

## **APPENDIX B: IMPLEMENTATION GUIDELINES**

---

This section contains some guidelines for implementations. These guidelines are not part of the compliance tests. At the moment they are simply suggestions on how to solve some difficult problems.

### **RTX TIMER**

Implementations should not start this timer on an L2CAP Connection Request packet unless the physical link has been established. Otherwise the Baseband paging mechanism might increase the cost of the request beyond that of the minimal timeout value. If an implementation performs some form of security check it is recommended that the connection pending response be sent back prior to any consultation with a security manager that might perform Baseband authentication commands. If any security check requires user interaction, the link might timeout waiting for the user to enter a PIN.

### **QOS MAPPING TO LM AND L2CAP IMPLEMENTATIONS**

#### **Token Rate**

The Link Manager (LM) should ensure data is removed from the transmission buffer at this rate. The LM should ensure the polling interval is fast enough to support this data rate. The polling interval should be adjusted if the packet type changes. If the buffer overflows, and the service type is Guaranteed, a QoS violation should be reported. If the service type is Best Effort, and a Token Rate was non-zero, a QoS violation should also be reported.

Given a Token Rate of 0xFFFFFFFF, and Service Type of Guaranteed, the LM should refuse any additional connections from remote devices and disable all periodic scans.

#### **Token Bucket Size**

L2CAP implementations should ensure that a buffer meeting the size request is allocated for the channel. If no buffer is available, and the service type is Guaranteed, the request should be rejected. If no appropriately sized buffer is available, and the service type is Best Effort, the largest available buffer should be allocated.

#### **Peak Bandwidth**

If the token bucket buffer overflows, a QoS violation should be raised.

**Latency**

The LM should ensure the polling interval is at least this value. If the polling interval necessary to support the token rate is less than this value, the smaller interval should be used. If this interval cannot be supported, a QoS violation should be raised.

**Delay Variation**

The LM may ignore this value because there is no clear mapping between L2CAP packet delays and the necessary polling interval without requiring the LM to comprehend the length field in L2CAP packets.

**COLLISION TABLES**

Current Value	Requested Value	Result
X	X	X
X	Y	If $(X < Y)$ then X, else Y

Table I: Result of Second Link Timeout Request

Current Value	Requested Value	Result
N	0	N
N	N	N
N	$M \neq N$	Reject

Table II: Result of Second Flush Timeout Request

**Part E**

**SERVICE DISCOVERY  
PROTOCOL (SDP)**

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





**CONTENTS**

<b>1</b>	<b>Introduction .....</b>	<b>327</b>
1.1	General Description .....	327
1.2	Motivation.....	327
1.3	Requirements.....	327
1.4	Non-requirements and Deferred Requirements.....	328
1.5	Conventions .....	329
1.5.1	Bit And Byte Ordering Conventions.....	329
<b>2</b>	<b>Overview .....</b>	<b>330</b>
2.1	SDP Client-Server Interaction.....	330
2.2	Service Record.....	332
2.3	Service Attribute.....	334
2.4	Attribute ID .....	335
2.5	Attribute Value.....	335
2.6	Service Class .....	336
2.6.1	A Printer Service Class Example .....	336
2.7	Searching for Services.....	337
2.7.1	UUID.....	337
2.7.2	Service Search Patterns.....	338
2.8	Browsing for Services .....	338
2.8.1	Example Service Browsing Hierarchy .....	339
<b>3</b>	<b>Data Representation .....</b>	<b>341</b>
3.1	Data Element .....	341
3.2	Data Element Type Descriptor .....	341
3.3	Data Element Size Descriptor .....	342
3.4	Data Element Examples.....	343
<b>4</b>	<b>Protocol Description .....</b>	<b>344</b>
4.1	Transfer Byte Order .....	344
4.2	Protocol Data Unit Format.....	344
4.3	Partial Responses and Continuation State .....	346
4.4	Error Handling .....	346
4.4.1	SDP_ErrorResponse PDU .....	347
4.5	ServiceSearch Transaction .....	348
4.5.1	SDP_ServiceSearchRequest PDU.....	348
4.5.2	SDP_ServiceSearchResponse PDU.....	349
4.6	ServiceAttribute Transaction .....	351
4.6.1	SDP_ServiceAttributeRequest PDU.....	351
4.6.2	SDP_ServiceAttributeResponse PDU.....	352

4.7	ServiceSearchAttribute Transaction .....	354
4.7.1	SDP_ServiceSearchAttributeRequest PDU .....	354
4.7.2	SDP_ServiceSearchAttributeResponse PDU .....	356
<b>5</b>	<b>Service Attribute Definitions.....</b>	<b>358</b>
5.1	Universal Attribute Definitions.....	358
5.1.1	ServiceRecordHandle Attribute.....	358
5.1.2	ServiceClassIDList Attribute.....	359
5.1.3	ServiceRecordState Attribute.....	359
5.1.4	ServiceID Attribute .....	359
5.1.5	ProtocolDescriptorList Attribute.....	360
5.1.6	BrowseGroupList Attribute .....	361
5.1.7	LanguageBaseAttributeIDList Attribute .....	361
5.1.8	ServiceInfoTimeToLive Attribute .....	362
5.1.9	ServiceAvailability Attribute.....	363
5.1.10	BluetoothProfileDescriptorList Attribute .....	363
5.1.11	DocumentationURL Attribute .....	364
5.1.12	ClientExecutableURL Attribute.....	364
5.1.13	IconURL Attribute.....	365
5.1.14	ServiceName Attribute .....	365
5.1.15	ServiceDescription Attribute.....	366
5.1.16	ProviderName Attribute.....	366
5.1.17	Reserved Universal Attribute IDs.....	366
5.2	ServiceDiscoveryServer Service Class Attribute Definitions ...	367
5.2.1	ServiceRecordHandle Attribute.....	367
5.2.2	ServiceClassIDList Attribute.....	367
5.2.3	VersionNumberList Attribute .....	367
5.2.4	ServiceDatabaseState Attribute.....	368
5.2.5	Reserved Attribute IDs.....	368
5.3	BrowseGroupDescriptor Service Class Attribute Definitions ...	369
5.3.1	ServiceClassIDList Attribute.....	369
5.3.2	GroupID Attribute .....	369
5.3.3	Reserved Attribute IDs.....	369
<b>Appendix A – Background Information .....</b>		<b>370</b>
<b>Appendix B – Example SDP Transactions .....</b>		<b>371</b>



## 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 "Appendix A – Background Information," on page 370, 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 Transfer Byte Order on page 344.



## 2 OVERVIEW

### 2.1 SDP CLIENT-SERVER INTERACTION

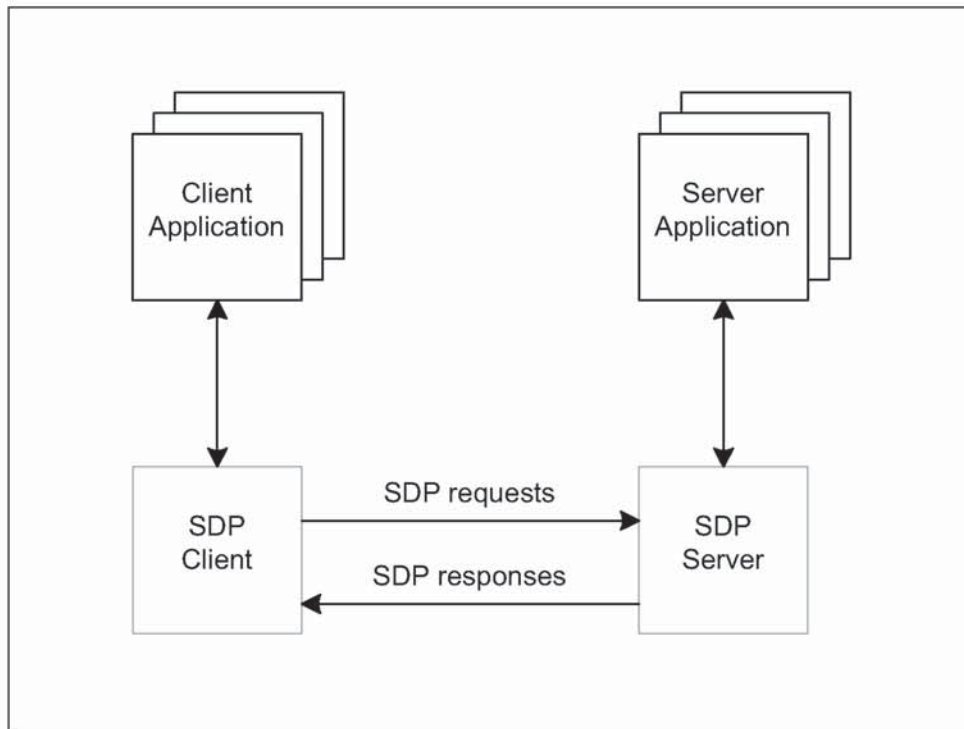


Figure 2.1:

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 Figure 1 may be simplified to that shown in Figure 2.

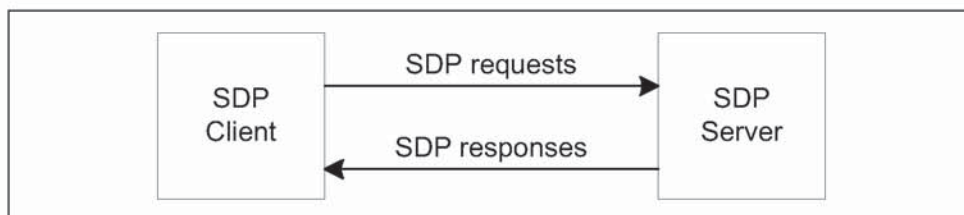


Figure 2.2:

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.

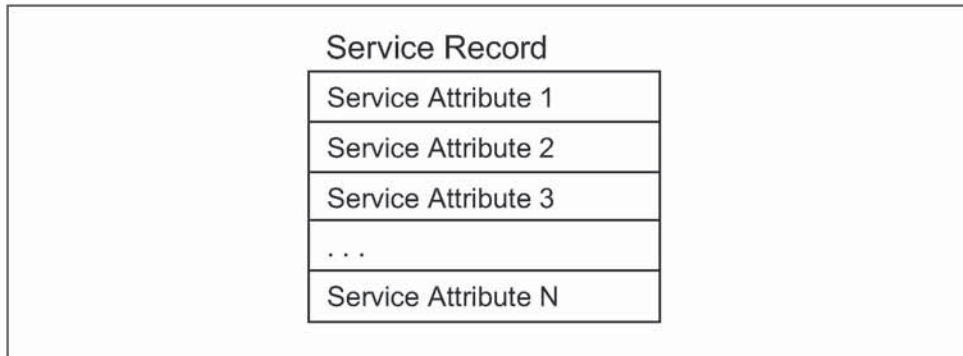


Figure 2.3: Service Record

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 Service Attribute Definitions on page 358.

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 Universal Attribute Definitions on page 358, 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.

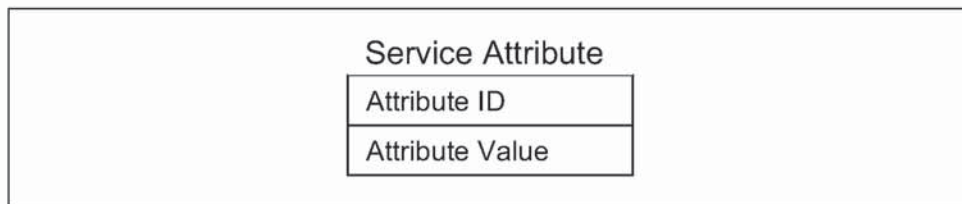


Figure 2.4: Service Attribute



## 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 Service Class on page 336.

In the Service Discovery Protocol, an attribute ID is often represented as a data element. See Section 3 Data Representation on page 341.

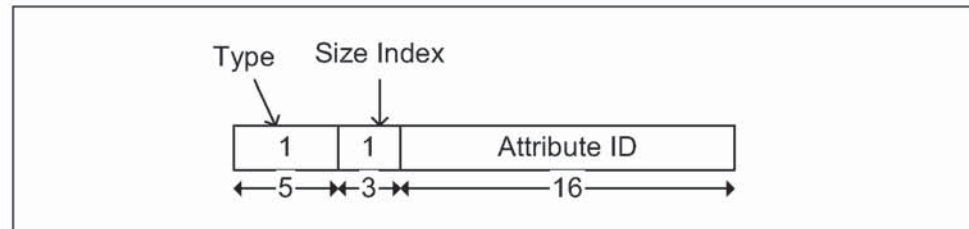


Figure 2.5:

## 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 Data Representation on page 341.) 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 Service Attribute Definitions on page 358, 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 UUID on page 337). 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 Service-Class definitions and have a ServiceClassIDList with UUIDs (See Section 2.7.1 UUID on page 337.) 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<sup>1</sup> (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 00000000-0000-1000-7007-00805F9B34FB, from the Bluetooth Assigned Numbers document. UUID values in the pre-allocated range have aliases that are represented as 16-bit or 32-bit values. These aliases are often called 16-bit and 32-bit UUIDs, but it is important to note that each actually represents a 128-bit UUID value.

The full 128-bit value of a 16-bit or 32-bit UUID may be computed by a simple arithmetic operation.

$$128\_bit\_value = 16\_bit\_value * 2^{96} + Bluetooth\_Base\_UUID$$

$$128\_bit\_value = 32\_bit\_value * 2^{96} + Bluetooth\_Base\_UUID$$

A 16-bit UUID may be converted to 32-bit UUID format by zero-extending the 16-bit value to 32-bits. An equivalent method is to add the 16-bit UUID value to a zero-valued 32-bit UUID.

Note that two 16-bit UUIDs may be compared directly, as may two 32-bit UUIDs or two 128-bit UUIDs. If two UUIDs of differing sizes are to be compared, the shorter UUID must be converted to the longer UUID format before comparison.

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)"

### 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).

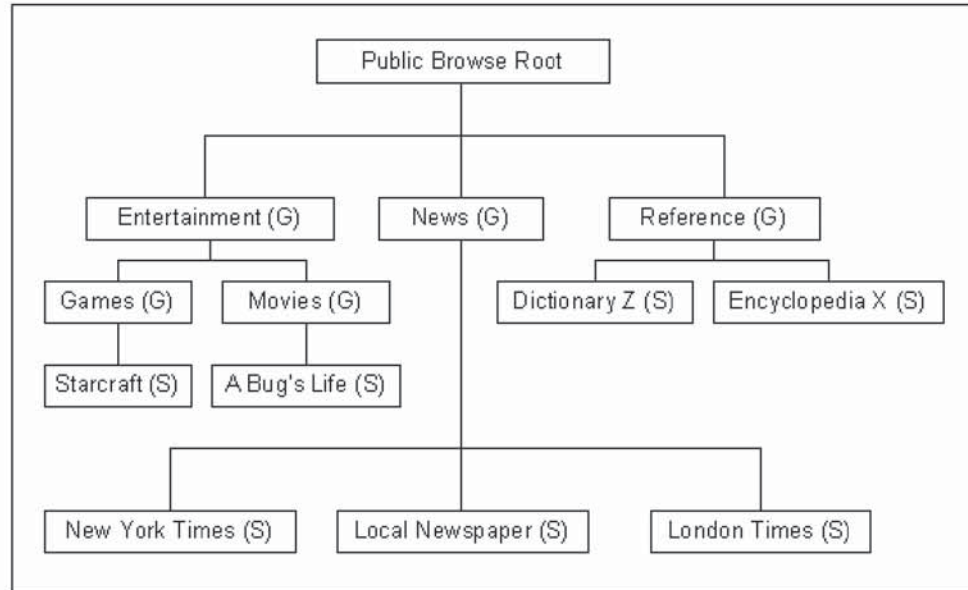


Figure 2.6:

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

Table 2.1:



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 Data Element Size Descriptor on page 342) 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
6	5, 6, 7	Data element sequence, a data element whose data field is a sequence of data elements
7	5, 6, 7	Data element alternative, data element whose data field is a sequence of data elements from which one data element is to be selected.
8	5, 6, 7	URL, a uniform resource locator
9-31		Reserved

Table 3.1:

### 3.3 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

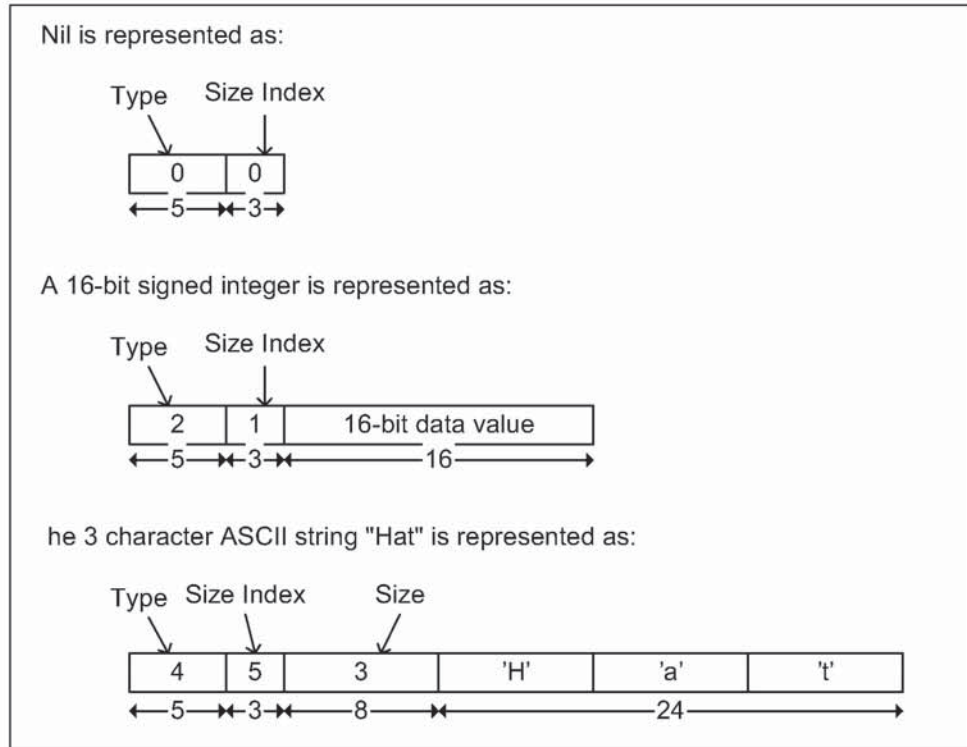


Figure 3.1:



## 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. However, the requests may potentially be pipelined and responses may potentially be returned out of order.

In the specific case where SDP utilises the Bluetooth L2CAP transport protocol, multiple SDP PDUs may be sent in a single L2CAP packet, but only one L2CAP packet per connection to a given SDP server may be outstanding at a given instant. Limiting SDP to sending one unacknowledged packet 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.

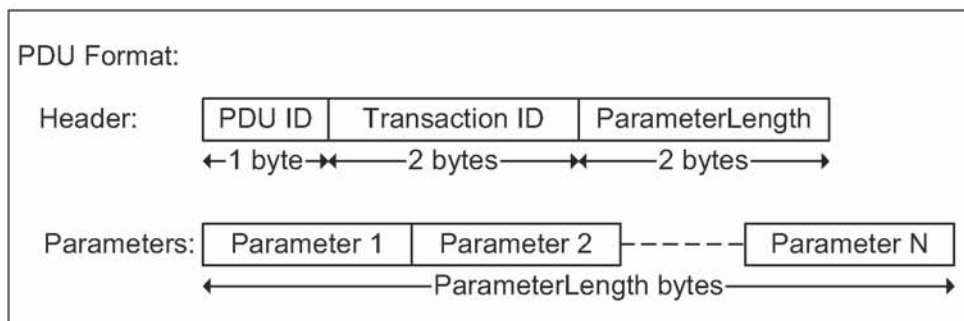


Figure 4.1:



*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.

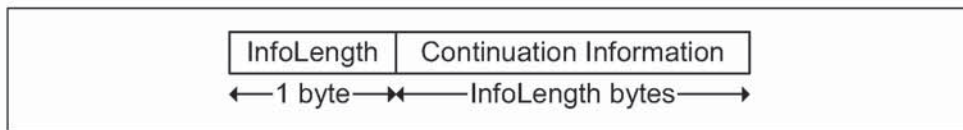


Figure 4.2: Continuation State Format

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.

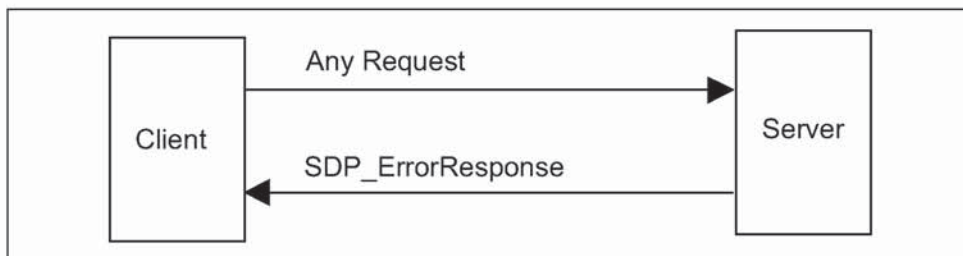


Figure 4.3:

**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 Browsing for Services on page 338 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*. The list of UUIDs constitutes a service search pattern.

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

*MaximumServiceRecordCount:*

*Size: 2 Bytes*

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

*ContinuationState:*

*Size: 1 to 17 Bytes*

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

#### 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

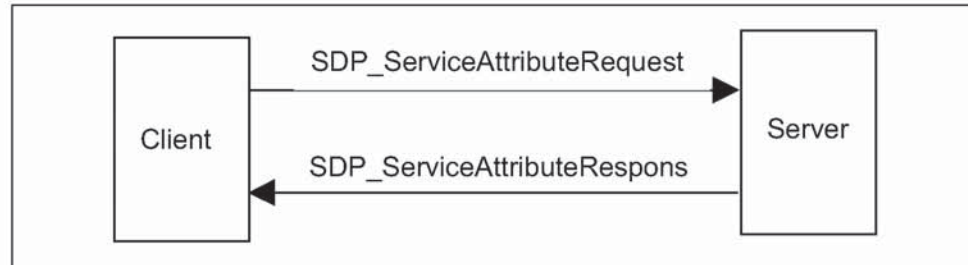


Figure 4.5:

#### 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(s) to this request. The SDP server should not return more than N bytes of attribute data in the response(s). If the requested attributes require more than N bytes, the SDP server determines how to truncate the list. 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

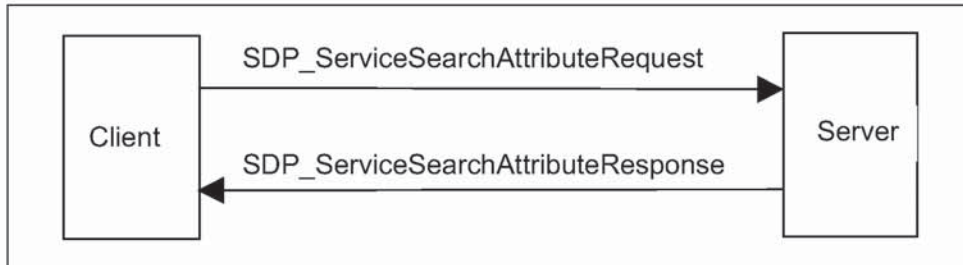


Figure 4.6:

### 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*. The list of UUIDs constitutes a service search pattern.

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

*MaximumAttributeByteCount:*

*Size: 2 Bytes*

Value	Parameter Description
N	MaximumAttributeByteCount specifies the maximum number of bytes of attribute data to be returned in the response(s) to this request. The SDP server should not return more than N bytes of attribute data in the response(s). If the requested attributes require more than N bytes, the SDP server determines how to truncate the list. Range: 0x0009-0xFFFF

*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.

