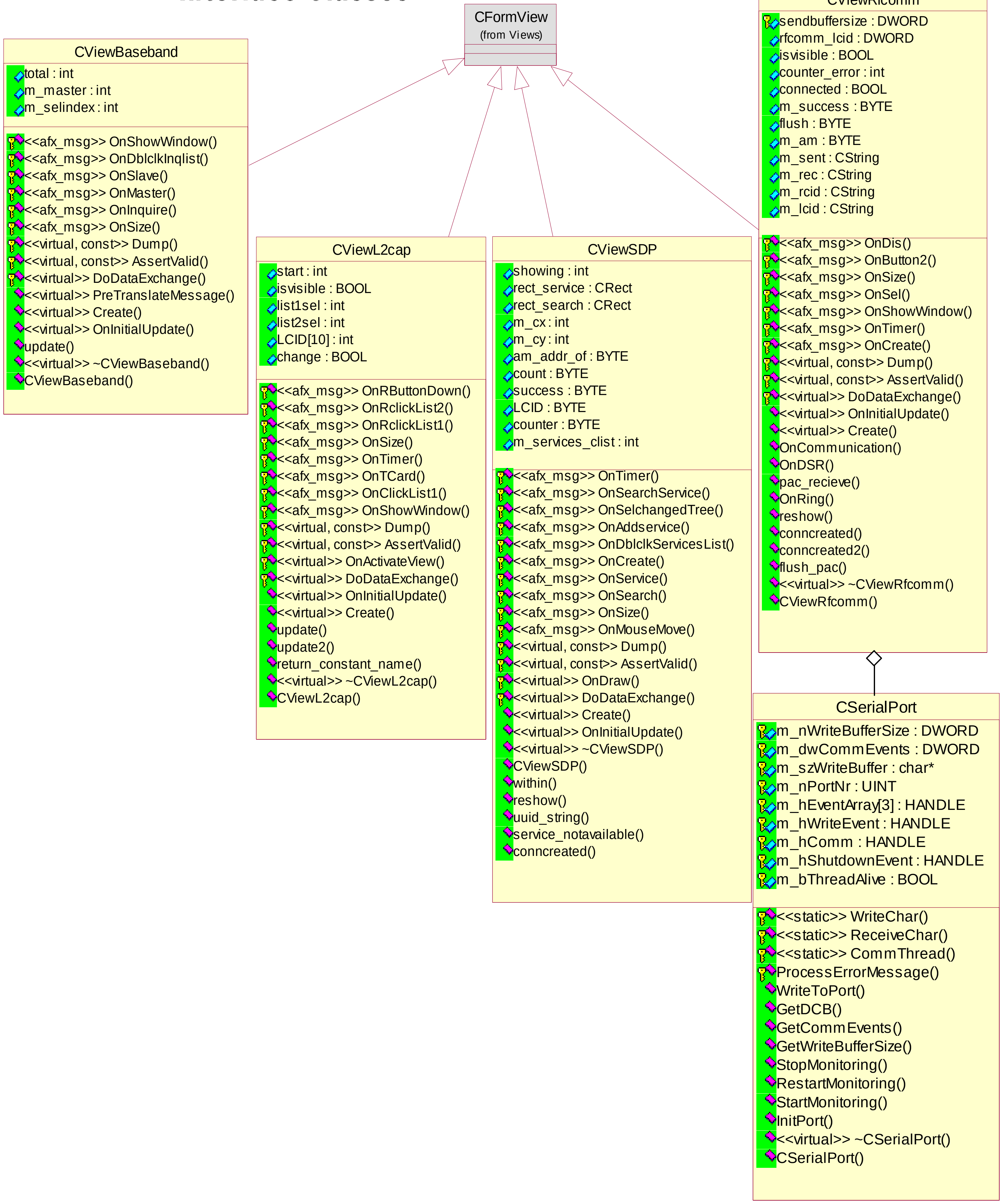
Overall Class Diagram



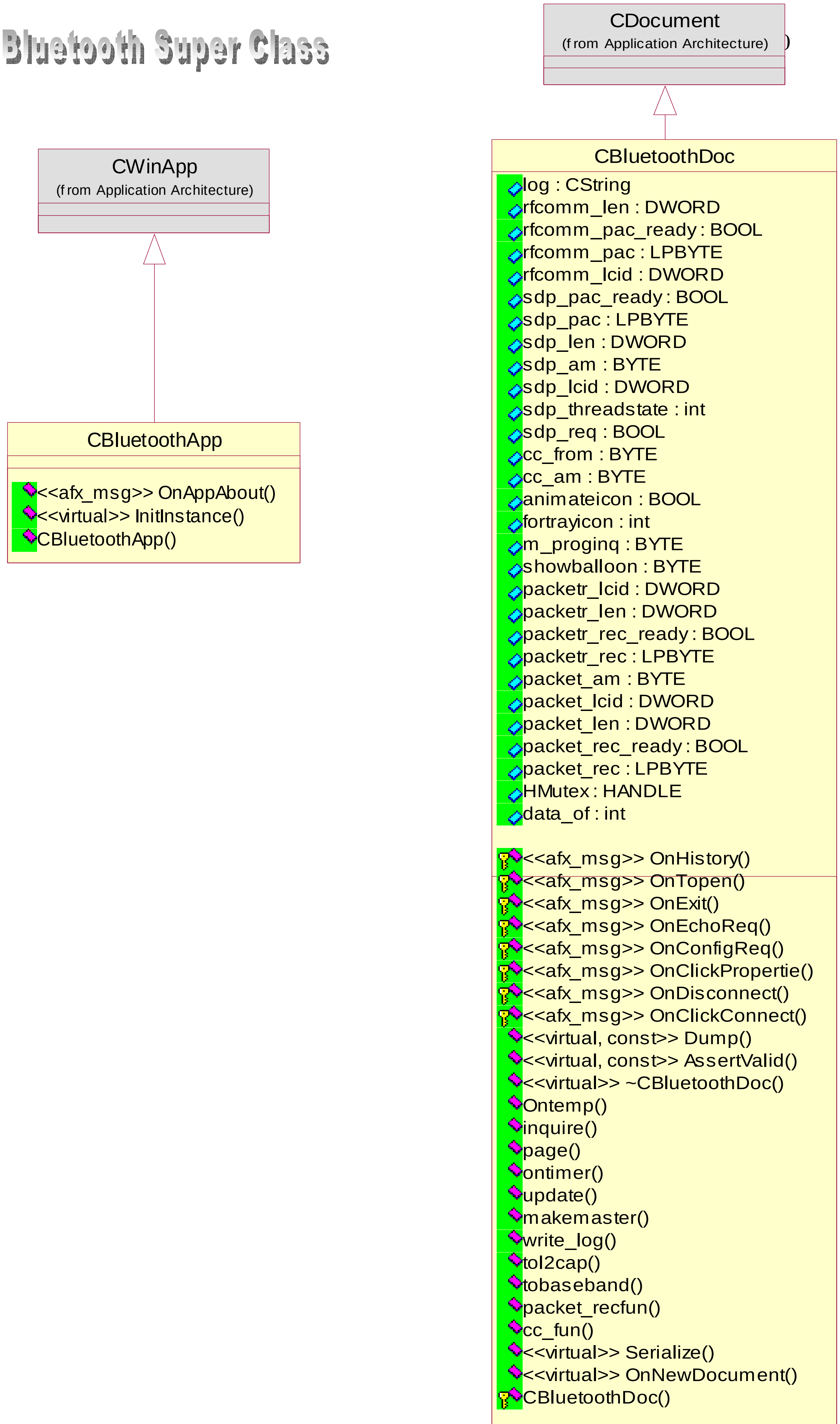
View Classes



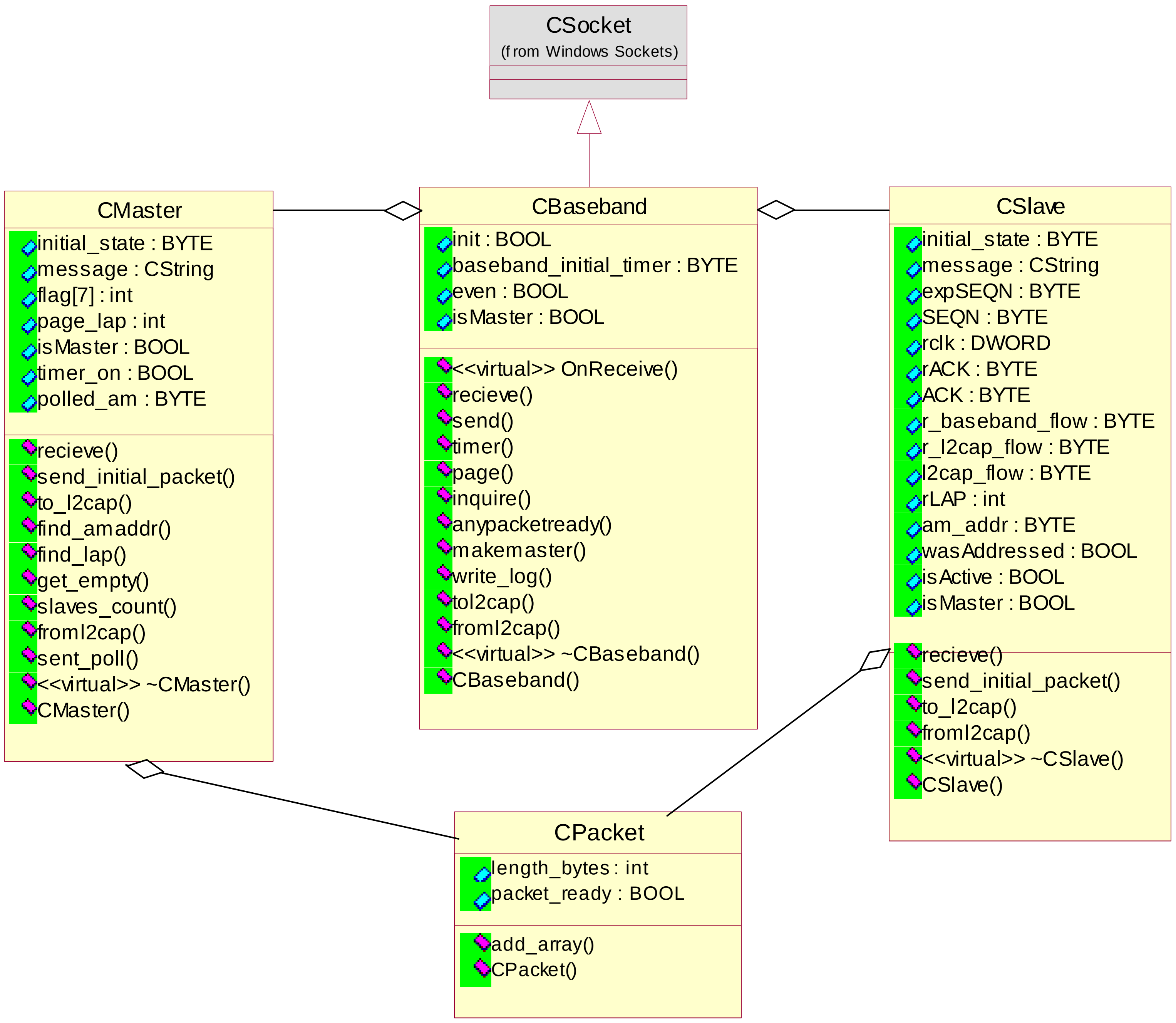
Interface Classes



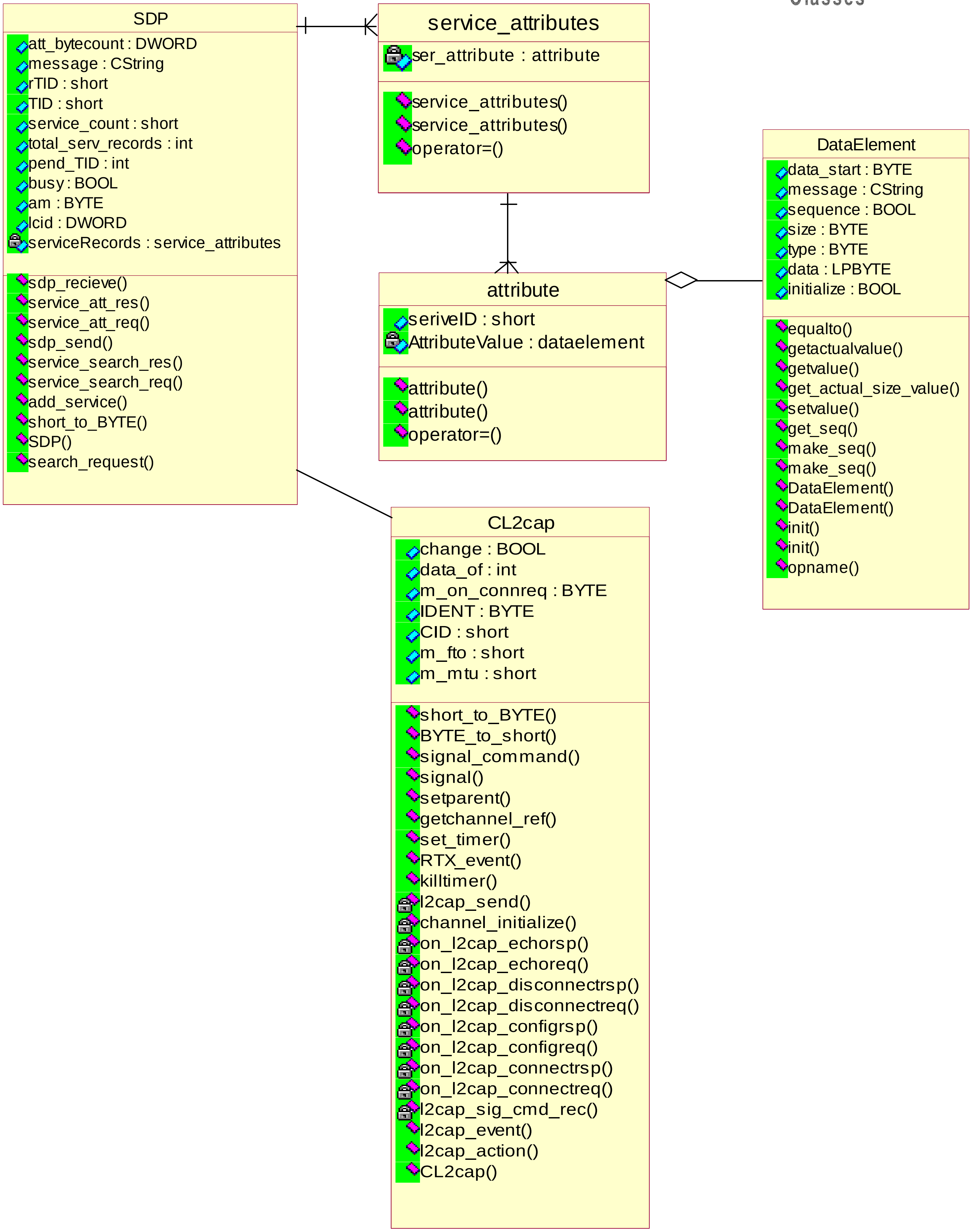
Bluetooth Super Class



Baseband Class



Service Discovery Protocol And Logical Link Control And Adaptation Protocol Classes



CHAPTER 6

SOFTWARE SIMULATION & TESTING

As specified above the protocol layers were implemented as a distributed simulation. Since the design of a radio chip, implementing a frequency hopping mechanism, to which the protocol stack could be ported was beyond the scope of this project, the sofware was adapted to run on a LAN in a distribute manner each PC running the software simulating the actions of a bluetooth device communicating using the rules and procedures defined in the stack.

