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

4.4.2 Searching and Browsing Services

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

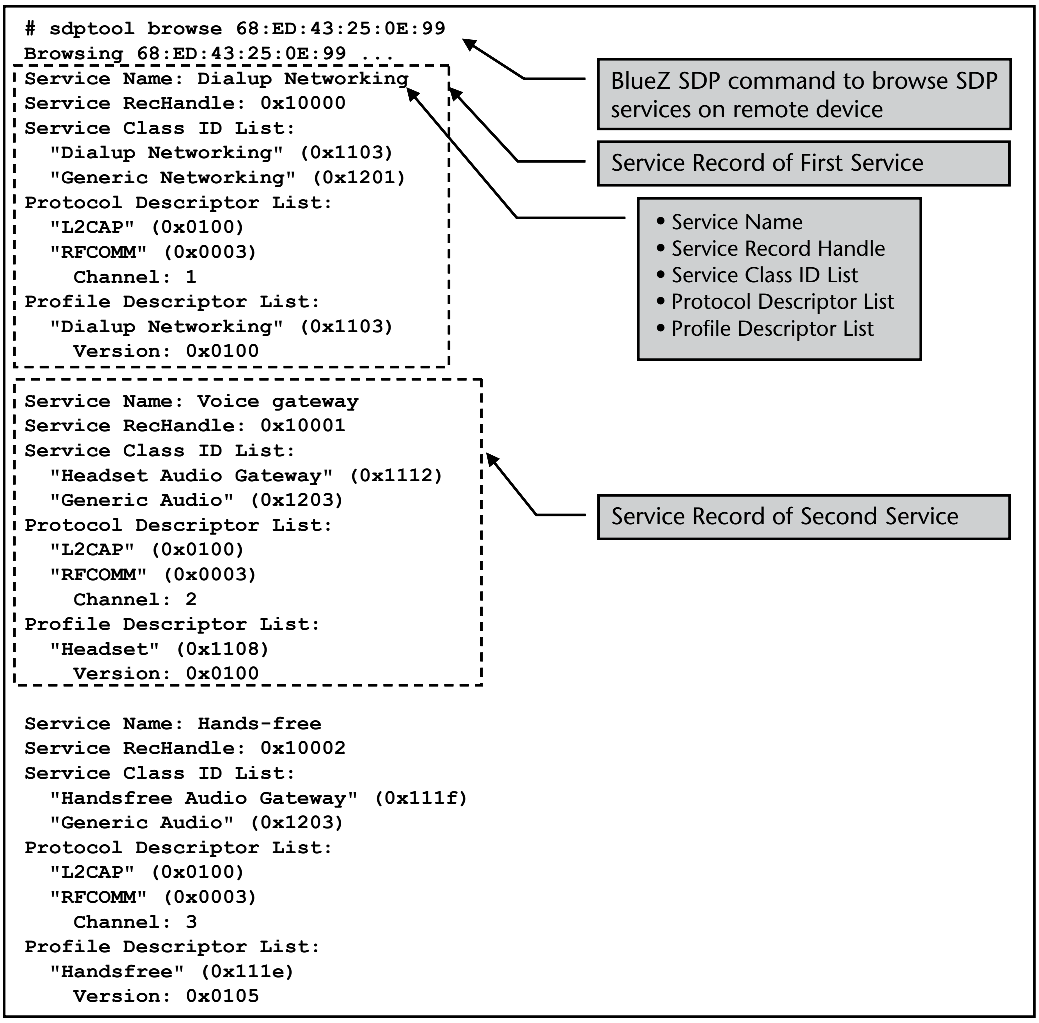
- Search for Services.
- Browse for Services.

Using the Service Search transaction, the client can search for service records for some desired characteristics. These characteristics are in the form of Attribute Values. The search is in the form of a list of UUIDs to be searched. A service search pattern is formed by the SDP client which is a list of all UUIDs to search. This service search pattern is sent to the SDP server in the service search request. This pattern is matched on the SDP server side if all the UUIDs in the list are contained in any specific service record. A handle to this service record is returned if a match is found.