*IrDA-like printer*

(( L2CAP, PSM=RFCOMM ), ( RFCOMM, CN=1 ), ( PostscriptStream ) )

*IP Network Printing*

(( L2CAP, PSM=RFCOMM ), ( RFCOMM, CN=2 ), ( PPP ), ( IP ), ( TCP ), ( IPP ) )

Synchronization Protocol Descriptor Example

(( L2CAP, PSM=0x1001 ), ( RFCOMM, CN=1 ), ( Obex ), ( vCal ) )

(( L2CAP, PSM=0x1002 ), ( RFCOMM, CN=1 ), ( Obex ), ( otherSynchronisationApplication ) )

**5.1.6 BrowseGroupList Attribute**

Attribute Name	Attribute ID	Attribute Value Type
BrowseGroupList	0x0005	Data Element Sequence

**Description:**

The BrowseGroupList attribute consists of a data element sequence in which each element is a UUID that represents a browse group to which the service record belongs. The top-level browse group ID, called PublicBrowseRoot and representing the root of the browsing hierarchy, has the value 00001002-0000-1000-7007-00805F9B34FB (UUID16: 0x1002) from the Bluetooth Assigned Numbers document.

**5.1.7 LanguageBaseAttributeIDList Attribute**

Attribute Name	Attribute ID	Attribute Value Type
LanguageBaseAttributeIDList	0x0006	Data Element Sequence

**Description:**

In order to support human-readable attributes for multiple natural languages in a single service record, a base attribute ID is assigned for each of the natural languages used in a service record. The human-readable universal attributes are then defined with an attribute ID offset from each of these base values, rather than with an absolute attribute ID.

The LanguageBaseAttributeIDList attribute is a list in which each member contains a language identifier, a character encoding identifier, and a base attribute



ID for each of the natural languages used in the service record. The LanguageBaseAttributeIDList attribute consists of a data element sequence in which each element is a 16-bit unsigned integer. The elements are grouped as triplets (threes).

The first element of each triplet contains an identifier representing the natural language. The language is encoded according to ISO 639:1988 (E/F): "Code for the representation of names of languages".

The second element of each triplet contains an identifier that specifies a character encoding used for the language. Values for character encoding can be found in IANA's database<sup>2</sup>, and have the values that are referred to as MIBenum values. The recommended character encoding is UTF-8.

The third element of each triplet contains an attribute ID that serves as the base attribute ID for the natural language in the service record. Different service records within a server may use different base attribute ID values for the same language.

To facilitate the retrieval of human-readable universal attributes in a principal language, the base attribute ID value for the primary language supported by a service record must be 0x0100. Also, if a LanguageBaseAttributeIDList attribute is contained in a service record, the base attribute ID value contained in its first element must be 0x0100.

**5.1.8 ServiceInfoTimeToLive Attribute**

Attribute Name	Attribute ID	Attribute Value Type
ServiceInfoTimeToLive	0x0007	32-bit unsigned integer

**Description:**

The ServiceTimeToLive attribute is a 32-bit integer that contains the number of seconds for which the information in a service record is expected to remain valid and unchanged. This time interval is measured from the time that the attribute value is retrieved from the SDP server. This value does not imply a guarantee that the service record will remain available or unchanged. It is simply a hint that a client may use to determine a suitable polling interval to re-validate the service record contents.

2. See <http://www.isi.edu/in-notes/iana/assignments/character-sets>



### 5.1.9 ServiceAvailability Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceAvailability	0x0008	8-bit unsigned integer

#### Description:

The ServiceAvailability attribute is an 8-bit unsigned integer that represents the relative ability of the service to accept additional clients. A value of 0xFF indicates that the service is not currently in use and is thus fully available, while a value of 0x00 means that the service is not accepting new clients. For services that support multiple simultaneous clients, intermediate values indicate the relative availability of the service on a linear scale.

For example, a service that can accept up to 3 clients should provide ServiceAvailability values of 0xFF, 0xAA, 0x55, and 0x00 when 0, 1, 2, and 3 clients, respectively, are utilising the service. The value 0xAA is approximately  $(2/3) * 0xFF$  and represents 2/3 availability, while the value 0x55 is approximately  $(1/3) * 0xFF$  and represents 1/3 availability. Note that the availability value may be approximated as

$$(1 - (\text{current\_number\_of\_clients} / \text{maximum\_number\_of\_clients})) * 0xFF$$

When the maximum number of clients is large, this formula must be modified to ensure that ServiceAvailability values of 0x00 and 0xFF are reserved for their defined meanings of unavailability and full availability, respectively.

Note that the maximum number of clients a service can support may vary according to the resources utilised by the service's current clients.

A non-zero value for ServiceAvailability does not guarantee that the service will be available for use. It should be treated as a hint or an approximation of availability status.

### 5.1.10 BluetoothProfileDescriptorList Attribute

Attribute Name	Attribute ID	Attribute Value Type
BluetoothProfileDescriptorList	0x0009	Data Element Sequence

#### Description:

The BluetoothProfileDescriptorList attribute consists of a data element sequence in which each element is a profile descriptor that contains information about a Bluetooth profile to which the service represented by this service record conforms. Each profile descriptor is a data element sequence whose

first element is the UUID assigned to the profile and whose second element is a 16-bit profile version number.

Each version of a profile is assigned a 16-bit unsigned integer profile version number, which consists of two 8-bit fields. The higher-order 8 bits contain the major version number field and the lower-order 8 bits contain the minor version number field. The initial version of each profile has a major version of 1 and a minor version of 0. When upward compatible changes are made to the profile, the minor version number will be incremented. If incompatible changes are made to the profile, the major version number will be incremented.

### 5.1.11 DocumentationURL Attribute

Attribute Name	Attribute ID	Attribute Value Type
DocumentationURL	0x000A	URL

#### Description:

This attribute is a URL which points to documentation on the service described by a service record.

### 5.1.12 ClientExecutableURL Attribute

Attribute Name	Attribute ID	Attribute Value Type
ClientExecutableURL	0x000B	URL

#### Description:

This attribute contains a URL that refers to the location of an application that may be used to utilize the service described by the service record. Since different operating environments require different executable formats, a mechanism has been defined to allow this single attribute to be used to locate an executable that is appropriate for the client device's operating environment. In the attribute value URL, the first byte with the value 0x2A (ASCII character '\*') is to be replaced by the client application with a string representing the desired operating environment before the URL is to be used.

The list of standardized strings representing operating environments is contained in the Bluetooth Assigned Numbers document.

For example, assume that the value of the ClientExecutableURL attribute is `http://my.fake/public*/client.exe`. On a device capable of executing SH3 WindowsCE files, this URL would be changed to `http://my.fake/public/sh3-microsoft-wince/client.exe`. On a device capable of executing Windows 98 binaries, this URL would be changed to `http://my.fake/public/i86-microsoft-win98/client.exe`.



**5.1.13 IconURL Attribute**

Attribute Name	Attribute ID	Attribute Value Type
IconURL	0x000C	URL

**Description:**

This attribute contains a URL that refers to the location of an icon that may be used to represent the service described by the service record. Since different hardware devices require different icon formats, a mechanism has been defined to allow this single attribute to be used to locate an icon that is appropriate for the client device. In the attribute value URL, the first byte with the value 0x2A (ASCII character ‘\*’) is to be replaced by the client application with a string representing the desired icon format before the URL is to be used.

The list of standardized strings representing icon formats is contained in the Bluetooth Assigned Numbers document.

For example, assume that the value of the IconURL attribute is `http://my.fake/public/icons/*`. On a device that prefers 24 x 24 icons with 256 colors, this URL would be changed to `http://my.fake/public/icons/24x24x8.png`. On a device that prefers 10 x 10 monochrome icons, this URL would be changed to `http://my.fake/public/icons/10x10x1.png`.

**5.1.14 ServiceName Attribute**

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

**Description:**

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



**5.1.15 ServiceDescription Attribute**

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

**Description:**

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

**5.1.16 ProviderName Attribute**

Attribute Name	Attribute ID Offset	Attribute Value Type
ProviderName	0x0002	String

**Description:**

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

**5.1.17 Reserved Universal Attribute IDs**

Attribute IDs in the range of 0x000D-0x01FF are reserved.

## 5.2 SERVICEDISCOVERYSERVER SERVICE CLASS ATTRIBUTE DEFINITIONS

This service class describes service records that contain attributes of service discovery server itself. The attributes listed in this section are only valid if the ServiceClassIDList attribute contains the ServiceDiscoveryServerServiceClassID. Note that all of the universal attributes may be included in service records of the ServiceDiscoveryServer class.

### 5.2.1 ServiceRecordHandle Attribute

Described in the universal attribute definition for ServiceRecordHandle.

#### Value

A 32-bit integer with the value 0x00000000.

### 5.2.2 ServiceClassIDList Attribute

Described in the universal attribute definition for ServiceClassIDList.

#### Value

A UUID representing the ServiceDiscoveryServerServiceClassID.

### 5.2.3 VersionNumberList Attribute

Attribute Name	Attribute ID	Attribute Value Type
VersionNumberList	0x0200	Data Element Sequence

#### Description:

The VersionNumberList is a data element sequence in which each element of the sequence is a version number supported by the SDP server.

A version number is a 16-bit unsigned integer consisting of two fields. The higher-order 8 bits contain the major version number field and the low-order 8 bits contain the minor version number field. The initial version of SDP has a major version of 1 and a minor version of 0. When upward compatible changes are made to the protocol, the minor version number will be incremented. If incompatible changes are made to SDP, the major version number will be incremented. This guarantees that if a client and a server support a common major version number, they can communicate if each uses only features of the specification with a minor version number that is supported by both client and server.

**5.2.4 ServiceDatabaseState Attribute**

Attribute Name	Attribute ID	Attribute Value Type
ServiceDatabaseState	0x0201	32-bit unsigned integer

**Description:**

The ServiceDatabaseState is a 32-bit integer that is used to facilitate caching of service records. If this attribute exists, its value is guaranteed to change when any of the other service records are added to or deleted from the server's database. If this value has not changed since the last time a client queried its value, the client knows that a) none of the other service records maintained by the SDP server have been added or deleted; and b) any service record handles acquired from the server are still valid. A client should query this attribute's value when a connection to the server is established, prior to using any service record handles acquired during a previous connection.

Note that the ServiceDatabaseState attribute does not change when existing service records are modified, including the addition, removal, or modification of service attributes. A service record's ServiceRecordState attribute indicates when that service record is modified.

**5.2.5 Reserved Attribute IDs**

Attribute IDs in the range of 0x0202-0x02FF are reserved.



### 5.3 BROWSEGROUPDESCRIPTOR SERVICE CLASS ATTRIBUTE DEFINITIONS

This service class describes the ServiceRecord provided for each BrowseGroupDescriptor service offered on a Bluetooth device. The attributes listed in this section are only valid if the ServiceClassIDList attribute contains the BrowseGroupDescriptorServiceClassID. Note that all of the universal attributes may be included in service records of the BrowseGroupDescriptor class.

#### 5.3.1 ServiceClassIDList Attribute

Described in the universal attribute definition for ServiceClassIDList.

#### Value

A UUID representing the BrowseGroupDescriptorServiceClassID.

#### 5.3.2 GroupID Attribute

Attribute Name	Attribute ID	Attribute Value Type
GroupID	0x0200	UUID

#### Description:

This attribute contains a UUID that can be used to locate services that are members of the browse group that this service record describes.

#### 5.3.3 Reserved Attribute IDs

Attribute IDs in the range of 0x0201-0x02FF are reserved.

---

## **APPENDIX A – BACKGROUND INFORMATION**

---

### **A.1. Service Discovery**

As computing continues to move to a network-centric model, finding and making use of services that may be available in the network becomes increasingly important. Services can include common ones such as printing, paging, FAX-ing, and so on, as well as various kinds of information access such as teleconferencing, network bridges and access points, eCommerce facilities, and so on — most any kind of service that a server or service provider might offer. In addition to the need for a standard way of discovering available services, there are other considerations: getting access to the services (finding and obtaining the protocols, access methods, “drivers” and other code necessary to utilize the service), controlling access to the services, advertising the services, choosing among competing services, billing for services, and so on. This problem is widely recognized; many companies, standards bodies and consortia are addressing it at various levels in various ways. Service Location Protocol (SLP), Jini™, and Salutation™, to name just a few, all address some aspect of service discovery.

### **A.2. Bluetooth Service Discovery**

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

## APPENDIX B – EXAMPLE SDP TRANSACTIONS

The following are simple examples of typical SDP transactions. These are meant to be illustrative of SDP flows. The examples do not consider:

- Caching (in a caching system, the SDP client would make use of the ServiceRecordState and ServiceDatabaseState attributes);
- Service availability (if this is of interest, the SDP client should use the ServiceAvailability and/or ServiceTimeToLive attributes);
- SDP versions (the VersionNumberList attribute could be used to determine compatible SDP versions);
- SDP Error Responses (an SDP error response is possible for any SDP request that is in error); and
- Communication connection (the examples assume that an L2CAP connection is established).

The examples are meant to be illustrative of the protocol. The format used is `ObjectName[ObjectSizeInBytes] {SubObjectDefinitions}`, but this is not meant to illustrate an interface. The `ObjectSizeInBytes` is the size of the object in decimal. The `SubObjectDefinitions` (inside of `{}` characters) are components of the immediately enclosing object. Hexadecimal values shown as lower-case letters, such as for transaction IDs and service handles, are variables (the particular value is not important for the illustration, but each such symbol always represents the same value). Comments are included in this manner: `/* comment text */`.

### B.1. SDP Example 1 – ServiceSearchRequest

The first example is that of an SDP client searching for a generic printing service. The client does not specify a particular type of printing service. In the example, the SDP server has two available printing services. The transaction illustrates:

1. SDP client to SDP server: `SDP_ServiceSearchRequest`, specifying the `PrinterServiceClassID` (represented as a `DataElement` with a 32-bit UUID value of `ppp . . . ppp`) as the only element of the `ServiceSearchPattern`. The `PrinterServiceClassID` is assumed to be a 32-bit UUID and the data element type for it is illustrated. The `TransactionID` is illustrated as `tttt`.
2. SDP server to SDP client: `SDP_ServiceSearchResponse`, returning handles to two printing services, represented as `qqqqqqqq` for the first printing service and `rrrrrrrr` for the second printing service. The `TransactionID` is the same value as supplied by the SDP client in the corresponding request (`tttt`).

```
/* Sent from SDP Client to SDP server */
SDP_ServiceSearchRequest[15] {
  PDUID[1] {
```



*Service Discovery Protocol***Bluetooth.**

```

    0x02
  }
  TransactionID[2] {
    0xtttt
  }
  ParameterLength[2] {
    0x000A
  }
  ServiceSearchPattern[7] {
    DataElementSequence[7] {
      0b00110 0b101 0x05
      UUID[5] {
        /* PrinterServiceClassID */
        0b00011 0b010 0xpppppppp
      }
    }
  }
  MaximumServiceRecordCount[2] {
    0x0003
  }
  ContinuationState[1] {
    /* no continuation state */
    0x00
  }
}

/* Sent from SDP server to SDP client */
SDP_ServiceSearchResponse[16] {
  PDUID[1] {
    0x03
  }
  TransactionID[2] {
    0xtttt
  }
  ParameterLength[2] {
    0x000D
  }
  TotalServiceRecordCount[2] {
    0x0002
  }
  CurrentServiceRecordCount[2] {
    0x0002
  }
  ServiceRecordHandleList[8] {
    /* print service 1 handle */
    0xqqqqqqqq
    /* print service 2 handle */
    0xrrrrrrrr
  }
  ContinuationState[1] {
    /* no continuation state */
    0x00
  }
}

```

## B.2. SDP Example 2 – ServiceAttributeTransaction

The second example continues the first example. In Example 1, the SDP client obtained handles to two printing services. In Example 2, the client uses one of those service handles to obtain the ProtocolDescriptorList attribute for that printing service. The transaction illustrates:

1. SDP client to SDP server: SDP\_ServiceAttributeRequest, presenting the previously obtained service handle (the one denoted as `qqqqqqqq`) and specifying the ProtocolDescriptorList attribute ID (AttributeID `0x0004`) as the only attribute requested (other attributes could be retrieved in the same transaction if desired). The TransactionID is illustrated as `uuuu` to distinguish it from the TransactionID of Example 1.
2. SDP server to SDP client: SDP\_ServiceAttributeResponse, returning the ProtocolDescriptorList for the specified printing service. This protocol stack is assumed to be ( (L2CAP), (RFCOMM, 2), (PostscriptStream) ). The ProtocolDescriptorList is a data element sequence in which each element is, in turn, a data element sequence whose first element is a UUID representing the protocol, and whose subsequent elements are protocol-specific parameters. In this example, one such parameter is included for the RFCOMM protocol, an 8-bit value indicating RFCOMM server channel 2. The Transaction ID is the same value as supplied by the SDP client in the corresponding request (`uuuu`). The Attributes returned are illustrated as a data element sequence where the protocol descriptors are 32-bit UUIDs and the RFCOMM server channel is a data element with an 8-bit value of 2.

```

/* Sent from SDP Client to SDP server */
SDP_ServiceAttributeRequest[17] {
  PDUID[1] {
    0x04
  }
  TransactionID[2] {
    0xuuuu
  }
  ParameterLength[2] {
    0x000C
  }
  ServiceRecordHandle[4] {
    0xqqqqqqqq
  }
  MaximumAttributeByteCount[2] {
    0x0080
  }
  AttributeIDList[5] {
    DataElementSequence[5] {
      0b00110 0b101 0x03
      AttributeID[3] {
        0b00001 0b001 0x0004
      }
    }
  }
}

```

*Service Discovery Protocol***Bluetooth.**

```

ContinuationState[1] {
  /* no continuation state */
  0x00
}
}

/* Sent from SDP server to SDP client */
SDP_ServiceAttributeResponse[36] {
  PDUID[1] {
    0x05
  }
  TransactionID[2] {
    0xuuuu
  }
  ParameterLength[2] {
    0x0021
  }
  AttributeListByteCount[2] {
    0x001E
  }
  AttributeList[30] {
    DataElementSequence[30] {
      0b00110 0b101 0x1C
      Attribute[28] {
        AttributeID[3] {
          0b00001 0b001 0x0004
        }
        AttributeValue[25] {
          /* ProtocolDescriptorList */
          DataElementSequence[25] {
            0b00110 0b101 0x17
            /* L2CAP protocol descriptor */
            DataElementSequence[7] {
              0b00110 0b101 0x05
              UUID[5] {
                /* L2CAP Protocol UUID */
                0b00011 0b010 <32-bit L2CAP UUID>
              }
            }
          }
          /* RFCOMM protocol descriptor */
          DataElementSequence[9] {
            0b00110 0b101 0x07
            UUID[5] {
              /* RFCOMM Protocol UUID */
              0b00011 0b010 <32-bit RFCOMM UUID>
            }
          }
          /* parameter for server 2 */
          Uint8[2] {
            0b00001 0b000 0x02
          }
        }
      }
    }
    /* PostscriptStream protocol descriptor */
    DataElementSequence[7] {
      0b00110 0b101 0x05
      UUID[5] {
        /* PostscriptStream Protocol UUID */
        0b00011 0b010 <32-bit PostscriptStream UUID>
      }
    }
  }
}

```



```
    }  
  }  
}  
ContinuationState[1] {  
  /* no continuation state */  
  0x00  
}
```

**B.3. SDP Example 3 – ServiceSearchAttributeTransaction**

The third example is a form of service browsing, although it is not generic browsing in that it does not make use of SDP browse groups. Instead, an SDP client is searching for available Synchronization services that can be presented to the user for selection. The SDP client does not specify a particular type of synchronization service. In the example, the SDP server has three available synchronization services: an address book synchronization service and a calendar synchronization service (both from the same provider), and a second calendar synchronization service from a different provider. The SDP client is retrieving the same attributes for each of these services; namely, the data formats supported for the synchronization service (vCard, vCal, iCal, etc.) and those attributes that are relevant for presenting information to the user about the services. Also assume that the maximum size of a response is 400 bytes. Since the result is larger than this, the SDP client will repeat the request supplying a continuation state parameter to retrieve the remainder of the response. The transaction illustrates:

1. SDP client to SDP server: `SDP_ServiceSearchAttributeRequest`, specifying the generic `SynchronisationServiceClassID` (represented as a data element whose 32-bit UUID value is `sss . . . sss`) as the only element of the `ServiceSearchPattern`. The `SynchronisationServiceClassID` is assumed to be a 32-bit UUID. The requested attributes are the `ServiceRecordHandle` (attribute ID `0x0000`), `ServiceClassIDList` (attribute ID `0x0001`), `IconURL` (attribute ID `0x000C`), `ServiceName` (attribute ID `0x0100`), `ServiceDescription` (attribute ID `0x0101`), and `ProviderName` (attribute ID `0x0102`) attributes; as well as the service-specific `SupportedDataStores` (AttributeID `0x0301`). Since the first two attribute IDs (`0x0000` and `0x0001`) and three other attribute IDs (`0x0100`, `0x0101`, and `0x0102`) are consecutive, they are specified as attribute ranges. The `TransactionID` is illustrated as `vvvv` to distinguish it from the `TransactionIDs` of the other Examples.

Note that values in the service record's primary language are requested for the text attributes (`ServiceName`, `ServiceDescription` and `ProviderName`) so that absolute attribute IDs may be used, rather than adding offsets to a base obtained from the `LanguageBaseAttributeIDList` attribute.

2. SDP server to SDP client: `SDP_ServiceSearchAttributeResponse`, returning the specified attributes for each of the three synchronization services. In the example, each `ServiceClassIDList` is assumed to contain a single element, the generic `SynchronisationServiceClassID` (a 32-bit UUID represented as `sss...sss`). Each of the other attributes contain illustrative data in the example (the strings have illustrative text; the icon URLs are illustrative, for each of the respective three synchronization services; and the `SupportedDataStore` attribute is represented as an unsigned 8-bit integer where `0x01` = `vCard2.1`, `0x02` = `vCard3.0`, `0x03` = `vCal1.0` and `0x04` = `iCal`). Note that one of the service records (the third for which data is returned) has no `ServiceDescription` attribute. The attributes are returned as a data element sequence, where each element is in turn a data element sequence repre-

sending a list of attributes. Within each attribute list, the ServiceClassIDList is a data element sequence while the remaining attributes are single data elements. The Transaction ID is the same value as supplied by the SDP client in the corresponding request (ωωωω). Since the entire result cannot be returned in a single response, a non-null continuation state is returned in this first response.