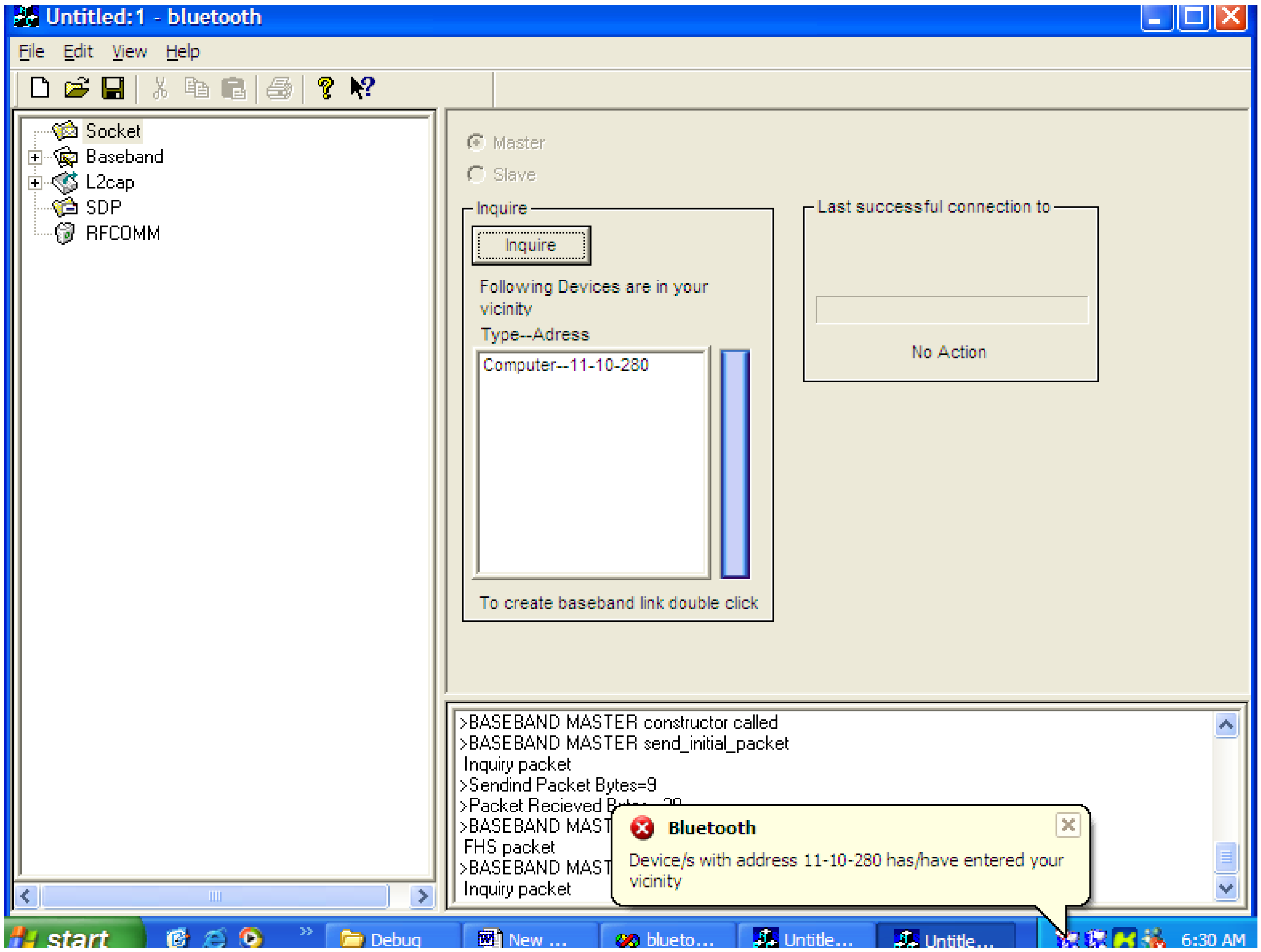
6.1 Visual C++: An Appropriate Choice

The choice of Visual C++ as the environment for implementing the protocol stack, on of the requirements of the project, was a natural one given the excellent support provided for hardware interaction using it's service primitives present in the language. The fact that the code is written in C++ means that the process of porting it to assembly code for hardware implementation is straightforward. The extensive GUI support present meant that a viable application interface for the simulation could be designed, allowing the detailed depiction of the features and capabilities of the protocol stack.

6.2 SIMULATION

The simulation demonstrates the entire bluetooth communication process. The part played by each protocol in the process and the actions carried out by each layer during connection establishment and data transfer is displayed sequentially. The next section explains in brief the steps involved:-

The left pane shows the layers of the bluetooth that are implemented. Clicking on one of them opens the view related to that layer.The right pane shows the current view.The pane at the bottom shows the history of messages and different events and action occurring in the protocol.



Following Devices are in your vicinity

Type--Adress

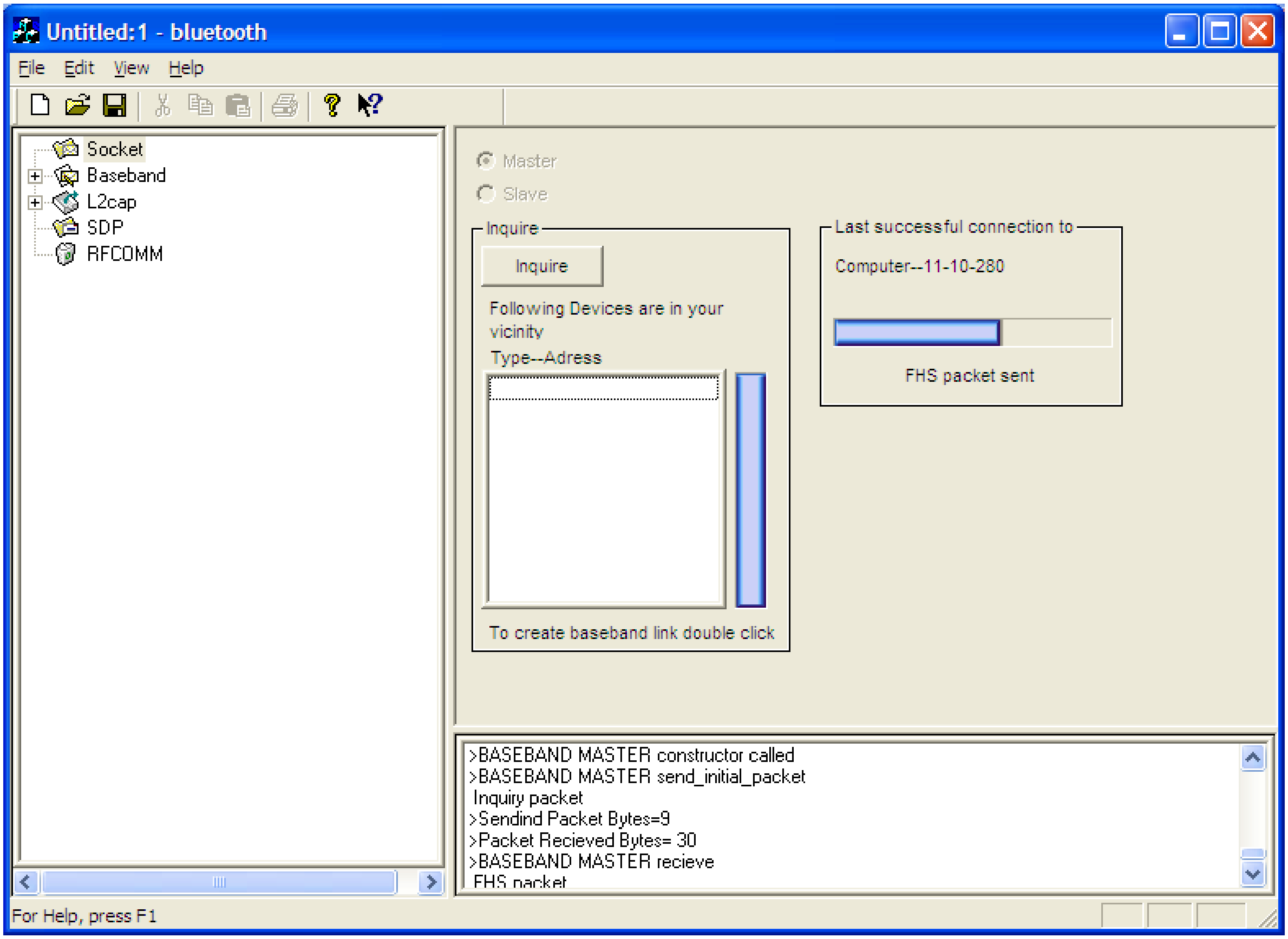
Computer--11-10-280

To create baseband link double click

```
>BASEBAND MASTER constructor called
>BASEBAND MASTER send_initial_packet
Inquiry packet
>Sendind Packet Bytes=9
>Packet Recieved B
>BASEBAND MAST
FHS packet
>BASEBAND MAST
Inquiry packet
```

