3.3.2 Control field (2 octets)

The Control field identifies the type of frame. The control field will contain sequence numbers where applicable. Its coding is shown in Table 3.1 on page 36. There are two different frame types, Information frame types and Supervisory frame types. Information and Supervisory frames types are distinguished by the rightmost bit in the Control field, as shown in Table 3.1 on page 36.

- *Information frame format (I-frame)*

  The I-frames are used to transfer information between L2CAP entities. Each I-frame has a TxSeq(Send sequence number), ReqSeq(Receive sequence number) which may or may not acknowledge additional I-frames received by the data link layer entity, and a retransmission bit (R bit) that affects whether I-frames are retransmitted.

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

- *Supervisory frame format (S-frame)*

  S-frames are used to acknowledge I-frames and request retransmission of I-frames. Each S-frame has an ReqSeq sequence number which may acknowledge additional I-frames received by the data link layer entity, and a retransmission bit (R bit) that affects whether I-frames are retransmitted.

  Defined types of S-frames are RR (Receiver Ready) and REJ (Reject).

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

X denotes reserved bits. Shall be coded 0.

Table 3.1: Control Field formats

- *Send Sequence Number - TxSeq (6 bits)*

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

- *Receive Sequence Number - ReqSeq (6 bits)*

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

- *Retransmission Disable Bit - R (1 bit)*

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

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

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

  The functions of ReqSeq and R are independent.

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

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

  An I-frame with SAR="Start of L2CAP SDU" also contains a length indicator, specifying the number of information octets in the total L2CAP SDU. The encoding of the SAR bits is shown in Table 3.2.

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

Table 3.2: SAR control element format.

- *Supervisory function - S (2 bits)*

  The S-bits mark the type of S-frame. There are two types defined: RR (Receiver Ready) and REJ (Reject). The encoding is shown in Table 3.3.

00	RR - Receiver Ready
01	REJ - Reject
10	Reserved
11	Reserved

Table 3.3: S control element format: type of S-frame.

3.3.3 L2CAP SDU length field (2 octets)

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

3.3.4 Information payload field (0 to 65531 octets)

The information payload field consists of an integral number of octets. The maximum number of octets in this field is the same as the negotiated value of the MPS configuration parameter. The maximum number of octets in this field is also limited by the range of the basic L2CAP header length field. For I-frames without an SDU length field, this limits the maximum number of octets in the field to 65531. For I-frames with an SDU length field, SAR=01, this limits maximum number of octets in the field to 65529. Thus, even if an MPS of 65531 has been negotiated, the range of the basic L2CAP header length field will restrict the number of octets in this field when an SDU length field is present to 65529.

3.3.5 Frame check sequence (2 octets)

The Frame Check Sequence (FCS) is 2 octets. The FCS is constructed using the generator polynomial $g(D) = D^{16} + D^{15} + D^2 + 1$ (see Figure 3.4). The 16 bit LFSR is initially loaded with the value 0x0000, as depicted in Figure 3.5. The switch S is set in position 1 while data is shifted in, LSB first for each octet. After the last bit has entered the LFSR, the switch is set in position 2, and the register contents are transmitted from right to left (i.e. starting with position 15, then position 14, etc.). The FCS covers the Basic L2CAP header, Control, L2CAP-SDU length and Information payload fields, if present, as shown in Figure 3.3 on page 35.

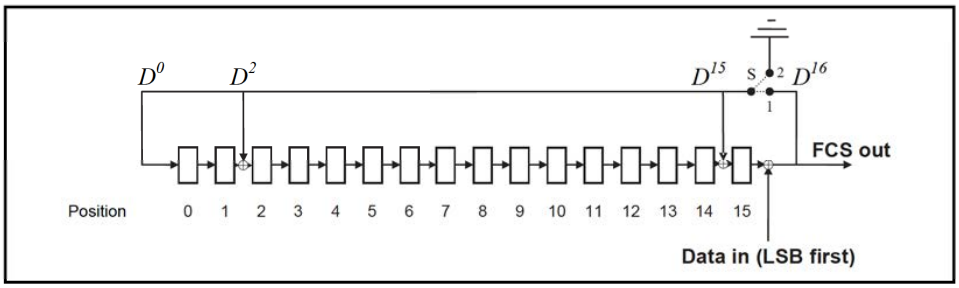


Figure 3.4: The LFSR circuit generating the FCS.

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

Figure 3.5: Initial state of the FCS generating circuit.

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

1. ***I Frame***
   Length = 14
   Control = 0x0002 (SAR=0, ReqSeq=0, R=0, TxSeq=1)
   Information Payload = 00 01 02 03 04 05 06 07 08 09 (10 octets, hexadecimal notation)
   ==> FCS = 0x6138
   ==> Data to Send = 0E 00 40 00 02 00 00 01 02 03 04 05 06 07 08 09 38 61 (hexadecimal notation)

2. ***RR Frame***
   Length = 4
   Control = 0x0101 (ReqSeq=1, R=0, S=0)
   ==> FCS = 0x14D4
   ==> Data to Send = 04 00 40 00 01 01 D4 14 (hexadecimal notation)

3.3.6 Invalid frame detection

A received PDU shall be regarded as invalid, if one of the following conditions occurs:

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

These error conditions may be used for error reporting.

4 SIGNALLING PACKET FORMATS

This section describes the signalling commands passed between two L2CAP entities on peer devices. All signalling commands are sent to the signalling channel with CID 0x0001. This signalling channel is available as soon as an ACL logical transport is set up and L2CAP traffic is enabled on the L2CAP logical link. Figure 4.1 on page 41 illustrates the general format of L2CAP PDUs containing signalling commands (C-frames). Multiple commands may be sent in a single C-frame. Commands take the form of Requests and Responses. All L2CAP implementations shall support the reception of C-frames with a payload length that does not exceed the signaling MTU. The minimum supported payload length for the C-frame (MTU_{sig}) is 48 octets. L2CAP implementations should not use C-frames that exceed the MTU_{sig} of the peer device. If they ever do, the peer device shall send a Command Reject containing the supported MTU_{sig}. Implementations must be able to handle the reception of multiple commands in an L2CAP packet.

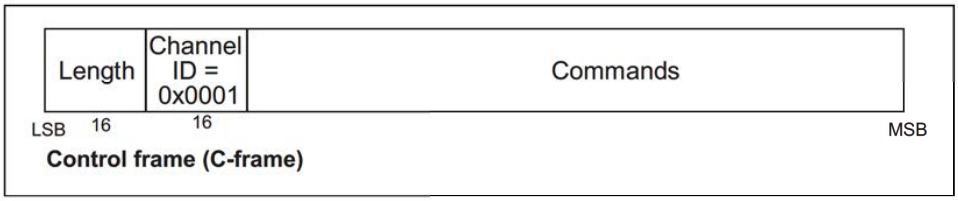


Figure 4.1: L2CAP PDU format on the signalling channel

Figure 4.2 displays the general format of all signalling commands.

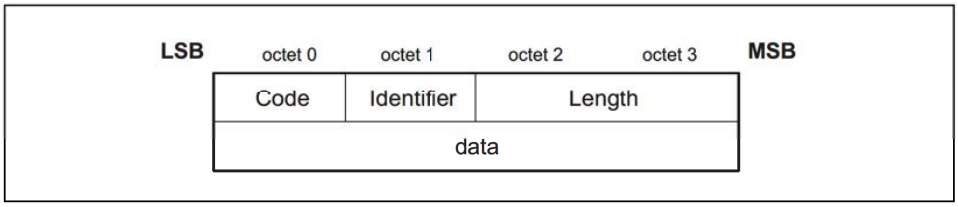


Figure 4.2: Command format

The fields shown are:

- *Code (1 octet)*

  The Code field is one octet long and identifies the type of command. When a packet is received with an unknown Code field, a Command Reject packet (defined in Section 4.1 on page 43) is sent in response.

  Table 4.1 on page 42 lists the codes defined by this document. All codes are specified with the most significant bit in the left-most position.

Code	Description
0x00	RESERVED
0x01	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 4.1: Signalling Command Codes

- *Identifier (1 octet)*

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

  RTX and ERTX timers are used to determine when the remote end point is not responding to signalling requests. On the expiration of a RTX or ERTX timer, the same identifier shall be used if a duplicate Request is re-sent as stated in Section 6.2 on page 75.

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

- *Length (2 octets)*

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

- *Data (0 or more octets)*

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

4.1 COMMAND REJECT (CODE 0x01)

A Command Reject packet shall be sent in response to a command packet with an unknown command code or when sending the corresponding response is inappropriate. Figure 4.3 on page 43 displays the format of the packet. The identifier shall match the identifier of the command packet being rejected. Implementations shall always send these packets in response to unidentified 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 signalling MTU (MTU_{sig}) of the receiver, a single Command Reject packet shall be sent in response. The identifier shall match the first Request command in the L2CAP packet. If only Responses are recognized, the packet shall be silently discarded.

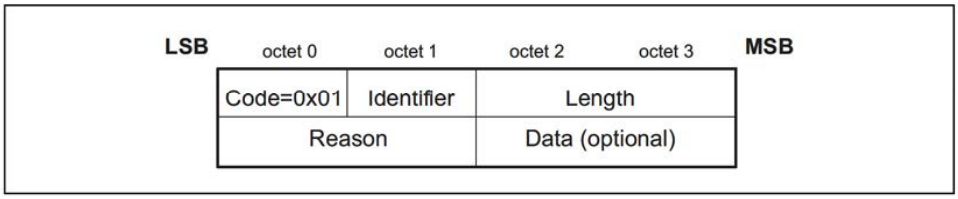


Figure 4.3: Command Reject packet

Figure 4.3 shows the format of the Command Reject packet. The data fields are:

- *Reason (2 octets)*

  The Reason field describes why the Request packet was rejected, and is set to one of the Reason codes in Table 4.2.

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

Table 4.2: 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. The data field shall be 4 octets containing the local (first) and remote (second) channel endpoints (relative to the sender of the Command Reject) of the disputed channel. The remote endpoint is the source CID from the rejected command. The local endpoint is the destination CID from the rejected command. If the rejected command contains only one of the channel endpoints, the other one shall be replaced by the null CID 0x0000.

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

Table 4.3: Reason Data values

4.2 CONNECTION REQUEST (CODE 0x02)

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

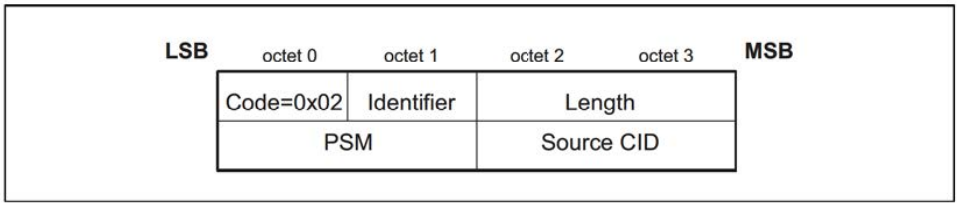


Figure 4.4: Connection Request Packet

The data fields are:

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

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

PSM value	Description
0x0001	Service Discovery Protocol
0x0003	RFCOMM
0x0005	Telephony Control Protocol
0x0007	TCS cordless
0x000F	BNEP
0x0011	HID Control
0x0013	HID Interrupt
0x0015	UPnP (ESDP)
0x0017	AVCTP
0x0019	AVDTP
Other < 1000	RESERVED
[0x1001-0xFFFF]	DYNAMICALLY ASSIGNED

Table 4.4: Defined PSM Values[1]

1. The most recent PSM assignments can be found in the Assigned Numbers database on the Bluetooth member web site.

- *Source CID - SCID (2 octets)*

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

4.3 CONNECTION RESPONSE (CODE 0x03)

When a device receives a Connection Request packet, it shall send a Connection Response packet. The format of the connection response packet is shown in Figure 4.5.

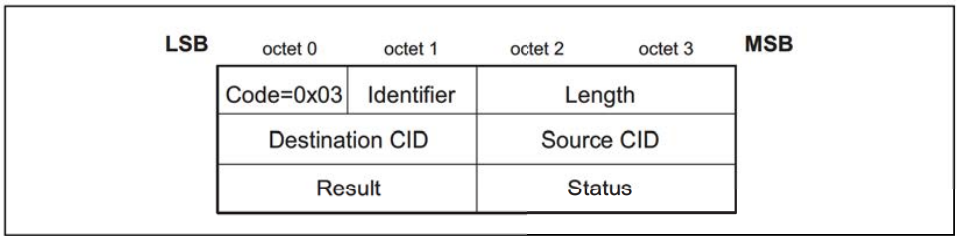


Figure 4.5: Connection Response Packet

The data fields are:

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

  This field contains the channel endpoint on the device sending this Response packet. Thus, the Destination CID represents the channel endpoint on the device receiving the request and sending the response.

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

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

- *Result (2 octets)*

  The result field indicates the outcome of the connection request. The result value of 0x0000 indicates success while a non-zero value indicates the connection request failed or is pending. A logical channel is established on the receipt of a successful result. Table 4.5 on page 46 defines values for this field. The DCID and SCID fields shall be ignored when the result field indicates the connection was refused.

Value	Description
0x0000	Connection successful.
0x0001	Connection pending
0x0002	Connection refused – PSM not supported.
0x0003	Connection refused – security block.
0x0004	Connection refused – no resources available.
Other	Reserved.

Table 4.5: Result values

- *Status (2 octets)*

  Only defined for Result = Pending. Indicates the status of the connection. The status is set to one of the values shown in Table 4.6 on page 47.

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

Table 4.6: Status values

4.4 CONFIGURATION REQUEST (CODE 0x04)

Configuration Request packets are sent to establish an initial logical link transmission contract between two L2CAP entities and also to re-negotiate this contract whenever appropriate. During a re-negotiation session, all data traffic on the channel should be suspended pending the outcome of the negotiation. Each configuration parameter in a Configuration Request shall be related exclusively to either the outgoing or the incoming data traffic but not both of them. In Section 5 on page 57, the various configuration parameters and their relation to the outgoing or incoming data traffic are shown. If an L2CAP entity receives a Configuration Request while it is waiting for a response it shall not block sending the Configuration Response, otherwise the configuration process may deadlock.

If no parameters need to be negotiated then no options shall be inserted and the continuation flag (C) shall be set to zero. L2CAP entities in remote devices shall negotiate all parameters defined in this document whenever the default values are not acceptable. Any missing configuration parameters are assumed to have their most recently explicitly or implicitly accepted values. Even if all default values are acceptable, a Configuration Request packet with no options shall be sent. Implicitly accepted values are default values for the configuration parameters that have not been explicitly negotiated for the specific channel under configuration.

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

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

See Section 7.1 on page 79 for details of the configuration procedure.

Figure 4.6 defines the format of the Configuration Request packet.

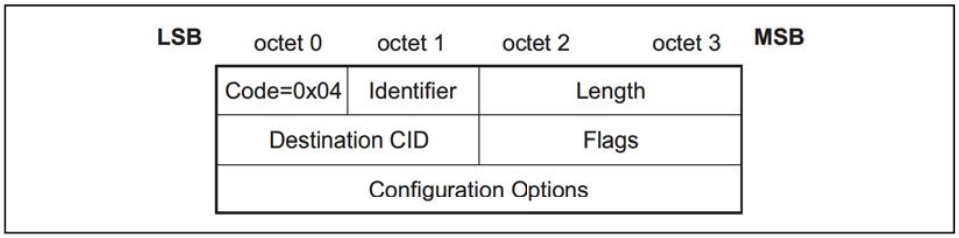


Figure 4.6: Configuration Request Packet

The data fields are:

- *Destination CID - DCID (2 octets)*

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

- *Flags (2 octets)*

  Figure 4.7 shows the two-octet Flags field. Note the most significant bit is shown on the left.