Note that the total length of the initial data element sequence (487 in the example) is indicated in the first response, even though only a portion of this data element sequence (368 bytes in the example, as indicated in the AttributeLists byte count) is returned in the first response. The remainder of this data element sequence is returned in the second response (without an additional data element header).

3. SDP client to SDP server: SDP\_ServiceSearchAttributeRequest, with the same parameters as in step 1, except that the continuation state received from the server in step 2 is included as a request parameter. The TransactionID is changed to ωωωω to distinguish it from previous request.
4. SDP server to SDP client: SDP\_ServiceSearchAttributeResponse, with the remainder of the result computed in step 2 above. Since all of the remaining result fits in this second response, a null continuation state is included.

```

/* Part 1 -- Sent from SDP Client to SDP server */
SdpSDP_ServiceSearchAttributeRequest[33] {
  PDUID[1] {
    0x06
  }
  TransactionID[2] {
    0xvvvv
  }
  ParameterLength[2] {
    0x001B
  }
  ServiceSearchPattern[7] {
    DataElementSequence[7] {
      0b00110 0b101 0x05
      UUID[5] {
        /* SynchronisationServiceClassID */
        0b00011 0b010 0xssssssss
      }
    }
  }
  MaximumAttributeByteCount[2] {
    0x0190
  }
  AttributeIDList[18] {
    DataElementSequence[18] {
      0b00110 0b101 0x10
      AttributeIDRange[5] {
        0b00001 0b010 0x00000001
      }
      AttributeID[3] {
        0b00001 0b001 0x000C
      }
    }
  }
}

```



*Service Discovery Protocol***Bluetooth.**

```

    }
    AttributeIDRange[5] {
        0b00001 0b010 0x01000102
    }
    AttributeID[3] {
        0b00001 0b001 0x0301
    } }
}
ContinuationState[1] {
    /* no continuation state */
    0x00
}
}

/* Part 2 -- Sent from SDP server to SDP client */
SdpSDP_ServiceSearchAttributeResponse[384] {
    PDUID[1] {
        0x07
    }
    TransactionID[2] {
        0xvvvv
    }
    ParameterLength[2] {
        0x017B
    }
    AttributeListByteCount[2] {
        0x0170
    }
    AttributeLists[368] {
        DataElementSequence[487] {
            0b00110 0b110 0x01E4
            DataElementSequence[178] {
                0b00110 0b101 0xB0
                Attribute[8] {
                    AttributeID[3] {
                        0b00001 0b001 0x0000
                    }
                    AttributeValue[5] {
                        /* service record handle */
                        0b00001 0b010 0xhhhhhhh
                    }
                }
                Attribute[10] {
                    AttributeID[3] {
                        0b00001 0b001 0x0001
                    }
                }
                AttributeValue[7] {
                    DataElementSequence[7] {
                        0b00110 0b101 0x05
                        UUID[5] {
                            /* SynchronisationServiceClassID */
                            0b00011 0b010 0xssssssss
                        }
                    }
                }
            }
        }
        Attribute[35] {

```

```

        AttributeID[3] {
            0b00001 0b001 0x000C
        }
        AttributeValue[32] {
            /* IconURL; '*' replaced by client application */
            0b01000 0b101 0x1E
            "http://Synchronisation/icons/*"
        }
    }
    Attribute[22] {
        AttributeID[3] {
            0b00001 0b001 0x0100
        }
        AttributeValue[19] {
            /* service name */
            0b00100 0b101 0x11
            "Address Book Sync"
        }
    }
    Attribute[59] {
        AttributeID[3] {
            0b00001 0b001 0x0101
        }
        AttributeValue[56] {
            /* service description */
            0b00100 0b101 0x36
            "Synchronisation Service for"
            " vCard Address Book Entries"
        }
    }
    Attribute[37] {
        AttributeID[3] {
            0b00001 0b001 0x0102
        }
        AttributeValue[34] {
            /* service provider */
            0b00100 0b101 0x20
            "Synchronisation Specialists Inc."
        }
    }
    Attribute[5] {
        AttributeID[3] {
            0b00001 0b001 0x0301
        }
        AttributeValue[2] {
            /* Supported Data Store 'phonebook' */
            0b00001 0b000 0x01
        }
    }
}
DataElementSequence[175] {
    0b00110 0b101 0xAD
    Attribute[8] {
        AttributeID[3] {
            0b00001 0b001 0x0000
        }
        AttributeValue[5] {

```

```

        /* service record handle */
        0b00001 0b010 0xmmmmmmmm
    }
}
Attribute[10] {
    AttributeID[3] {
        0b00001 0b001 0x0001
    }
    AttributeValue[7] {
        DataElementSequence[7] {
            0b00110 0b101 0x05
            UUID[5] {
                /* SynchronisationServiceClassID */
                0b00011 0b010 0xssssssss
            }
        }
    }
}
Attribute[35] {
    AttributeID[3] {
        0b00001 0b001 0x000C
    }
    AttributeValue[32] {
        /* IconURL; '*' replaced by client application */
        0b01000 0b101 0x1E
        "http://Synchronisation/icons/*"
    }
}
Attribute[21] {
    AttributeID[3] {
        0b00001 0b001 0x0100
    }
    AttributeValue[18] {
        /* service name */
        0b00100 0b101 0x10
        "Appointment Sync"
    }
}
Attribute[57] {
    AttributeID[3] {
        0b00001 0b001 0x0101
    }
    AttributeValue[54] {
        /* service description */
        0b00100 0b101 0x34
        "Synchronisation Service for"
        " vCal Appointment Entries"
    }
}
Attribute[37] {
    AttributeID[3] {
        0b00001 0b001 0x0102
    }
    AttributeValue[34] {
        /* service provider */
        0b00100 0b101 0x20
        "Synchronisation Specialists Inc."
    }
}

```



```

    }
  }
  Attribute[5] {
    AttributeID[3] {
      0b00001 0b001 0x0301
    }
    AttributeValue[2] {
      /* Supported Data Store 'calendar' */
      0b00001 0b000 0x03
    }
  }
}
/* } Data element sequence of attribute lists */
/* is not completed in this PDU. */
}
ContinuationState[9] {
  /* 8 bytes of continuation state */
  0x08 0xzzzzzzzzzzzzzzzzzz
}
}

/* Part 3 -- Sent from SDP Client to SDP server */
SdpSDP_ServiceSearchAttributeRequest[41] {
  PDUID[1] {
    0x06
  }
  TransactionID[2] {
    0xwww
  }
  ParameterLength[2] {
    0x0024
  }
  ServiceSearchPattern[7] {
    DataElementSequence[7] {
      0b00110 0b101 0x05
      UUID[5] {
        /* SynchronisationServiceClassID */
        0b00011 0b010 0xssssssss
      }
    }
  }
  MaximumAttributeByteCount[2] {
    0x0180
  }
  AttributeIDList[18] {
    DataElementSequence[18] {
      0b00110 0b101 0x10
      AttributeIDRange[5] {
        0b00001 0b010 0x00000001
      }
      AttributeID[3] {
        0b00001 0b001 0x000C
      }
      AttributeIDRange[5] {
        0b00001 0b010 0x01000102
      }
      AttributeID[3] {

```

*Service Discovery Protocol***Bluetooth.**

```

        0b00001 0b001 0x0301
    }
}
}
ContinuationState[9] {
    /* same 8 bytes of continuation state */
    /* received in part 2 */
    0x08 0xffffffffffffffff
}
}

```

Part 4 -- Sent from SDP server to SDP client

```

SdpSDP_ServiceSearchAttributeResponse[115] {
    PDUID[1] {
        0x07
    }
    TransactionID[2] {
        0xwww
    }
    ParameterLength[2] {
        0x006E
    }
    AttributeListByteCount[2] {
        0x006B
    }
    AttributeLists[107] {
        /* Continuing the data element sequence of */
        /* attribute lists begun in Part 2. */
        DataElementSequence[107] {
            0b00110 0b101 0x69
            Attribute[8] {
                AttributeID[3] {
                    0b00001 0b001 0x0000
                }
                AttributeValue[5] {
                    /* service record handle */
                    0b00001 0b010 0xffffffff
                }
            }
            Attribute[10] {
                AttributeID[3] {
                    0b00001 0b001 0x0001
                }
                AttributeValue[7] {
                    DataElementSequence[7] {
                        0b00110 0b101 0x05
                        UUID[5] {
                            /* SynchronisationServiceClassID */
                            0b00011 0b010 0xssssssss
                        }
                    }
                }
            }
            Attribute[35] {
                AttributeID[3] {
                    0b00001 0b001 0x000C
                }
            }
        }
    }
}

```

```

    }
    AttributeValue[32] {
        /* IconURL; '*' replaced by client application */
        0b01000 0b101 0x1E
        "http://DevManufacturer/icons/*"
    }
}
Attribute[18] {
    AttributeID[3] {
        0b00001 0b001 0x0100
    }
    AttributeValue[15] {
        /* service name */
        0b00100 0b101 0x0D
        "Calendar Sync"
    }
}
Attribute[29] {
    AttributeID[3] {
        0b00001 0b001 0x0102
    }
    AttributeValue[26] {
        /* service provider */
        0b00100 0b101 0x18
        "Device Manufacturer Inc."
    }
}
Attribute[5] {
    AttributeID[3] {
        0b00001 0b001 0x0301
    }
    AttributeValue[2] {
        /* Supported Data Store 'calendar' */
        0b00001 0b000 0x03
    }
}
}
/* This completes the data element sequence */
/* of attribute lists begun in Part 2.
}
ContinuationState[1] {
    /* no continuation state */
    0x00
}
}
}

```





**Part F:1**

**RFCOMM with TS 07.10**

**Serial Port Emulation**

**This document specifies the RFCOMM protocol by specifying a subset of the ETSI TS 07.10 standard, along with some Bluetooth-specific adaptations**





**CONTENTS**

<b>1</b>	<b>Introduction .....</b>	<b>389</b>
1.1	Overview .....	389
1.2	Device Types .....	389
1.3	Byte Ordering.....	390
<b>2</b>	<b>RFCOMM Service Overview .....</b>	<b>391</b>
2.1	RS-232 Control Signals.....	391
2.2	Null Modem Emulation.....	391
2.3	Multiple Emulated Serial Ports.....	393
2.3.1	Multiple Emulated Serial Ports between two Devices.....	393
2.3.2	Multiple Emulated Serial Ports and Multiple BT Devices.....	393
<b>3</b>	<b>Service Interface Description.....</b>	<b>395</b>
3.1	Service Definition Model .....	395
<b>4</b>	<b>TS 07.10 Subset Supported by RFCOMM .....</b>	<b>396</b>
4.1	Options and Modes.....	396
4.2	Frame Types.....	396
4.3	Commands.....	396
4.4	Convergence Layers.....	397
<b>5</b>	<b>TS 07.10 Adaptations for RFCOMM .....</b>	<b>398</b>
5.1	Media Adaptation .....	398
5.1.1	FCS calculation .....	398
5.2	TS 07.10 Multiplexer Start-up and Closedown Procedure .....	399
5.2.1	Start-up procedure .....	399
5.2.2	Close-down procedure .....	399
5.2.3	Link loss handling.....	399
5.3	System Parameters.....	400
5.4	DLCI allocation with RFCOMM server channels.....	400
5.5	Multiplexer Control Commands.....	401
5.5.1	Remote Port Negotiation Command (RPN) .....	401
5.5.2	Remote Line Status Command (RLS).....	402
5.5.3	DLC parameter negotiation (PN).....	402
<b>6</b>	<b>Flow Control .....</b>	<b>403</b>
6.1	L2CAP Flow Control in Overview.....	403
6.2	Wired Serial Port Flow Control.....	403
6.3	RFCOMM Flow Control.....	403
6.4	Port Emulation Entity Serial Flow Control .....	403

**Bluetooth.**

7 **Interaction with Other Entities** ..... 405

    7.1 Port Emulation and Port Proxy Entities ..... 405

        7.1.1 Port Emulation Entity ..... 405

        7.1.2 Port Proxy Entity ..... 405

    7.2 Service Registration and Discovery ..... 405

    7.3 Lower Layer Dependencies ..... 407

        7.3.1 Reliability ..... 407

        7.3.2 Low power modes ..... 407

8 **References** ..... 408

9 **Terms and Abbreviations** ..... 409

## 1 INTRODUCTION

The RFCOMM protocol provides emulation of serial ports over the L2CAP protocol. The protocol is based on the ETSI standard TS 07.10. This document does not contain a complete specification. Instead, references are made to the relevant parts of the TS 07.10 standard. Only a subset of the TS 07.10 standard is used, and some adaptations of the protocol are specified in this document.

### 1.1 OVERVIEW

RFCOMM is a simple transport protocol, with additional provisions for emulating the 9 circuits of RS-232 (EIA/TIA-232-E) serial ports.

The RFCOMM protocol supports up to 60 simultaneous connections between two BT devices. The number of connections that can be used simultaneously in a BT device is implementation-specific.

### 1.2 DEVICE TYPES

For the purposes of RFCOMM, a complete communication path involves two applications running on different devices (the communication endpoints) with a communication segment between them. Figure 1.1 shows the complete communication path. (In this context, the term *application* may mean other things than end-user application; e.g. higher layer protocols or other services acting on behalf of end-user applications.)

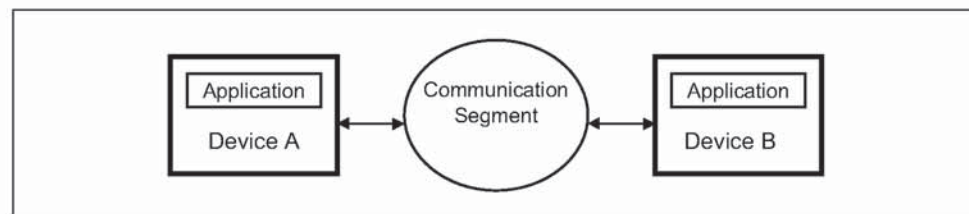


Figure 1.1: RFCOMM Communication Segment

RFCOMM is intended to cover applications that make use of the serial ports of the devices in which they reside. In the simple configuration, the communication segment is a BT link from one device to another (direct connect), see Figure 1.2. Where the communication segment is another network, BT is used for the path between the device and a network connection device like a modem. RFCOMM is only concerned with the connection between the devices in the direct connect case, or between the device and a modem in the network case. RFCOMM can support other configurations, such as modules that communicate via BT on one side and provide a wired interface on the other side, as shown in Figure 1.3. These devices are not really modems but offer a similar service. They are therefore not explicitly discussed here.



Basically two device types exist that RFCOMM must accommodate. Type 1 devices are communication end points such as computers and printers. Type 2 devices are those that are part of the communication segment; e.g. modems. Though RFCOMM does not make a distinction between these two device types in the protocol, accommodating both types of devices impacts the RFCOMM protocol.

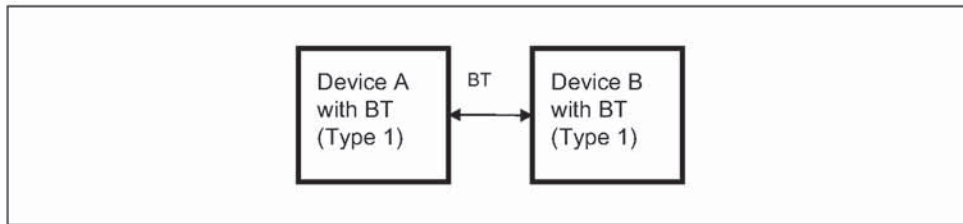


Figure 1.2: RFCOMM Direct Connect

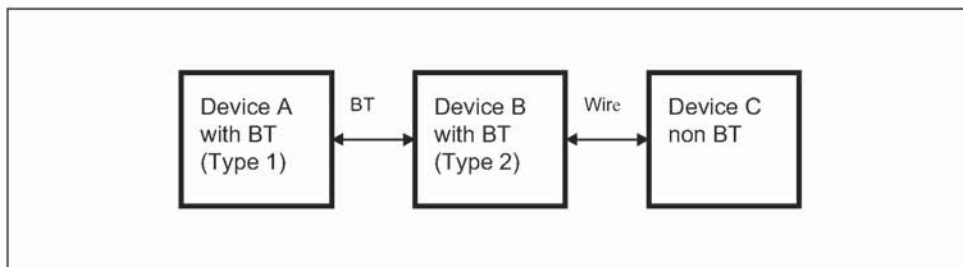


Figure 1.3: RFCOMM used with legacy COMM device

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

### 1.3 BYTE ORDERING

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

## 2 RFCOMM SERVICE OVERVIEW

RFCOMM emulates RS-232 (EIA/TIA-232-E) serial ports. The emulation includes transfer of the state of the non-data circuits. RFCOMM has a built-in scheme for null modem emulation.

In the event that a baud rate is set for a particular port through the RFCOMM service interface, that will not affect the actual data throughput in RFCOMM; i.e. RFCOMM does not incur artificial rate limitation or pacing. However, if either device is a type 2 device (relays data onto other media), or if data pacing is done above the RFCOMM service interface in either or both ends, actual throughput will, on an average, reflect the baud rate setting.

RFCOMM supports emulation of multiple serial ports between two devices and also emulation of serial ports between multiple devices, see Section 2.3 on page 393.

### 2.1 RS-232 CONTROL SIGNALS

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

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

Table 2.1: Emulated RS-232 circuits in RFCOMM

### 2.2 NULL MODEM EMULATION

RFCOMM is based on TS 07.10. When it comes to transfer of the states of the non-data circuits, TS 07.10 does not distinguish between DTE and DCE devices. The RS-232 control signals are sent as a number of DTE/DCE independent signals, see Table 2.2.

TS 07.10 Signals	Corresponding RS-232 Control Signals
RTC	DSR, DTR
RTR	RTS, CTS
IC	RI
DV	DCD

Table 2.2: TS 07.10 Serial Port Control Signals

The way in which TS 07.10 transfers the RS-232 control signals creates an implicit null modem when two devices of the same kind are connected together. Figure 2.1 shows the null modem that is created when two DTE are connected via RFCOMM. No single null-modem cable wiring scheme works in all cases; however the null modem scheme provided in RFCOMM should work in most cases.

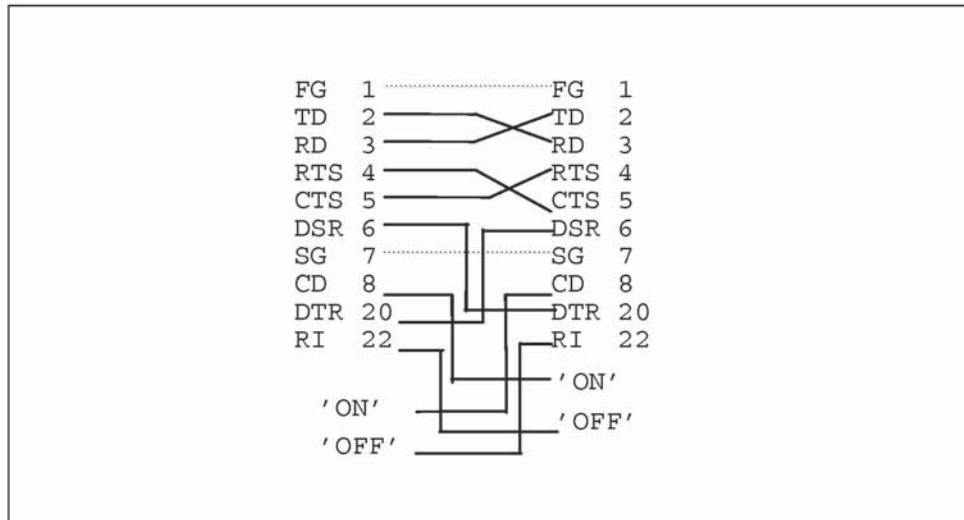


Figure 2.1: RFCOMM DTE-DTE Null Modem Emulation



## 2.3 MULTIPLE EMULATED SERIAL PORTS

### 2.3.1 Multiple Emulated Serial Ports between two Devices

Two BT devices using RFCOMM in their communication may open multiple emulated serial ports. RFCOMM supports up to 60 open emulated ports; however the number of ports that can be used in a device is implementation-specific.

A Data Link Connection Identifier (DLCI) [1] identifies an ongoing connection between a client and a server application. The DLCI is represented by 6 bits, but its usable value range is 2...61; in TS 07.10, DLCI 0 is the dedicated control channel, DLCI 1 is unusable due to the concept of Server Channels, and DLCI 62-63 is reserved. The DLCI is unique within one RFCOMM session between two devices. (This is explained further in Section 2.3.2) To account for the fact that both client and server applications may reside on both sides of an RFCOMM session, with clients on either side making connections independent of each other, the DLCI value space is divided between the two communicating devices using the concept of RFCOMM server channels. This is further described in Section 5.4.

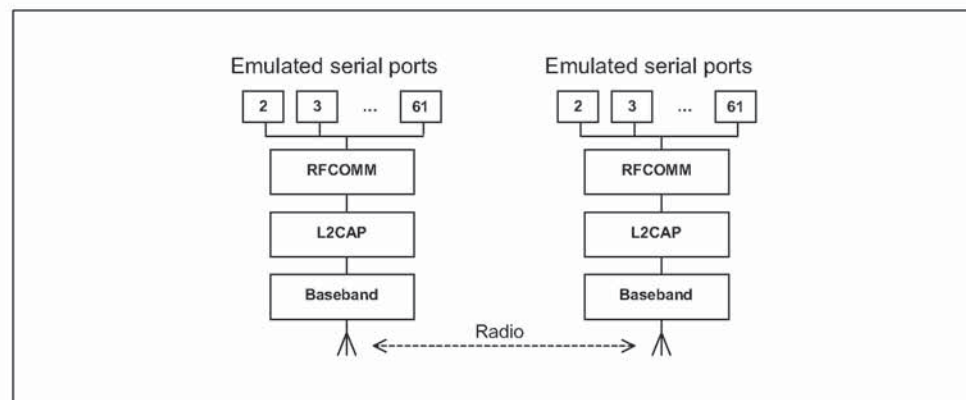


Figure 2.2: Multiple Emulated Serial Ports.

### 2.3.2 Multiple Emulated Serial Ports and Multiple BT Devices

If a BT device supports multiple emulated serial ports and the connections are allowed to have endpoints in different BT devices, then the RFCOMM entity must be able to run multiple TS 07.10 multiplexer sessions, see Figure 2.3. Note that each multiplexer session is using its own L2CAP channel ID (CID). The ability to run multiple sessions of the TS 07.10 multiplexer is optional for RFCOMM.