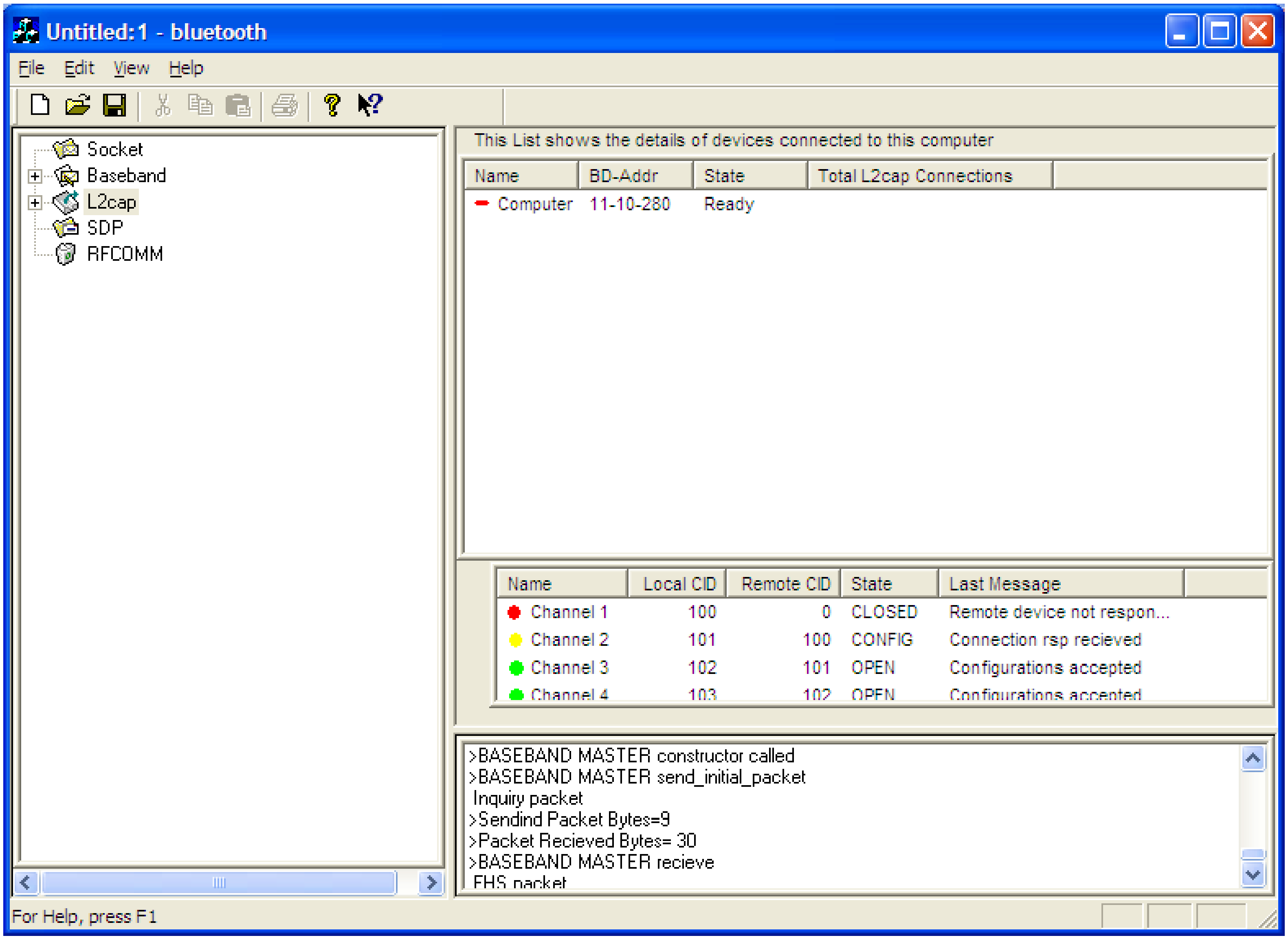
Device/s with address 11-10-280 has/have entered your vicinity

Fig 1:This figure shows the inquiry procedure that on completion is indicating that a device with addr 11-10-280 has entered the vicinity.



```
>BASEBAND MASTER constructor called
>BASEBAND MASTER send_initial_packet
 Inquiry packet
>Sendind Packet Bytes=9
>Packet Recieved Bytes= 30
>BASEBAND MASTER recieve
 FHS packet
```

Fig 2: Paging procedure is shown. The right progress bar shows the process of paging procedure.



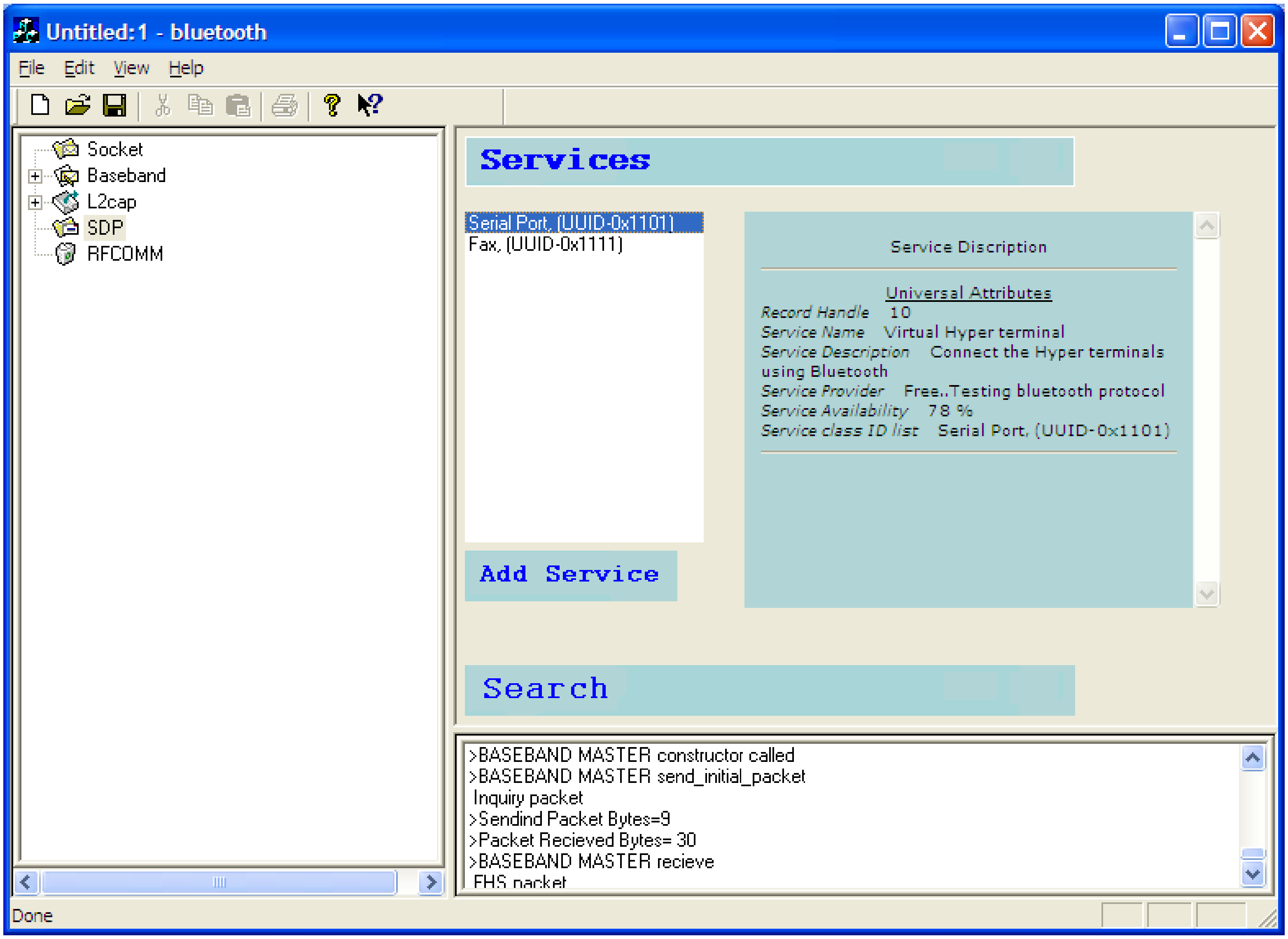
This List shows the details of devices connected to this computer

Name	BD-Addr	State	Total L2cap Connections
Computer	11-10-280	Ready	

Name	Local CID	Remote CID	State	Last Message
🔴 Channel 1	100	0	CLOSED	Remote device not respon...
🟡 Channel 2	101	100	CONFIG	Connection rsp recieved
🟢 Channel 3	102	101	OPEN	Configurations accepted
🟢 Channel 4	103	102	OPEN	Configurations accepted

```
>BASEBAND MASTER constructor called
>BASEBAND MASTER send_initial_packet
 Inquiry packet
>Sendind Packet Bytes=9
>Packet Recieved Bytes= 30
>BASEBAND MASTER recieve
 FHS packet
```

Fig 3: L2CAP view is shown in this figure. First list box shows devices connected with this computer. The second list box shows the L2CAP channels made on each baseband link. Red circles indicate the channel is closed and has expired. Yellow circle shows that the channel is in configuration state and green circle shows that the channel is open for data transfer