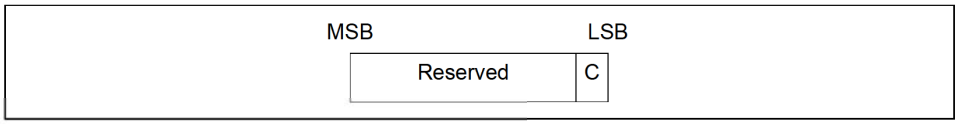


Figure 4.7: Configuration Request Flags field format

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

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

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

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

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

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

- *Configuration Options*

  A list of the parameters and their values to be negotiated shall be provided in the Configuration Options field. These are defined in Section 5 on page 57. A Configuration Request may contain no options (referred to as an empty or null configuration request) and can be used to request a response. For an empty configuration request the length field is set to 0x0004.

4.5 CONFIGURATION RESPONSE (CODE 0X05)

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

The options sent in the Response depend on the value in the Result field. Figure 4.8 on page 50 defines the format of the Configuration Response packet. See also Section 7.1 on page 79 for details of the configuration process.

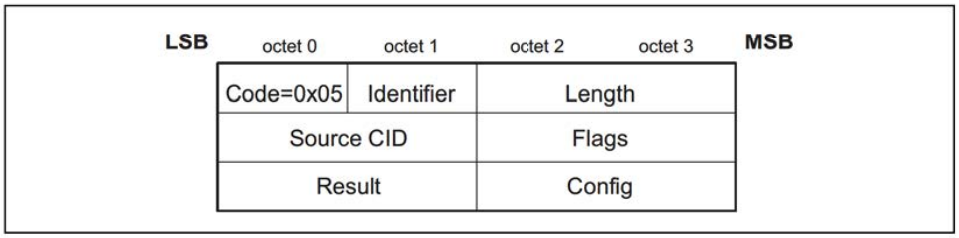


Figure 4.8: Configuration Response Packet

The data fields are:

- *Source CID - SCID (2 octets)*

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

- *Flags (2 octets)*

  Figure 4.9 displays the two-octet Flags field. Note the most significant bit is shown on the left.

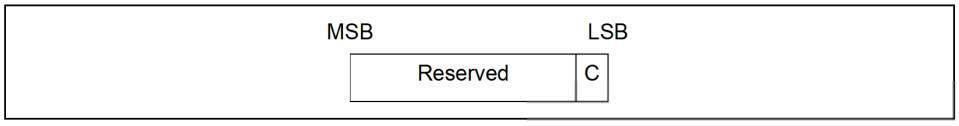


Figure 4.9: Configuration Response Flags field format

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

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

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

- *Result (2 octets)*

  The Result field indicates whether or not the Request was acceptable. See Table 4.7 on page 51 for possible result codes.

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

Table 4.7: Configuration Response Result codes

- *Configuration Options*

  This field contains the list of parameters being configured. These are defined in Section 5 on page 57. On a successful result, these parameters contain the return values for any wild card parameter values (see Section 5.3 on page 60) contained in the request.

  On an unacceptable parameters failure (Result=0x0001) the rejected parameters shall be sent in the response with the values that would have been accepted if sent in the original request. Any missing configuration parameters are assumed to have their most recently accepted values and they too shall be included in the Configuration Response if they need to be changed.

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

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

  On an unknown option failure (Result=0x0003), the option types not understood by the recipient of the Request shall be included in the Response unless they are hints. Hints are those options in the Request that are skipped if not understood (see Section 5 on page 57). Hints shall not be included in the Response and shall not be the sole cause for rejecting the Request.

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

4.6 DISCONNECTION REQUEST (CODE 0x06)

Terminating an L2CAP channel requires that a disconnection request be sent and acknowledged by a disconnection response. Figure 4.10 on page 52 shows a disconnection request. The receiver shall ensure that both source and destination CIDs match before initiating a channel disconnection.

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

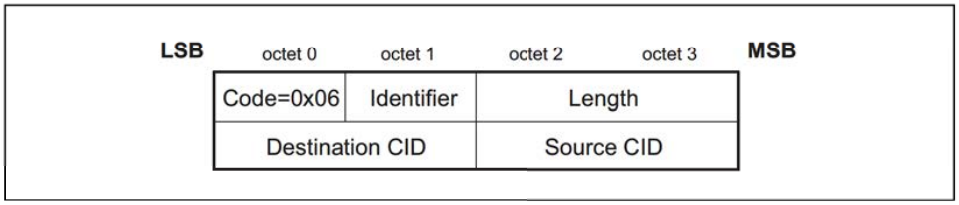


Figure 4.10: Disconnection Request Packet

The data fields are:

- *Destination CID - DCID (2 octets)*

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

- *Source CID - SCID (2 octets)*

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

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

4.7 DISCONNECTION RESPONSE (CODE 0x07)

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

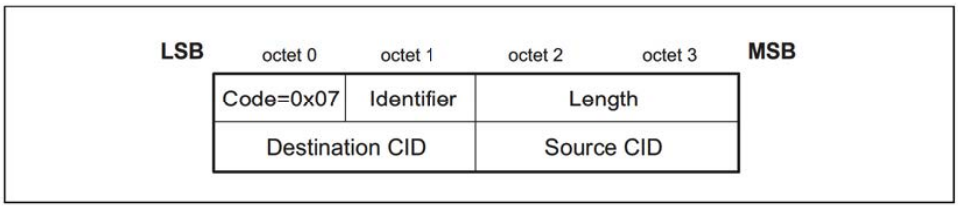


Figure 4.11: Disconnection Response Packet

The data fields are:

- *Destination CID - DCID (2 octets)*

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

- *Source CID - SCID (2 octets)*

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

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

4.8 ECHO REQUEST (CODE 0x08)

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

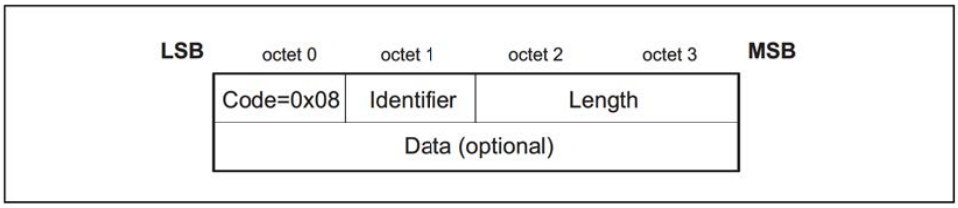


Figure 4.12: Echo Request Packet

4.9 ECHO RESPONSE (CODE 0x09)

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

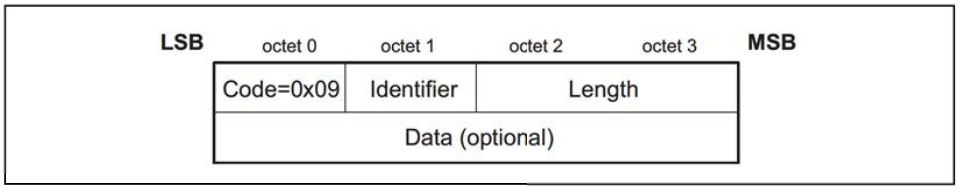


Figure 4.13: Echo Response Packet

4.10 INFORMATION REQUEST (CODE 0X0A)

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

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

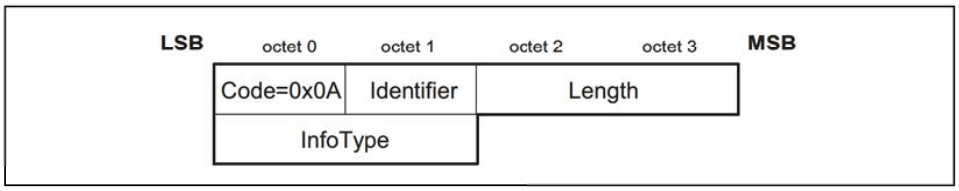


Figure 4.14: Information Request Packet

The data fields are:

- *InfoType (2 octets)*

  The InfoType defines the type of implementation specific information being requested. See Section 4.11 on page 55 for details on the type of information requested.

Value	Description
0x0001	Connectionless MTU
0x0002	Extended features supported
Other	Reserved

Table 4.8: InfoType definitions

4.11 INFORMATION RESPONSE (CODE 0X0B)

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

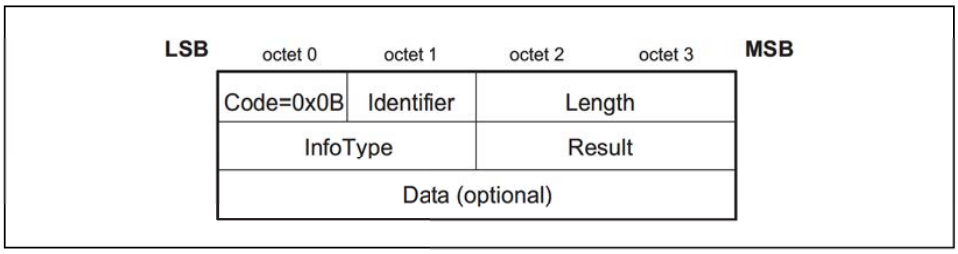


Figure 4.15: Information Response Packet

The data fields are:

- *InfoType (2 octets)*

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

- *Result (2 octets)*

  The Result contains information about the success of the request. If result is "Success", the data field contains the information as specified in Table 4.10 on page 56. If result is "Not supported", no data shall be returned.

Value	Description
0x0000	Success
0x0001	Not supported
Other	Reserved

Table 4.9: Information Response Result values

- *Data (0 or more octets)*

  The contents of the Data field depends on the InfoType.
  For InfoType = 0x0001 the data field contains the remote entity's 2-octet acceptable connectionless MTU. The default value is defined in Section 3.2 on page 34.

  For InfoType = 0x0002, the data field contains the 4 octet L2CAP extended feature mask. The feature mask refers to the extended features that the L2CAP entity sending the Information Response supports. The feature bits contained in the L2CAP feature mask are specified in Section 4.12 on page 56.

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

InfoType	Data	Data Length (octets)
0x0001	Connectionless MTU	2
0x0002	Extended feature mask	4

Table 4.10: Information Response Data fields

4.12 EXTENDED FEATURE MASK

The features are represented as a bit mask in the Information Response data field (see Section 4.11 on page 55). For each feature a single bit is specified which shall be set to 1 if the feature is supported and set to 0 otherwise. All unknown, reserved, or unassigned feature bits shall be set to 0.

The feature mask shown in Table 4.11 on page 56 consists of 4 octets (numbered octet 0 ... 3), with bit numbers 0 ... 7 each. Within the Information Response packet data field, bit 0 of octet 0 is aligned leftmost, bit 7 of octet 3 is aligned rightmost.

Note: the L2CAP feature mask is a new concept introduced in Bluetooth v1.2 and thus contains new features introduced after Bluetooth v1.1.

No.	Supported feature	Octet	Bit
0	Flow control mode	0	0
1	Retransmission mode	0	1
2	Bi-directional QoS[1]	0	2
31	Reserved for feature mask extension	3	7

Table 4.11: Extended feature mask.

1. Peer side supports upper layer control of the Link Manager's Bi-directional QoS, see Section 5.3 on page 60 for more details.

5 CONFIGURATION PARAMETER OPTIONS

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

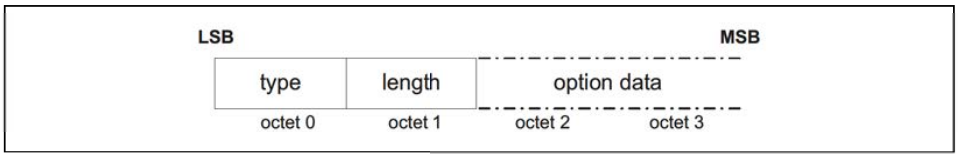


Figure 5.1: Configuration option format

The configuration option fields are:

- *Type (1 octet)*

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

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

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

- *Length (1 octet)*

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

- *Option data*

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

5.1 MAXIMUM TRANSMISSION UNIT (MTU)

This option specifies the maximum SDU size the sender of this option is capable of accepting for a channel. The type is 0x01, and the payload length is 2 octets, carrying the two-octet MTU size value as the only information element (see Figure 5.2 on page 58). Unlike the B-Frame length field, the I-Frame length field may be greater then the configured MTU because it includes the octet lengths of the Control, L2CAP SDU Length (when present), and frame check sequence fields as well as the Information octets.

MTU is not a negotiated value, it is an informational parameter that each device can specify independently. It indicates to the remote device that the local device can receive, in this channel, an MTU larger than the minimum required. All L2CAP implementations shall support a minimum MTU of 48 octets, however some protocols and profiles explicitly require support for a

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

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

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

The signalling described in Section 4.5 on page 50 may be used to reject an MTU smaller than the minimum MTU for a channel. The "failure-unacceptable parameters" result sent to reject the MTU shall include the proposed value of MTU that the remote device intends to transmit. It is implementation specific whether the local device continues the configuration process or disconnects the channel.

If the remote device sends a positive configuration response it shall include the actual MTU to be used on this channel for traffic flowing into the local device. This is the minimum of the MTU in the configuration request and the outgoing MTU capability of the device sending the configuration response. The new agreed value (the default value in a future re-configuration) is the value specified in the request.

The MTU to be used on this channel for the traffic flowing in the opposite direction will be established when the remote device sends its own Configuration Request as explained in Section 4.4 on page 47.

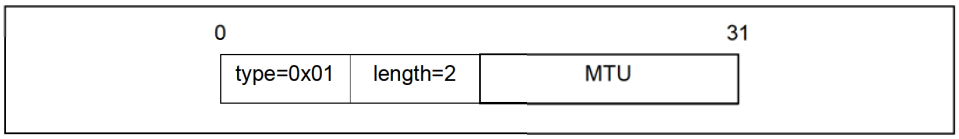


Figure 5.2: MTU Option Format

The option data field is:

- Maximum Transmission Unit - MTU (2 octets)

  The MTU field is the maximum SDU size, in octets, that the originator of the Request can accept for this channel. The MTU is asymmetric and the sender of the Request shall specify the MTU it can receive on this channel if it differs from the default value. L2CAP implementations shall support a minimum MTU size of 48 octets. The default value is 672 octets[1].

5.2 FLUSH TIMEOUT OPTION

This option is used to inform the recipient of the Flush Timeout the sender is going to use. The Flush Timeout is defined in the Baseband specification "Flushing payloads" on page 150[vol. 3]. The type is 0x02 and the payload size is 2 octets.

If the remote device returns a negative response to this option and the local device cannot honor the proposed value, then it shall either continue the configuration process by sending a new request with the original value, or disconnect the channel. The flush timeout applies to all channels on the same ACL logical transport and other channels on the same ACL logical transport may therefore have other values.

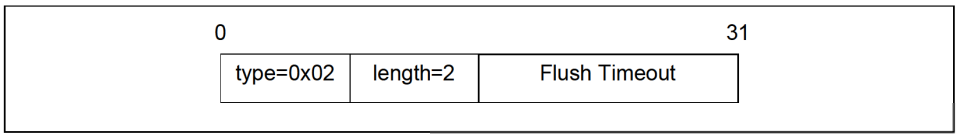


Figure 5.3: Flush Timeout option format.

The option data field is:

- *Flush Timeout*

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

  Possible values are:

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

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

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

1. The default MTU was selected based on the payload carried by two baseband DH5 packets (2*341=682) minus the baseband ACL headers (2*2=4) and L2CAP header (6).

5.3 QUALITY OF SERVICE (QOS) OPTION

This option specifies a flow specification similar to RFC 1363[1]. Although the RFC flow specification addresses only the transmit characteristics, the Bluetooth QoS interface can handle the two directions (Tx and Rx) in the negotiation as described below.

If no QoS configuration parameter is negotiated the link shall assume the default parameters. The QoS option is type 0x03.