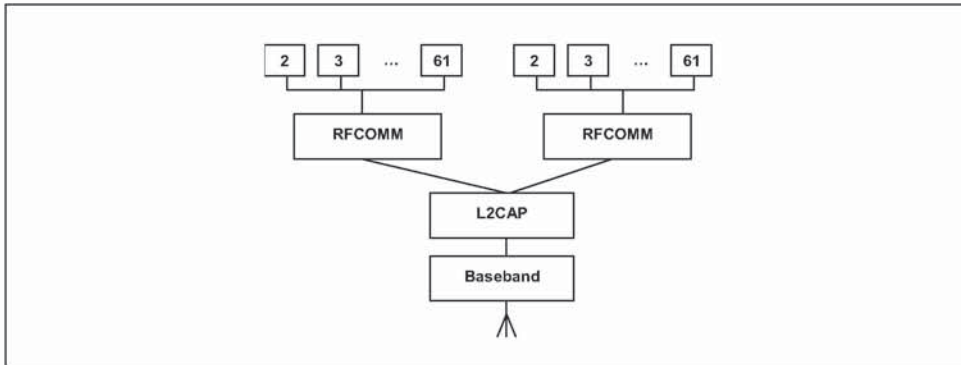


Figure 2.3: Emulating serial ports coming from two BT devices.

### 3 SERVICE INTERFACE DESCRIPTION

RFCOMM is intended to define a protocol that can be used to emulate serial ports. In most systems, RFCOMM will be part of a port driver which includes a serial port emulation entity.

#### 3.1 SERVICE DEFINITION MODEL

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

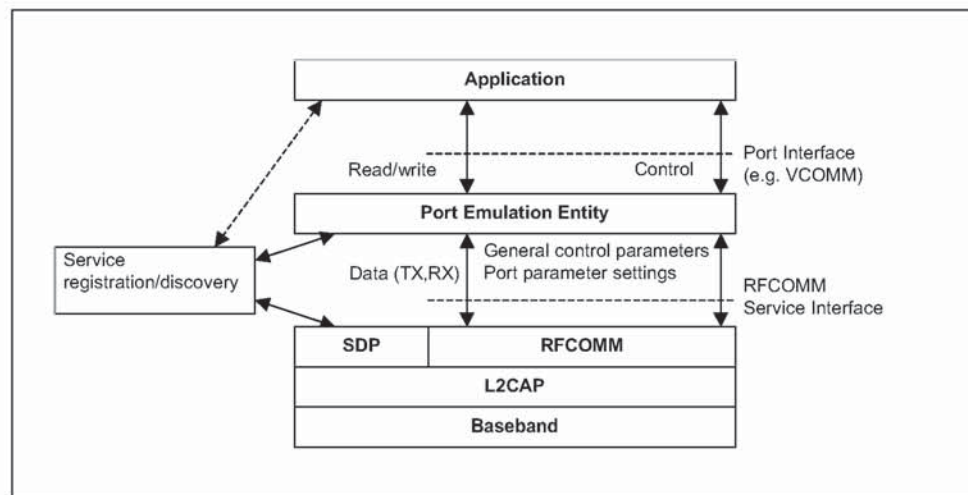


Figure 3.1: RFCOMM reference model

The elements of the RFCOMM reference model are described below.

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



## 4 TS 07.10 SUBSET SUPPORTED BY RFCOMM

### 4.1 OPTIONS AND MODES

RFCOMM uses the basic option of TS 07.10.

### 4.2 FRAME TYPES

Table 4.1 shows the TS 7.10 frame types that are supported in RFCOMM.

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

Table 4.1: Supported frame types in RFCOMM

The 'Unnumbered Information (UI) command and response' are not supported by RFCOMM. Since the error recovery mode option of the TS 07.10 protocol is not used in RFCOMM none of the associated frame types are supported.

### 4.3 COMMANDS

TS 07.10 defines a multiplexer that has a dedicated control channel, DLCI 0. The control channel is used to convey information between two multiplexers. The following commands in TS 07.10 are supported by RFCOMM:

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

Whenever a non-supported command type is received a 'Non-Supported Command Response (NSC)' should be sent.

#### **4.4 CONVERGENCE LAYERS**

RFCOMM only supports the type 1 convergence layer in TS 07.10.

The Modem Status Command (MSC) shall be used to convey the RS-232 control signals and the break signal for all emulated serial ports.

## 5 TS 07.10 ADAPTATIONS FOR RFCOMM

### 5.1 MEDIA ADAPTATION

The opening flag and the closing flags in the 07.10 basic option frame are not used in RFCOMM, instead it is only the fields contained between the flags that are exchanged between the L2CAP layer and RFCOMM layer, see Figure 5.1.

Flag	Address	Control	Length Indicator	Information	FCS	Flag
0111 1101	1 octet	1 octet	1 or 2 octets	Unspecified length but integral number of octets	1 octet	0111 1101

Figure 5.1: Frame Structure for Basic option. Note that the opening and closing flags from the 07.10 Basic option are excluded in RFCOMM.

#### 5.1.1 FCS calculation

In 07.10, the frame check sequence (FCS) is calculated on different sets of fields for different frame types. These are the fields that the FCS are calculated on:

For SABM, DISC, UA, DM frames: on Address, Control and length field.

For UIH frames: on Address and Control field.

(This is stated here for clarification, and to set the standard for RFCOMM; the fields included in FCS calculation have actually changed in version 7.0.0 of TS 07.10, but RFCOMM will not change the FCS calculation scheme from the one above.)



## 5.2 TS 07.10 MULTIPLEXER START-UP AND CLOSDOWN PROCEDURE

The start-up and closedown procedures as specified in section 5.7 in TS 07.10 are not supported. This means that the AT-command AT+CMUX is not supported by RFCOMM, neither is the multiplexer close down (CLD) command.

At any time, there must be at most one RFCOMM session between any pair of devices. When establishing a new DLC, the initiating entity must check if there already exists an RFCOMM session with the remote device, and if so, establish the new DLC on that. A session is identified by the Bluetooth BD\_ADDR of the two endpoints<sup>1</sup>.

### 5.2.1 Start-up procedure

The device opening up the first emulated serial port connection between two devices is responsible for first establishing the multiplexer control channel. This involves the following steps:

- Establish an L2CAP channel to the peer RFCOMM entity, using L2CAP service primitives, see L2CAP "Service Primitives" on page 295.
- Start the RFCOMM multiplexer by sending SABM command on DLCI 0, and await UA response from peer entity. (Further optional negotiation steps are possible.)

After these steps, DLCs for user data traffic can be established.

### 5.2.2 Close-down procedure

The device closing the last connection (DLC) on a particular session is responsible for closing the multiplexer by closing the corresponding L2CAP channel.

Closing the multiplexer by first sending a DISC command frame on DLCI 0 is optional, but it is mandatory to respond correctly to a DISC (with UA response).

### 5.2.3 Link loss handling

If an L2CAP link loss notification is received, the local RFCOMM entity is responsible for sending a connection loss notification to the port emulation/proxy entity for each active DLC. Then all resources associated with the RFCOMM session should be freed.

The appropriate action to take in the port emulation/proxy entity depends on the API on top. For example, for an emulated serial port (vCOMM), it would be suitable to drop CD, DSR and CTS signals (assuming device is a DTE).

---

1. This implies that, when responding to an L2CAP connection indication, the RFCOMM entity should save and associate the new RFCOMM session with the remote BD\_ADDR. This is, at least, necessary if subsequent establishment of a DLC in the opposite direction is possible (which may depend on device capabilities).

### 5.3 SYSTEM PARAMETERS

Table 5.1 contains all the applicable system parameters for the RFCOMM implementation of the TS 07.10 multiplexer.

System Parameter	Value
Maximum Frame Size ( <i>N1</i> )	Default: 127 (negotiable range 23 – 32767)
Acknowledgement Timer ( <i>T1</i> )	60 seconds
Response Timer for Multiplexer Control Channel ( <i>T2</i> )	60 seconds

Table 5.1: System parameter values

Note: The timer T1 is the timeout for *frames* sent with the P/F-bit set to one (this applies only to SABM and DISC frames in RFCOMM). T2 is the timeout for *commands* sent in UIH frames on DLCI 0.

Since RFCOMM relies on lower layers to provide reliable transmission, the default action performed on timeouts is to close down the multiplexer session. The only exception to this is when trying to set up a new DLC on an existing session; i.e. waiting for the UA response for a SABM command. In this case, the initiating side may defer the timeout by an unspecified amount of time if it has knowledge that the delay is due to user interaction (e.g. authentication procedure in progress). When/if the connection attempt is eventually considered to have timed out, the initiating side must send a DISC command frame on the same DLCI as the original SABM command frame, in order to notify the other party that the connection attempt is aborted. (After that the initiating side will, as usual, expect a UA response for the DISC command.)

### 5.4 DLCI ALLOCATION WITH RFCOMM SERVER CHANNELS

To account for the fact that both client and server applications may reside on both sides of an RFCOMM session, with clients on either side making connections independent of each other, the DLCI value space is divided between the two communicating devices using the concept of RFCOMM server channels and a direction bit.

The RFCOMM server channel number is a subset of the bits in the DLCI part of the address field in the TS 07.10 frame.

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

Table 5.2: The format of the Address Field



Server applications registering with an RFCOMM service interface are assigned a Server Channel number in the range 1...30. [0 and 31 should not be used since the corresponding DLCIs are reserved in TS 07.10] It is this value that should be registered in the Service Discovery Database, see Section 7.2.

For an RFCOMM session, the initiating device is given the direction bit D=1 (and conversely, D=0 in the other device). When establishing a new data link connection on an existing RFCOMM session, the direction bit is used in conjunction with the Server Channel to determine the DLCI to use to connect to a specific application. This DLCI is thereafter used for all packets in both directions between the endpoints.

In effect, this partitions the DLCI value space such that server applications on the non-initiating device are reachable on DLCIs 2,4,6,...,60; and server applications on the initiating device are reachable on DLCIs 3,5,7,...,61. (Note that for a device that supports multiple simultaneous RFCOMM sessions to two or more devices, the direction bit might not be the same on all sessions.)

An RFCOMM entity making a new DLC on an existing session forms the DLCI by combining the Server Channel for the application on the other device, and the inverse of its own direction bit for the session.

DLCI 1 and 62-63 are reserved and never used in RFCOMM.

## 5.5 MULTIPLEXER CONTROL COMMANDS

Note that in TS 07.10, some Multiplexer Control commands pertaining to specific DLCIs may be exchanged on the control channel (DLCI 0) *before* the corresponding DLC has been established. (This refers the PN and RPN commands.) All such states associated with an individual DLC must be reset to their default values upon receiving a DISC command frame, or when closing the DLC from the local side. This is to ensure that all DLC (re-)establishments on the same session will have predictable results, irrespective of the session history.

### 5.5.1 Remote Port Negotiation Command (RPN)

The RPN command can be used before a new DLC is opened and should be used whenever the port settings change.

The RPN command is specified as optional in TS 07.10, but it is mandatory to recognize and respond to it in RFCOMM. (Although the handling of individual settings are implementation-dependent.)



### 5.5.2 Remote Line Status Command (RLS)

This command is used for indication of remote port line status.

The RLS command is specified as optional in TS 07.10, but it is mandatory to recognize and respond to it in RFCOMM. (Although the handling of individual settings are implementation-dependent.)

### 5.5.3 DLC parameter negotiation (PN)

The PN command is specified as optional in TS 07.10, but it is mandatory to recognize and respond to it in RFCOMM. This command can be used before a new DLC is opened.

There are some parameters in the PN command which convey information not applicable to RFCOMM. These fields must therefore be set to predetermined values by the sender, and they must be ignored by the receiver. This concern the following fields (see table 3 in ref. [1]):

- I1-I4 must be set to 0. (Meaning: use UIH frames.)
- CL1-CL4 must be set to 0. (Meaning: use convergence layer type 1.)
- T1-T8 must be set to 0. (Meaning: acknowledgment timer  $T1$ , which is not negotiable in RFCOMM.)
- NA1-NA8 must be set to 0. (Meaning: number of retransmissions  $N2$ ; always 0 for RFCOMM)
- K1-K3 must be set to 0. (Meaning: defines the window size for error recovery mode, which is not used for RFCOMM.)

If a command is received with invalid (or for some reason unacceptable) values in any field, a DLC parameter negotiation response must be issued with values that are acceptable to the responding device.

## 6 FLOW CONTROL

---

Wired ports commonly use flow control such as RTS/CTS to control communications. On the other hand, the flow control between RFCOMM and the lower layer L2CAP depends on the service interface supported by the implementation. In addition RFCOMM has its own flow control mechanisms. This section describes the different flow control mechanisms.

### 6.1 L2CAP FLOW CONTROL IN OVERVIEW

L2CAP relies on the flow control mechanism provided by the Link Manager layer in the baseband. The flow control mechanism between the L2CAP and RFCOMM layers is implementation-specific.

### 6.2 WIRED SERIAL PORT FLOW CONTROL

Wired Serial ports falls into two camps – software flow control using characters such as XON/XOFF, and flow control using RTS/CTS or DTR/DSR circuits. These methods may be used by both sides of a wired link, or may be used only in one direction.

### 6.3 RFCOMM FLOW CONTROL

The RFCOMM protocol provides two flow control mechanisms:

1. The RFCOMM protocol contains flow control commands that operate on the aggregate data flow between two RFCOMM entities; i.e. all DLCIs are affected. The control channel commands, FCon and FCoff, are defined in section 5.4.6.3 in ref [1].
2. The Modem Status command as defined in section 5.4.6.3 in ref [1] is the flow control mechanism that operates on individual DLCI.

### 6.4 PORT EMULATION ENTITY SERIAL FLOW CONTROL

On Type 1 devices some port drivers (Port Emulation Entities plus RFCOMM) will need to provide flow control services as specified by the API they are emulating. An application may request a particular flow control mechanism like XON/XOFF or RTS/CTS and expect the port driver to handle the flow control. On type 2 devices the port driver may need to perform flow control on the non-RFCOMM part of the communication path; i.e. the physical RS-232 port. This flow control is specified via the control parameters sent by the peer RFCOMM entity (usually a type 1 device). The description of flow control in this section is for port drivers on type 1 devices.

Since RFCOMM already has its own flow control mechanism, the port driver does not need to perform flow control using the methods requested by the application. In the ideal case, the application sets a flow control mechanism



and assumes that the COMM system will handle the details. The port driver could then simply ignore the request and rely on RFCOMM's flow control. The application is able to send and receive data, and does not know or care that the port driver did not perform flow control using the mechanism requested. However, in the real world some problems arise.

- The RFCOMM-based port driver is running on top of a packet-based protocol where data may be buffered somewhere in the communication path. Thus, the port driver cannot perform flow control with the same precision as in the wired case.
- The application may decide to apply the flow control mechanism itself in addition to requesting flow control from the port driver.

These problems suggest that the port driver must do some additional work to perform flow control emulation properly. Here are the basic rules for flow control emulation.

- The port driver will not solely rely on the mechanism requested by the application but use a combination of flow control mechanisms.
- The port driver must be aware of the flow control mechanisms requested by the application and behave like the wired case when it sees changes on the non-data circuits (hardware flow control) or flow control characters in the incoming data (software flow control). For example, if XOFF and XON characters would have been stripped in the wired case they must be stripped by the RFCOMM based port driver.
- If the application sets a flow control mechanism via the port driver interface and then proceeds to invoke the mechanism on its own, the port driver must behave in a manner similar to that of the wired case (e.g. If XOFF and XON characters would have been passed through to the wire in the wired case the port driver must also pass these characters).

These basic rules are applied to emulate each of the wired flow control schemes. Note that multiple types of flow control can be set at the same time. Section 5.4.8 in ref [1] defines each flow control mechanism.



## 7 INTERACTION WITH OTHER ENTITIES

### 7.1 PORT EMULATION AND PORT PROXY ENTITIES

This section defines how the RFCOMM protocol should be used to emulate serial ports. Figure 7.1 shows the two device types that the RFCOMM protocol supports.

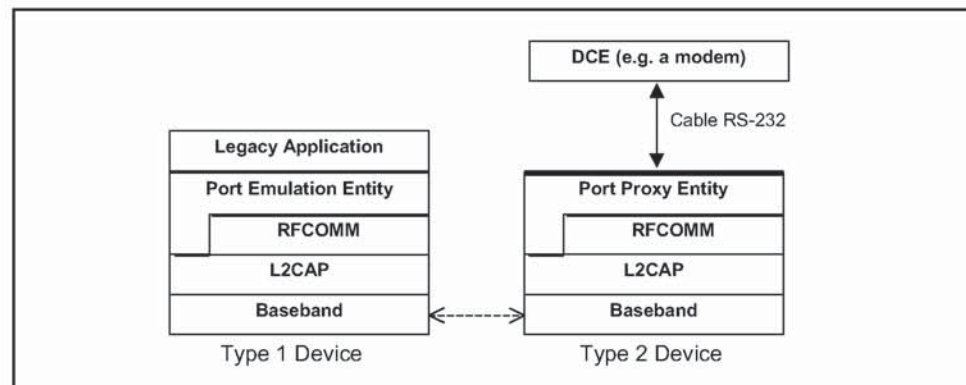


Figure 7.1: The RFCOMM communication model

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

#### 7.1.1 Port Emulation Entity

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

#### 7.1.2 Port Proxy Entity

The port proxy entity relays data from RFCOMM to an external RS-232 interface linked to a DCE. The communications parameters of the RS-232 interface are set according to received RPN commands, see Section 5.5.1.

## 7.2 SERVICE REGISTRATION AND DISCOVERY

Registration of individual applications or services, along with the information needed to reach those (i.e. the RFCOMM Server Channel) is the responsibility of each application respectively (or possibly a Bluetooth configuration application acting on behalf of legacy applications not directly aware of Bluetooth).

Below is a template/example for developing service records for a given service or profile using RFCOMM. It illustrates the inclusion of the ServiceClassList with a single service class, a ProtocolDescriptor List with two protocols

(although there may be more protocols on top of RFCOMM). The example shows the use of one other universal attribute (ServiceName). For each service running on top of RFCOMM, appropriate SDP-defined universal attributes and/or service-specific attributes will apply. For additional information on Service Records, see the SDP Specification, Section 2.2 on page 332.

The attributes that a client application needs (at a minimum) to connect to a service on top of RFCOMM are the ServiceClassIDList and the ProtocolDescriptorList (corresponding to the shaded rows in the table below).

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

**Notes:**

1. Defined in "Bluetooth Assigned Numbers" on page 1009.
2. For national language support for all 'displayable' text string attributes, an offset has to be added to the LanguageBaseAttributeIDList value for the selected language (see the SDP Specification, Section 5.1.14 on page 365 for details).
3. To be defined (where necessary) for the specific service.
4. For a specific service some of the SDP-defined universal attributes may apply. See the SDP Specification, Section 5.1 on page 358.
5. This indicates the class of service. It can be a single entry or a list of service classes ranging from generic to most specific.



## 7.3 LOWER LAYER DEPENDENCIES

### 7.3.1 Reliability

RFCOMM uses the services of L2CAP to establish L2CAP channels to RFCOMM entities on other devices. An L2CAP channel is used for the RFCOMM/TS 07.10 multiplexer session. On such a channel, the TS 07.10 frames listed in Section 4.2 are sent, with the adaptation defined in Section 5.1.

Some frame types (SABM and DISC) as well as UIH frames with multiplexer control commands sent on DLCI 0 always require a response from the remote entity, so they are acknowledged on the RFCOMM level (but not retransmitted in the absence of acknowledgment, see Section 5.3). Data frames do not require any response in the RFCOMM protocol, and are thus unacknowledged.

Therefore, RFCOMM must require L2CAP to provide channels with maximum reliability, to ensure that all frames are delivered in order, and without duplicates. Should an L2CAP channel fail to provide this, RFCOMM expects a link loss notification, which should be handled by RFCOMM as described in Section 5.2.3.

### 7.3.2 Low power modes

If all L2CAP channels towards a certain device are idle for a certain amount of time, a decision may be made to put that device in a low power mode (i.e. use hold, sniff or park, see 'Baseband Specification' Section 10.10.3 on page 125). This will be done without any interference from RFCOMM. RFCOMM can state its latency requirements to L2CAP. This information may be used by lower layers to decide which low power mode(s) to use.

The RFCOMM protocol as such does not suffer from latency delays incurred by low power modes, and consequentially, this specification does not state any maximum latency requirement on RFCOMM's behalf. Latency sensitivity inherently depends on application requirements, which suggests that an RFCOMM service interface implementation could include a way for applications to state latency requirements, to be aggregated and conveyed to L2CAP by the RFCOMM implementation. (That is if such procedures make sense for a particular platform.)



## **8 REFERENCES**

---

- [1] TS 07.10, ver 6.3.0, ETSI
- [2] Bluetooth L2CAP Specification
- [3] Bluetooth SDP Specification
- [4] Bluetooth Assigned Numbers

## 9 TERMS AND ABBREVIATIONS

---

The following terms are used throughout the document.

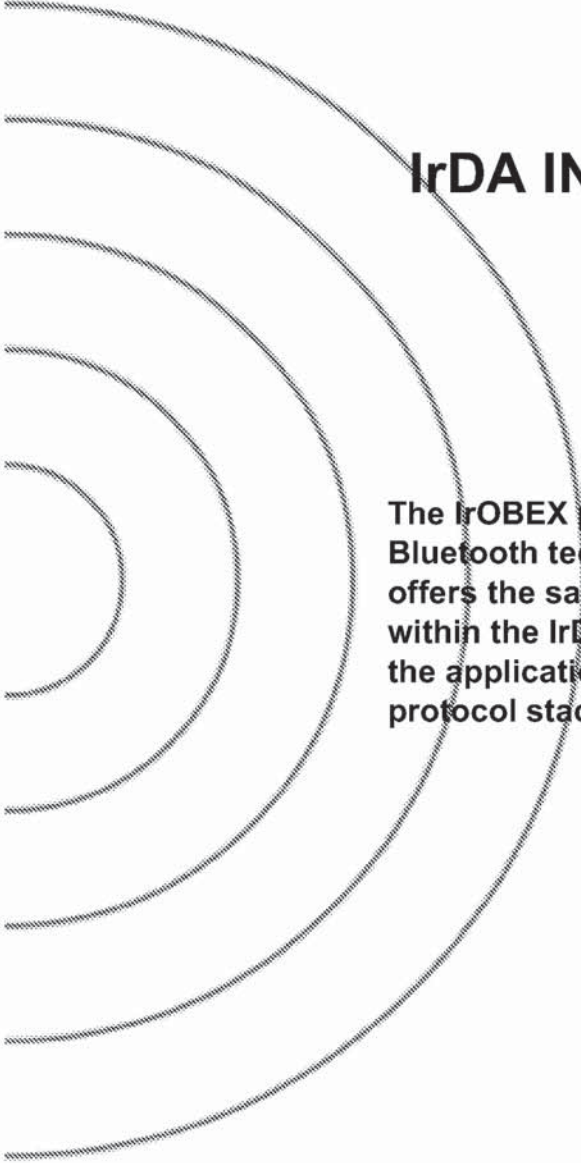
DTE	Data Terminal Equipment – in serial communications, DTE refers to a device at the endpoint of the communications path; typically a computer or terminal
DCE	Data Circuit-Terminating Equipment – in serial communications, DCE refers to a device between the communication endpoints whose sole task is to facilitate the communications process; typically a modem
RFCOMM initiator	The device initiating the RFCOMM session; i.e. setting up RFCOMM channel on L2CAP and starting RFCOMM multiplexing with the SABM command frame on DLCI 0 (zero)
RFCOMM Client	An RFCOMM client is an application that requests a connection to another application (RFCOMM server)
RFCOMM Server	An RFCOMM server is an application that awaits a connection from an RFCOMM client on another device. What happens after such a connection is established is not within the scope of this definition
RFCOMM Server Channel	This is a subfield of the TS 07.10 DLCI number. This abstraction is used to allow both server and client applications to reside on both sides of an RFCOMM session





## Part F:2

# IrDA INTEROPERABILITY



The IrOBEX protocol is utilized by the Bluetooth technology. In Bluetooth, OBEX offers the same features for applications as within the IrDA protocol hierarchy, enabling the applications to work over the Bluetooth protocol stack as well as the IrDA stack.