Serial Port, (UUID-0x1101)
Fax, (UUID-0x1111)

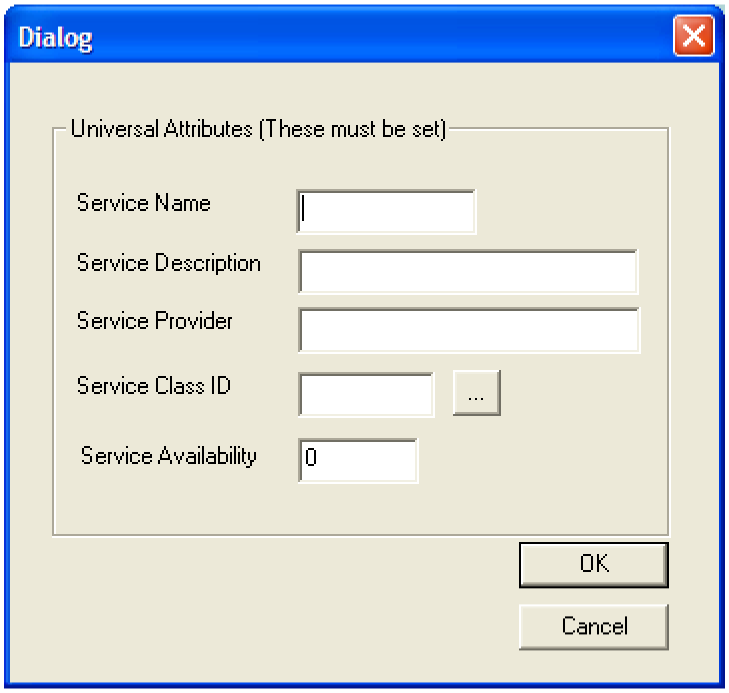
Service Discription

Universal Attributes

Record Handle	10
Service Name	Virtual Hyper terminal
Service Description	Connect the Hyper terminals using Bluetooth
Service Provider	Free..Testing bluetooth protocol
Service Availability	78 %
Service class ID list	Serial Port, (UUID-0x1101)

```
>BASEBAND MASTER constructor called
>BASEBAND MASTER send_initial_packet
 Inquiry packet
>Sendind Packet Bytes=9
>Packet Recieved Bytes= 30
>BASEBAND MASTER recieve
 FHS packet
```

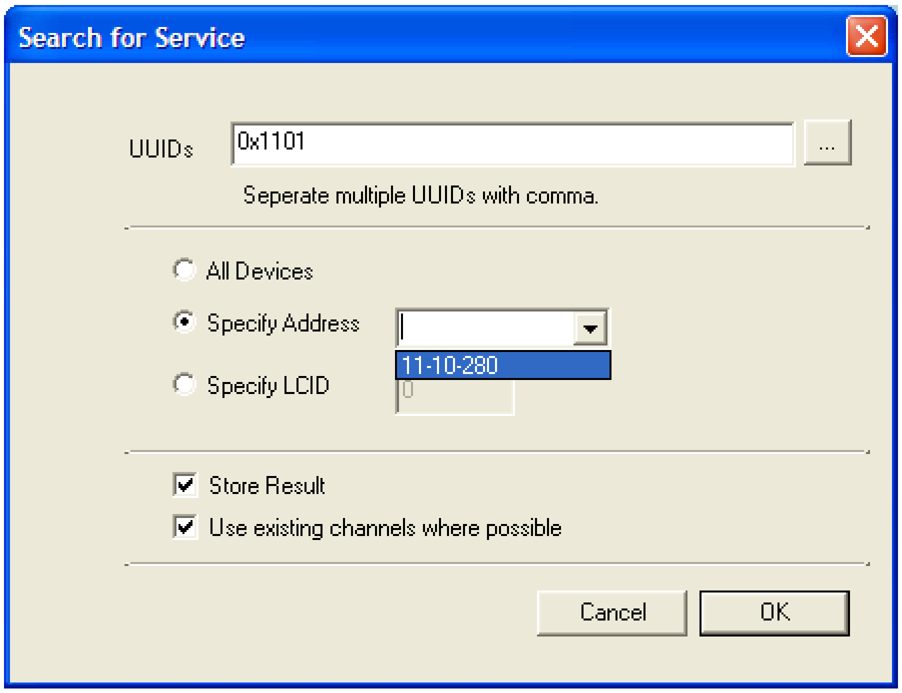
Fig 4: This figure shows SDP (services on this computer) view. The green box shows the attributes of the services listed on left list box.



Universal Attributes (These must be set)

Service Name	
Service Description	
Service Provider	
Service Class ID	
Service Availability	0

Fig 5: Dialog box for adding a service



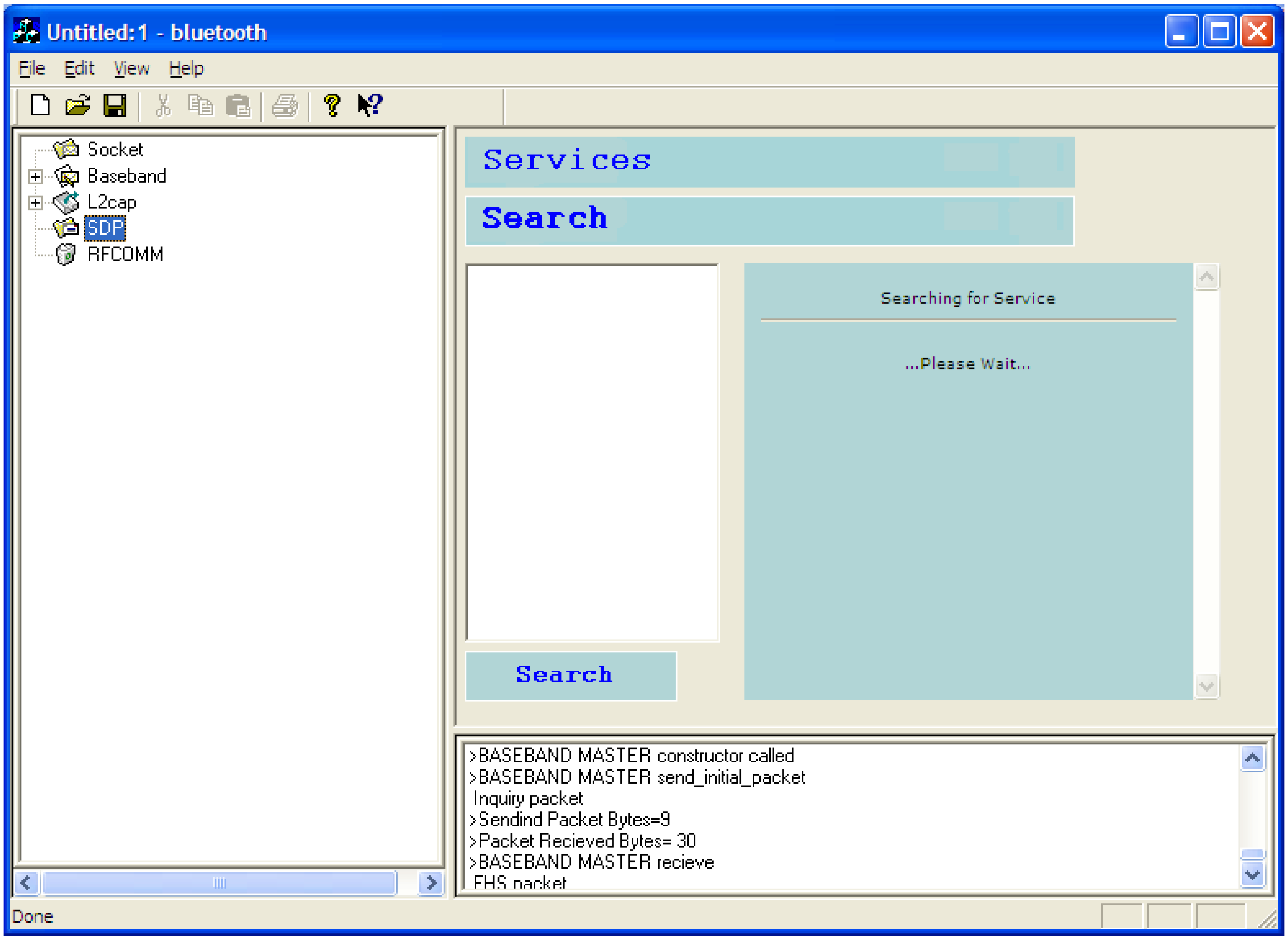
UUIDs: 0x1101

Seperate multiple UUIDs with comma.