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

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

The remote device's Configuration Response contains information that depends on the value of the result field (see Section 4.5 on page 50). If the request was for Guaranteed Service, the response shall include specific values for any wild card parameters (see Token Rate and Token Bucket Size descriptions) contained in the request. If the result is "Failure – unacceptable parameters", the response shall include a list of outgoing flow specification parameters and parameter values that would make a new Connection Request from the local device acceptable by the remote device. Both explicitly referenced in a Configuration Request or implied configuration parameters can be included in a Configuration Response. Recall that any missing configuration parameters from a Configuration Request are assumed to have their most recently accepted values.

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

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

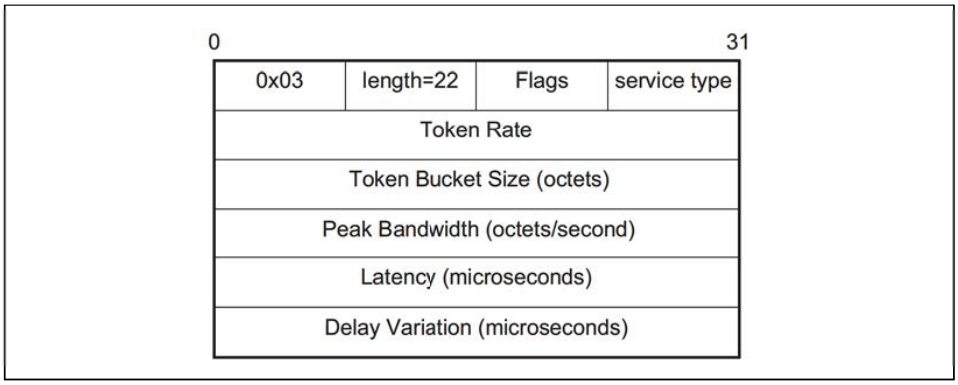


Figure 5.4: Quality of Service (QoS) option format containing Flow Specification.

The option data fields are:

- *Flags (1 octet)*

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

- *Service Type (1 octet)*

  This field indicates the level of service required. Table 5.1 on page 61 defines the different services available. The default value is 'Best effort'.

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

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

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

Table 5.1: Service type definitions

- *Token Rate (4 octets)*

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

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

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

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

- *Token Bucket Size (4 octets)*

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

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

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

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

- *Peak Bandwidth (4 octets)*

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

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

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

- *Access Latency (4 octets)*

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

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

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

- *Delay Variation (4 octets)*

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

5.4 RETRANSMISSION AND FLOW CONTROL OPTION

This option specifies whether retransmission and flow control is used. If the feature is used incoming parameters are specified by this option.

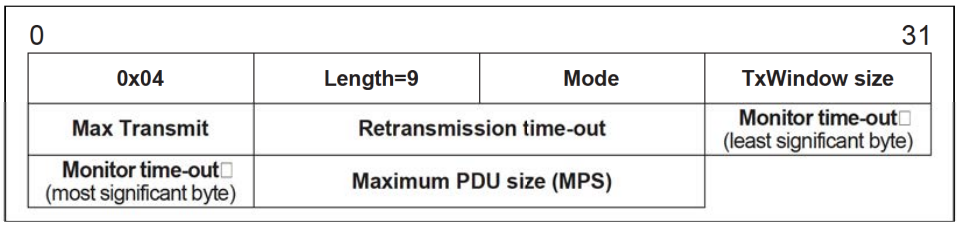


Figure 5.5: Retransmission and Flow Control option format.

The option data fields are:

- *Mode (1 octet)*

  The field contain the requested mode of the link. Possible values are shown in Table 5.2 on page 64.

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

Table 5.2: Mode definitions.

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

  Retransmission mode should be enabled if a reliable channel has been requested, or if the L2CAP Flush Time-Out is long enough to contain the round-trip delay of a retransmission request.

- *TxWindow size (1 octet)*

  This field specifies the size of the transmission window for flow control mode and retransmission mode. The range is 1 to 32.

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

- *MaxTransmit (1 octet)*

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

  MaxTransmit controls the number of retransmissions that L2CAP is allowed to try in Retransmission mode before accepting that a packet and the link is lost. Lower values might be appropriate for services requiring low latency. Higher values will be suitable for a link requiring robust operation. A value of 1 means that no retransmissions will be made but also means that the link will be dropped as soon as a packet is lost. MaxTransmit shall not be set to zero.

- *Retransmission time-out (2 octets)*

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

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

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

- *Monitor time-out (2 octets)*

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

  This timer ensures that lost acknowledgements are retransmitted. Its main use is to recover Retransmission Disable Bit changes in lost frames when no data is being sent. The timer shall be started immediately upon transitioning to the open state. It shall remain active as long as the connection is in the open state and the retransmission timer is not active. Upon expiration of the Monitor timer an S-frame shall be sent and the timer shall be restarted. If the monitor timer is already active when an S-frame is sent, the timer shall be restarted. An idle connection will have periodic monitor traffic sent in both directions. The value for this time-out should also be set to 100's of milliseconds or higher.

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

  The maximum size of payload data in octets that the L2CAP layer entity is capable of accepting, i.e. the MPS corresponds to the maximum PDU payload size.

The settings are configured separately for the two directions of an L2CAP connection. For example, an L2CAP connection can be configured as Flow Control

mode in one direction and Retransmission mode in the other direction. If Basic L2CAP mode is configured in one direction and Retransmission mode or Flow control mode is configured in the other direction on the same L2CAP channel then the channel shall not be used.
Note: this asymmetric configuration only occurs during configuration.

6 STATE MACHINE

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

6.1 GENERAL RULES FOR THE STATE MACHINE:

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

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

- CLOSED – channel not connected.
- WAIT_CONNECT – a connection request has been received, but only a connection response with indication “pending” can be sent.
- WAIT_CONNECT_RSP – a connection request has been sent, pending a positive connect response.
- CONFIG – the different options are being negotiated for both sides; this state comprises a number of substates, see Section 6.1.3 on page 69
- OPEN – user data transfer state.
- WAIT_DISCONNECT – a disconnect request has been sent, pending a disconnect response.

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

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

- *OpenChannel_Req* – a local L2CAP entity is requested to set up a new connection-oriented channel.
- *OpenChannel_Rsp* – a local L2CAP entity is requested to finally accept or refuse a pending connection request.
- *ConfigureChannel_Req* – a local L2CAP entity is requested to initiate an outgoing configuration request.
- *CloseChannel_Req* – a local L2CAP entity is requested to close a channel.
- *SendData_Req* – a local L2CAP entity is requested to transmit an SDU.
- *ReconfigureChannel_Req* – a local L2CAP entity is requested to reconfigure the parameters of a connection-oriented channel.

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

To simplify the state event tables, the RTX and ERTX timers, as well as the handling of request retransmissions are described in Section 6.2 on page 75 and not included in the state tables.

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

The following states and transitions are illustrated in Figure 6.1 on page 77.

6.1.1 CLOSED state

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

Table 6.1: CLOSED state event table.

Notes:

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

6.1.2 WAIT_CONNECT_RSP state

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

Table 6.2: WAIT_CONNECT_RSP state event table.

Notes:

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

6.1.3 WAIT_CONNECT state

Event	Condition	Action	Next State
OpenChannel_Rsp	Pending connection request is finally acceptable	Send L2CAP_Connect_Rsp (success)	CONFIG (substate WAIT_CONFIG)
OpenChannel_Rsp	Pending connection request is finally refused	Send L2CAP_Connect_Rsp (refused)	CLOSED
L2CAP_ConnectRsp	-	Ignore	WAIT_CONNECT
L2CAP_ConfigRsp	-	Ignore	WAIT_CONNECT
L2CAP_DisconnectRsp	-	Ignore	WAIT_CONNECT
L2CAP_Data	-	Ignore	WAIT_CONNECT

Table 6.3: WAIT_CONNECT state event table.

Notes:

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

6.1.4 CONFIG state

As it is also described in Section 7.1 on page 79, both L2CAP entities initiate a configuration request during the configuration process. This means that each device adopts an initiator role for the outgoing configuration request, and an acceptor role for the incoming configuration request. Configurations in both directions may occur sequentially, but can also occur in parallel.

The following substates are distinguished within the CONFIG state:

- WAIT_CONFIG – a device has sent or received a connection response, but has neither initiated a configuration request yet, nor received a configuration request with acceptable parameters.
- WAIT_SEND_CONFIG – for the initiator path, a configuration request has not yet been initiated, while for the response path, a request with acceptable options has been received.
- WAIT_CONFIG_REQ_RSP – for the initiator path, a request has been sent but a positive response has not yet been received, and for the acceptor path, a request with acceptable options has not yet been received.
- WAIT_CONFIG_RSP – the acceptor path is complete after having responded to acceptable options, but for the initiator path, a positive response on the recent request has not yet been received.
- WAIT_CONFIG_REQ – the initiator path is complete after having received a positive response, but for the acceptor path, a request with acceptable options has not yet been received.

According to Section 6.1.1 on page 68 and Section 6.1.2 on page 69, the CONFIG state is entered via WAIT_CONFIG substate from either the CLOSED state, the WAIT_CONNECT state, or the WAIT_CONNECT_RSP state. The CONFIG state is left for the OPEN state if both the initiator and acceptor paths complete successfully.

For better overview, separate tables are given: Table 6.4 shows the success transitions; therein, transitions on one of the minimum paths (no previous non-success transitions) are shaded. Table 6.5 on page 71 shows the non-success transitions within the configuration process, and Table 6.6 on page 72 shows further transition cause by events not belonging to the configuration process itself. The following configuration states and transitions are illustrated in Figure 6.2 on page 78.

Previous state	Event	Condition	Action	Next State
WAIT_CONFIG	*ConfigureChannel_Req*	-	Send L2CAP_ConfigReq	WAIT_CONFIG_REQ_RSP
WAIT_CONFIG	L2CAP_ConfigReq	Options acceptable	Send L2CAP_ConfigRsp (success)	WAIT_SEND_CONFIG

Table 6.4: CONFIG state/substates event table: success transitions within configuration process.

Previous state	Event	Condition	Action	Next State
WAIT_CONFIG_REQ_RSP	L2CAP_ConfigReq	Options acceptable	Send L2CAP_Config Rsp (success)	WAIT_CONFIG_RSP
WAIT_CONFIG_REQ_RSP	L2CAP_ConfigRsp	Remote side accepts options	- (continue waiting for configuration request)	WAIT_CONFIG_REQ
WAIT_CONFIG_REQ	L2CAP_ConfigReq	Options acceptable	Send L2CAP_Config Rsp (success)	OPEN
WAIT_SEND_CONFIG	*ConfigureChannel_Req*	-	Send L2CAP_Config Req	WAIT_CONFIG_RSP
WAIT_CONFIG_RSP	L2CAP_ConfigRsp	Remote side accepts options	-	OPEN

Table 6.4: CONFIG state/substates event table: success transitions within configuration process.

Previous state	Event	Condition	Action	Next State
WAIT_CONFIG	L2CAP_ConfigReq	Options not acceptable	Send L2CAP_Config Rsp (fail)	WAIT_CONFIG
WAIT_CONFIG	L2CAP_ConfigRsp	-	Ignore	WAIT_CONFIG
WAIT_SEND_CONFIG	L2CAP_ConfigRsp	-	Ignore	WAIT_SEND_CONFIG
WAIT_CONFIG_REQ_RSP	L2CAP_ConfigReq	Options not acceptable	Send L2CAP_Config Rsp (fail)	WAIT_CONFIG_REQ_RSP
WAIT_CONFIG_REQ_RSP	L2CAP_ConfigRsp	Remote side rejects options	Send L2CAP_Config Req (new options)	WAIT_CONFIG_REQ_RSP
WAIT_CONFIG_REQ	L2CAP_ConfigReq	Options not acceptable	Send L2CAP_Config Rsp (fail)	WAIT_CONFIG_REQ
WAIT_CONFIG_REQ	L2CAP_ConfigRsp	-	Ignore	WAIT_CONFIG_REQ
WAIT_CONFIG_RSP	L2CAP_ConfigRsp	Remote side rejects options	Send L2CAP_Config Req (new options)	WAIT_CONFIG_RSP

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

Previous state	Event	Condition	Action	Next State
CONFIG (any substate)	*CloseChannel_Req*	Any internal reason to stop	Send L2CAP_ Disconnect Req	WAIT_ DISCONNECT
CONFIG (any substate)	L2CAP_Disconnect Req	-	Send L2CAP_ Disconnect Rsp	CLOSED
CONFIG (any substate)	L2CAP_Disconnect Rsp	-	Ignore	CONFIG (remain in sub-state)
CONFIG (any substate)	L2CAP_Data	-	Process the PDU	CONFIG (remain in sub-state)

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

Notes:

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

6.1.5 OPEN state

Event	Condition	Action	Next State
SendData_req	-	Send L2CAP_Data packet according to configured mode	OPEN
ReconfigureChannel_req	-	Complete outgoing SDU Send L2CAP_ConfigReq	CONFIG (substate WAIT_CONFIG_RSP)
CloseChannel_req	-	Send L2CAP_DisconnectReq	WAIT_DISCONNECT
L2CAP_ConnectRsp	-	Ignore	OPEN
L2CAP_ConfigReq	Incoming config. options acceptable	Complete outgoing SDU Send L2CAP_ConfigRsp (ok)	CONFIG (substate WAIT_CONFIG_REQ)
L2CAP_ConfigReq	Incoming config. options not acceptable	Complete outgoing SDU Send L2CAP_ConfigRsp (fail)	OPEN
L2CAP_DisconnectReq	-	Send L2CAP_DisconnectRsp	CLOSED
L2CAP_DisconnectRsp	-	Ignore	OPEN
L2CAP_Data	-	Process the PDU	OPEN

Table 6.7: OPEN state event table.

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

6.1.6 WAIT_DISCONNECT state

Event	Condition	Action	Next State
L2CAP_ConnectRsp	-	Ignore	WAIT_DISCONNECT
L2CAP_ConfigReq	-	Send L2CAP_CommandReject with reason Invalid CID	WAIT_DISCONNECT
L2CAP_ConfigRsp	-	Ignore	WAIT_DISCONNECT
L2CAP_DisconnectReq	-	Send L2CAP_DisconnectRsp	CLOSED
L2CAP_DisconnectRsp	-	-	CLOSED

Table 6.8: WAIT_DISCONNECT state event table.

Event	Condition	Action	Next State
L2CAP_Data	-	Ignore	WAIT_DISCONNECT

Table 6.8: WAIT_DISCONNECT state event table.

6.2 TIMERS EVENTS

6.2.1 RTX

The Response Timeout eXpired (RTX) timer is used to terminate the channel when the remote endpoint is unresponsive to signalling requests. This timer is started when a signalling request (see Section 7 on page 79) is sent to the remote device. This timer is disabled when the response is received. If the initial timer expires, a duplicate Request message may be sent or the channel identified in the request may be disconnected. If a duplicate Request message is sent, the RTX timeout value shall be reset to a new value at least double the previous value. When retransmitting the Request message, the context of the same state shall be assumed as with the original transmission. If a Request message is received that is identified as a duplicate (retransmission), it shall be processed in the context of the same state which applied when the original Request message was received.

Implementations have the responsibility to decide on the maximum number of Request retransmissions performed at the L2CAP level before terminating the channel identified by the Requests. The one exception is the signalling CID that should never be terminated. The decision should be based on the flush timeout of the signalling link. The longer the flush timeout, the more retransmissions may be performed at the physical layer and the reliability of the channel improves, requiring fewer retransmissions at the L2CAP level.

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

The value of this timer is implementation-dependent but the minimum initial value is 1 second and the maximum initial value is 60 seconds. One RTX timer shall exist for each outstanding 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.