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

Table 4.4 Commonly Used Service Attributes

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



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

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

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

BlueZ SDP command to browse SDP services on remote device
Service Record of First Service
• Service Name
• Service Record Handle
• Service Class ID List
• Protocol Descriptor List
• Profile Descriptor List
Service Record of Second Service

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

4.4.3 SDP Transactions

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

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

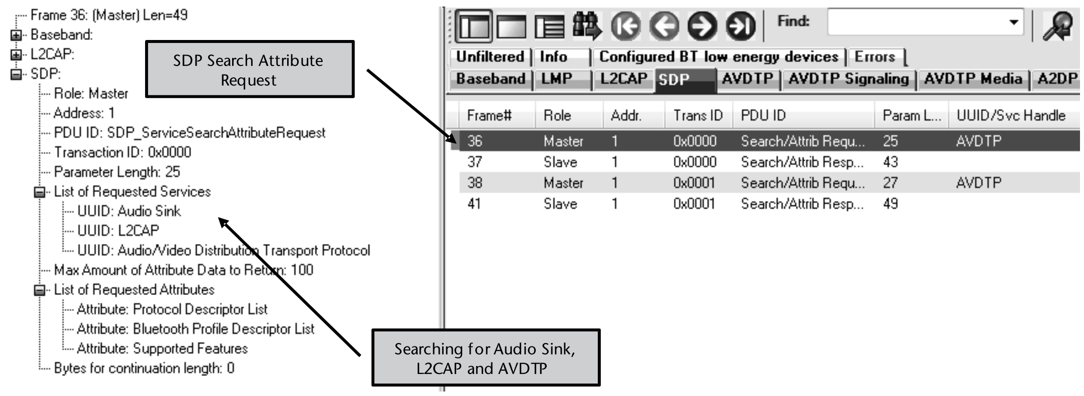
An example of the SDP transactions between a mobile phone and an A2DP headset is shown in Figure 4.7 and Figure 4.8.

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

- Audio Sink;

Table 4.5 SDP Transactions

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


```
Frame 36: (Master) Len=49
⊞ Baseband:
⊞ L2CAP:
⊟ SDP:
   Role: Master
   Address: 1
   PDU ID: SDP_ServiceSearchAttributeRequest
   Transaction ID: 0x0000
   Parameter Length: 25
   ⊟ List of Requested Services
      UUID: Audio Sink
      UUID: L2CAP
      UUID: Audio/Video Distribution Transport Protocol
   Max Amount of Attribute Data to Return: 100
   ⊟ List of Requested Attributes
      Attribute: Protocol Descriptor List
      Attribute: Bluetooth Profile Descriptor List
      Attribute: Supported Features
   Bytes for continuation length: 0
```

Frame#	Role	Addr.	Trans ID	PDU ID	Param L...	UUID/Svc Handle
36	Master	1	0x0000	Search/Attrib Requ...	25	AVDTP
37	Slave	1	0x0000	Search/Attrib Resp...	43	
38	Master	1	0x0001	Search/Attrib Requ...	27	AVDTP
41	Slave	1	0x0001	Search/Attrib Resp...	49	