( ) All Devices
(●) Specify Address: 11-10-280
( ) Specify LCID: 0

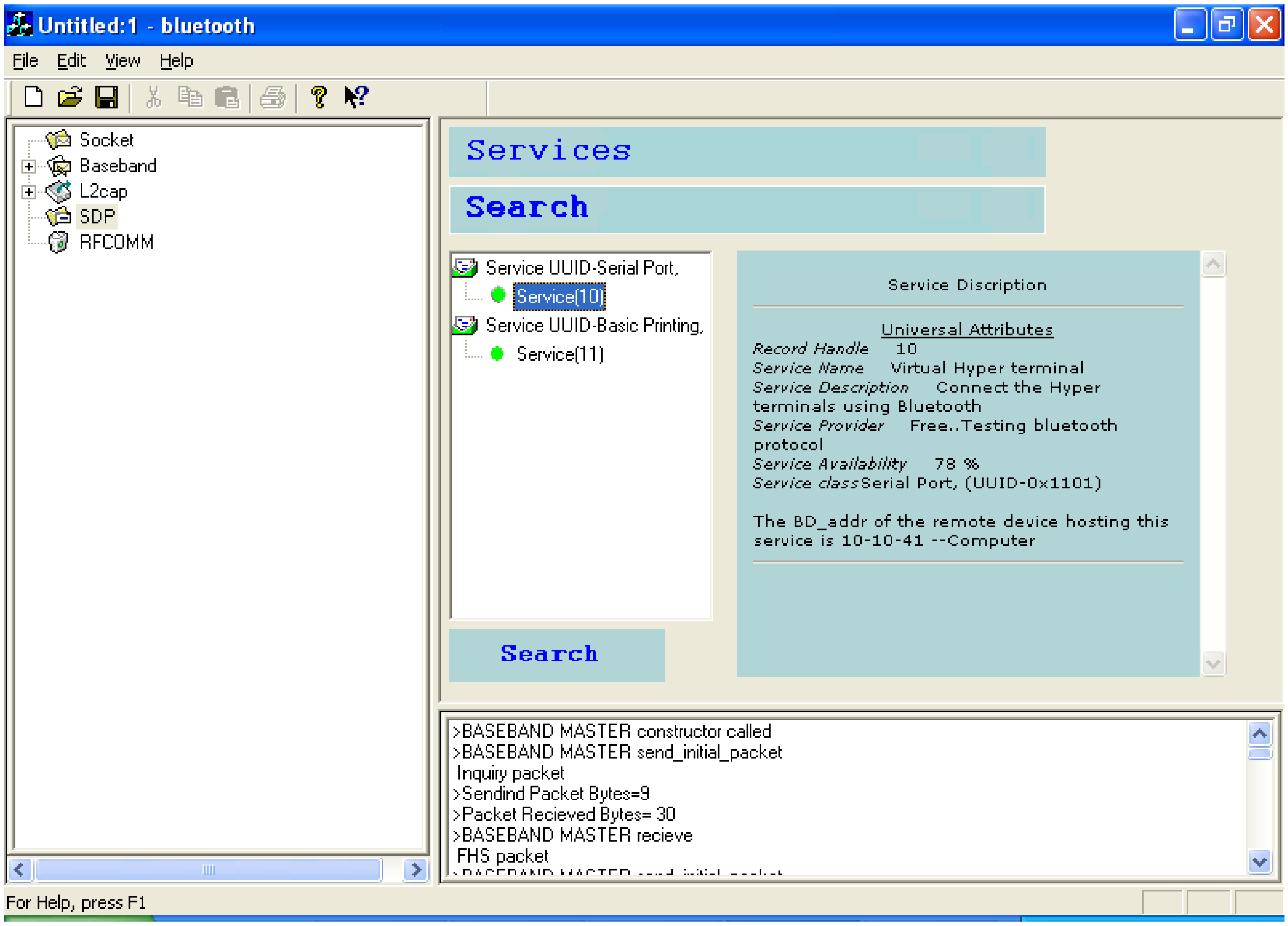
[✓] Store Result
[✓] Use existing channels where possible

Fig 6: Dialog box for searching a service. Three choices are available for searching. To search all the connected devices for a particular service or to search a particular device or to search through a manually created channel.



```
>BASEBAND MASTER constructor called
>BASEBAND MASTER send_initial_packet
 Inquiry packet
>Sendind Packet Bytes=9
>Packet Recieved Bytes= 30
>BASEBAND MASTER recieve
 FHS packet
```

Fig 7: The figure shows the view when a search is in progress.



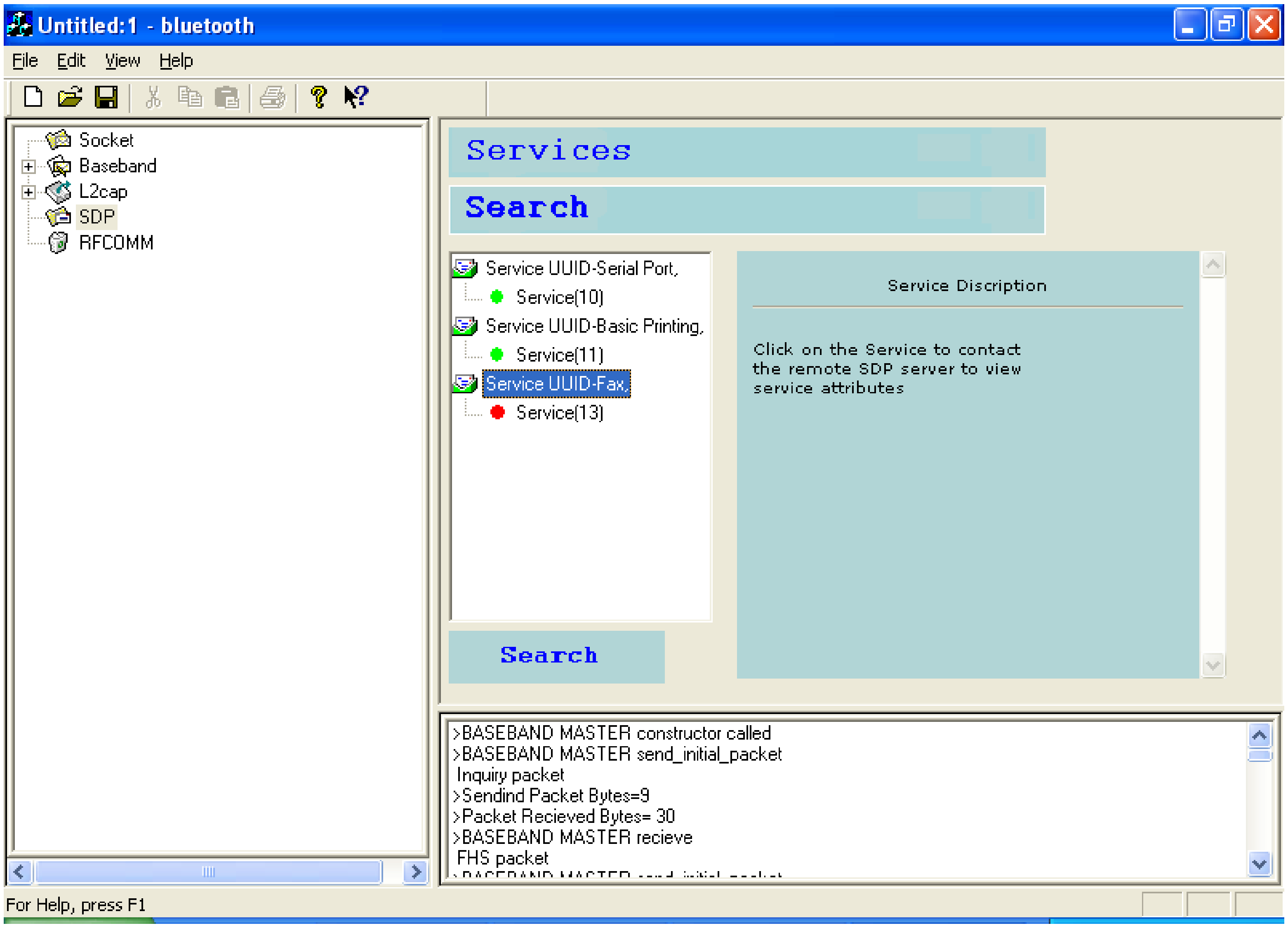
Service Discription

Universal Attributes
Record Handle 10
Service Name Virtual Hyper terminal
Service Description Connect the Hyper terminals using Bluetooth
Service Provider Free..Testing bluetooth protocol
Service Availability 78 %
Service class Serial Port, (UUID-0x1101)