6.2.2 ERTX

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

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

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

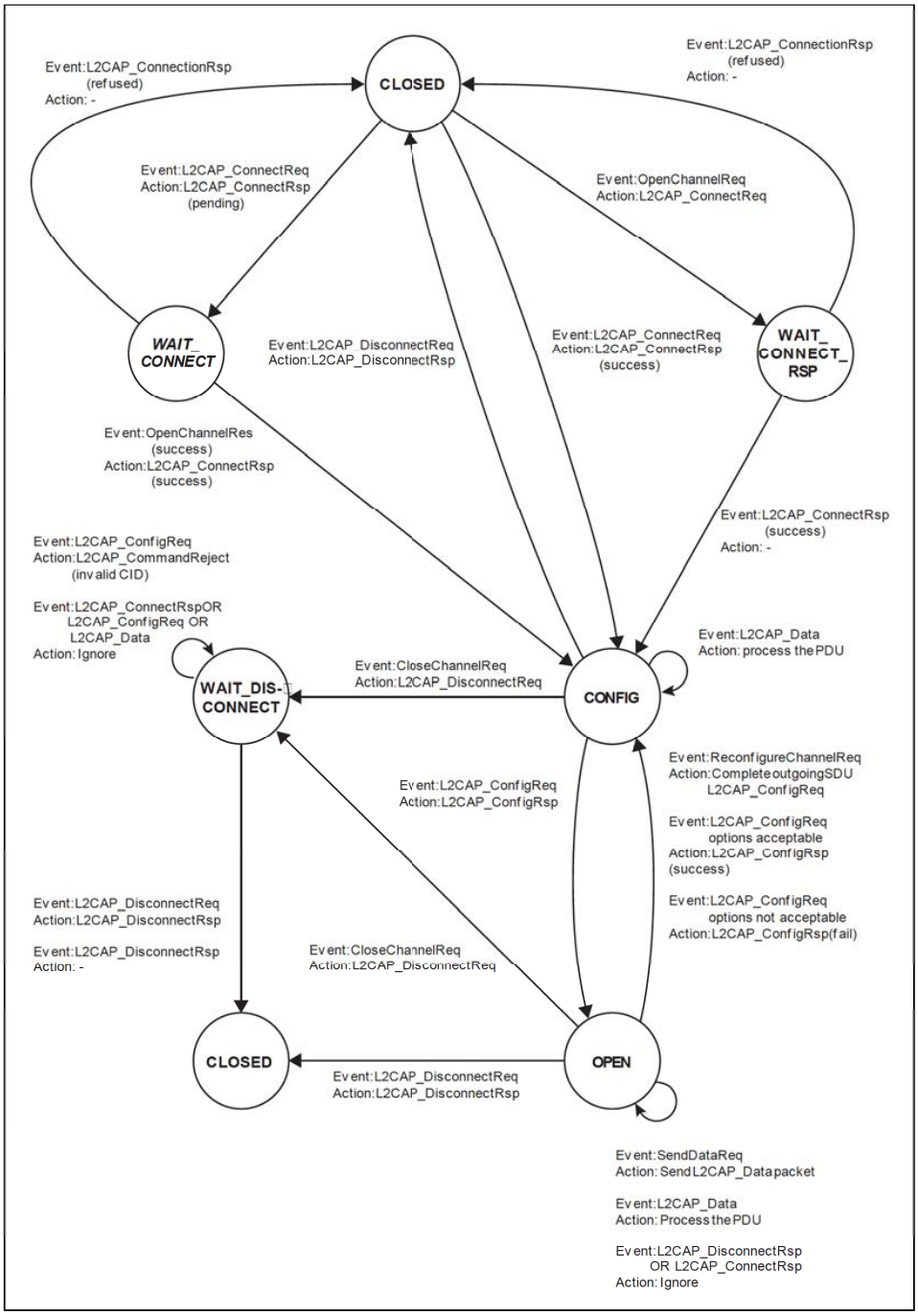


Figure 6.1: States and transitions.

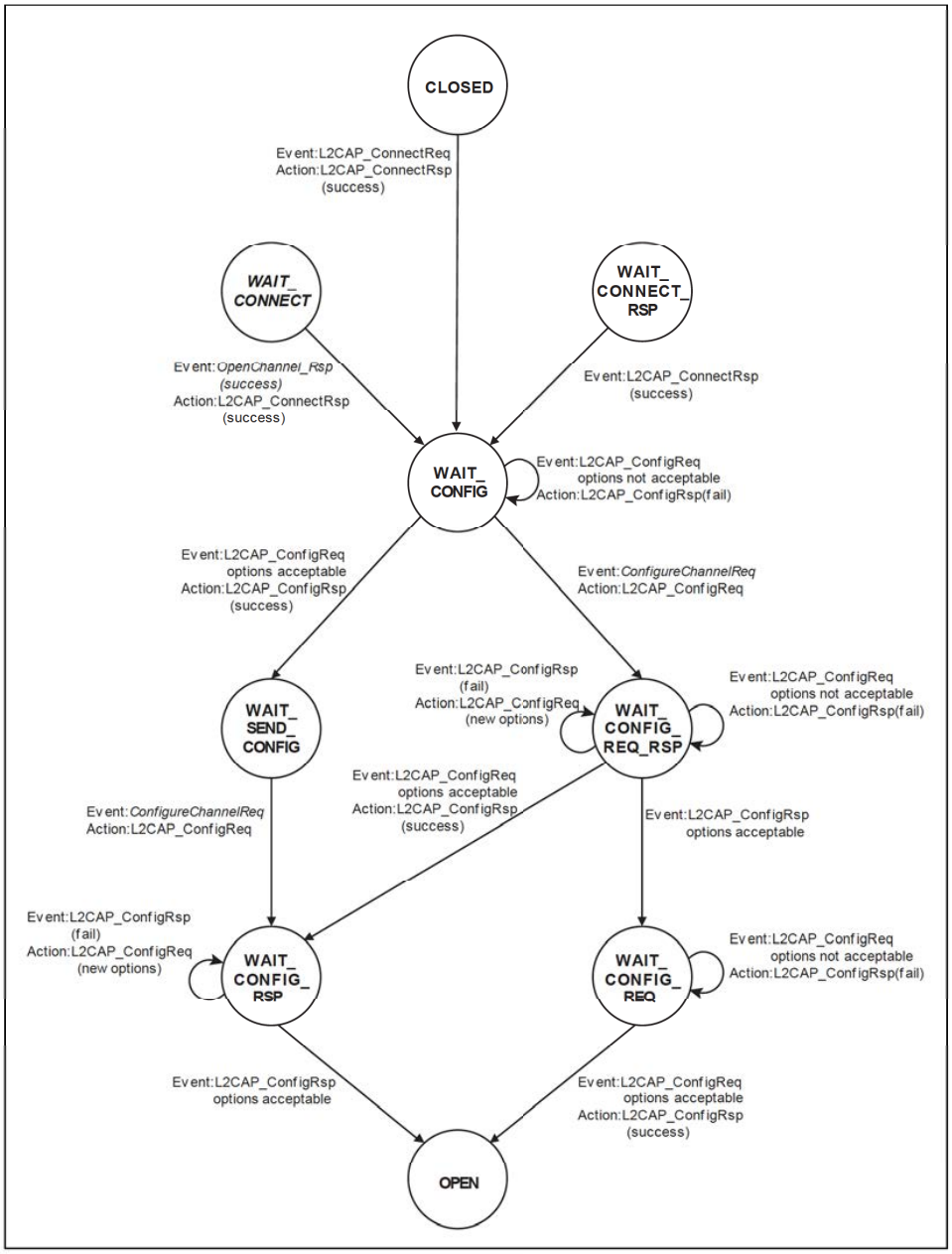


Figure 6.2: Configuration states and transitions

7 GENERAL PROCEDURES

This section describes the general operation of L2CAP, including the configuration process, the handling and the processing of user data for transportation over the air interface. This section also describes the operation of L2CAP features including the delivery of erroneous packets, the flushing of expired data and operation in connectionless mode.

Procedures for the flow control and retransmission modes are described in Section 8 on page 87.

7.1 CONFIGURATION PROCESS

Configuring the channel parameters shall be done independently for both directions. Both configurations may be done in parallel. For each direction the following procedure shall be used:

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 with these values, including the default ones, using a Configuration Response.
3. The local and remote devices repeat steps (1) and (2) until agreement on all parameters is reached.

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

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

7.1.1 Request path

The Request Path can configure the following:

- requester's incoming MTU.
- requester's outgoing flush timeout.
- requester's outgoing QoS.
- requester's incoming flow and error control information.

Table 7.1 on page 80 defines the configuration options that may be placed in a Configuration Request.

Parameter	Description
MTU	Incoming MTU information
FlushTO	Outgoing flush timeout
QoS	Outgoing QoS information
RFCMode	Incoming Retransmission and Flow Control Mode

Table 7.1: Parameters allowed in Request

The state machine for the configuration process is described in Section 6 on page 67.

7.1.2 Response path

The Response Path can configure the following:

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

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

Parameter	Description
MTU	Outgoing MTU information
FlushTO	Incoming flush timeout
QoS	Incoming QoS information
RFCMode	Outgoing Retransmission and Flow Control Mode

Table 7.2: Parameters allowed in Response

7.2 FRAGMENTATION AND RECOMBINATION

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

7.2.1 Fragmentation of L2CAP PDUs

An L2CAP implementation may fragment any L2CAP PDU for delivery to the lower layers. If L2CAP runs directly over the link controller protocol, then an implementation may fragment the PDU into multiple baseband packets for transmission over the air. If L2CAP runs above the host controller interface, then an implementation may send HCI transport sized fragments to the Controller which passes them to the baseband. All L2CAP fragments associated with an L2CAP PDU shall be passed to the baseband before any other L2CAP PDU for the same logical transport shall be sent.

The two LLID bits defined in the first octet of baseband payload (also called the frame header) are used to signal the start and continuation of L2CAP PDUs. LLID shall be '10' for the first segment in an L2CAP PDU and '01' for a continuation segment. An illustration of fragmentation is shown in Figure 7.1 on page 81. An example of how fragmentation might be used in a device with HCI is shown in Figure 7.2 on page 82.

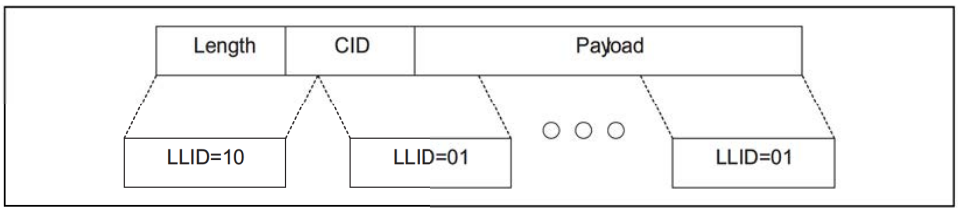


Figure 7.1: L2CAP fragmentation.

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

7.2.2 Recombination of L2CAP PDUs

The link controller protocol attempts to deliver ACL packets in sequence and protects the integrity of the data using a 16-bit CRC. When errors are detected by the baseband it uses an automatic repeat request (ARQ) mechanism.

Recombination of fragments may occur in the Controller but ultimately it is the responsibility of L2CAP to reassemble PDUs and SDUs and to check the length field of the SDUs. As the baseband controller receives ACL packets, it either signals the L2CAP layer on the arrival of each baseband packet, or accumulates a number of packets (before the receive buffer fills up or a timer expires) before passing fragments to the L2CAP layer.

An L2CAP implementation shall use the length field in the header of L2CAP PDUs, see Section 3 on page 33, as a consistency check and shall discard any L2CAP PDUs that fail to match the length field. If channel reliability is not needed, packets with invalid lengths may be silently discarded. For reliable channels, an L2CAP implementation shall indicate to the upper layer that the channel has become unreliable. Reliable channels are defined by having an infinite flush timeout value as specified in Section 5.2 on page 59. For higher data integrity L2CAP should be operated in the Retransmission Mode.

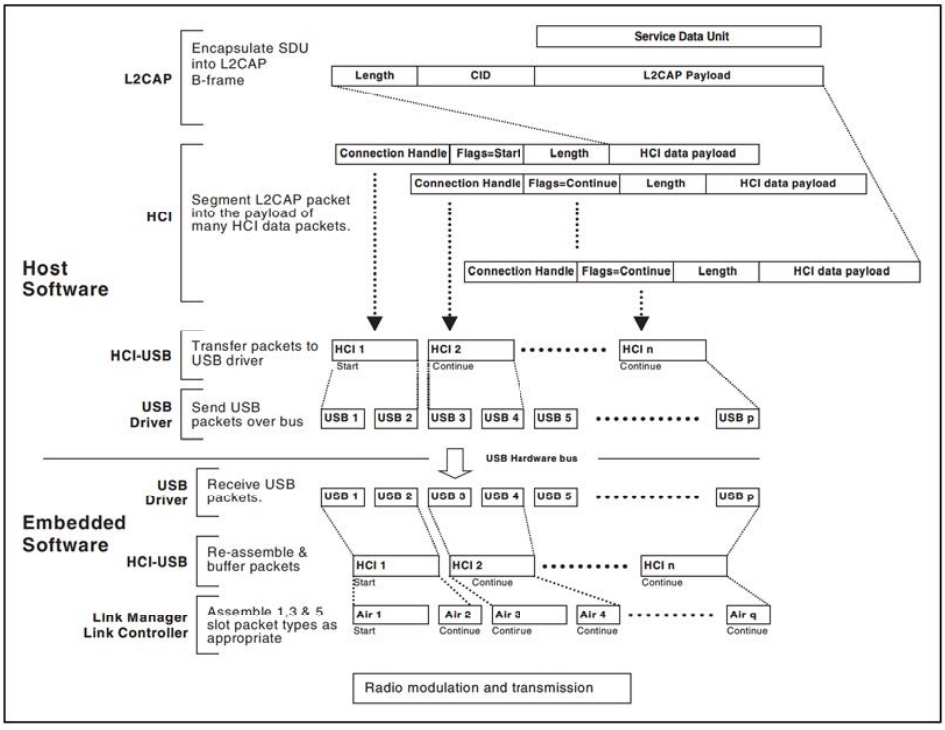


Figure 7.2: Example of fragmentation processes in a device with HCI.

7.3 ENCAPSULATION OF SDUs

All SDUs are encapsulated into one or more L2CAP PDUs.

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

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

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

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

7.3.1 Segmentation of L2CAP SDUs

In Flow Control or Retransmission modes, incoming SDUs may be broken down into segments, which shall then be individually encapsulated with L2CAP protocol elements (header and checksum fields) to form I-frames. I-frames are subject to flow control and may be subject to retransmission procedures. The header carries a 2 bit SAR field that is used to identify whether the I-frame is a 'start', 'end' or 'continuation' packet or whether it carries a complete, unsegmented SDU. Figure 7.3 on page 83 illustrates segmentation and fragmentation.

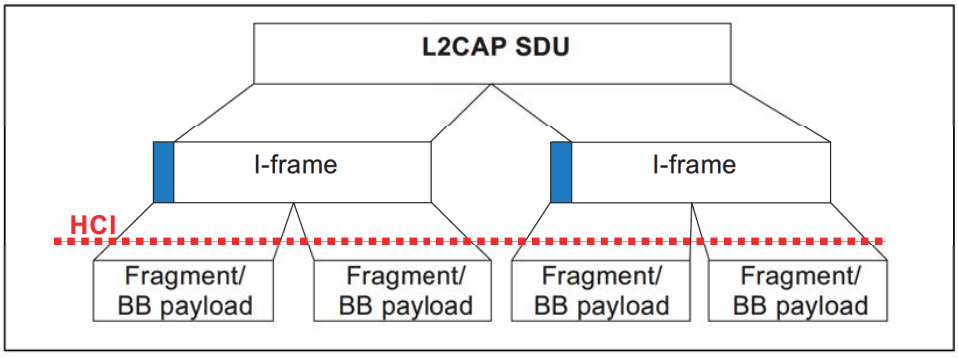


Figure 7.3: Segmentation and fragmentation of an SDU.