**CONTENTS**

<b>1</b>	<b>Introduction .....</b>	<b>414</b>
1.1	OBEX and Bluetooth Architecture.....	415
1.2	Bluetooth OBEX-Related Specifications .....	415
1.3	Other IrOBEX Implementations.....	416
<b>2</b>	<b>OBEX Object and Protocol .....</b>	<b>417</b>
2.1	Object.....	417
2.2	Session Protocol .....	417
2.2.1	Connect Operation .....	418
2.2.2	Disconnect Operation.....	419
2.2.3	Put Operation .....	419
2.2.4	Get Operation.....	420
2.2.5	Other Operations.....	420
<b>3</b>	<b>OBEX over RFCOMM .....</b>	<b>421</b>
3.1	OBEX Server Start-up on RFCOMM.....	421
3.2	Receiving OBEX Packets from Serial Port.....	421
3.3	Connection Establishment .....	422
3.4	Disconnection .....	422
3.5	Pushing and Pulling OBEX Packets over RFCOMM .....	422
<b>4</b>	<b>OBEX over TCP/IP .....</b>	<b>423</b>
4.1	OBEX Server Start-up on TCP/IP .....	423
4.2	Connection Establishment .....	423
4.3	Disconnection .....	424
4.4	Pushing and Pulling OBEX Packets over TCP .....	424
<b>5</b>	<b>Bluetooth Application Profiles using OBEX.....</b>	<b>425</b>
5.1	Synchronization .....	425
5.2	File Transfer .....	425
5.3	Object Push .....	426
<b>6</b>	<b>References .....</b>	<b>427</b>
<b>7</b>	<b>List of Acronyms and Abbreviations.....</b>	<b>428</b>



## 1 INTRODUCTION

---

The goal of this document is to enable the development of application programs that function well over both short-range RF and IR media. Each media type has its advantages and disadvantages but the goal is for applications to work over both. Rather than fragment the application domain, this document defines the intersection point where Bluetooth and IrDA applications may converge. That intersection point is IrOBEX [1].

IrOBEX is a session protocol defined by IrDA. This protocol is now also utilized by the Bluetooth technology, making it possible for applications to use either the Bluetooth radio technology or the IrDA IR technology. However, even though both IrDA and Bluetooth are designed for short-range wireless communications, they have some fundamental differences relating to the lower-layer protocols. IrOBEX will therefore be mapped over the lower layer protocols which are adopted by Bluetooth.

This document defines how IrOBEX (OBEX for short) is mapped over RFCOMM [2] and TCP/IP [3]. Originally, OBEX (Object Exchange Protocol) was developed to exchange data objects over an infrared link and was placed within the IrDA protocol hierarchy. However, it can appear above other transport layers, now RFCOMM and TCP/IP. At this moment, it is worth mentioning that the OBEX over TCP/IP implementation is an optional feature for Bluetooth devices supporting the OBEX protocol.

The IrOBEX specification [1] provides a model for representing objects and a session protocol, which structures the dialogue between two devices. The IrOBEX protocol follows a client/server **request-response** paradigm for the conversation format.

Bluetooth uses only the connection-oriented OBEX even though IrDA has specified the connectionless OBEX also. The reasons for the connection-oriented approach are:

- OBEX is mapped over the connection-oriented protocols in the Bluetooth architecture.
- Most of application profiles using OBEX and Bluetooth needs a connection-oriented OBEX to provide the functionality described for the features included in these profiles.
- The connectionless OBEX with the connection-oriented one would raise the interoperability problems, which are not desirable.

## 1.1 OBEX AND BLUETOOTH ARCHITECTURE

Figure 1.1 depicts part of the hierarchy of the Bluetooth architecture and shows the placement of the OBEX protocol and the application profiles using it (See also Section 5 on page 425). The protocols can also communicate with the service discovery DB even though the figure does not show it.

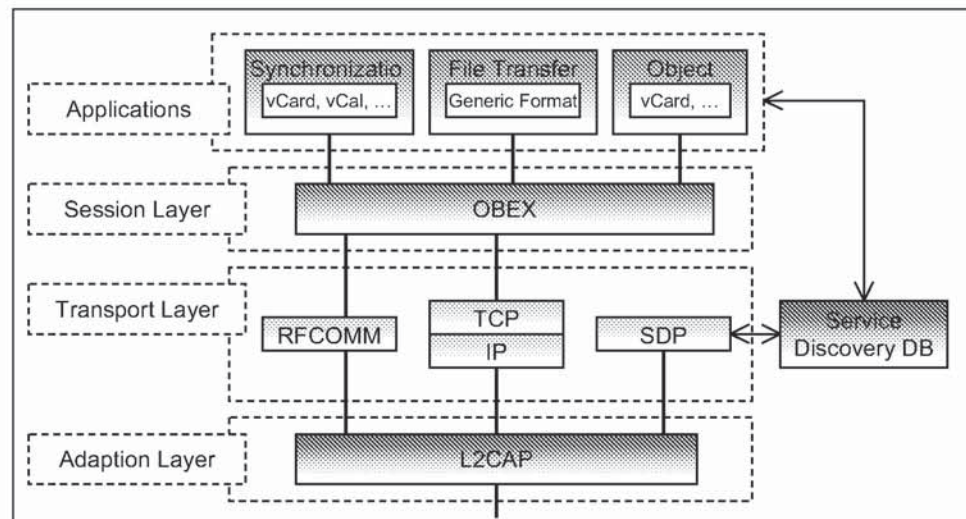


Figure 1.1: Part of Bluetooth Protocol Hierarchy

In the Bluetooth system, the purpose of the OBEX protocol is to enable the exchange of data objects. The typical example could be an object push of business cards to someone else. A more complex example is synchronizing calendars on multiple devices using OBEX. Also, the File Transfer applications can be implemented using OBEX. For the Object Push and Synchronization applications, content formats can be the vCard [4], vCalendar [5], vMessage [6], and vNotes [6] formats. The vCard, vCalendar, vMessage, and vNotes describe the formats for the electronic business card, the electronic calendar-ing and scheduling, the electronic message and mails, and the electronic notes, respectively.

## 1.2 BLUETOOTH OBEX-RELATED SPECIFICATIONS

Bluetooth Specification includes five separate specifications related to OBEX and applications using it:

### 1. Bluetooth IrDA Interoperability Specification (This specification)

- Defines how the applications can function over both Bluetooth and IrDA
- Specifies how OBEX is mapped over RFCOMM and TCP
- Defines the application profiles using OBEX over Bluetooth



## 2. Bluetooth Generic Object Exchange Profile Specification [7]

- Generic interoperability specification for the application profiles using OBEX
- Defines the interoperability requirements of the lower protocol layers (e.g. Baseband and LMP) for the application profiles

## 3. Bluetooth Synchronization Profile Specification [8]

- Application Profile for the Synchronization applications
- Defines the interoperability requirements for the applications within the Synchronization application profile
- Does not define the requirements for the Baseband, LMP, L2CAP, or RFCOMM.

## 4. Bluetooth File Transfer Profile Specification [9]

- Application Profile for the File Transfer applications
- Defines the interoperability requirements for the applications within the File Transfer application profile.
- Does not define the requirements for the Baseband, LMP, L2CAP, or RFCOMM.

## 5. Bluetooth Object Push Profile Specification [10]

- Application Profile for the Object Push applications
- Defines the interoperability requirements for the applications within the Object Push application profile.
- Does not define the requirements for the Baseband, LMP, L2CAP, or RFCOMM.

### 1.3 OTHER IROBEX IMPLEMENTATIONS

Over IR, OBEX has also been implemented over IrCOMM and Tiny TP. The Bluetooth technology does not define these protocols as transport protocols for OBEX, but they can be supported by independent software vendors if desired.



## 2 OBEX OBJECT AND PROTOCOL

---

This section is dedicated to the model of OBEX objects and the OBEX session protocol. The section is intended to be read with the IrOBEX specification [1].

### 2.1 OBJECT

The OBEX object model (Section 2 in [1]) describes how OBEX objects are presented. The OBEX protocol can transfer an object by using the **Put-** and **Get-**operations (See Section 2.2.3 and 2.2.4). One object can be exchanged in one or more **Put-**requests or **Get-**responses.

The model handles both information about the object (e.g. type) and object itself. It is composed of headers, which consist of a header ID and value (See Section 2.1 in [1]). The header ID describes what the header contains and how it is formatted, and the header value consists of one or more bytes in the format and meaning specified by Header ID. The specified headers are **Count**, **Name**, **Type**, **Length**, **Time**, **Description**, **Target**, **HTTP**, **Body**, **End of Body**, **Who**, **Connection ID**, **Application Parameters**, **Authenticate Challenge**, **Authenticate Response**, **Object Class**, and User-Defined Headers. These are explained in detail by Section 2.2 in the IrOBEX specification.

### 2.2 SESSION PROTOCOL

The OBEX operations are formed by **response-request** pairs. Requests are issued by the client and responses by the server. After sending a request, the client waits for a response from the server before issuing a new request. Each request packet consists of a one-byte opcode (See Section 3.3 in [1]), a two-byte length indicator, and required or optional data depending on the operation. Each response packet consists of a one-byte response code (See Section 3.2.1 in [1]), a two-byte length indicator, and required or optional data depending on the operation.

In the following subsections, the OBEX operations are explained in general.

### 2.2.1 Connect Operation

An OBEX session is started, when an application asks the first time to transmit an OBEX object. An OBEX client starts the establishment of an OBEX connection. The session is started by sending a **Connect**-request (See Section 3.3.1 in [1]). The request format is:

Byte 0	Bytes 1 and 2	Byte 3	Byte 4	Bytes 5 and 6	Byte 7 to n
0x80 (opcode)	Connect request packet length	OBEX version number	Flags	Maximum OBEX packet length	Optional headers

Note. The Big Endian format is used to define the byte ordering for the PDUs (requests and responses) in this specification as well as in the IrOBEX specification; i.e. the most significant byte (MSB) is always on left and the least significant byte (LSB) on right.

At the remote host, the **Connect**-request is received by the OBEX server, if it exists. The server accepts the connection by sending the successful response to the client. Sending any other response (i.e. a non-successful response) back to the client indicates a failure to make a connection. The response format is:

Byte 0	Bytes 1 and 2	Byte 3	Byte 4	Bytes 5 and 6	Byte 7 to n
Response code	Connect response packet length	OBEX version number	Flags	Maximum OBEX packet length	Optional headers

The response codes are list in the Section 3.2.1 in the IrOBEX specification. The bytes 5 and 6 define the maximum OBEX packet length, which can be received by the server. This value may differ from the length, which can be received by the client. These **Connect**-request and response packets must each fit in a single packet.

Once a connection is established it remains 'alive', and is only disconnected by requests/responses or by failures (i.e. the connection is not automatically disconnected after each OBEX object has completely transmitted).

### 2.2.2 Disconnect Operation

The disconnection of an OBEX session occurs when an application, which is needed for an OBEX connection, is closed or the application wants to change the host to which the requests are issued. The client issues the **Disconnect**-request (See Section 3.3.2 in [1]) to the server. The request format is:

Byte 0	Bytes 1 and 2	Byte 3
0x81	Packet length	Optional headers

The request cannot be refused by the server. Thus, it has to send the response, and the response format is:

Byte 0	Bytes 1 and 2	Byte 3
0xA0	Response packet length	Optional response headers

### 2.2.3 Put Operation

When the connection has been established between the client and server the client is able to push OBEX objects to the server. The **Put**-request is used to push an OBEX object (See Section 3.3.3 in [1]). The request has the following format.

Byte 0	Bytes 1 and 2	Byte 3
0x02 (0x82 when Final bit set)	Packet length	Sequence of headers

A **Put**-request consists of one or more request packets, depending on how large the transferred object is, and how large the packet size is. A response packet from the server is required for every **Put**-request packet. Thus, one response is not permitted for several request packets, although they consist of one OBEX object. The response format is:

Byte 0	Bytes 1 and 2	Byte 3
Response code	Response packet length	Optional response headers



### 2.2.4 Get Operation

When the connection has been established between the client and server, the client is also able to pull OBEX objects from the server. The **Get**-request is used to pull an OBEX object (See Section 3.3.4 in [1]). The request has the following format.

Byte 0	Bytes 1 and 2	Byte 3
0x03 (0x83 when Final bit set)	Packet length	Sequence of headers starting with Name

The object is returned as a sequence of headers, and the client has to send a request packet for every response packet. The response format is:

Byte 0	Bytes 1 and 2	Byte 3
Response code	Response packet length	Optional response headers

### 2.2.5 Other Operations

Other OBEX operations consist of a **SetPath**-, and an **Abort**-operation. These are carefully explained in the Sections 3.3.5-6 in the IrOBEX specification. It is important to note that the client can send an **Abort**-request after each response – even in the middle of a request/response sequence. Thus, the whole OBEX object does not have to be received before sending an **Abort**-request. In addition to these operations, the IrOBEX specification facilitates user-defined operations, but their use may not necessarily be adopted in Bluetooth.

### 3 OBEX OVER RFCOMM

---

This section specifies how OBEX is mapped over RFCOMM, which is the multiplexing and transport protocol based on ETSI TS 07.10 [11] and which also provides a support for serial cable emulation. The Bluetooth devices supporting the OBEX protocol must satisfy the following requirements.

1. The device supporting OBEX must be able to function as either a client, a server, or both
2. All servers running simultaneously on a device must use separate RFCOMM server channels
3. Applications (service/server) using OBEX must be able to register the proper information into the service discovery database. This information for different application profiles is specified in the profile specifications

#### 3.1 OBEX SERVER START-UP ON RFCOMM

When a client sends a connecting request, a server is assumed to be ready to receive requests. However, before the server is ready to receive (i.e. is running) certain prerequisites must be fulfilled before the server can enter the listening mode:

1. The server must open an RFCOMM server channel
2. The server must register its capabilities into the service discovery database

After this, other hosts are able to find the server if needed, and the server listens for get requests from clients.

#### 3.2 RECEIVING OBEX PACKETS FROM SERIAL PORT

As discussed earlier, one object can be exchanged over one or more **Put**-requests or **Get**-responses (i.e. the object is received in one or several packets). However, if OBEX is running directly over the serial port, it does not receive packets from RFCOMM. Instead, a byte stream is received by OBEX from a serial port emulated by RFCOMM.

To detect packets in the byte stream, OBEX has to look for opcodes or response codes (See Chapter 2.2) depending on whether a packet is a request or a response. The opcodes and response code can be thought of as the start flags of packets. In OBEX packets, there is no 'end flag' that would indicate the end of a packet. However, after the opcode or response code, the length of a packet is received in the next two bytes. Thus, the whole length of a packet is known, and the boundary of two packets can be determined.

All data that is not recognized must be dumped. This could cause a synchronization problem but, considering the nature of the OBEX protocol, this is not a problem over RFCOMM, which provides reliable transport over Bluetooth.

### 3.3 CONNECTION ESTABLISHMENT

A client initiates the establishment of a connection. However, the following sequence of tasks must occur before the client is able to send the first request for data:

1. By using the SD protocol described in the SDP specification [12], the client must discover the proper information (e.g. RFCOMM channel) associated with the server on which the connection can be established
2. The client uses the discovered RFCOMM channel to establish the RFCOMM connection
3. The client sends the **Connect**-request to the server, to establish an OBEX session. The session is established correctly if the client receives a successful response from the server

### 3.4 DISCONNECTION

The disconnection of an OBEX session over RFCOMM is straightforward. The disconnection is done by using the **Disconnect**-request (See Section 2.2.2). When the client has received the response, the next operation is to close the RFCOMM channel assigned to the OBEX client.

### 3.5 PUSHING AND PULLING OBEX PACKETS OVER RFCOMM

Data is pushed in OBEX packets over RFCOMM by using **Put**-requests (See Section 2.2.3). After each request, a response is required before the next request with the data can be pushed.

Pulling data from a remote host happens by sending a **Get**-request (See Section 2.2.4). The data arrives in OBEX response packets. After each response, a new request has to be sent, to pull more data.



## 4 OBEX OVER TCP/IP

---

This section specifies how OBEX is mapped over the TCP/IP creating reliable connection-oriented services for OBEX. This specification does not define how TCP/IP is mapped over Bluetooth.

The Bluetooth devices, which support the OBEX protocol over TCP/IP, must satisfy the following requirements:

1. The device supporting OBEX must be able to function as either a client, or a server, or both
2. For the server, the TCP port number 650 is assigned by IANA. If an assigned number is not desirable, the port number can be a value above 1023. However, the use of the TCP port number (650) defined by IANA is highly recommended. The 0-1023 range is reserved by IANA (See [13])
3. The client must use a port number (on the client side), which is not within the 0-1023 range
4. Applications (service/server) using OBEX must be able to register the proper information into the service discovery database. This information for different application profiles is specified in the profile specifications

### 4.1 OBEX SERVER START-UP ON TCP/IP

When a client sends a **Put-** or **Get-**request, a server is assumed to be ready to receive requests. However, when the server is ready (i.e. is running), certain prerequisites must be fulfilled before the server can enter the listening mode:

1. The server must initialize a TCP port with the value 650 or value above 1023
2. The server registers its capabilities into the service discovery database

After this, other devices are able to find the server if needed, and the server listens for get requests from clients.

### 4.2 CONNECTION ESTABLISHMENT

A client initiates a connection. However, the following sequence of tasks must occur before a connection can be established:

1. By using, the SD protocol described in the SDP specification [12], the client discovers the proper information (e.g. TCP port number) associated with the server, to enable the connection can be established
2. The client initializes a socket associated to a TCP port number above 1023, and establishes a TCP connection with the host of the server
3. The client sends the **Connect-**request to the server, to establish an OBEX session. The session is established correctly if the client receives a successful response from the server.

### 4.3 DISCONNECTION

The disconnection of an OBEX session over TCP is straightforward. The disconnection is done by using the **Disconnect**-request (See Section 2.2.2). When the client has received the response, the next operation is to close the TCP port dedicated for this session.

### 4.4 PUSHING AND PULLING OBEX PACKETS OVER TCP

See Section 3.5.

## 5 BLUETOOTH APPLICATION PROFILES USING OBEX

Bluetooth SIG (Special Interest Group) has defined three separate application profiles using OBEX. These profiles are briefly introduced in this section.

### 5.1 SYNCHRONIZATION

Basically, the synchronization means comparing two object stores, determining their inequalities, and then unifying these two object stores. The Bluetooth devices supporting the synchronization may be desktop PCs, notebooks, PDAs, cellular phones, or smart phones.

The Bluetooth Synchronization profile uses the servers and clients compliant to the IrMC synchronization specified by IrDA (See Section 5 in [6]).

The Bluetooth Synchronization servers and clients must support the level 4 synchronization functionality specified in the IrMC specification.

The actual logic of the synchronization engines which process the synchronization algorithm at the client device is implementation-specific. It is therefore left to the participating software vendors, and is not considered in the Bluetooth specifications.

The synchronization is not limited to one type of application. The Bluetooth synchronization (i.e. the IrMC synchronization) enables four different application classes:

1. Phone Book – provides a means for a user to manage contact records
2. Calendar – enables a user to manage calendar items, and can also be used for 'to-do' or task lists
3. Messaging – lets a user manage messages (e.g. e-mails)
4. Notes – provides a means for a user to manage small notes

The interoperability requirements for the Bluetooth Synchronization profile are defined in the Synchronization Profile [8] and Generic Object Exchange Profile [7] specifications.

### 5.2 FILE TRANSFER

At the minimum, the File Transfer profile is intended for sending and retrieving generic files to and from the Bluetooth device. The File Transfer service also facilitates the browsing of the remote Bluetooth device's folder.

The interoperability requirements for the Bluetooth File Transfer profile are defined in the File Transfer Profile [9] and Generic Object Exchange Profile [7] specifications.



### 5.3 OBJECT PUSH

The Object Push profile is the special case of the File Transfer Profile for beaming objects and optionally pulling the default objects. At a minimum, it offers the capability to exchange business cards, but is not limited to this service.

The interoperability requirements for the Object Push profile are defined in the Object Push Profile [10] and Generic Object Exchange Profile [7] specifications.

## 6 REFERENCES

---

- [1] Infrared Data Association, IrDA Object Exchange Protocol (IrOBEX), Version 1.2, April 1999
- [2] Bluetooth RFCOMM with TS 07.10, on page 385
- [3] Internet Engineering Task Force, IETF Directory List of RFCs (<http://www.ietf.org/rfc/>), May 1999.
- [4] The Internet Mail Consortium, vCard - The Electronic Business Card Exchange Format, Version 2.1, September 1996.
- [5] The Internet Mail Consortium, vCalendar - The Electronic Calendaring and Scheduling Exchange Format, Version 1.0, September 1996.
- [6] Infrared Data Association, IrMC (Ir Mobile Communications) Specification, Version 1.1, February 1999.
- [7] Bluetooth Generic Object Exchange Profile, see Volume 2.
- [8] Bluetooth Synchronization Profile, see Volume 2.
- [9] Bluetooth File Transfer Profile, see Volume 2.
- [10] Bluetooth Object Push Profile, see Volume 2.
- [11] ETSI, TS 07.10, Version 6.3.0
- [12] Bluetooth Service Discovery Protocol, see Volume 2.
- [13] Internet Assigned Numbers Authority, IANA Protocol/Number Assignments Directory (<http://www.iana.org/numbers.html>), May 1999.