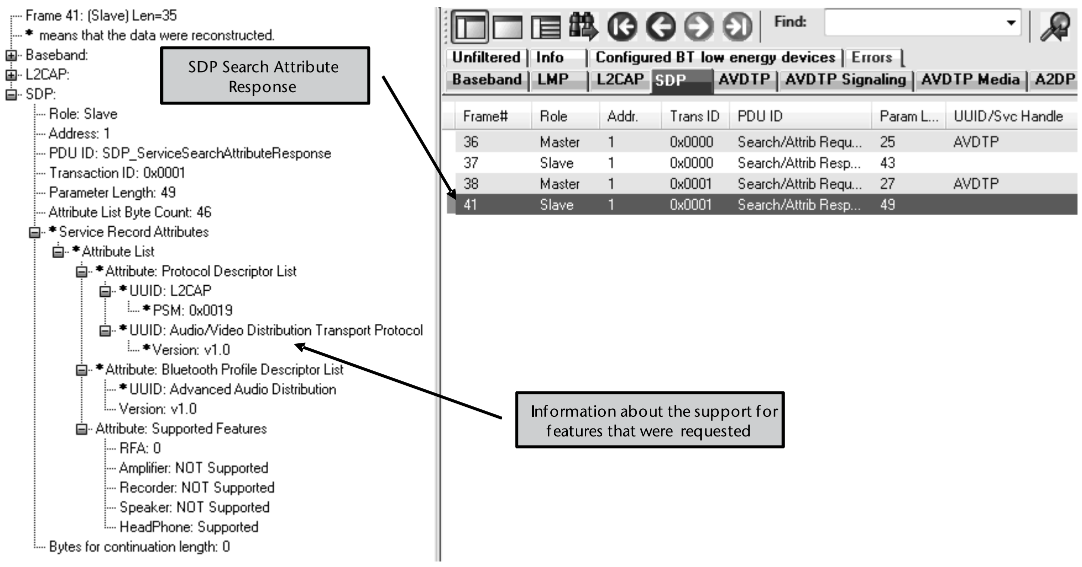
Find:
Unfiltered
Info
Configured BT low energy devices
Errors
Baseband
LMP
L2CAP
SDP
AVDTP
AVDTP Signaling
AVDTP Media
A2DP
SDP Search Attribute Request
Searching for Audio Sink, L2CAP and AVDTP

Figure 4.7 Example of SDP_ServiceSearchAttributeRequest.

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

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

- Audio/Video Distribution Transport Protocol (AVDTP) Version 1.0
- AVDTP is using PSM 0x0019 of L2CAP.
- Advanced Audio Distribution (A2DP) Version 1.0.


SDP Search Attribute Response
Information about the support for features that were requested
Find:
Unfiltered
Info
Configured BT low energy devices
Errors
Baseband
LMP
L2CAP
SDP
AVDTP
AVDTP Signaling
AVDTP Media
A2DP

```
Frame 41: (Slave) Len=35
* means that the data were reconstructed.
Baseband:
L2CAP:
SDP:
  Role: Slave
  Address: 1
  PDU ID: SDP_ServiceSearchAttributeResponse
  Transaction ID: 0x0001
  Parameter Length: 49
  Attribute List Byte Count: 46
  *Service Record Attributes
    *Attribute List
      *Attribute: Protocol Descriptor List
        *UUID: L2CAP
          *PSM: 0x0019
        *UUID: Audio/Video Distribution Transport Protocol
          *Version: v1.0
      *Attribute: Bluetooth Profile Descriptor List
        *UUID: Advanced Audio Distribution
        Version: v1.0
      Attribute: Supported Features
        RFA: 0
        Amplifier: NOT Supported
        Recorder: NOT Supported
        Speaker: NOT Supported
        HeadPhone: Supported
  Bytes for continuation length: 0
```

Frame#	Role	Addr.	Trans ID	PDU ID	Param L...	UUID/Svc Handle
36	Master	1	0x0000	Search/Attrib Requ...	25	AVDTP
37	Slave	1	0x0000	Search/Attrib Resp...	43	
38	Master	1	0x0001	Search/Attrib Requ...	27	AVDTP
41	Slave	1	0x0001	Search/Attrib Resp...	49	

Figure 4.8 Example of SDP_ServiceSearchAttributeResponse.

- Headphone is supported.
- Amplifier, Recorder and Speaker are NOT supported.

4.5 RFCOMM

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

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

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

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

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

There are broadly two types of communications devices:

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

RFCOMM supports both these types of devices.