7.3.2 Reassembly of L2CAP SDUs

The receiving side uses the SAR field of incoming 'I-frames' for the reassembly process. The L2CAP SDU length field, present in the "start of SDU" I-frame, is an extra integrity check, and together with the sequence numbers may be used to indicate lost L2CAP SDUs to the application. Figure 7.3 on page 83 illustrates segmentation and fragmentation.

7.3.3 Segmentation and fragmentation

Figure 7.4 on page 84 illustrates the use of segmentation and fragmentation operations to transmit a single SDU. Note that while SDUs and L2CAP PDUs are transported in peer-to-peer fashion, the fragment size used by the Fragmentation and Recombination routines is implementation specific and may not be the same in the sender and the receiver. The over-the-air sequence of baseband packets as created by the sender is common to both devices.

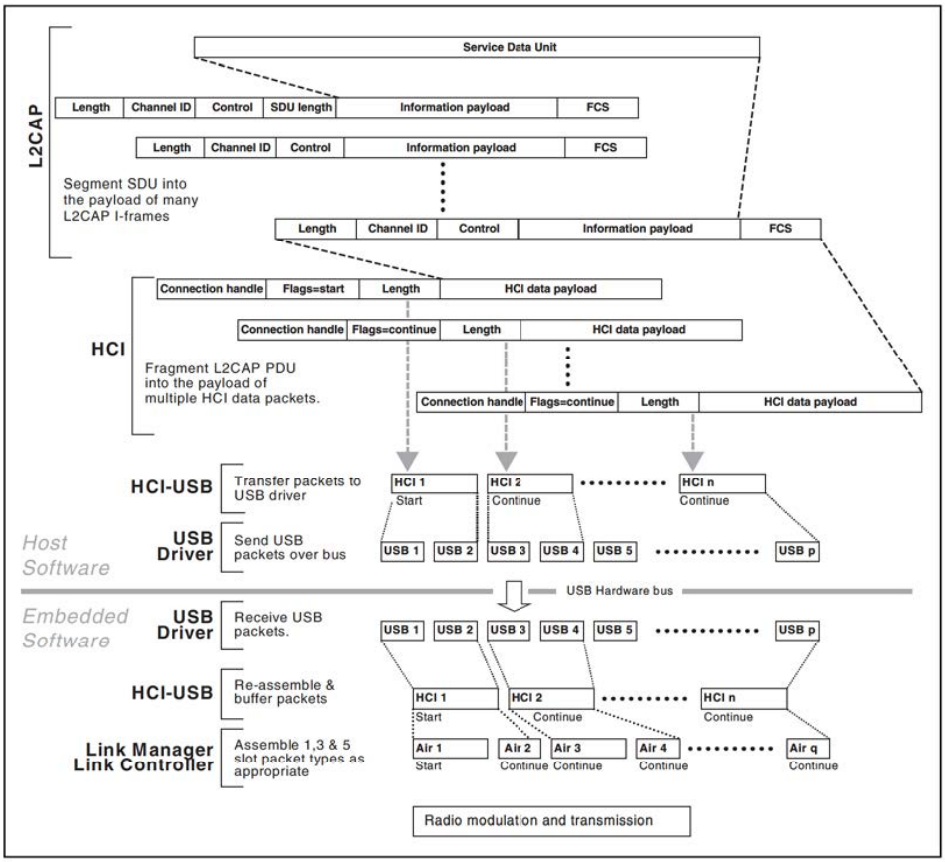


Figure 7.4: Example of segmentation and fragment processes in a device with HCI[1]

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

7.4 DELIVERY OF ERRONEOUS L2CAP SDUS

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

7.5 OPERATION WITH FLUSHING

In the L2CAP configuration, the Flush Time-Out may be set separately per L2CAP channel, but there is only one flush mechanism per ACL logical transport in the baseband.

When there is more than one L2CAP channel mapped to the same ACL logical transport, the automatic flush time-out does not discriminate between L2CAP channels. The automatic flush time-out flushes a specific L2CAP PDU. The HCI Flush command flushes all outstanding L2CAP PDUs for the ACL logical transport. Therefore, care has to be taken when using the Automatic Flush Time-out and the HCI Flush command:

1. For any connection to be reliable at the L2CAP level, it should use L2CAP retransmission mode if it is mapped to an ACL logical transport with a finite automatic flush time-out. In retransmission mode, loss of flushed L2CAP PDUs on the channel is detected by the L2CAP ARQ mechanism and they are retransmitted.
2. There is only one automatic flush time-out setting per ACL logical transport. Therefore, all time bounded L2CAP channels on an ACL logical transport with a flush time-out setting should configure the same flush time-out value at the L2CAP level.
3. If Automatic Flush Time-out is used, then it should be taken into account that it only flushes one L2CAP PDU. If one PDU has timed out and needs flushing, then others on the same logical transport are also likely to need flushing. Therefore, when retransmission mode is used, flushing should be handled by the HCI Flush command so that all outstanding L2CAP PDUs are flushed.

7.6 CONNECTIONLESS DATA CHANNEL

In addition to connection-oriented channels, L2CAP also has a connectionless channel. The connectionless channel allows transmission to all members of the piconet. Data sent through the connectionless channel is sent in a best-effort manner. The connectionless channel has no quality of service and is unreliable. L2CAP makes no guarantee that data sent through the connectionless channel successfully reaches all members of the piconet. If reliable group transmission is required, it must be implemented at a higher layer.

Transmissions to the connectionless channel will be sent to all members of the piconet. If this data is not for transmission to all members of the piconet, then higher level encryption is required to support private communication.

The local device will not receive transmissions on the conectionless channel, therefore, higher layer protocols must loopback any data traffic being sent to the local device.

An L2CAP service interface could provide basic group management mechanisms including creating a group, adding members to a group, and removing members from a group.

Connectionless data channels shall not be used with Retransmission Mode or Flow Control Mode.

8 PROCEDURES FOR FLOW CONTROL AND RETRANSMISSION

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

8.1 INFORMATION RETRIEVAL

Before attempting to configure flow control- or retransmission mode on a channel, it is mandatory to verify that the suggested mode is supported by performing an information retrieval for the "Extended features supported" information type (0x0002). If the information retrieval is not successful or the "Extended features mask" bit is not set for the wanted mode, the mode shall not be suggested in a configuration request.

8.2 FUNCTION OF PDU TYPES FOR FLOW CONTROL AND RETRANSMISSION

Two frame formats are defined for flow control and retransmission modes (see Section 3.3 on page 35). The I-frame is used to transport user information instead of the B-frame. The S-frame is used for signalling.

8.2.1 Information frame (I-frame)

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

8.2.2 Supervisory Frame (S-frame)

The S-frame is used to control the transmission of I-frames. The S-frame has two formats: Receiver Ready (RR) and Reject (REJ).

8.2.2.1 Receiver Ready (RR)

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

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

The RR frame has no information field.

8.2.2.2 Reject (REJ)

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

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

8.3 VARIABLES AND SEQUENCE NUMBERS

The sending peer uses the following variables and Sequence numbers:

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

The receiving peer uses the following variables and Sequence numbers:

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

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

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

8.3.1 Sending peer

8.3.1.1 Send sequence number TxSeq

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

8.3.1.2 Send state variable NextTXSeq

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

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

8.3.1.3 Acknowledge state variable ExpectedAckSeq

The CID sent in the information frame is the destination CID and identifies the remote endpoint of the channel. An acknowledge state variable ExpectedAck-Seq shall be maintained for each remote endpoint. ExpectedAckSeq is the sequence number of the next in-sequence I-frame that the remote receiving peer is expected to acknowledge. (ExpectedAckSeq – 1 equals the TxSeq of the last acknowledged I-frame). When the link is created ExpectedAckSeq shall be initialized to 0.
Note that if the next acknowledgement acknowledges a single I-frame then it's ReqSeq will be expectedAckSeq + 1.

If a valid ReqSeq is received from the peer then ExpectedAckSeq is set to ReqSeq. A valid ReqSeq value is one that is in the range ExpectedAckSeq ≤ ReqSeq ≤ NextTxSeq.

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

These inequalities shall be interpreted in the following way: ReqSeq is valid, if and only if (ReqSeq-ExpectedAckSeq) mod 64 ≤ (NextTXSeq-ExpectedAck-Seq) mod 64. Furthermore, from the description of NextTXSeq, it can be seen that (NextTXSeq-ExpectedAckSeq) mod 64 ≤ TxWindow.

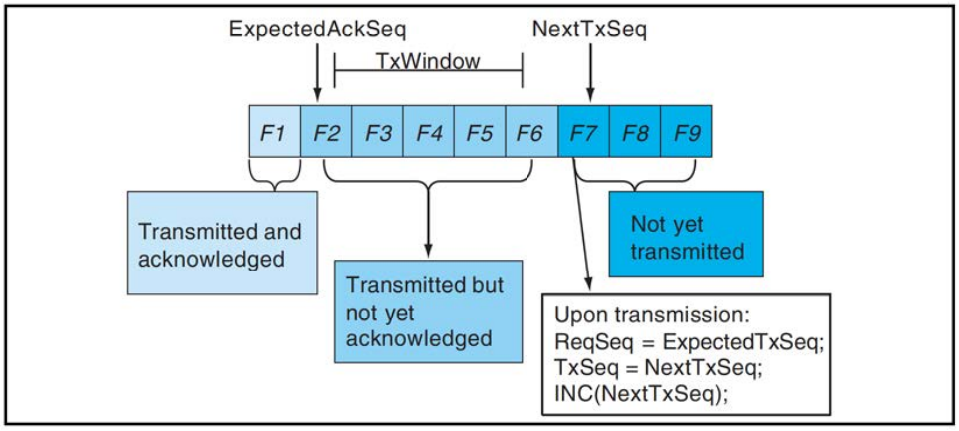


Figure 8.1: Example of the transmitter side

Figure 8.1 on page 90 shows TxWindow=5, and three frames awaiting transmission. The frame with number *F7* may be transmitted when the frame with *F2* is acknowledged. When the frame with *F7* is transmitted, TxSeq is set to the value of NextTXSeq. After TxSeq has been set, NextTxSeq is incremented.

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

8.3.2 Receiving peer

8.3.2.1 Receive sequence number ReqSeq

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

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

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

8.3.2.2 Receive state variable, ExpectedTxSeq

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

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

8.3.2.3 Buffer state variable BufferSeq

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

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

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

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

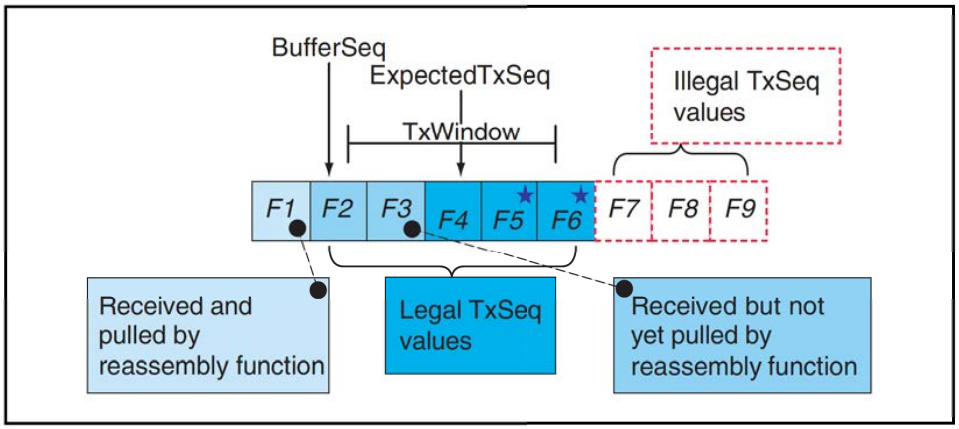


Figure 8.2: Example of the receiver side

Figure 8.2 on page 92 shows TxWindow=5. *F1* is successfully received and pulled by the upper layer. BufferSeq shows that *F2* is the next I-frame to be pulled, and ExpectedTxSeq points to the next I-frame expected from the peer. An I-frame with TxSeq equal to 5 has been received thus triggering an REJ exception. The star indicates I-frames received but discarded owing to the REJ exception. They will be resent as part of the error recovery procedure.

In Figure 8.2 on page 92 there are several I-frames in a buffer awaiting the SDU reassembly function to pull them and the TxWindow is full. The receiver would usually disable retransmission by setting the Retransmission Disable Bit to 1 and send an RR back to the sending side. This tells the transmitting peer that there is no point in performing retransmissions. Both sides will send S-frames to make sure the peer entity knows the current value of the Retransmission Disable Bit.

8.4 RETRANSMISSION MODE

8.4.1 Transmitting frames

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

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

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

8.4.1.1 Last received R was set to zero

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

a) If unacknowledged I-frames have been sent and the RetransmissionTimer has elapsed then an unacknowledged I-frame shall be retransmitted. The RetransmissionTimer shall be restarted.

b) If unacknowledged I-frames have been sent, the Retransmission timer has not elapsed then a new I-frame shall be sent if one is waiting and no timer action shall be taken.

c) If no unacknowledged I-frames have been sent, and a new I-frame is waiting, then the new I-frame shall be sent, the RetransmissionTimer shall be started and if the MonitorTimer is running, it shall be stopped.

d) If no unacknowledged I-frames have been sent and no new I-frames are waiting to be transmitted, and the RetransmissionTimer is running, then the retransmission timer shall be stopped and the monitor timer shall be started.

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

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

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

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

a) If the MonitorTimer is running and has not elapsed then no transmit action shall be taken and no timer action shall be taken.

b) If the MonitorTimer has elapsed then an S-frame shall be sent and the MonitorTimer shall be restarted.

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

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

8.4.1.2 Last received R was set to one

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

a) If the MonitorTimer is running and has not elapsed then no transmit action shall be taken and no timer action shall be taken.

b) If the MonitorTimer has elapsed then an S-frame shall be sent and the MonitorTimer shall be restarted.

8.4.2 Receiving I-frames

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

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

The ReqSeq shall be processed according to Section 8.4.6 on page 97.

If a valid I-frame with TxSeq ≠ ExpectedTxSeq is received then an exception condition shall be triggered which is handled according to Section 8.4.7 on page 97.

8.4.3 I-frames pulled by the SDU reassembly function

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

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

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

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

8.4.4 Sending and receiving acknowledgements

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

8.4.4.1 Sending acknowledgements

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

8.4.4.2 Receiving acknowledgements

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

The following rules shall be applied:

1. If the RetransmissionDisableBit changed value from 0 to 1 (stop retransmissions) then the receiving entity shall
    a) If the RetransmissionTimer is running then stop it and start the MonitorTimer.
    b) Store the state of the RetransmissionDisableBit received.
2. If the RetransmissionDisableBit changed value from 1 to 0 (start retransmissions) then the receiving entity shall
    a) Store the state of the RetransmissionDisableBit received.
    b) If there are any I-frames that have been sent but not acknowledged, then stop the MonitorTimer and start the RetransmissionTimer.
    c) Any buffered I-frames shall be transmitted according to Section 8.4.1 on page 93.
3. If any unacknowledged I-frames were acknowledged by the ReqSeq contained in the frame, and the RetransmissionDisableBit equals 1 (retransmissions stopped), then the receiving entity shall
    a) Follow the rules in Section 8.4.1 on page 93.
4. If any unacknowledged I-frames were acknowledged by the ReqSeq contained in the frame and the RetransmissionDisableBit equals 0 (retransmissions started) then the receiving entity shall
    a) If the RetransmissionTimer is running, then stop it.
    b) If any unacknowledged I-frames have been sent then the RetransmissionTimer shall be restarted.

c) Follow the rules in Section 8.4.1 on page 93.

d) If the RetransmissionTimer is not running and the MonitorTimer is not running, then start the MonitorTimer.