**7 LIST OF ACRONYMS AND ABBREVIATIONS**

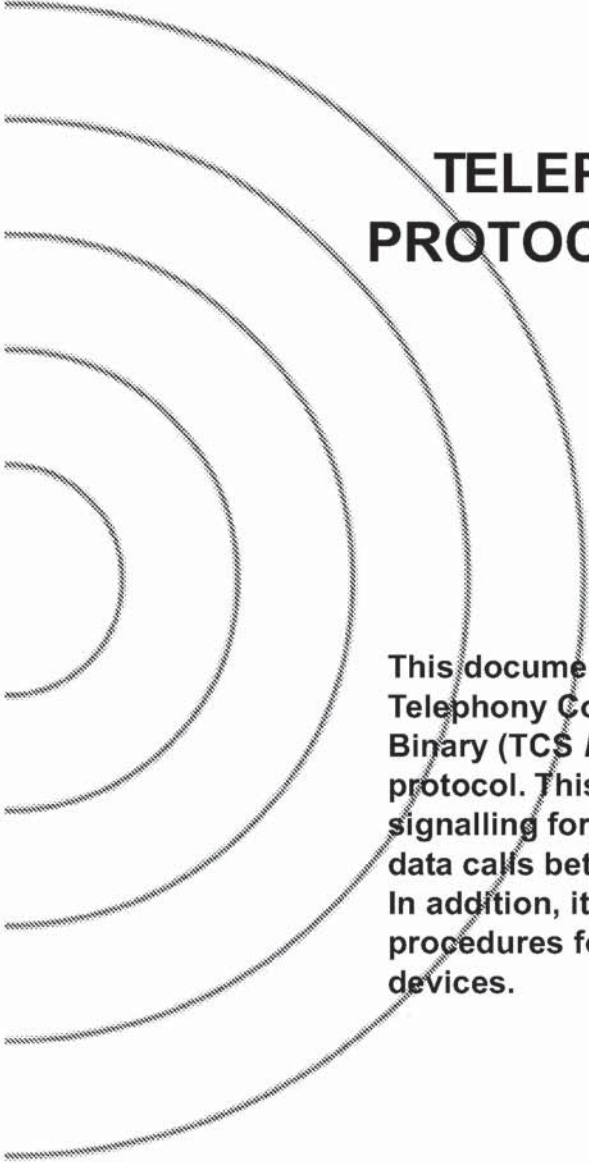
<b>Abbreviation or Acronym</b>	<b>Meaning</b>
GEOP	Generic Object Exchange Profile
IrDA	Infrared Data Association
IrMC	Ir Mobile Communications
L2CAP	Logical Link Control and Adaptation Protocol
LSB	Least Significant Byte
MSB	Most Significant Byte
OBEX	Object exchange protocol
PDU	Protocol Data Unit
RFCOMM	Serial cable emulation protocol based on ETSI TS 07.10
SD	Service Discovery
SDP	Service Discovery Protocol
SDDB	Service Discovery Database
TCP/IP	Transport Control Protocol/Internet Protocol



## Part F:3

# TELEPHONY CONTROL PROTOCOL SPECIFICATION

TCS Binary



This document describes the Bluetooth Telephony Control protocol Specification – Binary (TCS *Binary*), using a bit-oriented protocol. This protocol defines the call control signalling for the establishment of speech and data calls between Bluetooth devices. In addition, it defines mobility management procedures for handling Bluetooth TCS devices.



**CONTENTS**

<b>1</b>	<b>General Description .....</b>	<b>435</b>
1.1	Overview .....	435
1.2	Operation between devices.....	435
1.3	Operation between layers .....	437
<b>2</b>	<b>Call Control (CC) .....</b>	<b>439</b>
2.1	Call States .....	439
2.2	Call Establishment .....	439
2.2.1	Call Request.....	439
2.2.2	Bearer selection .....	440
2.2.3	Overlap Sending.....	441
2.2.4	Call Proceeding .....	441
2.2.4.1	Call proceeding, enbloc sending.....	441
2.2.4.2	Call proceeding, overlap sending.....	442
2.2.4.3	Expiry of timer T310.....	442
2.2.5	Call Confirmation.....	442
2.2.6	Call Connection .....	442
2.2.7	Call Information .....	443
2.2.8	Non-selected user clearing.....	443
2.2.9	In-band tones and announcements.....	443
2.2.10	Failure of call establishment.....	444
2.2.11	Call Establishment Message Flow .....	445
2.3	Call Clearing.....	446
2.3.1	Normal Call Clearing .....	446
2.3.2	Abnormal Call Clearing .....	447
2.3.3	Clear Collision .....	447
2.3.4	Call Clearing Message Flow.....	448



<b>3</b>	<b>Group Management (GM)</b> .....	<b>449</b>
3.1	Overview.....	449
3.2	The Wireless User Group.....	449
3.2.1	Description.....	449
3.2.2	Encryption within the WUG.....	450
3.2.3	Unconscious pairing.....	450
3.3	Obtain Access Rights.....	451
3.3.1	Procedure description.....	451
3.3.2	Message flow.....	451
3.4	Configuration Distribution.....	452
3.4.1	Procedure Description.....	452
3.4.2	Message flow.....	452
3.5	Fast inter-member Access.....	453
3.5.1	Listen Request.....	453
3.5.2	Listen Accept.....	453
3.5.3	Listen Reject by the WUG Master.....	454
3.5.4	Listen Reject by the WUG Member.....	454
3.5.5	Message flow.....	454
<b>4</b>	<b>Connectionless TCS (CL)</b> .....	<b>455</b>
<b>5</b>	<b>Supplementary Services (SS)</b> .....	<b>456</b>
5.1	Calling Line Identity.....	456
5.2	DTMF start & stop.....	456
5.2.1	Start DTMF request.....	457
5.2.2	Start DTMF response.....	457
5.2.3	Stop DTMF request.....	457
5.2.4	Stop DTMF response.....	457
5.2.5	Message flow.....	457
5.3	Register Recall.....	458

<b>6</b>	<b>Message formats</b> .....	<b>459</b>
6.1	Call Control Message Formats.....	460
6.1.1	ALERTING .....	460
6.1.2	CALL PROCEEDING .....	460
6.1.3	CONNECT.....	461
6.1.4	CONNECT ACKNOWLEDGE .....	461
6.1.5	DISCONNECT.....	462
6.1.6	INFORMATION .....	462
6.1.7	PROGRESS .....	463
6.1.8	RELEASE.....	463
6.1.9	RELEASE COMPLETE .....	464
6.1.10	SETUP .....	464
6.1.11	SETUP ACKNOWLEDGE .....	465
6.1.12	Start DTMF .....	465
6.1.13	Start DTMF Acknowledge.....	466
6.1.14	Start DTMF Reject.....	466
6.1.15	Stop DTMF .....	466
6.1.16	Stop DTMF Acknowledge.....	467
6.2	Group Management Message Formats .....	467
6.2.1	ACCESS RIGHTS REQUEST.....	467
6.2.2	ACCESS RIGHTS ACCEPT .....	467
6.2.3	ACCESS RIGHTS REJECT .....	468
6.2.4	INFO SUGGEST .....	468
6.2.5	INFO ACCEPT .....	468
6.2.6	LISTEN REQUEST .....	469
6.2.7	LISTEN SUGGEST .....	469
6.2.8	LISTEN ACCEPT .....	469
6.2.9	LISTEN REJECT.....	470
6.3	TCS Connectionless Message Formats.....	470
6.3.1	CL INFO .....	470
<b>7</b>	<b>Message coding</b> .....	<b>471</b>
7.1	Overview .....	471
7.2	Protocol Discriminator.....	472
7.3	Message Type.....	472
7.4	Other Information Elements .....	474
7.4.1	Coding rules .....	474
7.4.2	Audio control .....	476
7.4.3	Bearer capability.....	476
7.4.4	Call class.....	479

7.4.5	Called party number .....	480
7.4.6	Calling party number .....	481
7.4.7	Cause .....	482
7.4.8	Clock offset .....	482
7.4.9	Company specific .....	483
7.4.10	Configuration data .....	484
7.4.11	Destination CID .....	485
7.4.12	Keypad facility .....	485
7.4.13	Progress indicator .....	485
7.4.14	SCO Handle .....	486
7.4.15	Sending complete .....	486
7.4.16	Signal .....	486
<b>8</b>	<b>Message Error handling .....</b>	<b>487</b>
8.1	Protocol Discrimination Error .....	487
8.2	Message Too Short or Unrecognized .....	487
8.3	Message Type or Message Sequence Errors .....	487
8.4	Information Element Errors .....	487
<b>9</b>	<b>Protocol Parameters .....</b>	<b>489</b>
9.1	Protocol Timers .....	489
<b>10</b>	<b>References .....</b>	<b>490</b>
<b>11</b>	<b>List of Figures .....</b>	<b>491</b>
<b>12</b>	<b>List of Tables .....</b>	<b>492</b>
	<b>Appendix 1 - TCS Call States .....</b>	<b>493</b>



## 1 GENERAL DESCRIPTION

### 1.1 OVERVIEW

The Bluetooth Telephony Control protocol Specification Binary (TCS *Binary*) is based on the ITU-T Recommendation Q.931[1], applying the symmetrical provisions as stated in Annex D of Q.931. The resulting text does not discriminate between user and network side, but merely between Outgoing Side (the party originating the call) and Incoming Side (the party terminating the call). Effort was made to only apply those changes necessary for Bluetooth and foreseen applications, enabling re-use of Q.931 to the largest extent possible.

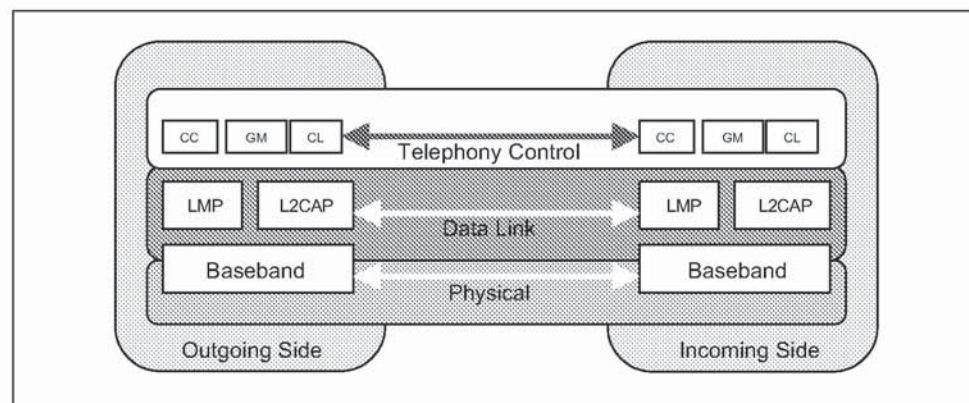


Figure 1.1: TCS within the Bluetooth stack

The TCS contains the following functionality:

- Call Control (CC) – signalling for the establishment and release of speech and data calls between Bluetooth devices
- Group Management – signalling to ease the handling of groups of Bluetooth devices
- ConnectionLess TCS (CL) – provisions to exchange signalling information not related to an ongoing call

### 1.2 OPERATION BETWEEN DEVICES

TCS uses point-to-point signalling and may use point-to-multipoint signalling. Point-to-point signalling is used when it is known to which side (Bluetooth device) a call needs to be established (*single-point configuration*).

Point-to-multipoint signalling may be used when there are more sides available for call establishment (*multi-point configuration*); e.g. when, for an incoming call, a home base station needs to alert all phones in range.

Point-to-point signalling is mapped towards a connection-oriented L2CAP channel, whereas point-to-multipoint signalling is mapped towards the connectionless L2CAP channel, which in turn is sent as broadcast information on the beacon channel (piconet broadcast).

Figure 1.2 illustrates point-to-point signalling to establish a voice or data call in a single-point configuration. First the other device is notified of the call request using the point-to-point signalling channel (A). Next, this signalling channel is used to further establish the speech or data channel (B).

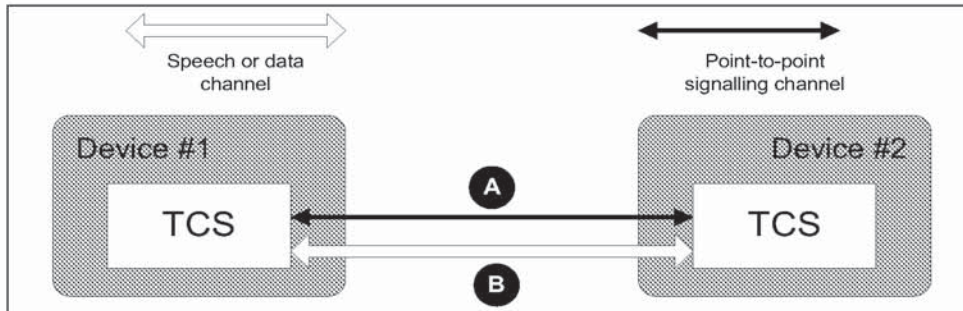


Figure 1.2: Point-to-point signalling in a single-point configuration

Figure 1.3 below illustrates how point-to-multipoint signalling and point-to-point signalling is used to establish a voice or data call in a multi-point configuration. First all devices are notified of the call request using point-to-multipoint signalling channel (A). Next, one of the devices answers the call on the point-to-point signalling channel (B); this signalling channel is used to further establish the speech or data channel (C).

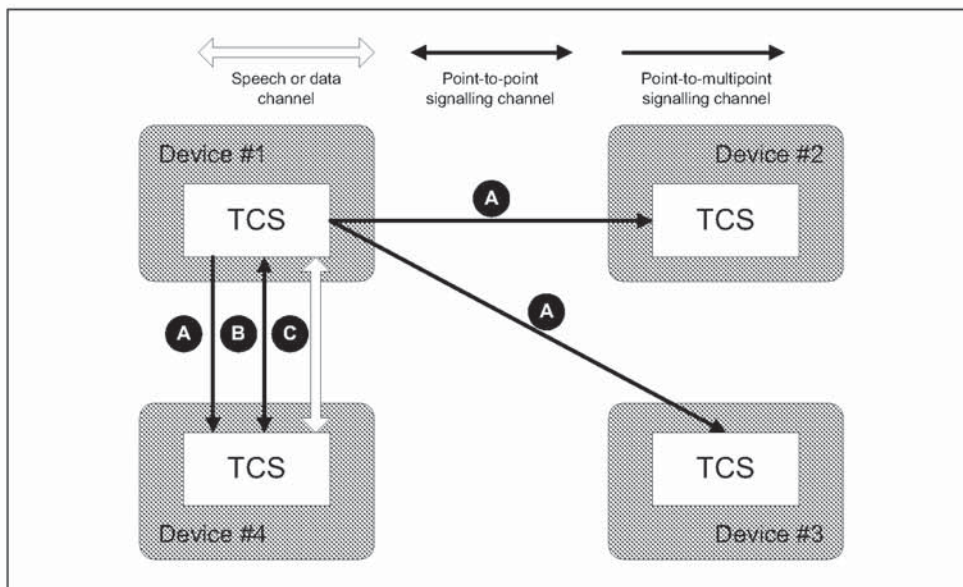


Figure 1.3: Signalling in a multi-point configuration

### 1.3 OPERATION BETWEEN LAYERS

TCS implementations should follow the general architecture described below (note that, for simplicity, handling of data calls is not drawn).

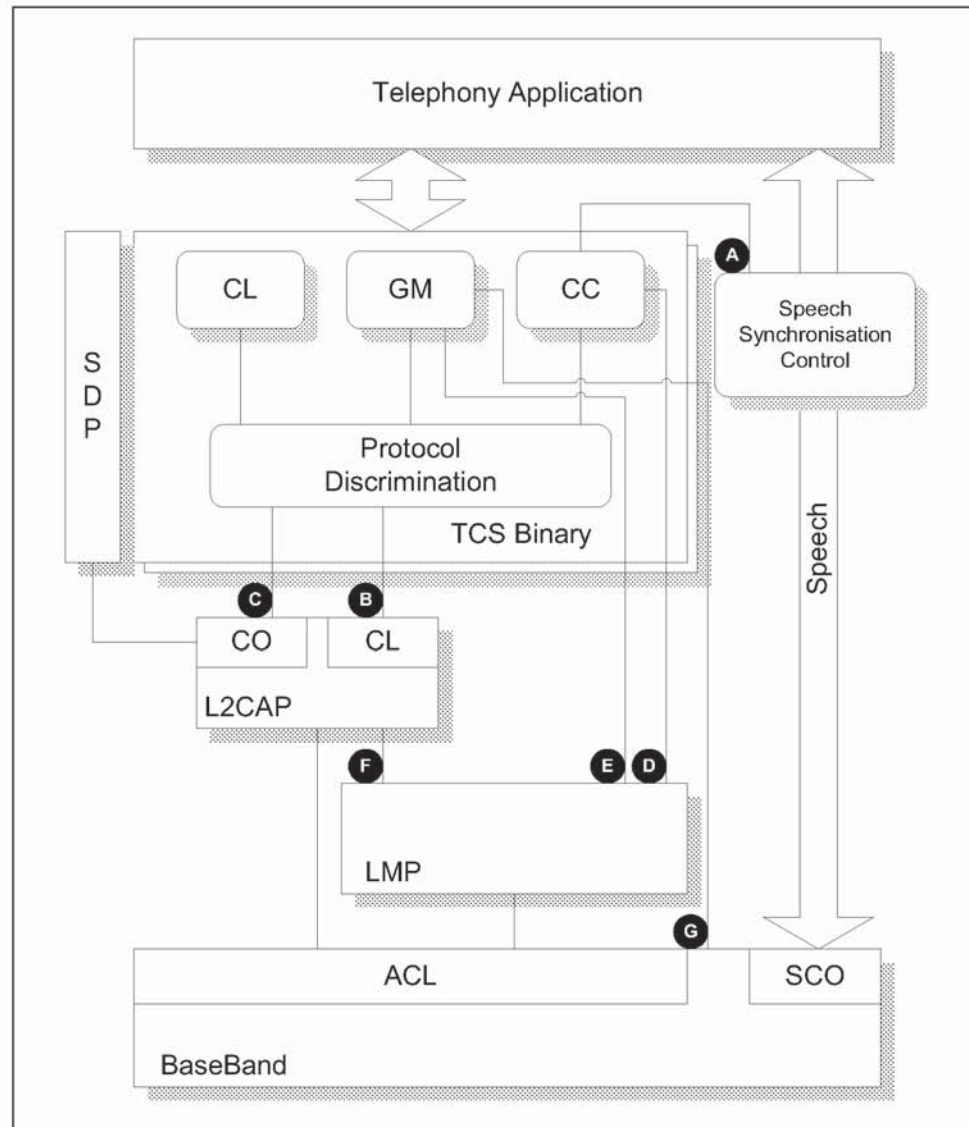


Figure 1.4: TCS Architecture

The internal structure of TCS Binary contains the functional entities Call Control, Group Management and ConnectionLess as described in Section 1.1 on page 435, complemented with the Protocol Discrimination which, based upon the TCS internal protocol discriminator, routes traffic to the appropriate functional entity.



To handle more calls simultaneously, multiple instances of TCS Binary may exist at the same time. Discrimination between the multiple instances can be based on the L2CAP channel identifier.

TCS Binary interfaces with a number of other (Bluetooth) entities to provide its (telephone) services to the application. The interfaces are identified in Figure 1.4 above, and information is exchanged across these interfaces for the following purposes:

- A The Call Control entity provides information to the speech synchronization control about when to connect (disconnect) the speech paths. This information is based upon the call control messages (e.g. reception of CONNECT ACKNOWLEDGE or DISCONNECT, see Section 2 on page 439)
- B To send a SETUP message (see Section 2.2.1 on page 439) using point-to-multipoint signalling, it is delivered on this interface to L2CAP for transmission on the connectionless channel. The other way round – L2CAP uses this interface to inform TCS of a SETUP message received on the connectionless channel. The connectionless L2CAP channel maps onto the piconet broadcast
- C Whenever a TCS message needs to be sent using point-to-point signalling, it is delivered on this interface to L2CAP for transmission on a connection-oriented channel. During L2CAP channel establishment specific quality of service to be used for the connection will be indicated, in particular the usage of low power modes (L2CAP will inform LMP about this – interface F)
- D The Call Control entity controls the LMP directly, for the purpose of establishing and releasing SCO links
- E & G. The Group Management entity controls the LMP and LC/Baseband directly during initialization procedures to control (for example) the inquiry, paging and pairing.

## 2 CALL CONTROL (CC)

### 2.1 CALL STATES

The call states used by the TCS are those identified in Q.931[1], for the user side only. To allow for implementation within computing power- and memory-restricted devices, only a subset of the states is mandatory for TCS based implementations. This mandatory subset is termed **Lean TCS**.

The states are named as follows. States in bold are mandatory states, part of Lean TCS:

#### General States

- Null (0)**
- Active (10)**
- Disconnect request (11)**
- Disconnect indication (12)**
- Release request (19)**

#### Outgoing Side States

- Call initiated (1)**
- Overlap sending (2)
- Outgoing call proceeding (3)
- Call delivered (4)

#### Incoming Side States

- Call present (6)**
- Call received (7)
- Connect request (8)**
- Incoming call proceeding (9)
- Overlap receiving (25)

These states, together with the state transitions, have been indicated in the state diagram contained in Appendix 1 – TCS Call States. For clarity, a separate state diagram has been included for Lean TCS.

### 2.2 CALL ESTABLISHMENT

A connection-oriented L2CAP channel between the Outgoing and Incoming Side shall be available before any of the CC procedures can operate.

Additionally, in a multi-point configuration (see Section 1.2 on page 435), a connectionless L2CAP channel shall be available between the Outgoing and Incoming Side.

#### 2.2.1 Call Request

The Outgoing Side initiates call establishment by sending a SETUP message, and starting timer T303.



In case of a single-point configuration (see Section 1.2 on page 435), the SETUP message is delivered on the connection-oriented channel.

In case of a multi-point configuration (see Section 1.2 on page 435), the SETUP message may be delivered on the connection-less channel. This causes the SETUP message to be transmitted as a broadcast message at every beacon instant (as described in Baseband Specification Section 10.8.4 on page 115).

If no response (as prescribed in Section 2.2.4 on page 441) is received from the Incoming Side before timer T303 expires, the Outgoing Side shall:

1. If the SETUP message was delivered on a connection-less channel, return to the Null state. This stops the transmission of the SETUP message.
2. If the SETUP message was delivered on a connection-oriented channel, send a RELEASE COMPLETE message to the Incoming Side. This message should contain cause # 102, *recovery on timer expiry*.

The SETUP message shall always contain the call class. It shall also contain all the information required by the Incoming Side to process the call. The number digits within the Called party number information element may optionally be incomplete, thus requiring the use of overlap sending (Section 2.2.3 on page 441). The SETUP message may optionally contain the Sending complete information element in order to indicate that the number is complete.

Following the transmission of the SETUP message, the Outgoing Side shall enter the Call initiated state. On receipt of the SETUP message the Incoming Side shall enter the Call present state.

### **2.2.2 Bearer selection**

The SETUP message sent during the Call Request may contain the Bearer capability information element, to indicate the requested bearer. The Incoming Side may negotiate on the requested bearer by including a Bearer capability information element in the first message in response to the SETUP message.