The BD_addr of the remote device hosting this service is 10-10-41 --Computer

```
>BASEBAND MASTER constructor called
>BASEBAND MASTER send_initial_packet
 Inquiry packet
>Sendind Packet Bytes=9
>Packet Recieved Bytes= 30
>BASEBAND MASTER recieve
 FHS packet
```

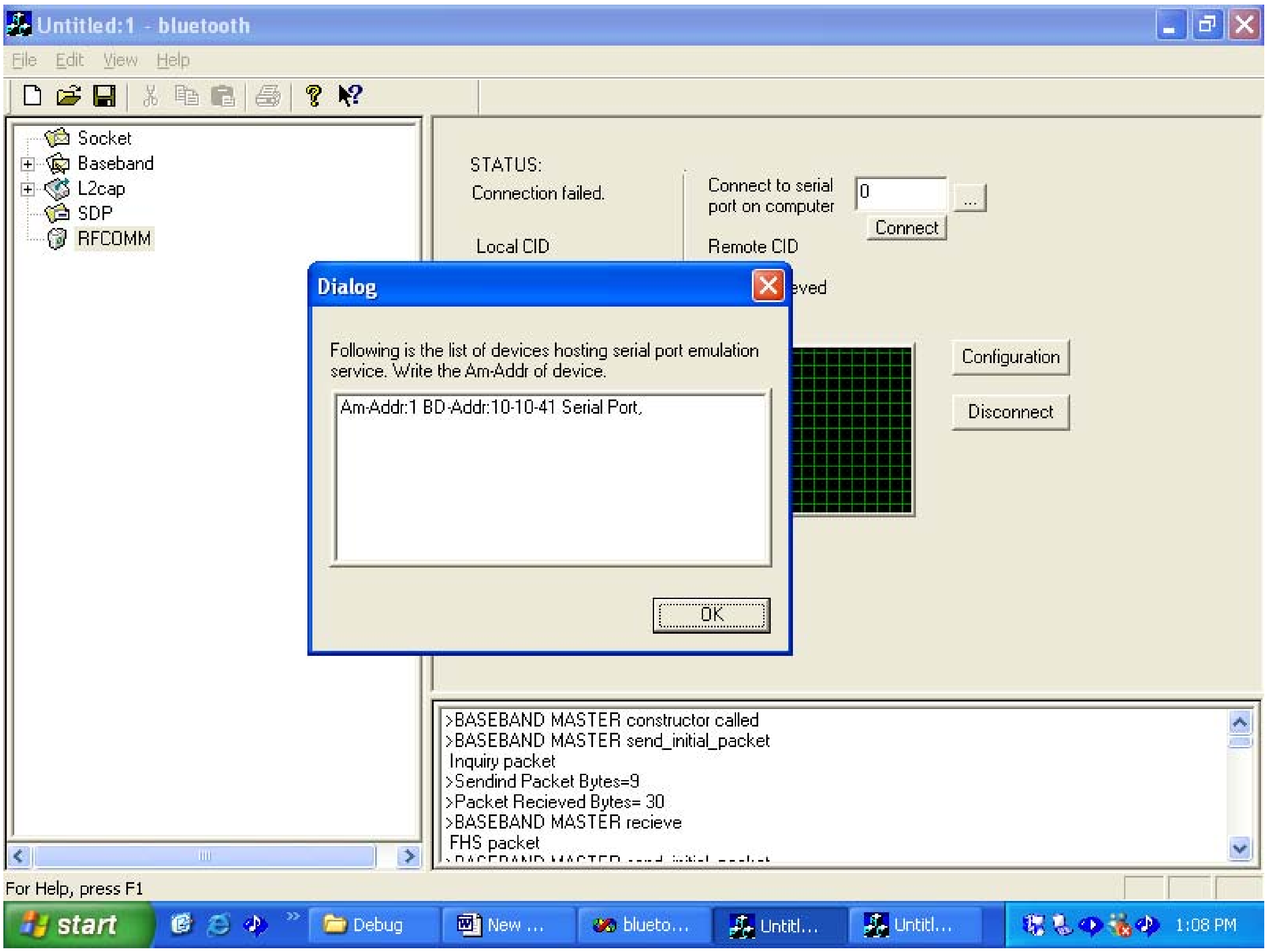
Fig 8: This figure shows the result of a service search.(Serial port and Basic Printing service)



Click on the Service to contact the remote SDP server to view service attributes

```
>BASEBAND MASTER constructor called
>BASEBAND MASTER send_initial_packet
 Inquiry packet
>Sendind Packet Bytes=9
>Packet Recieved Bytes= 30
>BASEBAND MASTER recieve
 FHS packet
```

Fig 9: This figure shows that some of the services are not available (red circle) and some are available (green circle).To find services attributes the service is clicked and then the SDP client contacts the server hosting that particular service to find its attributes.



STATUS:
Connection failed.

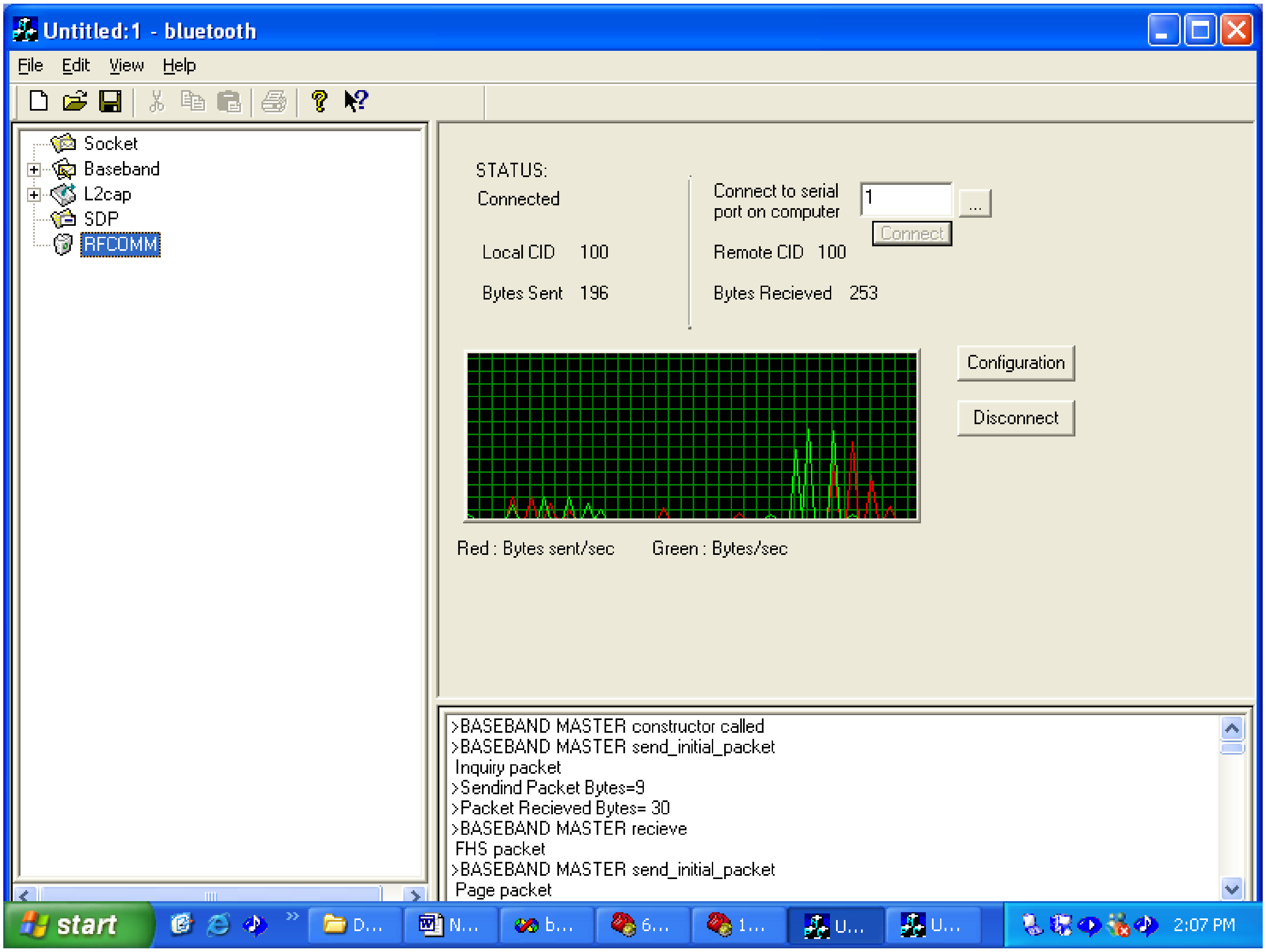
Local CID

Following is the list of devices hosting serial port emulation service. Write the Am-Addr of device.