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

8.4.5 Receiving REJ frames

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

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

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

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

8.4.6 Waiting acknowledgements

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

1. If the TransmitCounter is less than MaxTransmit then:
   a) Increment the TransmitCounter
   b) Retransmit the last unacknowledged I-frame, according to Section 8.4.1 on page 93.
2. If the TransmitCounter is equal to MaxTransmit this channel to the peer entity shall be assumed lost. The channel shall move to the CLOSED state and appropriate action shall be taken to report this to the upper layers.

8.4.7 Exception conditions

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

8.4.7.1 TxSeq Sequence error

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

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

- *Duplicated I-frame*

  The duplicated I-frame is identified by a TxSeq in the range BufferSeq to ExpectedTxSeq – 1 (BufferSeq ≤TxSeq<ExpectedTxSeq). The ReqSeq and RetransmissionDisableBit shall be processed according to Section 8.4.4 on page 96. The Information field shall be discarded since it has already been received.

- *Out-of-sequence I-frame*

  The out-of-sequence I-frame is identified by a TxSeq within the legal range. The ReqSeq and RetransmissionDisableBit shall be processed according to Section 8.4.4 on page 96.

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

- *Invalid TxSeq*

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

8.4.7.2 ReqSeq Sequence error

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

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

8.4.7.3 Timer recovery error

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

The L2CAP entity that transmitted an unacknowledged I-frame shall, on the expiry of the RetransmissionTimer, take appropriate recovery action as defined in Section 8.4.6 on page 97.

8.4.7.4 Invalid frame

Any frame received which is invalid (as defined in Section 3.3.6 on page 40) shall be discarded, and no action shall be taken as a result of that frame.

8.5 FLOW CONTROL MODE

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

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

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

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

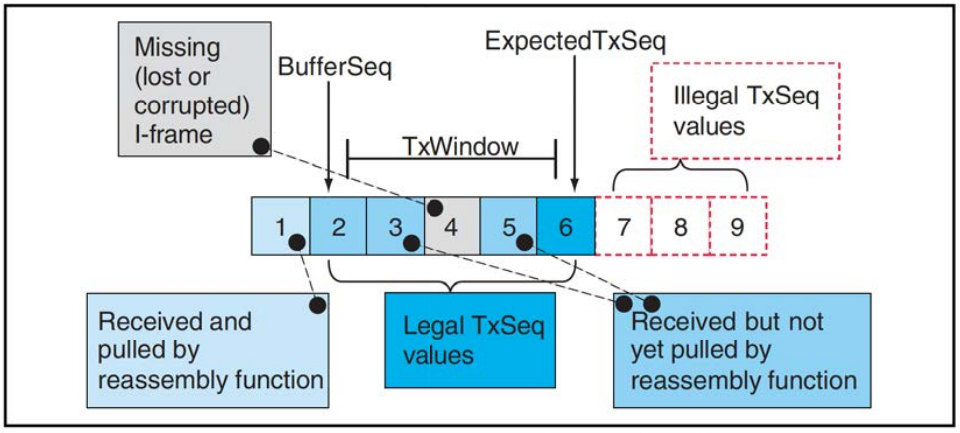


Figure 8.3: Overview of the receiver side when operating in flow control mode

8.5.1 Transmitting I-frames

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

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

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

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

8.5.2 Receiving I-frames

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

Upon receipt of a valid I-frame with an out-of-sequence TxSeq (see Section 8.5.6 on page 101) all frames with a sequence number less than TxSeq shall be assumed lost and marked as missing. The missing I-frames are in the range from ExpectedTxSeq (the frame that the device was expecting to receive) up to TxSeq-1, (the frame that the device actually received). ExpectedTxSeq shall be set to TxSeq +1. The received I-frame shall be made available for pulling by the reassembly function. The acknowledgement shall not occur until the SDU reassembly function has pulled the I-frame. The ReqSeq shall be processed according to Section 8.5.4 on page 100.

8.5.3 I-frames pulled by the SDU reassembly function

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

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

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

8.5.4 Sending and receiving acknowledgements

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

8.5.4.1 Sending acknowledgements

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

8.5.4.2 Receiving acknowledgements

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

1. If any outstanding I-frames were acknowledged then
    a) Stop the RetransmissionTimer
    b) If there are still unacknowledged I-frames then restart the RetransmissionTimer, otherwise start the MonitorTimer.
    c) Transmit any I-frames awaiting transmission according to Section 8.5.1 on page 99.

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

8.5.5 Waiting acknowledgements

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

1. If I-frames are waiting to be sent
    a) The RetransmissionTimer is restarted
    b) I-frames awaiting transmission are transmitted according to Section 8.5.1 on page 99.
2. If there are no I-frames waiting to be sent
    a) If there are still unacknowledged I-frames the RetransmissionTimer is restarted, otherwise the MonitorTimer is started.

8.5.6 Exception conditions

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

8.5.6.1 TxSeq Sequence error

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

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

- *Duplicated I-frame*

  The duplicated I-frame is identified by a TxSeq in the range BufferSeq to ExpectedTxSeq – 1. The ReqSeq shall be processed according to Section 8.5.4 on page 100. The Information field shall be discarded since it has already been received.

- *Out-of-sequence I-frame*

  The out-of-sequence I-frame is identified by a TxSeq within the legal range ExpectedTxSeq < TxSeq < (BufferSeq + TxWindow). The ReqSeq shall be processed according to Section 8.5.4 on page 100.

  The missing I-frame(s) are considered lost and ExpectedTXSeq is set equal to TxSeq+1 as specified in Section 8.5.2 on page 100. The missing I-frame(s) are reported as lost to the SDU reassembly function.

- *Invalid TxSeq*

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

8.5.6.2 ReqSeq Sequence error

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

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

8.5.6.3 Timer recovery error

An L2CAP entity that fails to receive an acknowledgement for an I-frame shall, on the expiry of the RetransmissionTimer, take appropriate recovery action as defined in Section 8.5.5 on page 101.

8.5.6.4 Invalid frame

Any frame received that is invalid (as defined in Section 3.3.6 on page 40) shall be discarded, and no action shall be taken as a result of that frame, unless the receiving L2CAP entity is configured to deliver erroneous frames to the layer above L2CAP. In that case, the data contained in invalid frames may also be added to the receive buffer and made available for pulling from the SDU reassembly function.

9 LIST OF FIGURES

Figure 1.1: L2CAP within protocol layers 19
Figure 1.2: L2CAP data flows in Bluetooth Protocol Architecture 20
Figure 1.3: L2CAP architectural blocks 20
Figure 2.1: Channels between devices 28
Figure 2.2: L2CAP transaction model. 29
Figure 3.1: PDU format in Basic L2CAP mode on connection-oriented channels (field sizes in bits) 31
Figure 3.2: L2CAP PDU format in Basic L2CAP mode on Connectionless channel 32
Figure 3.3: L2CAP PDU formats in Flow Control and Retransmission Modes 33
Figure 3.4: The LFSR circuit generating the FCS. 37
Figure 3.5: Initial state of the FCS generating circuit. 37
Figure 4.1: L2CAP PDU format on the signalling channel 39
Figure 4.2: Command format 39
Figure 4.3: Command Reject packet 41
Figure 4.4: Connection Request Packet 42
Figure 4.5: Connection Response Packet 44
Figure 4.6: Configuration Request Packet 46
Figure 4.7: Configuration Request Flags field format 46
Figure 4.8: Configuration Response Packet 48
Figure 4.9: Configuration Response Flags field format 48
Figure 4.10: Disconnection Request Packet 50
Figure 4.11: Disconnection Response Packet 51
Figure 4.12: Echo Request Packet 51
Figure 4.13: Echo Response Packet 52
Figure 4.14: Information Request Packet 52
Figure 4.15: Information Response Packet 53
Figure 5.1: Configuration option format 55
Figure 5.2: MTU Option Format 56
Figure 5.3: Flush Timeout option format. 57
Figure 5.4: Quality of Service (QoS) option format containing Flow Specification. 59
Figure 5.5: Retransmission and Flow Control option format. 62
Figure 6.1: States and transitions. 75
Figure 6.2: Configuration states and transitions. 76
Figure 7.1: L2CAP fragmentation. 79
Figure 7.2: Example of fragmentation processes in a device with HCI. 80
Figure 7.3: Segmentation and fragmentation of an SDU. 81
Figure 7.4: Example of segmentation and fragment processes in a device with HCI 82

Figure 8.1: Example of the transmitter side 88
Figure 8.2: Example of the receiver side 90
Figure 8.3: Overview of the receiver side when operating in flow control mode 97

10 LIST OF TABLES

Table 1.1: Terminology....24
Table 2.1: CID name space....27
Table 2.2: Types of Channel Identifiers....28
Table 3.1: Control Field formats....34
Table 3.2: SAR control element format....35
Table 3.3: S control element format: type of S-frame....35
Table 4.1: Signalling Command Codes....40
Table 4.2: Reason Code Descriptions....41
Table 4.3: Reason Data values....42
Table 4.4: Defined PSM Values....43
Table 4.5: Result values....44
Table 4.6: Status values....45
Table 4.7: Configuration Response Result codes....49
Table 4.8: InfoType definitions....52
Table 4.9: Information Response Result values....53
Table 4.10: Information Response Data fields....54
Table 4.11: Extended feature mask....54
Table 5.1: Service type definitions....59
Table 5.2: Mode definitions....62
Table 6.1: CLOSED state event table....66
Table 6.2: WAIT_CONNECT_RSP state event table....67
Table 6.3: WAIT_CONNECT state event table....67
Table 6.4: CONFIG state/substates event table: success transitions within configuration process....68
Table 6.5: CONFIG state/substates event table: non-success transitions within configuration process....69
Table 6.6: CONFIG state/substates event table: events not related to configuration process....70
Table 6.7: OPEN state event table....71
Table 6.8: WAIT_DISCONNECT state event table....71
Table 7.1: Parameters allowed in Request....78
Table 7.2: Parameters allowed in Response....78
Table 8.1: Summary of actions when the RetransmissionTimer is in use and R=0....92



APPENDIX A: CONFIGURATION MSCs

The examples in this appendix describe a sample of the multiple possible configuration scenarios that might occur.

Figure I illustrates the basic configuration process. In this example, the devices exchange MTU information. All other values are assumed to be default.

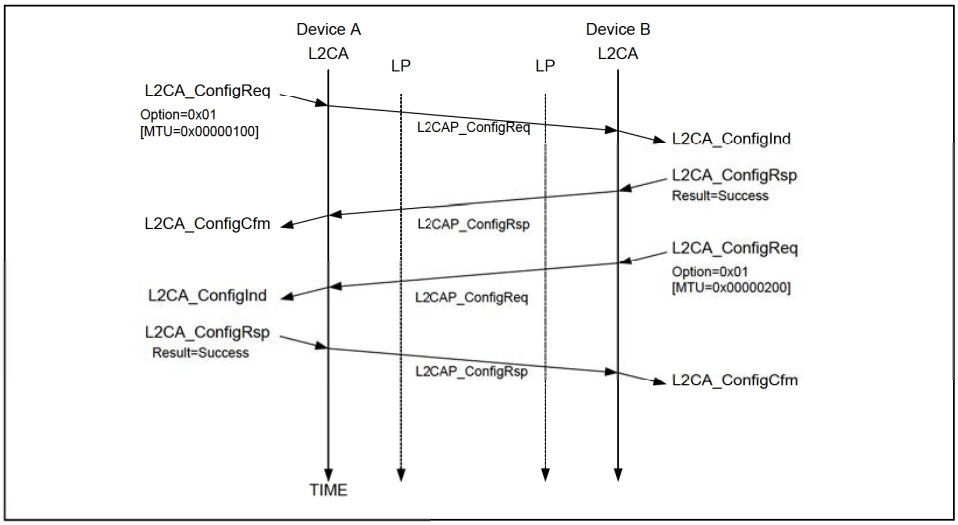


Figure I: Basic MTU exchange

Figure II on page 108 illustrates how two devices interoperate even though one device supports more options than the other does. Device A is an upgraded version. It uses a hypothetically defined option type 0x20 for link-level security. Device B rejects the command using the Configuration Response packet with result 'unknown parameter' informing Device A that option 0x20 is not understood. Device A then resends the request omitting option 0x20. Device B notices that it does not need to such a large MTU and accepts the request but includes in the response the MTU option informing Device A that Device B will not send an L2CAP packet with a payload larger than 0x80 octets over this channel. On receipt of the response, Device A could reduce the buffer allocated to hold incoming traffic.

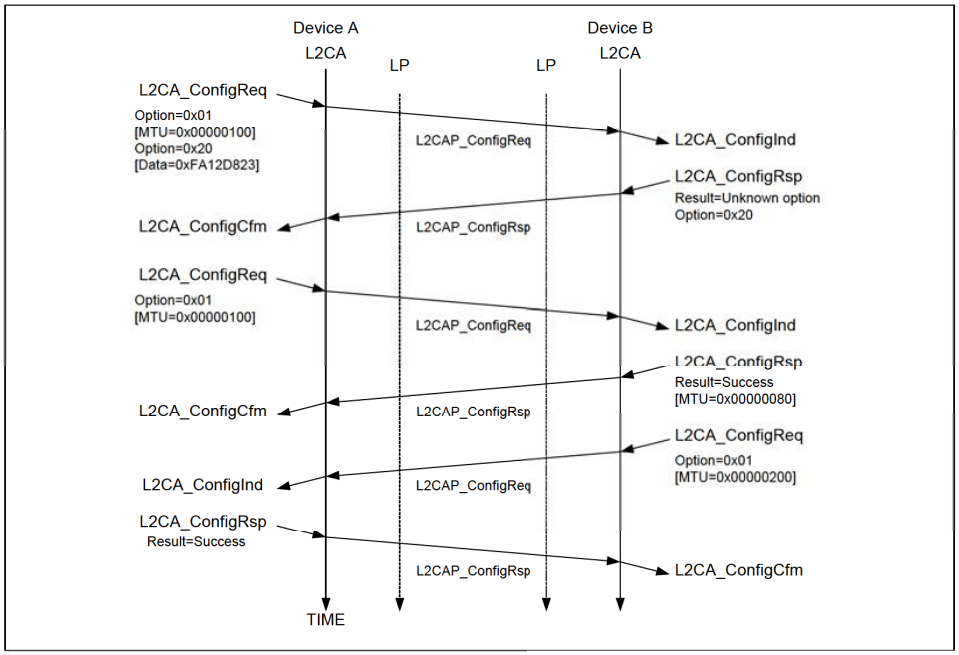


Figure II: Dealing with Unknown Options

Figure III on page 109 illustrates an unsuccessful configuration request. There are two problems described by this example. The first problem is that the configuration request is placed in an L2CAP packet that cannot be accepted by the remote device, due to its size. The remote device informs the sender of this problem using the Command Reject message. Device A then resends the configuration options using two smaller L2CAP_ConfigReq messages.

The second problem is an attempt to configure a channel with an invalid CID. For example device B may not have an open connection on that CID (0x01234567 in this example case).