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

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

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

As shown in Figure 4.9, up to 60 emulated ports can be active simultaneously though in practice a much lesser number of ports may be supported by the

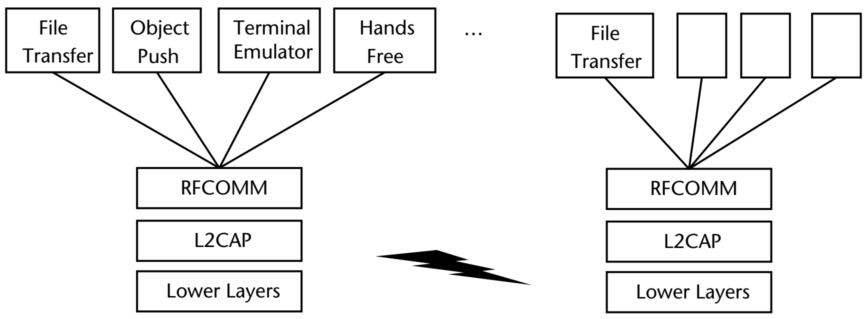


Figure 4.9 RFCOMM multiplexing.

implementation. Each connection is identified by a DLCI (Data Link Connection Identifier).

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

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

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

One of the enhancements that RFCOMM made to TS 07.10 is the credit based flow control. It was introduced after Bluetooth spec 1.0b to provide a flow control mechanism between the two devices. At the time of DLCI establishment, the receiving entity provides a number of credits to the transmitter. The transmitter can send as many frames as it has credits and decrement its credit count accordingly. Once the credit count reaches zero, it stops further transmission and waits for further credits from the receiver. Once the receiver is ready to receive more packets (For example after it has processed the previous packets) it provides further credits to the transmitter. This provides a simple and effective mechanism to emulate the flow control circuits of RS-232.

4.6 Object Exchange Protocol (OBEX)

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

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

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

Some examples of usage of OBEX are provided below:

- Synchronization: OBEX could be used for the synchronization of data between two devices. For example to keep the contacts information and calendar in sync between the laptop and mobile phone.
- File Transfer: OBEX can be used for sending and receiving files, browsing folders, deleting files, etc.

- Object Push: OBEX can be used for sending (pushing) and fetching (pulling) objects like business cards, calendars, notes, etc. Standard formats for these objects are used to ensure interoperability. These formats are referred to as vCard, vCalendar, vMessage and vNotes (electronic business card, electronic calendar, electronic message, and electronic notes).

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

4.6.1 OBEX Operations

OBEX follows a client/server request-response mechanism as shown in Figure 4.10. The client is the initiator of the OBEX connection. Requests are issued by the client and the server responds to these requests. The request/response pair is referred to as an operation.

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

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

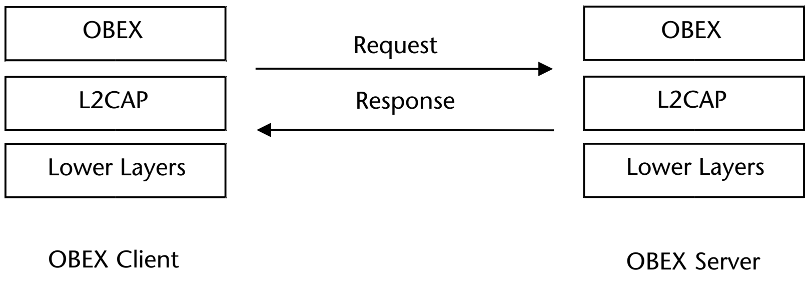


Figure 4.10 OBEX Client and Server (with OBEX over L2CAP).

Table 4.6 OBEX Operations

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

4.7 Audio/Video Control Transport Protocol (AVCTP)

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

Two roles are defined:

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

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

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

4.8 Audio/Video Distribution Transport Protocol (AVDTP)

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

Two roles are defined:

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

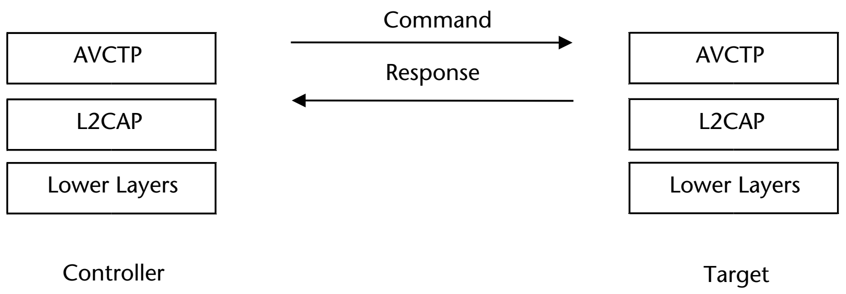


Figure 4.11 AVCTP Controller and Target.