Am-Addr:1 BD-Addr:10-10-41 Serial Port,

```
>BASEBAND MASTER constructor called
>BASEBAND MASTER send_initial_packet
 Inquiry packet
>Sendind Packet Bytes=9
>Packet Recieved Bytes= 30
>BASEBAND MASTER recieve
 FHS packet
```

Fig 10: This figure shows RFCOMM view. The dialog box shows all the searched serial port services on the connected devices. In order to connect to one of them Am-Addr of the device is written in the edit box.



STATUS:
Connected

Connect to serial port on computer 1

Local CID 100

Remote CID 100

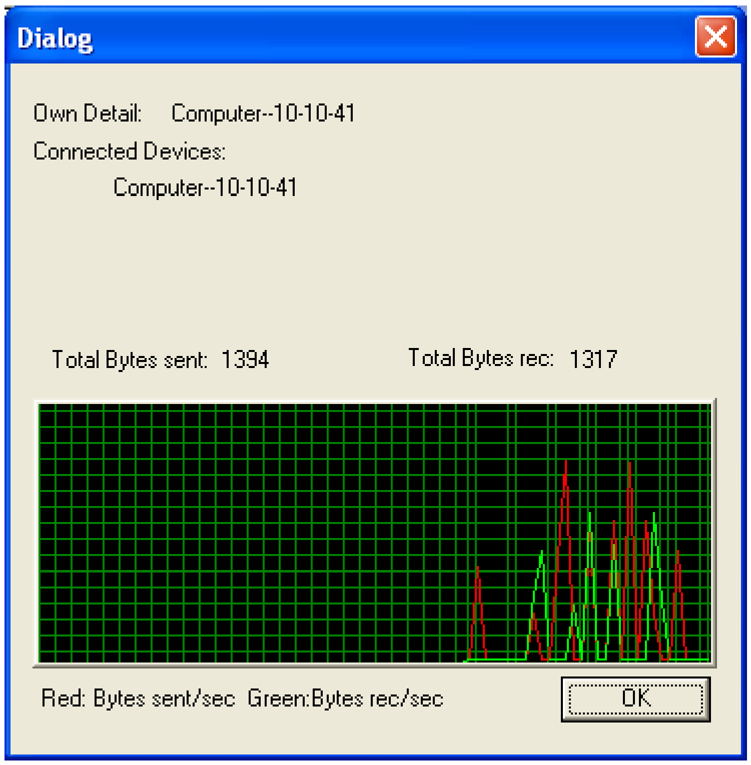
Bytes Sent 196

Bytes Recieved 253

Red : Bytes sent/sec Green : Bytes/sec

```
>BASEBAND MASTER constructor called
>BASEBAND MASTER send_initial_packet
 Inquiry packet
>Sendind Packet Bytes=9
>Packet Recieved Bytes= 30
>BASEBAND MASTER recieve
 FHS packet
>BASEBAND MASTER send_initial_packet
 Page packet
```

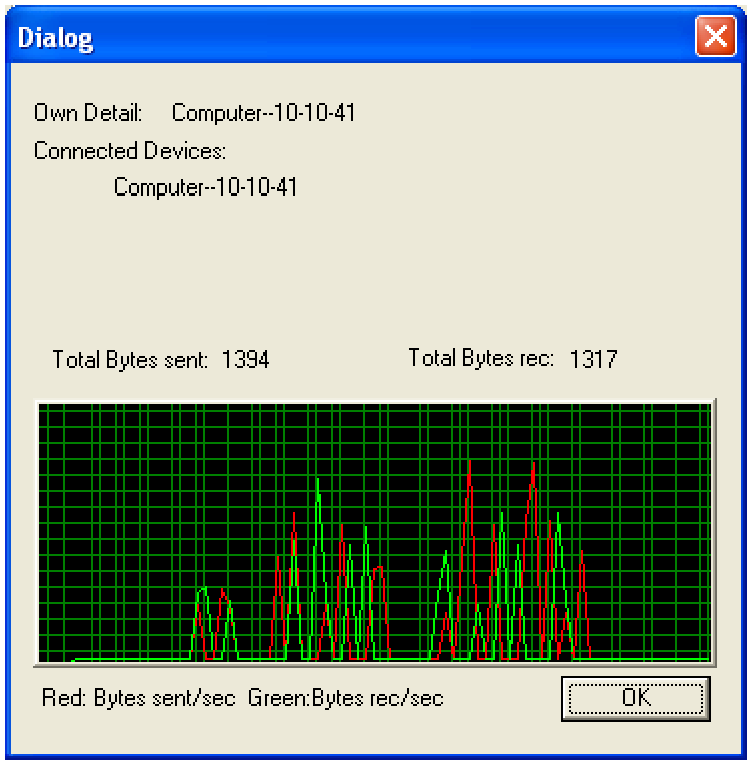
Fig 11: This figures shows that this computer is connected to the remote device through local CID = 100. It also shows the number of bytes sent and received. The graph shows bytes /sec sent and received.



Own Detail: Computer--10-10-41
Connected Devices:
Computer--10-10-41

Total Bytes sent: 1394 Total Bytes rec: 1317

Red: Bytes sent/sec Green:Bytes rec/sec



Own Detail: Computer--10-10-41
Connected Devices:
Computer--10-10-41

Total Bytes sent: 1394 Total Bytes rec: 1317

Red: Bytes sent/sec Green:Bytes rec/sec

Fig 12: This figure shows the dialog box that appears when we click the status area icon of this program. It shows the devices connected and transfer of data.

CHAPTER 7

RECOMMENDATIONS & CONCLUSIONS

CONCLUSION

The work done in our project was mainly aimed at the implementation of the communication and data transfer aspects of Bluetooth. The protocols implemented provide the primary blueprint for carrying out communication between two Bluetooth capable devices. The baseband layer is used to establish a connection between communicating Bluetooth devices by establishing a piconet. The innovative use of frequency hopping and synchronization between devices using the Bluetooth clock of the master are some of the unique features that give Bluetooth it's capability to function over short ranges without interference and without needing the line of sight requirements necessary for other short range communication methods such as Infrared. The L2CAP Layer further facilitates the process of communication, it's segmentation and reassembly, negotiated parameter features allowing the integration of higher level protocols such as WAP, TCP/IP and PTP on top of the main Bluetooth protocol stack. This increases the range and use of Bluetooth immeasurably allowing a Bluetooth capable device to access internet services using the above mentioned protocols. This capability is further enhanced by the guarantees provided in L2CAP for reliable communication. The Bluetooth SDP Layer's ability to dynamically detect all services provided by Bluetooth capable devices without needing to register them separately allow great flexibility in choices and probable actions to a device implanting the protocol. The specific attributes defined for each class of services enable their rapid detection as soon as the associated Bluetooth device comes within range. The RFCOMM use as a serial port emulating protocol over the Bluetooth stack allows it carry out data transfer and other command calls using the unique capabilities of the protocol stack.

Although this stack has been implemented as a simulation the features and capabilities of these layer are sufficiently well implemented to provide an excellent display of Bluetooth's practical application. Infact the software has been designed so that it can easily be hardcoded into a chip carrying out frequency hopping at the radio level, creating a complete Bluetooth unit.

RECOMMENDATIONS

The following recommendations are made on the basis of the work done in this project:

1. As designing and implementing a hardware device carrying out frequency hopping was outside our area of expertise, this project is in the form of a simulation. Further work may involve integrating the software stack design here with frequency hopping hardware to obtain a complete Bluetooth unit.

2. Communication has been implemented for asynchronous data channels. Support may be developed for synchronous data channels allowing for transfer of voice data.

3. User Authentication: Only a device may be authenticated under the current security architecture. In order to authenticate a user, application level security has to be used.

4. Bi-directional traffic: Once a connection is established, data flow is bi-directional. It is not possible to enforce uni-directional data flow.

5. Preset service authorization: There is no mechanism to define preset authorizations for services.

REFERENCES

1. Bluetooth Specifications-Core 1.1.pdf.
2. www.palowireless.com/infotooth
3. www.bluetooth.com
4. www.ibm.com/bluetoothresources
5. www.bluetooth.net