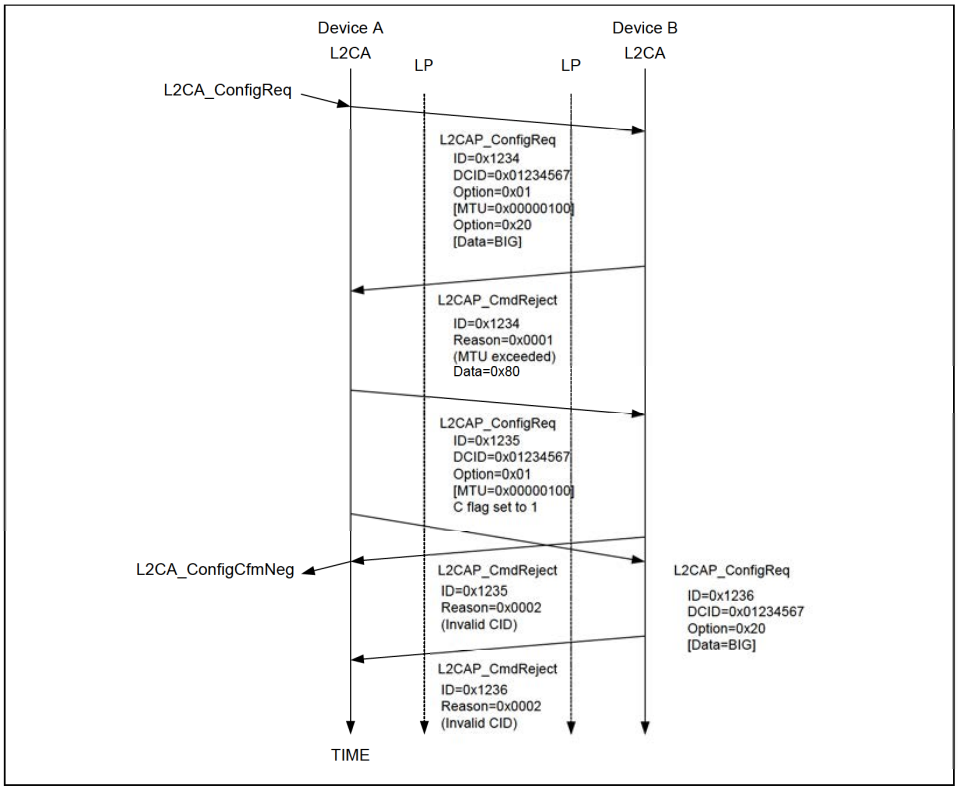


Figure III: Unsuccessful Configuration Request





SERVICE DISCOVERY PROTOCOL (SDP)

This specification defines a protocol for locating services provided by or available through a Bluetooth device.

CONTENTS

1 Introduction 115
1.1 General Description 115
1.2 Motivation 115
1.3 Requirements 115
1.4 Non-requirements and Deferred Requirements 116
1.5 Conventions 116
1.5.1 Bit And Byte Ordering Conventions 116
2 Overview 117
2.1 SDP Client-Server Interaction 117
2.2 Service Record 118
2.3 Service Attribute 120
2.4 Attribute ID 120
2.5 Attribute Value 121
2.6 Service Class 121
2.6.1 A Printer Service Class Example 122
2.7 Searching for Services 123
2.7.1 UUID 123
2.7.2 Service Search Patterns 124
2.8 Browsing for Services 124
2.8.1 Example Service Browsing Hierarchy 125
3 Data Representation 127
3.1 Data Element 127
3.2 Data Element Type Descriptor 127
3.3 Data Element Size Descriptor 128
3.4 Data Element Examples 129
4 Protocol Description 131
4.1 Transfer Byte Order 131
4.2 Protocol Data Unit Format 131
4.3 Partial Responses and Continuation State 133
4.4 Error Handling 133
4.4.1 SDP_ErrorResponse PDU 134
4.5 ServiceSearch Transaction 135
4.5.1 SDP_ServiceSearchRequest PDU 135
4.5.2 SDP_ServiceSearchResponse PDU 136
4.6 ServiceAttribute Transaction 138
4.6.1 SDP_ServiceAttributeRequest PDU 138
4.6.2 SDP_ServiceAttributeResponse PDU 140
4.7 ServiceSearchAttribute Transaction 141

4.7.1 SDP_ServiceSearchAttributeRequest PDU 141
4.7.2 SDP_ServiceSearchAttributeResponse PDU 143

5 Service Attribute Definitions 145
5.1 Universal Attribute Definitions 145
5.1.1 ServiceRecordHandle Attribute 145
5.1.2 ServiceClassIDList Attribute 146
5.1.3 ServiceRecordState Attribute 146
5.1.4 ServiceID Attribute 146
5.1.5 ProtocolDescriptorList Attribute 147
5.1.6 BrowseGroupList Attribute 148
5.1.7 LanguageBaseAttributeIDList Attribute 148
5.1.8 ServiceInfoTimeToLive Attribute 149
5.1.9 ServiceAvailability Attribute 150
5.1.10 BluetoothProfileDescriptorList Attribute 150
5.1.11 DocumentationURL Attribute 151
5.1.12 ClientExecutableURL Attribute 151
5.1.13 IconURL Attribute 152
5.1.14 ServiceName Attribute 152
5.1.15 ServiceDescription Attribute 153
5.1.16 ProviderName Attribute 153
5.1.17 Reserved Universal Attribute IDs 153
5.2 ServiceDiscoveryServer Service Class Attribute Definitions ... 154
5.2.1 ServiceRecordHandle Attribute 154
5.2.2 ServiceClassIDList Attribute 154
5.2.3 VersionNumberList Attribute 154
5.2.4 ServiceDatabaseState Attribute 155
5.2.5 Reserved Attribute IDs 155
5.3 BrowseGroupDescriptor Service Class Attribute Definitions ... 155
5.3.1 ServiceClassIDList Attribute 155
5.3.2 GroupID Attribute 156
5.3.3 Reserved Attribute IDs 156

6 Appendix 156

1 INTRODUCTION

1.1 GENERAL DESCRIPTION

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

1.2 MOTIVATION

Service Discovery in the Bluetooth environment, where the set of services that are available changes dynamically based on the RF proximity of devices in motion, is qualitatively different from service discovery in traditional network-based environments. The service discovery protocol defined in this specification is intended to address the unique characteristics of the Bluetooth environment. See Section , “Appendix A – Background Information,” on page 157, for further information on this topic.

1.3 REQUIREMENTS

The following capabilities have been identified as requirements for version 1.0 of the Service Discovery Protocol.

1. SDP shall provide the ability for clients to search for needed services based on specific attributes of those services.
2. SDP shall permit services to be discovered based on the class of service.
3. SDP shall enable browsing of services without a priori knowledge of the specific characteristics of those services.
4. SDP shall provide the means for the discovery of new services that become available when devices enter RF proximity with a client device as well as when a new service is made available on a device that is in RF proximity with the client device.
5. SDP shall provide a mechanism for determining when a service becomes unavailable when devices leave RF proximity with a client device as well as when a service is made unavailable on a device that is in RF proximity with the client device.
6. SDP shall provide for services, classes of services, and attributes of services to be uniquely identified.
7. SDP shall allow a client on one device to discover a service on another device without consulting a third device.
8. SDP should be suitable for use on devices of limited complexity.
9. SDP shall provide a mechanism to incrementally discover information about the services provided by a device. This is intended to minimize the quantity

of data that must be exchanged in order to determine that a particular service is not needed by a client.

10. SDP should support the caching of service discovery information by intermediary agents to improve the speed or efficiency of the discovery process.
11. SDP should be transport independent.
12. SDP shall function while using L2CAP as its transport protocol.
13. SDP shall permit the discovery and use of services that provide access to other service discovery protocols.
14. SDP shall support the creation and definition of new services without requiring registration with a central authority.

1.4 NON-REQUIREMENTS AND DEFERRED REQUIREMENTS

The Bluetooth SIG recognizes that the following capabilities are related to service discovery. These items are not addressed in SDP version 1.0. However, some may be addressed in future revisions of the specification.

1. SDP 1.0 does not provide access to services. It only provides access to information about services.
2. SDP 1.0 does not provide brokering of services.
3. SDP 1.0 does not provide for negotiation of service parameters.
4. SDP 1.0 does not provide for billing of service use.
5. SDP 1.0 does not provide the means for a client to control or change the operation of a service.
6. SDP 1.0 does not provide an event notification when services, or information about services, become unavailable.
7. SDP 1.0 does not provide an event notification when attributes of services are modified.
8. This specification does not define an application programming interface for SDP.
9. SDP 1.0 does not provide support for service agent functions such as service aggregation or service registration.

1.5 CONVENTIONS

1.5.1 Bit And Byte Ordering Conventions

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

Multiple-byte fields are drawn with the more significant bytes toward the left and the less significant bytes toward the right. Multiple-byte fields are transferred in network byte order. See section 4.1 on page 131.

2 OVERVIEW

2.1 SDP CLIENT-SERVER INTERACTION

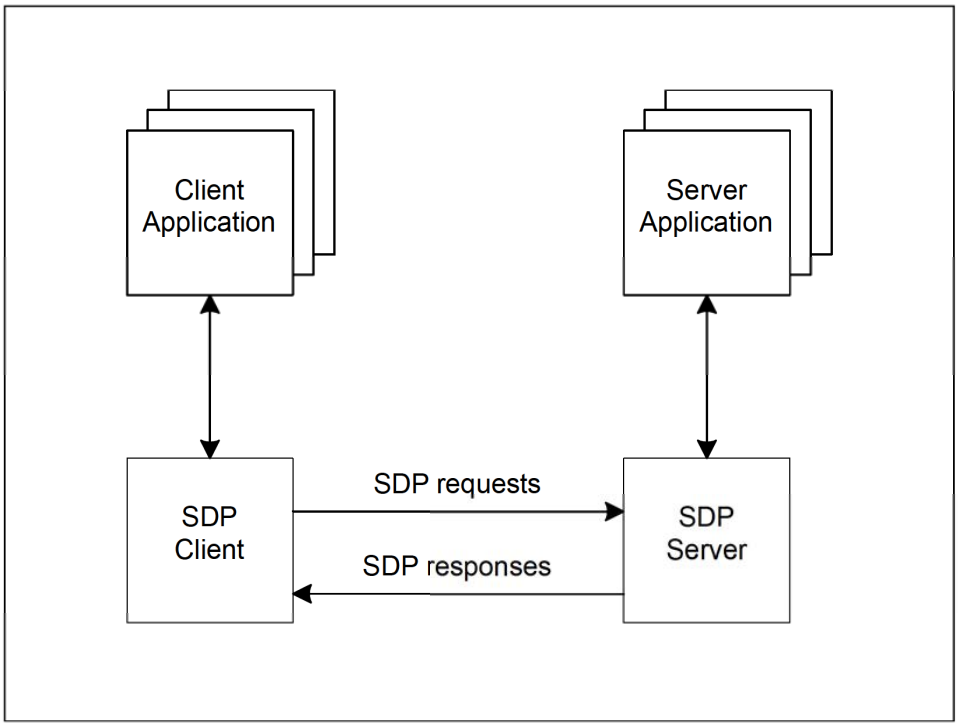


Figure 2.1:

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 may be simplified to that shown in Figure 2.1 may be simplified to that shown in Figure 2.2.

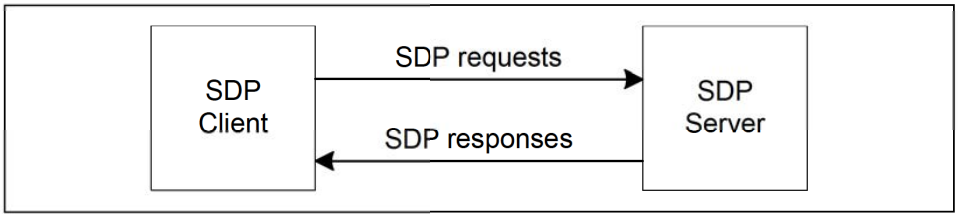


Figure 2.2:

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.

Additional information regarding application interaction with SDP is contained in the Bluetooth Service Discovery Profile document.

2.2 SERVICE RECORD

A service is any entity that can provide information, perform an action, or control a resource on behalf of another entity. A service may be implemented as software, hardware, or a combination of hardware and software.

All of the information about a service that is maintained by an SDP server is contained within a single service record. The service record consists entirely of a list of service attributes.

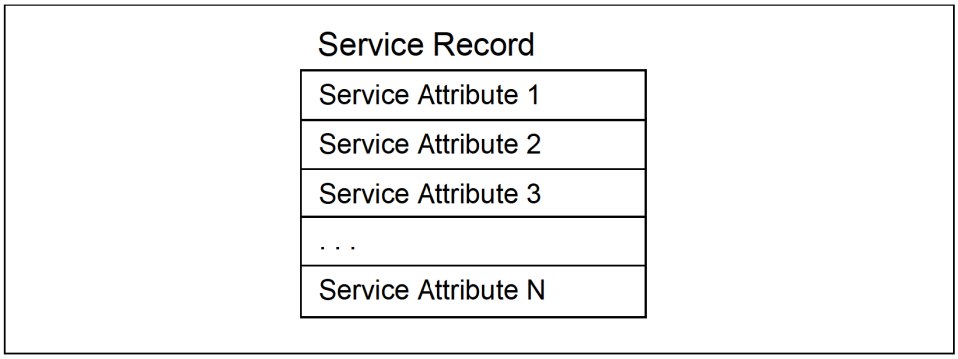


Figure 2.3: Service Record

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 re-used while an L2CAP connection remains established. Note that service record handles are known to remain valid across successive L2CAP connections while the ServiceDatabaseState attribute value remains unchanged. See the ServiceRecordState and ServiceDatabaseState attributes in section 5 on page 145.

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.

2.3 SERVICE ATTRIBUTE

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

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

See section 5.1 on page 145, for attribute definitions that are common to all service records. Service providers can also define their own service attributes.

A service attribute consists of two components: an attribute ID and an attribute value.

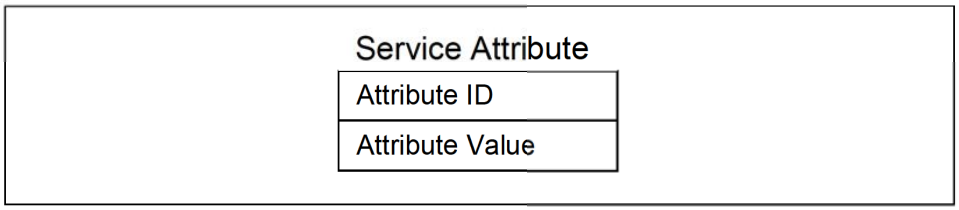


Figure 2.4: Service Attribute

2.4 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. See section 2.6 on page 121.

In the Service Discovery Protocol, an attribute ID is often represented as a data element. See section 3 on page 127.

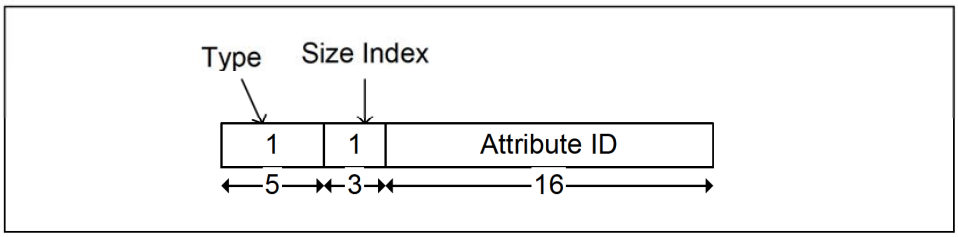


Figure 2.5:

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. (See section 3 on page 127.) Generally, any type of data element is permitted as an attribute value, subject to the constraints specified in the service class definition that assigns an attribute ID to the attribute and assigns a meaning to the attribute value. See section 5 on page 145, for attribute value examples.

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 (see section 2.7.1 on page 123). 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 should be examined or verified before any class-specific attributes are used. Since all of the attributes in a service record must conform to all of the service's classes, the service class identifiers contained in the ServiceClassIDList attribute are related. Typically, each service class is a subclass

of another class whose identifier is contained in the list. A service subclass definition differs from its superclass in that the subclass contains additional attribute definitions that are specific to the subclass. 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 superclass. 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.

2.6.1 A Printer Service Class Example

A color postscript printer with duplex capability might conform to 4 ServiceClass definitions and have a ServiceClassIDList with UUIDs (See section 2.7.1 on page 123.) representing the following ServiceClasses:

DuplexColorPostscriptPrinterServiceClassID,
ColorPostscriptPrinterServiceClassID,
PostscriptPrinterServiceClassID,
PrinterServiceClassID

Note that this example is only illustrative. This may not be a practical printer class hierarchy.