The Bearer capability information element indicates which lower layer resources (the *bearer channel*) are used during a call. If bearer capability 'Synchronous Connection-Oriented (SCO)' is indicated, an SCO link will be used, with the indicated packet type and voice coding to enable speech calls. If bearer capability 'Asynchronous Connection-Less (ACL)' is indicated, an ACL link will be used. On top of this, there will be an L2CAP channel with indicated QoS requirements, to enable data calls. If bearer capability 'None' is indicated, no separate bearer channel will be established.

*Note: it is the responsibility of the implementation to assure that the bearer capability as indicated is available to the call.*



### 2.2.3 Overlap Sending

If the received SETUP message does not contain a Sending complete indication information element, and contains either –

- a) incomplete called-number information, or
- b) called-number information which the Incoming Side cannot determine to be complete,

then the Incoming Side shall start timer T302, send a SETUP ACKNOWLEDGE message to the Outgoing Side, and enter the Overlap receiving state.

When the SETUP ACKNOWLEDGE message is received, the Outgoing Side shall enter the Overlap sending state, stop timer T303, and start timer T304.

After receiving the SETUP ACKNOWLEDGE message, the Outgoing Side shall send the remainder of the call information (if any) in the called party number information element of one or more INFORMATION messages.

The Outgoing Side shall restart timer T304 when each INFORMATION message is sent.

The INFORMATION message, which completes the information sending, may contain a sending complete information element. The Incoming Side shall restart timer T302 on receipt of every INFORMATION message not containing a sending complete indication, if it cannot determine that the called party number is complete.

At the expiry of timer T304, the Outgoing Side shall initiate call clearing in accordance with Section 2.3.1 with cause #102, *recovery on timer expiry*.

At the expiry of timer T302, the Incoming Side shall:

- if it determines that the call information is incomplete, initiate call clearing in accordance with Section 2.3.1 with cause #28, *invalid number format*.
- otherwise the Incoming Side shall reply with a CALL PROCEEDING, ALERTING or CONNECT message.

### 2.2.4 Call Proceeding

#### 2.2.4.1 Call proceeding, enbloc sending

If enbloc sending is used (i.e. the Incoming Side can determine it has received sufficient information in the SETUP message from the Outgoing Side to establish the call) the Incoming Side shall send a CALL PROCEEDING message to the Outgoing Side to acknowledge the SETUP message and to indicate that the call is being processed. Upon receipt of the CALL PROCEEDING message, the Outgoing Side shall enter the Outgoing Call proceeding state stop

timer T303 and start timer T310. After sending the CALL PROCEEDING message, the Incoming Side shall enter the Incoming Call proceeding state.

#### 2.2.4.2 Call proceeding, overlap sending

Following the occurrence of one of these conditions –

- the receipt by the Incoming Side of a Sending complete indication, or
- analysis by the Incoming Side that all call information necessary to effect call establishment has been received,

the Incoming Side shall send a CALL PROCEEDING message to the Outgoing Side, stop timer T302, and enter the Incoming Call proceeding state.

When the Outgoing Side receives of the CALL PROCEEDING message it shall enter the Outgoing Call proceeding state, stop timer T304 and, if applicable, start timer T310.

#### 2.2.4.3 Expiry of timer T310

On expiry of T310 (i.e. if the Outgoing Side does not receive an ALERTING, CONNECT, DISCONNECT or PROGRESS message), the Outgoing Side shall initiate call clearing in accordance with Section 2.3.1 on page 446 with cause #102, *recovery on timer expiry*.

### **2.2.5 Call Confirmation**

Upon receiving an indication that user alerting has been initiated at the called address, the Incoming Side shall send an ALERTING message, and shall enter the Call received state.

When the Outgoing Side receives the ALERTING message, the Outgoing Side may begin an internally generated alerting indication and shall enter the Call delivered state. The Outgoing Side shall stop timer T304 (in case of overlap receiving), stop timer T303 or T310 (if running), and start timer T301 (unless another internal altering supervision timer function exists).

On expiry of T301, the Outgoing Side shall initiate call clearing in accordance with Section 2.3.1 on page 446 with cause #102, *recovery on timer expiry*.

### **2.2.6 Call Connection**

An Incoming Side indicates acceptance of an incoming call by sending a CONNECT message to the Outgoing Side, and stopping the user alerting. Upon sending the CONNECT message the Incoming Side shall start timer T313.



On receipt of the CONNECT message, the Outgoing Side shall stop any internally generated alerting indications, shall stop (if running) timers T301, T303, T304, and T310, shall complete the requested bearer channel to the Incoming Side, shall send a CONNECT ACKNOWLEDGE message, and shall enter the Active state.

The CONNECT ACKNOWLEDGE message indicates completion of the requested bearer channel. Upon receipt of the CONNECT ACKNOWLEDGE message, the Incoming Side shall connect to the bearer channel, stop timer T313 and enter the Active state.

When timer T313 expires prior to the receipt of a CONNECT ACKNOWLEDGE message, the Incoming Side shall initiate call clearing in accordance with Section 2.3.1 on page 446 with cause #102, *recovery on timer expiry*.

### 2.2.7 Call Information

While in the Active state, both sides may exchange any information related to the ongoing call using INFORMATION messages.

### 2.2.8 Non-selected user clearing

When the call has been delivered on a connection-less channel (in case of a multi-point configuration), in addition to sending a CONNECT ACKNOWLEDGE message to the Incoming Side selected for the call, the Outgoing Side shall send a RELEASE message (indicating cause #26, *non-selected user clearing*) to all other Incoming Sides that have sent SETUP ACKNOWLEDGE, CALL PROCEEDING, ALERTING, or CONNECT messages in response to the SETUP message. These RELEASE messages are used to notify the Incoming Sides that the call is no longer offered to them.

### 2.2.9 In-band tones and announcements

When the Incoming Side provides in-band tones/announcements, and if the requested bearer implies speech call, the Incoming Side will first complete the bearer channel (if not already available). Then a progress indicator #8, *in-band information or appropriate pattern is now available* is sent simultaneously with the application of the in-band tone/announcement. This progress indicator may be included in any call control message that is allowed to contain the progress indicator information element or, if there is no call state change, in a dedicated PROGRESS message.

Upon receipt of this message, the Outgoing Side may connect (if not already connected) to the bearer channel to receive the in-band tone/announcement.



### 2.2.10 Failure of call establishment

In the Call present, Overlap receiving, Incoming call proceeding, or Call received states, the Incoming Side may initiate clearing as described in Section 2.3 on page 446 with a cause value indicated. Examples of some the cause values that may be used to clear the call, when the Incoming Side is in the Call present, Overlap receiving, or Incoming call proceeding state are the following:

- #1 unassigned (unallocated) number*
- #3 no route to destination*
- #17 user busy*
- #18 no user responding*
- #22 number changed*
- #28 invalid number format (incomplete number)*
- #34 no circuit/channel available*
- #44 requested circuit/channel not available*
- #58 bearer capability not presently available*
- #65 bearer capability not implemented*

Examples of two of the cause values that may be used to clear the call when the Incoming Side is in the Call received state are as follows:

- #19 no answer from user (user alerted)*
- #21 call rejected by user*

**2.2.11 Call Establishment Message Flow**

The figure below provides a complete view of the messages exchanged during successful Call Establishment, as described in the sections above. The mandatory messages, part of the Lean TCS, are indicated by a solid arrow. A dotted arrow indicates the optional messages. A triangle indicates a running timer.

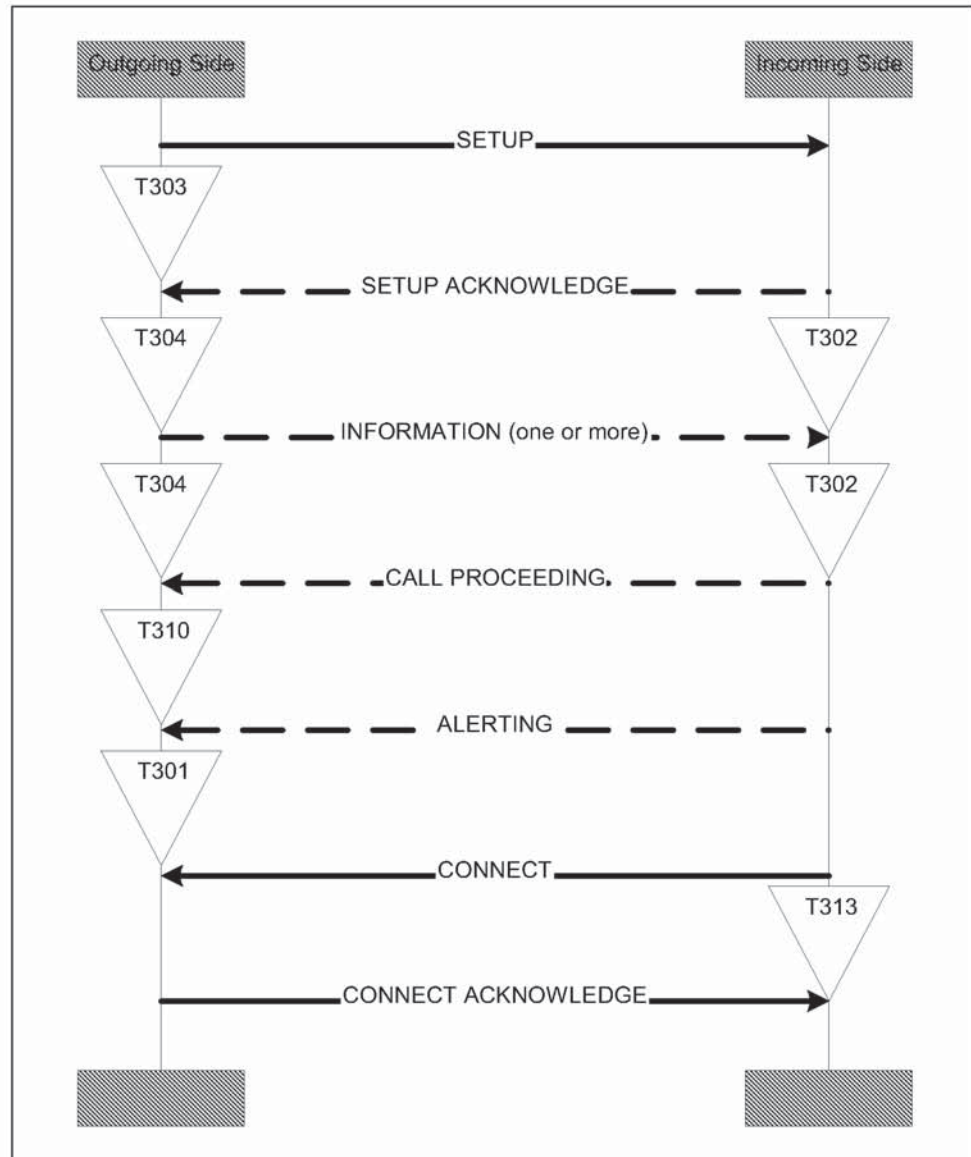


Figure 2.1: Call establishment message flow

## 2.3 CALL CLEARING

### 2.3.1 Normal Call Clearing

Apart from the exceptions identified in Section 2.3.2 on page 447, the clearing procedures are symmetrical and may be initiated by either the Outgoing or the Incoming Side. In the interest of clarity, the following procedures describe only the case where the Outgoing Side initiates clearing.

On sending or receiving any call clearing message, any protocol timer other than T305 and T308 shall be stopped.

The Outgoing Side shall initiate clearing by sending a DISCONNECT message, starting timer T305, disconnecting from the bearer channel, and entering the Disconnect request state.

The Incoming Side shall enter the Disconnect indication state upon receipt of a DISCONNECT message. This message prompts the Incoming Side to disconnect from the bearer channel. Once the channel used for the call has been disconnected, the Incoming Side shall send a RELEASE message to the Outgoing Side, start timer T308, and enter the Release request state.

On receipt of the RELEASE message the Outgoing Side shall cancel timer T305, release the bearer channel, send a RELEASE COMPLETE message, and return to the Null state.

Following the receipt of a RELEASE COMPLETE message from the Outgoing Side, the Incoming Side shall stop timer T308, release the bearer channel, and return to the Null state.

If the Outgoing Side does not receive a RELEASE message in response to the DISCONNECT message before timer T305 expires, it shall send a RELEASE message to the Incoming Side with the cause number originally contained in the DISCONNECT message, start timer T308 and enter the Release request state.

If in the Release request state, a RELEASE COMPLETE message is not received before timer T308 expires, the side that expected the message shall return to the Null state.

#### **Clearing by the called user employing user-provided tones/announcements**

In addition to the procedures described above, if the requested bearer signals a speech call, the Outgoing Side may apply in-band tones/announcements in the clearing phase. When in-band tones/announcements are provided, the Outgoing Side will first complete the bearer channel (if not already available), and next send the DISCONNECT message containing progress indicator #8, *in-band information or appropriate pattern is now available*.



Upon receipt of this message, the Incoming Side may connect (if not already connected) to the bearer channel to receive the in-band tone/announcement, and enter the Disconnect indication state.

The Incoming Side may subsequently continue clearing (before the receipt of a RELEASE from the Outgoing Side) by disconnecting from the bearer channel, sending a RELEASE message, starting timer T308, and entering the Release request state.

### **2.3.2 Abnormal Call Clearing**

Under normal conditions, call clearing is initiated when either side sends a DISCONNECT message and follows the procedures defined in Section 2.3.1 on page 446. The only exceptions to the above rule are as follows:

- a In response to a SETUP message, the Incoming Side can reject a call (e.g. because of unavailability of suitable resources) by responding with a RELEASE COMPLETE message provided no other response has previously been sent, and enter the Null state
- b In case of a multi-point configuration, non-selected user call clearing will be initiated with RELEASE message(s) from the Outgoing Side (Section 2.2.8 on page 443)
- c In case of a multi-point configuration, where the SETUP message is delivered on an connection-less channel, if a remote (calling) user disconnect indication is received during call establishment, any Incoming Side which has responded, or subsequently responds, shall be cleared by a RELEASE message, and the procedures of Section 2.3.1 on page 446 are then followed for that user. The Outgoing Side enters the Null state upon completion of clearing procedures for all responding Incoming Sides.

### **2.3.3 Clear Collision**

Clear collision occurs when the Incoming and the Outgoing Sides simultaneously transfer DISCONNECT messages. When either side receives a DISCONNECT message while in the Disconnect request state, the side shall stop timer T305, disconnect the bearer channel (if not disconnected), send a RELEASE message, start timer T308, and enter the Release request state.

Clear collision can also occur when both sides simultaneously transfer RELEASE messages. The entity receiving such a RELEASE message while within the Release request state shall stop timer T308, release the bearer channel, and enter the Null state (without sending or receiving a RELEASE COMPLETE message).

**2.3.4 Call Clearing Message Flow**

The figure below provides the complete view on the messages exchanged during normal Call Clearing, as described in the sections above. All messages are mandatory.

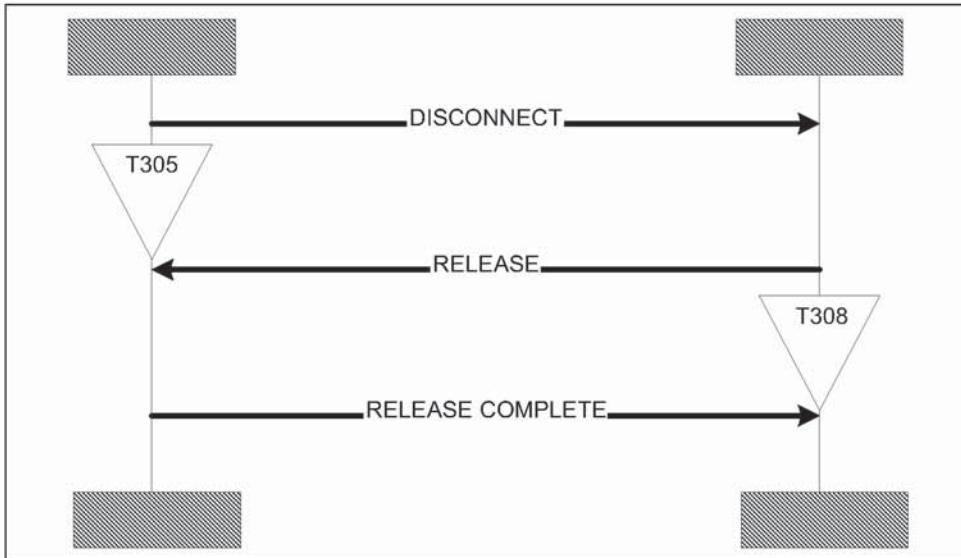


Figure 2.2: Call clearing message flow

## **3 GROUP MANAGEMENT (GM)**

---

### **3.1 OVERVIEW**

The Group Management entity provides procedures for managing a group of devices.

The following procedures are supported:

- Obtain access rights (Section 3.3 on page 451)  
enables the requesting device to use the telephony services of another device, part of a group of devices
- Configuration distribution (Section 3.4 on page 452)  
facilitates the handling and operation of a group of devices
- Fast inter-member access (Section 3.5 on page 453)  
enables faster contact establishment between devices of the same group

A connection-oriented L2CAP channel between devices shall be available before any of the GM procedures can operate.

For group management, the concept of Wireless User Group (WUG) is used.

### **3.2 THE WIRELESS USER GROUP**

#### **3.2.1 Description**

A WUG consists of a number of Bluetooth units supporting TCS. One of the devices is called the WUG master. The WUG master is typically a gateway, providing the other Bluetooth devices – called WUG members – with access to an external network. All members of the WUG in range are members of a piconet (active or parked). Master of this piconet is always the WUG master.

The main relational characteristics of a WUG are:

- All units that are part of a WUG know which unit is the WUG master and which other units are member of this WUG. WUG members receive this information from the WUG master.
- When a new unit has paired with the WUG master, it is able to communicate and perform authentication and encryption with any other unit part of the WUG without any further pairing/initialization. The WUG master provides the required authentication and encryption parameters to the WUG members.

Both relational characteristics are maintained through the Configuration distribution procedure.



### 3.2.2 Encryption within the WUG

In order to allow for encrypted transmission on the connectionless L2CAP channel, the WUG master issues a temporary key ( $K_{\text{master}}$ ). As a Bluetooth unit is not capable of switching between two or more encryption keys in real time, this key is normally also used for encrypted transmission on the connection-oriented channel (individually addressed traffic). Since the WUG master piconet may be in operation for extended periods without interruption, the  $K_{\text{master}}$  shall be changed periodically.

In order to allow for authentication and encryption to be performed between WUG members, the WUG master may use the Configuration distribution procedure to issue link keys that the WUG members use for communication with each other. Just as if pairing had created these keys, the keys are unique to a pair of WUG members and hence a WUG member uses a different key for every other WUG member it connects to.

The Configuration distribution shall always be performed using encrypted links. The  $K_{\text{master}}$  shall not be used for encryption; rather the WUG master shall ensure that the semi-permanent key for the specific WUG member addressed shall be used.

### 3.2.3 Unconscious pairing

For TCS, pairing a device with the WUG master implies pairing a device with all members of the WUG. This is achieved using the Configuration distribution procedure. This avoids the user of the device having to pair with each and every device of the WUG individually.

In Bluetooth, pairing is not related to a specific service but rather to a specific device. After pairing, all services provided by a device are accessible, if no further application- or device-specific protection is provided.

Without further provisions, pairing a device with the WUG master implies that all services provided by the new device are accessible to all other WUG members. And vice versa, without further provisions, the new device can access all services provided by other WUG members.

For this reason, implementers of TCS – and in particular the Configuration distribution procedure – are recommended to add provisions where:

1. a new device entering the WUG is not mandated to initiate the Obtain access rights procedure to become a WUG member, and is consequently only able to use the services provided by the WUG master (gateway)
2. a WUG master can reject a request to obtain access rights
3. a WUG member is not forced to accept the pairing information received during the Configuration distribution

This applies in particular to devices offering more than just TCS- related services.

### 3.3 OBTAIN ACCESS RIGHTS

Using the Obtain access rights procedure, a device can obtain the rights to use the telephony services provided by another device, part of a WUG.

#### 3.3.1 Procedure description

A device requests access rights by sending an ACCESS RIGHTS REQUEST message and starting timer T401. Upon receipt of the ACCESS RIGHTS REQUEST message, the receiving device accepts the request for access rights by sending an ACCESS RIGHTS ACCEPT.

When the requesting device receives the ACCESS RIGHTS ACCEPT, it shall stop timer T401. Then, the access rights procedure has completed successfully.

If no response has been received before the expiration of timer T401, the requesting device shall consider the request for access rights to be denied.

If, upon receipt of the ACCESS RIGHTS REQUEST message, the receiving device is for some reason unable to accept the access rights, it shall reply with an ACCESS RIGHTS REJECT message. Upon receipt of an ACCESS RIGHTS REJECT message, the requesting device shall stop timer T401 and consider the request for access rights to be denied.

#### 3.3.2 Message flow

The figure below provides the complete view on the messages exchanged during the Obtain access rights procedure.

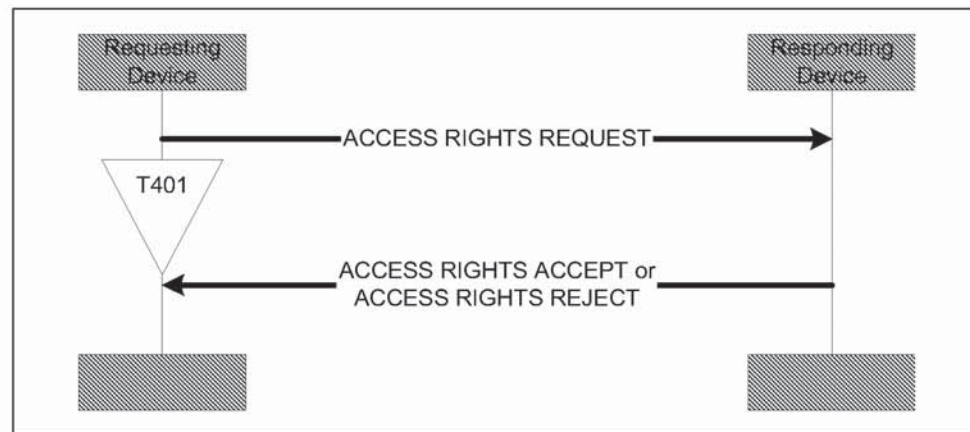


Figure 3.1: Obtain access rights message flow



### 3.4 CONFIGURATION DISTRIBUTION

The units in the WUG need to be informed about changes in the WUG; e.g. when a unit is added or removed. The Configuration distribution procedure is used to exchange this data.

When a WUG configuration change occurs, the WUG master initiates the Configuration distribution procedure on all WUG members. The WUG master keeps track of which WUG members have been informed of WUG configuration changes.