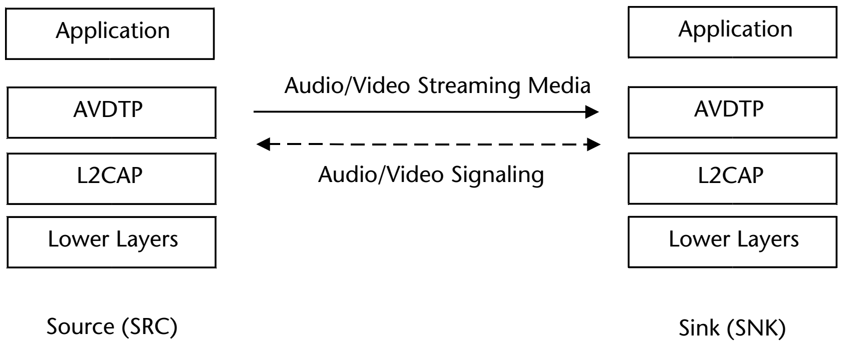


Figure 4.12 AVDTP Source and Sink.

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

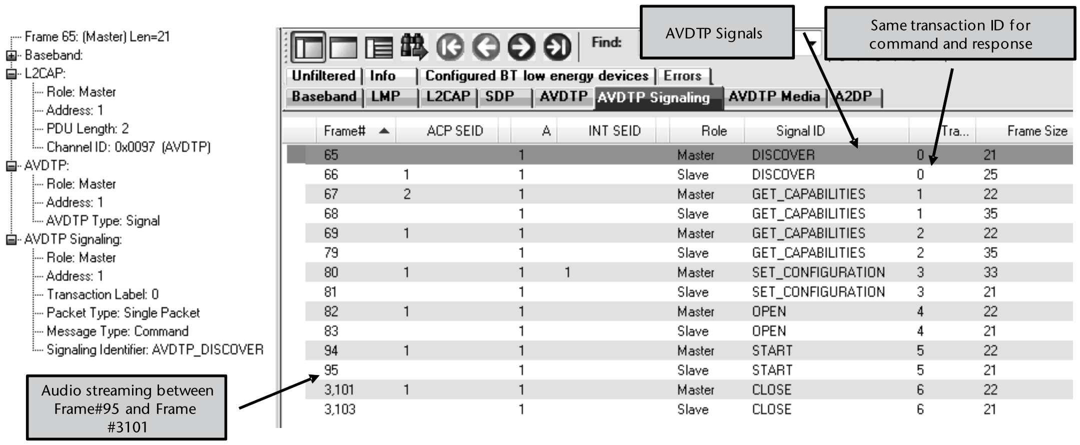
A Stream End Point (SEP) is a concept to expose the available transport services and AV capabilities of the application in order to negotiate a stream. An application registers its SEPs in AVDTP to allow other devices to discover and connect to them.

AVDTP defines procedures for the following:

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

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

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


AVDTP Signals
Same transaction ID for command and response
Audio streaming between Frame#95 and Frame #3101
Find:
Unfiltered | Info | Configured BT low energy devices | Errors
Baseband | LMP | L2CAP | SDP | AVDTP | AVDTP Signaling | AVDTP Media | A2DP

```
Frame 65: (Master) Len=21
Baseband:
L2CAP:
    Role: Master
    Address: 1
    PDU Length: 2
    Channel ID: 0x0097 (AVDTP)
AVDTP:
    Role: Master
    Address: 1
    AVDTP Type: Signal
AVDTP Signaling:
    Role: Master
    Address: 1
    Transaction Label: 0
    Packet Type: Single Packet
    Message Type: Command
    Signaling Identifier: AVDTP_DISCOVER
```