2.7 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.

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 com-

1. The format of UUIDs is defined by the International Organization for Standardization in ISO/IEC 11578:1996. “Information technology – Open Systems Interconnection – Remote Procedure Call (RPC)”

pared, the shorter UUID must be converted to the longer UUID format before comparison.

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.

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.

2.8.1 Example Service Browsing Hierarchy

Here is a fictitious service browsing hierarchy that may illuminate the manner in which browse group descriptors are used. Browse group descriptor service records are identified with (G); other service records with (S).

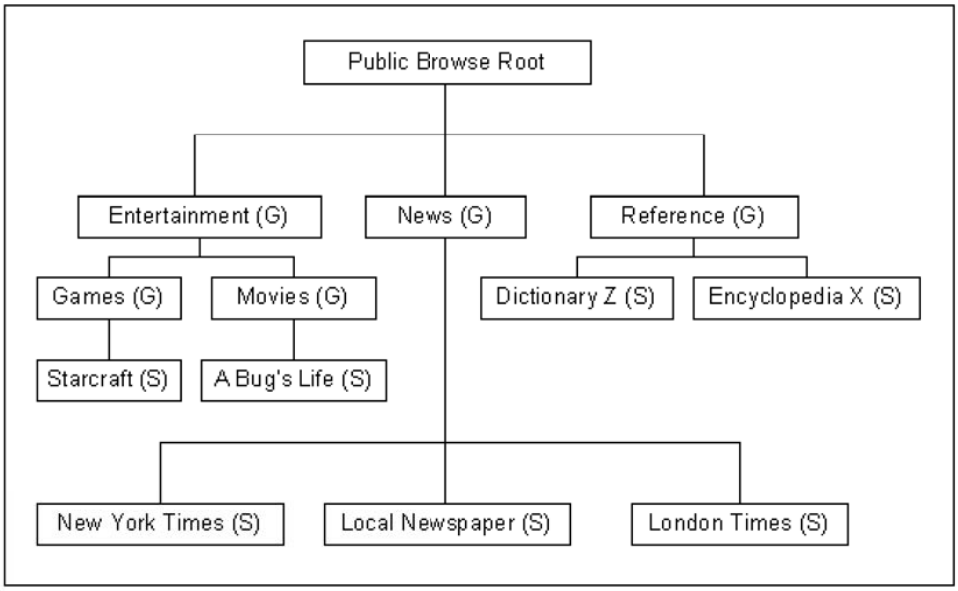


Figure 2.6:

This table shows the services records and service attributes necessary to implement the browse hierarchy.

Service Name	Service Class	Attribute Name	Attribute Value
Entertainment	BrowseGroupDescriptor	BrowseGroupList	PublicBrowseRoot
		GroupID	EntertainmentID
News	BrowsegroupDescriptor	BrowseGroupList	PublicBrowseRoot
		GroupID	NewsID
Reference	BrowseGroupDescriptor	BrowseGroupList	PublicBrowseRoot
		GroupID	ReferenceID
Games	BrowseGroupDescriptor	BrowseGroupList	EntertainmentID
		GroupID	GamesID
Movies	BrowseGroupDescriptor	BrowseGroupList	EntertainmentID
		GroupID	MoviesID
Starcraft	Video Game Class ID	BrowseGroupList	GamesID
A Bug's Life	Movie Class ID	BrowseGroupList	MovieID
Dictionary Z	Dictionary Class ID	BrowseGroupList	ReferenceID
Encyclopedia X	Encyclopedia Class ID	BrowseGroupList	ReferenceID
New York Times	Newspaper ID	BrowseGroupList	NewspaperID
London Times	Newspaper ID	BrowseGroupList	NewspaperID
Local Newspaper	Newspaper ID	BrowseGroupList	NewspaperID

Table 2.1:

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.

3.1 DATA ELEMENT

A data element is a typed data representation. It consists of two fields: a header field and a data field. The header field, in turn, is composed of two parts: a type descriptor and a size descriptor. The data is a sequence of bytes whose length is specified in the size descriptor (described in section 3.3 on page 128) and whose meaning is (partially) specified by the type descriptor.

3.2 DATA ELEMENT TYPE DESCRIPTOR

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

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

Table 3.1: Data Element Type.

1. False is represented by the value 0, and true is represented by the value 1. However, to maximize interoperability, any non-zero value received must be accepted as representing true.

3.3 DATA ELEMENT SIZE DESCRIPTOR

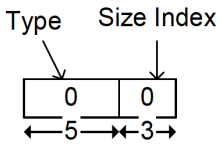
The data element size descriptor is represented as a 3-bit size index followed by 0, 8, 16, or 32 bits. The size index is contained in the least significant (low-order) 3 bits of the first byte of the data element header. The size index is encoded as follows.

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

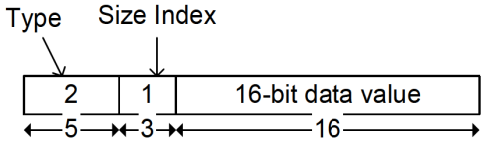
Table 3.2:

3.4 DATA ELEMENT EXAMPLES

Nil is represented as:



A 16-bit signed integer is represented as:



The 3 character ASCII string "Hat" is represented as:

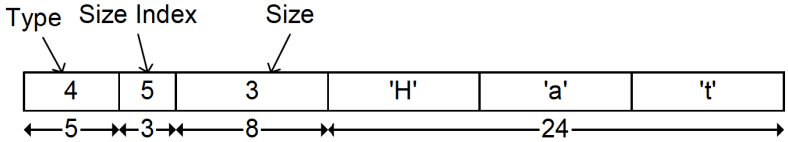


Figure 3.1:

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.

The protocol examples found in Appendix B – Example SDP Transactions, may be helpful in understanding the protocol transactions.

4.1 TRANSFER BYTE ORDER

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

4.2 PROTOCOL DATA UNIT FORMAT

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

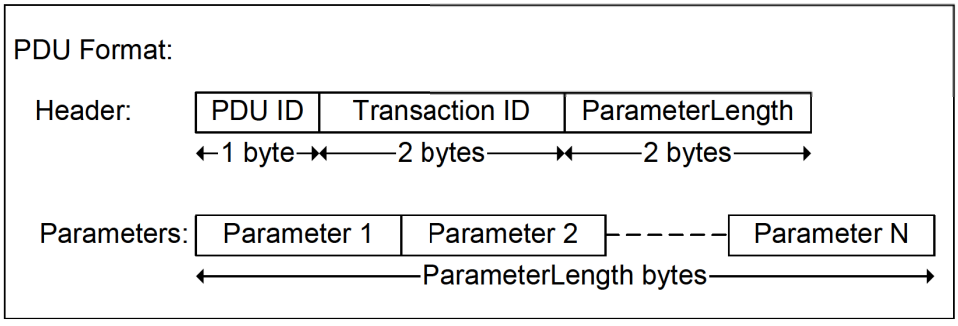


Figure 4.1:

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

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.

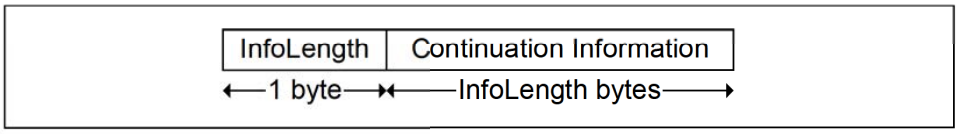


Figure 4.2: Continuation State Format

After a client receives a partial response and the accompanying continuation state parameter, it can re-issue the original request (with a new transaction ID) and include the continuation state in the new request indicating to the server that the remainder of the original response is desired. The maximum allowable value of the InfoLength field is 16 (0x10).

Note that 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.

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.

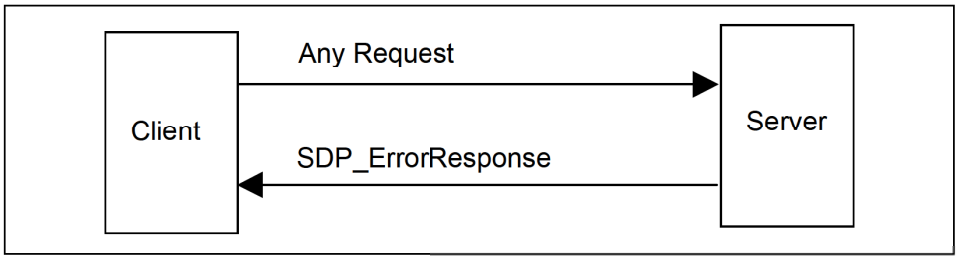


Figure 4.3:

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.

4.5 SERVICESEARCH TRANSACTION

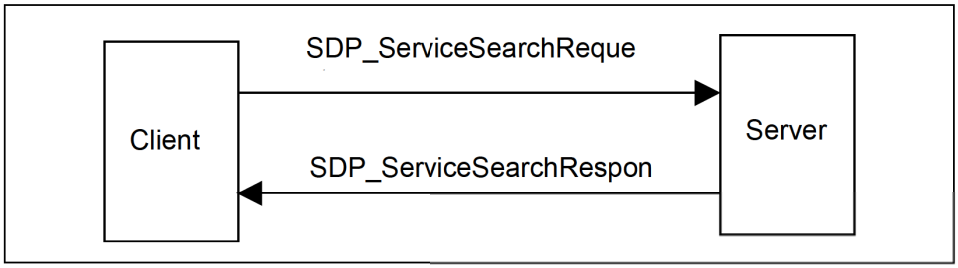


Figure 4.4:

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.

Note that no mechanism is provided to request information for all service records. However, see section 2.8 on page 124 for a description of a mechanism that permits browsing for non-specific services without a priori knowledge of the services.

PDU Parameters:

ServiceSearchPattern: *Size: Varies*

Value	Parameter Description
Data Element Sequence	The ServiceSearchPattern is a data element sequence where each element in the sequence is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in the sequence is 12[1]. The list of UUIDs constitutes a service search pattern.

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

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.

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.

4.6 SERVICEATTRIBUTE TRANSACTION

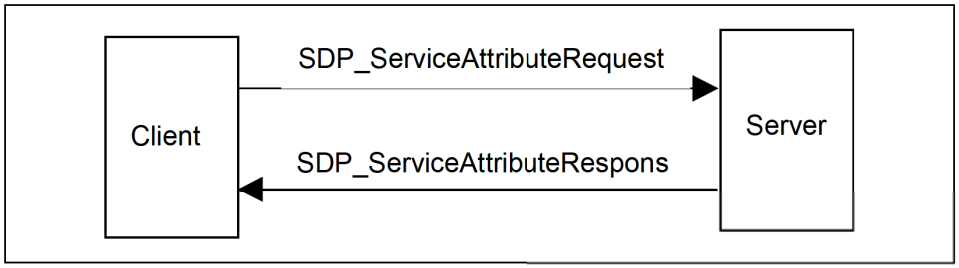


Figure 4.5:

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.

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 MaximumAttributeByteCount value specified in the SDP_ServiceAttributeRequest. Range: 0x0002-0xFFFF

AttributeList: *Size: AttributeListByteCount*

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

ContinuationState: *Size: 1 to 17 Bytes*

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

4.7 SERVICESEARCHATTRIBUTE TRANSACTION

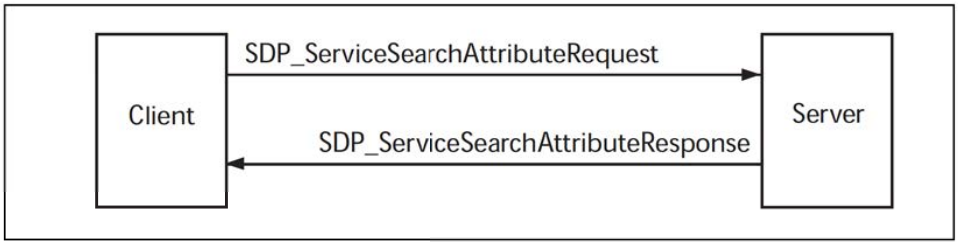


Figure 4.6:

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.

Note that the service record handle for each service record is contained in the ServiceRecordHandle attribute of that service and may be requested along with other attributes.

PDU Parameters:

ServiceSearchPattern: *Size: Varies*

Value	Parameter Description
Data Element Sequence	The ServiceSearchPattern is a data element sequence where each element in the sequence is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in the sequence is 12[1]. The list of UUIDs constitutes a service search pattern.

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

MaximumAttributeByteCount: *Size: 2 Bytes*

Value	Parameter Description
N	MaximumAttributeByteCount specifies the maximum number of bytes of attribute data to be returned in the response to this request. The SDP server 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.

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.

5 SERVICE ATTRIBUTE DEFINITIONS

The service classes and attributes contained in this document are necessarily a partial list of the service classes and attributes supported by SDP. Only service classes that directly support the SDP server are included in this document. Additional service classes will be defined in other documents and possibly in future revisions of this document. Also, it is expected that additional attributes will be discovered that are applicable to a broad set of services; these may be added to the list of Universal attributes in future revisions of this document.

5.1 UNIVERSAL ATTRIBUTE DEFINITIONS

Universal attributes are those service attributes whose definitions are common to all service records. Note that this does not mean that every service record 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.

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.

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.

5.1.3 ServiceRecordState Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceRecordState	0x0002	32-bit unsigned integer

Description:

The ServiceRecordState is a 32-bit integer that is used to facilitate caching of ServiceAttributes. If this attribute is contained in a service record, its value is guaranteed to change when any other attribute value is added to, deleted from or changed within the service record. This permits a client to check the value of this single attribute. If its value has not changed since it was last checked, the client knows that no other attribute values within the service record have changed.

5.1.4 ServiceID Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceID	0x0003	UUID

Description:

The ServiceID is a UUID that universally and uniquely identifies the service instance described by the service record. This service attribute is particularly useful if the same service is described by service records in more than one SDP server.

5.1.5 ProtocolDescriptorList Attribute

Attribute Name	Attribute ID	Attribute Value Type
ProtocolDescriptorList	0x0004	Data Element Sequence or Data Element Alternative

Description:

The ProtocolDescriptorList attribute describes one or more protocol stacks that may be used to gain access to the service described by the service record.

If the ProtocolDescriptorList describes a single stack, it takes the form of a data element sequence in which each element of the sequence is a protocol descriptor. Each protocol descriptor is, in turn, a data element sequence whose first element is a UUID identifying the protocol and whose successive elements are protocol-specific parameters. Potential protocol-specific parameters are a protocol version number and a connection-port number. The protocol descriptors are listed in order from the lowest layer protocol to the highest layer protocol used to gain access to the service.

If it is possible for more than one kind of protocol stack to be used to gain access to the service, the ProtocolDescriptorList takes the form of a data element alternative where each member is a data element sequence as described in the previous paragraph.

Protocol Descriptors

A protocol descriptor identifies a communications protocol and provides protocol-specific parameters. A protocol descriptor is represented as a data element sequence. The first data element in the sequence must be the UUID that identifies the protocol. Additional data elements optionally provide protocol-specific information, such as the L2CAP protocol/service multiplexer (PSM) and the RFCOMM server channel number (CN) shown below.

ProtocolDescriptorList Examples

These examples are intended to be illustrative. The parameter formats for each protocol are not defined within this specification.

In the first two examples, it is assumed that a single RFCOMM instance exists on top of the L2CAP layer. In this case, the L2CAP protocol specific information (PSM) points to the single instance of RFCOMM. In the last example, two different and independent RFCOMM instances are available on top of the L2CAP layer. In this case, the L2CAP protocol specific information (PSM) points to a distinct identifier that distinguishes each of the RFCOMM instances. According to the L2CAP specification, this identifier takes values in the range 0x1000-0xFFFF.