Frame#	ACP SEID	A	INT SEID	Role	Signal ID	Tra...	Frame Size
65		1		Master	DISCOVER	0	21
66	1	1		Slave	DISCOVER	0	25
67	2	1		Master	GET_CAPABILITIES	1	22
68		1		Slave	GET_CAPABILITIES	1	35
69	1	1		Master	GET_CAPABILITIES	2	22
79		1		Slave	GET_CAPABILITIES	2	35
80	1	1	1	Master	SET_CONFIGURATION	3	33
81		1		Slave	SET_CONFIGURATION	3	21
82	1	1		Master	OPEN	4	22
83		1		Slave	OPEN	4	21
94	1	1		Master	START	5	22
95		1		Slave	START	5	21
3,101	1	1		Master	CLOSE	6	22
3,103		1		Slave	CLOSE	6	21

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

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

4.9 Profiles

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

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

Generic Access Profile (GAP) is mandatory to be implemented for all devices that support Bluetooth. Devices may implement more profiles depending on the requirements of the application. The dependencies amongst profiles are depicted in

Figure 4.14. A profile is dependent on another profile if it uses parts of that profile. A dependent profile is shown in an inner box and the outer box indicates the profiles on which it is directly or indirectly dependent. GAP is shown in the outermost box since all other profiles are dependent on it.

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

4.10 Generic Access Profile (GAP)

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

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

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

- Device discovery;
- Connection establishment;

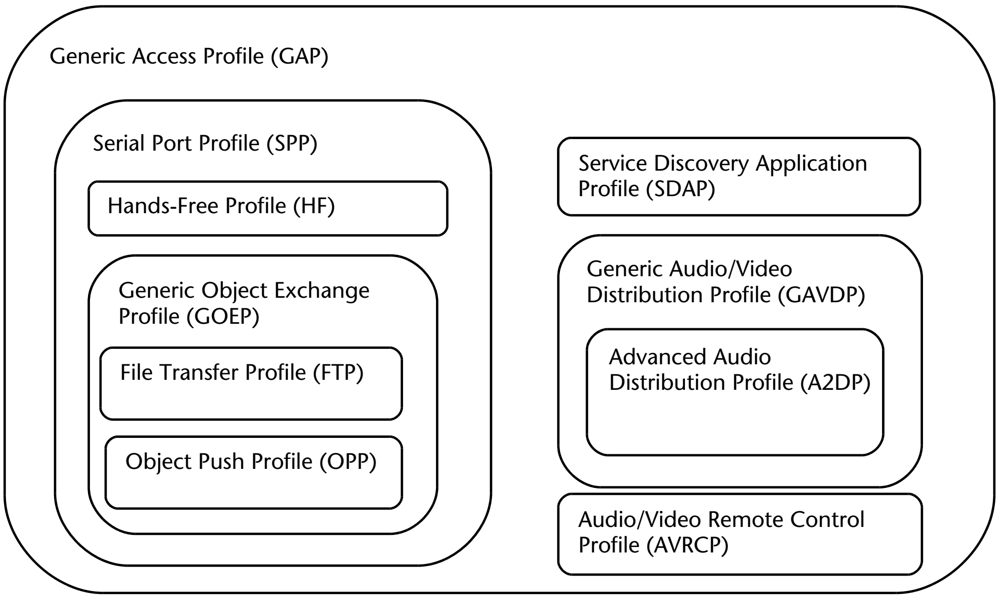

Generic Access Profile (GAP)
Serial Port Profile (SPP)
Hands-Free Profile (HF)
Generic Object Exchange Profile (GOEP)
File Transfer Profile (FTP)
Object Push Profile (OPP)
Service Discovery Application Profile (SDAP)
Generic Audio/Video Distribution Profile (GAVDP)
Advanced Audio Distribution Profile (A2DP)
Audio/Video Remote Control Profile (AVRCP)

Figure 4.14 Profile dependencies.