Some WUG members may be out of range and may therefore not be reached. The update of these WUG members will be performed when these members renew contact with the WUG master.

#### 3.4.1 Procedure Description

The WUG master initiates the Configuration distribution procedure by starting timer T403, and transferring the INFO SUGGEST message. The INFO SUGGEST message contains the complete WUG configuration information. Upon receipt of the INFO SUGGEST message, the WUG member shall send an INFO ACCEPT message, to acknowledge the proper receipt of the WUG configuration information.

When the WUG master receives the INFO ACCEPT, the timer T401 is stopped, and the Configuration distribution procedure has completed successfully. On expiry of timer T403, the Configuration distribution procedure is terminated.

#### 3.4.2 Message flow

The figure below provides the complete view on the messages exchanged during the Configuration distribution procedure, as described in the sections above.

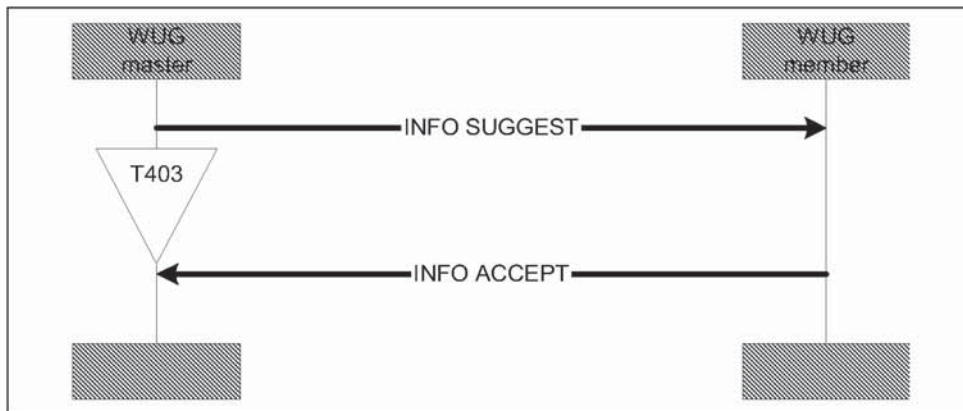


Figure 3.2: Configuration distribution message flow



### 3.5 FAST INTER-MEMBER ACCESS

When two WUG members are both active in the WUG master piconet, a WUG member can use the Fast inter-member access procedure to obtain fast access to another WUG member. With the Fast inter-member access procedure, the originating WUG member obtains clock information from the terminating WUG member and forces the terminating WUG member to go into PAGE\_SCAN for a defined period (T406).

#### 3.5.1 Listen Request

The originating WUG member initiates the Fast inter-member access procedure by starting timer T404 and transferring the LISTEN REQUEST message to the WUG master, indicating the WUG member with which it wishes to establish contact.

If, before expiry of timer T404, the originating WUG member receives no response to the LISTEN REQUEST message, the originating WUG member shall terminate the procedure.

#### 3.5.2 Listen Accept

Upon receipt of the LISTEN REQUEST message, the WUG master checks that the indicated WUG member is part of the WUG. If this is the case, the WUG master initiates the Fast inter-member access towards the terminating WUG member side by starting timer T405 and sending the LISTEN SUGGEST message to the terminating WUG member.

Upon receipt of the LISTEN SUGGEST message, the terminating WUG member confirms the suggested action (internal call) by sending a LISTEN ACCEPT message to the WUG master. This message contains the terminating WUG member's clock offset. After sending the LISTEN ACCEPT, the terminating WUG member shall go to PAGE-SCAN state, for T406 seconds, to enable connection establishment by the originating WUG member.

Upon receipt of the LISTEN ACCEPT message, the WUG master stops timer T405, and informs the originating WUG member of the result of the WUG fast inter-member access by sending a LISTEN ACCEPT message. This message contains the terminating WUG member's clock offset. Upon receipt of the LISTEN ACCEPT message, the originating WUG member stops timer T404, and starts paging the terminating WUG member.

If no response to the LISTEN SUGGEST message is received by the WUG master before the first expiry of timer T405, then the WUG master shall terminate the Fast inter-member access procedure by sending a LISTEN REJECT message to both originating and terminating WUG member using cause #102, *recovery on timer expiry*.

### 3.5.3 Listen Reject by the WUG Master

If the WUG master rejects the Fast inter-member access procedure, it sends a LISTEN REJECT message to the originating WUG member.

Valid cause values are:

#1, *Unallocated (unassigned) number* (when the indicated WUG member is not part of the WUG)

#17, *User busy* (in case terminating WUG member is engaged in an external call)

#20, *Subscriber absent* (upon failure to establish contact with the terminating WUG member), or

any cause value indicated in a LISTEN REJECT message received from/sent to the terminating WUG member.

Upon receipt of the LISTEN REJECT message, the originating WUG member stops timer T404, and terminates the procedure.

### 3.5.4 Listen Reject by the WUG Member

If the terminating WUG member rejects the suggested action received in the LISTEN SUGGEST message, it sends a LISTEN REJECT message to the WUG master. Valid cause value is #17, *User busy* (in case terminating WUG member is engaged in another internal call).

Upon receipt of the LISTEN REJECT, the WUG master stops timer T405, and continues as described in Section 3.5.3 on page 454.

### 3.5.5 Message flow

The figure below provides a view of the messages exchanged during Fast inter-member access, as described in the sections above. A successful Fast inter-member access procedure ends with the terminating WUG member going into page scan, thus allowing the originating WUG member to contact him directly.

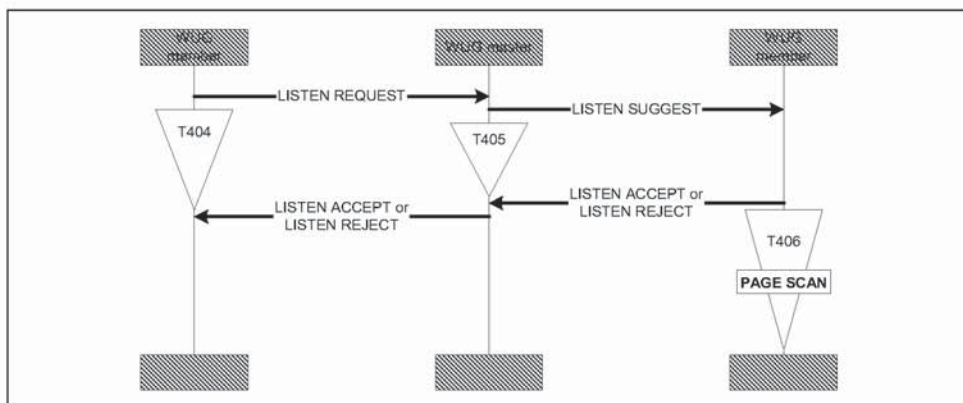


Figure 3.3: Fast inter-member access message flow

## 4 CONNECTIONLESS TCS (CL)

A connectionless TCS message can be used to exchange signalling information without establishing a TCS call. It is thus a connectionless service offered by TCS.

A connectionless TCS message is a CL INFO message (as defined in Section 6.3.1 on page 470).

A connection-oriented L2CAP channel between the Outgoing and Incoming Side shall be available before a CL INFO message can be sent.

*Note: In the case of a connection-oriented channel, it may choose to delay the termination of the channel for a defined period to exchange more CL INFO messages.*

Alternatively, in a multi-point configuration (see Section 1.2 on page 435), a connectionless L2CAP channel may be used and, if so, shall be available before a CL INFO can be sent.

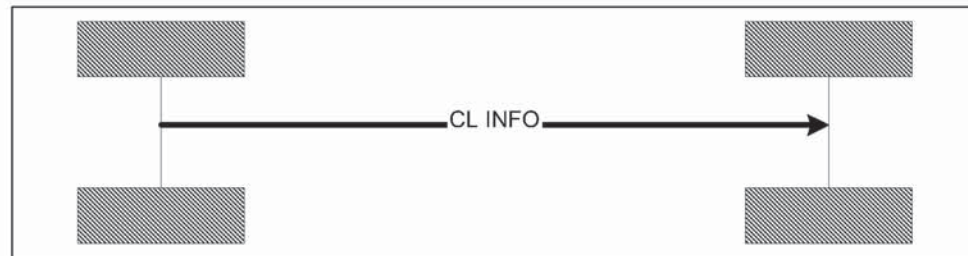


Figure 4.1: Connectionless TCS message flow



## 5 SUPPLEMENTARY SERVICES (SS)

The TCS provides explicit support for only one supplementary service, the Calling Line Identity (see Section 5.1 on page 456).

For supplementary services provided by an external network, using DTMF sequences for the activation/de-activation and interrogation of supplementary services, the DTMF start & stop procedure is supported (see Section 5.2 on page 456). This procedure allows both finite and infinite tone lengths.

Section 5.3 on page 458 specifies how a specific supplementary service, provided by an external network, called register recall is supported.

For other means of supplementary service control, no explicit support is specified. Support may be realized by either using the service call, or use the company specific information element, or a combination.

### 5.1 CALLING LINE IDENTITY

To inform the Incoming Side of the identity of the originator of the call, the Outgoing Side may include the calling party number information element (see Section 7.4.6 on page 481) in the SETUP message transferred as part of the call request.

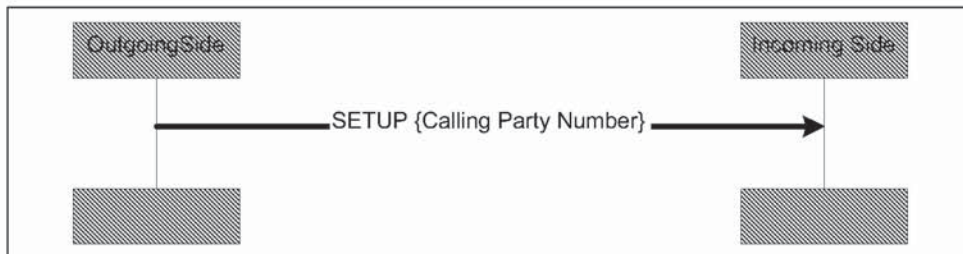


Figure 5.1: Calling line identity message flow

### 5.2 DTMF START & STOP

The DTMF start & stop procedure is supported to provide supplementary service control on PSTN type of networks.

In principle DTMF messages can be initiated by either (Outgoing or Incoming) Side; in practice, however, the Side (gateway) connected to the external PSTN network will be the recipient.

DTMF messages can be transmitted only in the active state of a call. Tone generation shall end when the call is disconnected.

**5.2.1 Start DTMF request**

A user may cause a DTMF tone to be generated; e.g. by depression of a key. The relevant action is interpreted as a requirement for a DTMF digit to be sent in a START DTMF message on an established signalling channel. This message contains the value of the digit to be transmitted (0, 1..9, A, B, C, D, \*, #).

Only a single digit will be transferred in each START DTMF message.

**5.2.2 Start DTMF response**

The side receiving the START DTMF message will reconvert the received digit back into a DTMF tone which is applied toward the remote user, and return a START DTMF ACKNOWLEDGE message to the initiating side. This acknowledgment may be used to generate an indication as a feedback for a successful transmission.

If the receiving side cannot accept the START DTMF message, a START DTMF REJECT message will be sent to the initiating side, using cause #29, *Facility rejected*, indicating that sending DTMF is not supported by the external network.

**5.2.3 Stop DTMF request**

When the user indicates the DTMF sending should cease (e.g. by releasing the key) the initiating side will send a STOP DTMF message to the other side.

**5.2.4 Stop DTMF response**

Upon receiving the STOP DTMF message, the receiving side will stop sending the DTMF tone (if still being sent) and return a STOP DTMF ACKNOWLEDGE message to the initiating side.

**5.2.5 Message flow**

The figure below provides a view of the messages exchanged when a single DTMF tone needs to be generated.

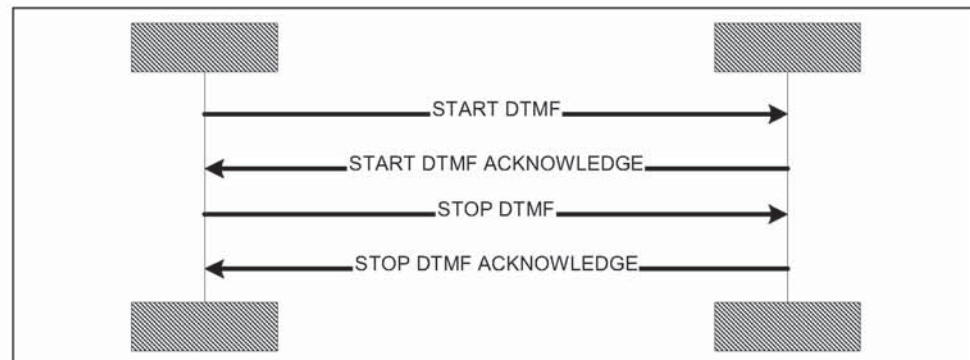


Figure 5.2: DTMF start & stop message flow

### 5.3 REGISTER RECALL

Register recall means to seize a register (with dial tone) to permit input of further digits or other action. In some markets, this is referred to as 'hook flash'. Register recall is supported by sending an INFORMATION message with a keypad facility information element, indicating 'register recall' (value 16H). Further digits are sent using the procedures as indicated in Section 5.2 above.



## 6 MESSAGE FORMATS

---

This section provides an overview of the structure of messages used in this specification, and defines the function and information contents (i.e. semantics) of each message.

Whenever a message is sent according to the procedures of Sections 2, 3 and 4, it shall contain the mandatory information elements, and optionally any combination of the optional information elements specified in this section for that message.

A message shall always be delivered in a single L2CAP packet. The start of a message (the LSB of the first octet) shall be aligned with the start of the L2CAP payload.

Each definition includes:

- a) A brief description of the message direction and use
- b) A table listing the information elements in order of their appearance in the message (same relative order for all message types)
- c) Indications for each information element in the table, specifying –
  - the section of this specification describing the information element
  - whether inclusion is mandatory ('M') or optional ('O')
  - the length (or length range) of the information element, where '\*' denotes an undefined maximum length which may be application dependent.
- d) Further explanatory notes, as necessary

All message formats are denoted in octets.

## 6.1 CALL CONTROL MESSAGE FORMATS

### 6.1.1 ALERTING

This message is sent by the incoming side to indicate that the called user alerting has been initiated.

Message Type: ALERTING

Direction: incoming to outgoing

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Bearer capability	7.4.3	O Note 1)	4(26)
Progress indicator	7.4.13	O	2
SCO Handle	7.4.14	O	2
Destination CID	7.4.11	O	4
Company specific	7.4.9	O	3-*

Table 6.1: ALERTING message content

Note 1: Allowed only in the first message sent by the incoming side.

### 6.1.2 CALL PROCEEDING

This message is sent by the incoming side to indicate that the requested call establishment has been initiated and no more call establishment information will be accepted.

Message Type: CALL PROCEEDING

Direction: incoming to outgoing

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Bearer capability	7.4.3	O Note 1)	4(26)
Progress indicator	7.4.13	O	2
SCO Handle	7.4.14	O	2
Destination CID	7.4.11	O	4
Company specific	7.4.9	O	3-*

Table 6.2: CALL PROCEEDING message content

Note 1: Allowed only in the first message sent by the incoming side.

**6.1.3 CONNECT**

This message is sent by the incoming side to indicate call acceptance by the called user.

Message Type: CONNECT

Direction: incoming to outgoing

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Bearer capability	7.4.3	O <sup>Note 1)</sup>	4(26)
SCO Handle	7.4.14	O	2
Company specific	7.4.9	O	3-*

Table 6.3: CONNECT message content

Note 1: Allowed only in the first message sent by the incoming side.

**6.1.4 CONNECT ACKNOWLEDGE**

This message is sent by the outgoing side to acknowledge the receipt of a CONNECT message.

Message Type: CONNECT ACKNOWLEDGE

Direction: outgoing to incoming

Information Element	Ref.	Type	Length
Message type	7.3	M	1
SCO Handle	7.4.14	O	2
Destination CID	7.4.11	O	4
Company specific	7.4.9	O	3-*

Table 6.4: CONNECT ACKNOWLEDGE message content



**6.1.5 DISCONNECT**

This message is sent by either side as an invitation to terminate the call.

Message Type: DISCONNECT

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Cause	7.4.7	O	2
Progress indicator	7.4.13	O	2
SCO Handle	7.4.14	O	2
Destination CID	7.4.11	O	4
Company specific	7.4.9	O	3-*

Table 6.5: DISCONNECT message content

**6.1.6 INFORMATION**

This message is sent by either side to provide additional information during call establishment (in case of overlap sending).

Message Type: INFORMATION

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Sending complete	7.4.15	O	1
Keypad facility	7.4.12	O	2
Called party number	7.4.5	O	3-*
Audio control	7.4.2	O	3-*
Company specific	7.4.9	O	3-*

Table 6.6: INFORMATION message content

### 6.1.7 PROGRESS

This message is sent by the incoming side to indicate the progress of a call in the event of interworking or by either side in the call with the provision of optional in-band information/patterns.

Message Type: PROGRESS

Direction: incoming to outgoing

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Progress indicator	7.4.13	M	2
SCO Handle	7.4.14	O	2
Destination CID	7.4.11	O	4
Company specific	7.4.9	O	3-*

Table 6.7: PROGRESS message content

### 6.1.8 RELEASE

This message is used to indicate that the device sending the message had disconnected the channel (if any) and intends to release the channel, and that receiving device should release the channel after sending RELEASE COMPLETE.

Message Type: RELEASE

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Cause	7.4.7	O <sup>Note 1)</sup>	2
Company specific	7.4.9	O	3-*

Table 6.8: RELEASE message content

Note 1: Mandatory in the first call clearing message.

### 6.1.9 RELEASE COMPLETE

This message is used to indicate that the device sending the message has released the channel (if any), and that the channel is available for re-use.

Message Type: RELEASE COMPLETE

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Cause	7.4.7	O <sup>Note 1)</sup>	2
Company specific	7.4.9	O	3-*

Table 6.9: RELEASE COMPLETE message content

Note 1: Mandatory in the first call clearing message.

### 6.1.10 SETUP

This message is sent by the outgoing side to initiate call establishment.

Message Type:

Direction:

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Call class	7.4.4	M	2
Sending complete	7.4.15	O	1
Bearer capability	7.4.3	O	4(26)
Signal	7.4.16	O	2
Calling party number	7.4.6	O	3-*
Called party number	7.4.5	O	3-*
Company specific	7.4.9	O	3-*

Table 6.10: SETUP message content



**6.1.11 SETUP ACKNOWLEDGE**

This message is sent by the incoming side to indicate that call establishment has been initiated, but additional information may be required.

Message Type: SETUP ACKNOWLEDGE

Direction: incoming to outgoing

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Bearer capability	7.4.3	O <sup>Note 1)</sup>	4(26)
Progress indicator	7.4.13	O	2
SCO Handle	7.4.14	O	2
Destination CID	7.4.11	O	4
Company specific	7.4.9	O	3-*

Table 6.11: SETUP ACKNOWLEDGE message content

Note 1: Allowed only in the first message sent by the incoming side.

**6.1.12 Start DTMF**

This message contains the digit the other side should reconvert back into a DTMF tone, which is then applied towards the remote user.

Message Type: Start DTMF

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Keypad facility	7.4.12	M	2

Table 6.12: Start DTMF message content

**6.1.13 Start DTMF Acknowledge**

This message is sent to indicate the successful initiation of the action required by the Start DTMF message.

Message Type: Start DTMF Acknowledge

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Keypad facility	7.4.12	M	2

Table 6.13: Start DTMF Acknowledge message content

**6.1.14 Start DTMF Reject**

This message is sent to indicate that the other side cannot accept the Start DTMF message.

Message Type: Start DTMF Reject

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Cause	7.4.7	O	2

Table 6.14: Start DTMF Reject message content

**6.1.15 Stop DTMF**

This message is used to stop the DTMF tone sent towards the remote user.

Message Type: Stop DTMF

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1

Table 6.15: Stop DTMF message content

### 6.1.16 Stop DTMF Acknowledge

This message is sent to indicate that the sending of the DTMF tone has been stopped.

Message Type: Stop DTMF Acknowledge

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Keypad facility	7.4.12	M	2

Table 6.16: Stop DTMF Acknowledge message content

## 6.2 GROUP MANAGEMENT MESSAGE FORMATS

### 6.2.1 ACCESS RIGHTS REQUEST

This message is sent by the initiating side to obtain access rights.

Message Type: ACCESS RIGHTS REQUEST

Direction:

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Company specific	7.4.9	O	3-*

Table 6.17: ACCESS RIGHTS REQUEST message content

### 6.2.2 ACCESS RIGHTS ACCEPT

This message is sent by the responding side to indicate granting of access rights.

Message Type: ACCESS RIGHTS ACCEPT

Direction:

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Company specific	7.4.9	O	3-*

Table 6.18: ACCESS RIGHTS ACCEPT message content



**6.2.3 ACCESS RIGHTS REJECT**

This message is sent by the responding side to indicate denial of access rights.

Message Type: ACCESS RIGHTS REJECT

Direction:

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Company specific	7.4.9	O	3-*

Table 6.19: ACCESS RIGHTS REJECT message content

**6.2.4 INFO SUGGEST**

This message is sent by the WUG master to indicate that a change has occurred in the WUG configuration.

Message Type: INFO SUGGEST

Direction: WUG master to WUG member

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Configuration Data	7.4.10	M	*
Company specific	7.4.9	O	3-*

Table 6.20: INFO SUGGEST message content

**6.2.5 INFO ACCEPT**

This message is sent by the WUG member to indicate the acceptance of the updated WUG configuration.

Message Type: INFO ACCEPT

Direction: WUG member to WUG master

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Company specific	7.4.9	O	3-*

Table 6.21: INFO ACCEPT message content

**6.2.6 LISTEN REQUEST**

This message is sent by a WUG member to indicate to the WUG master the request for a Fast inter-member access to the indicated WUG member.

Message Type: LISTEN REQUEST

Direction: WUG member to WUG master

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Called party number	7.4.6	M	3-*
Company specific	7.4.9	O	3-*

Table 6.22: LISTEN REQUEST message content

**6.2.7 LISTEN SUGGEST**

This message is sent by a WUG master to indicate to the WUG member the request for a Fast inter-member access.

Message Type: LISTEN SUGGEST

Direction: WUG master to WUG member

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Company specific	7.4.9	O	3-*

Table 6.23: LISTEN SUGGEST message content

**6.2.8 LISTEN ACCEPT**

This message is sent to indicate the acceptance of the previous request for a Fast inter-member access.

Message Type: LISTEN ACCEPT

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Clock offset	7.4.8	O	4
Company specific	7.4.9	O	3-*

Table 6.24: LISTEN ACCEPT message content

### 6.2.9 LISTEN REJECT

This message is sent to indicate the rejection of the previous request for a Fast inter-member access.

Message Type: LISTEN REJECT

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Cause	7.4.7	O	2
Company specific	7.4.9	O	3-*

Table 6.25: LISTEN REJECT message content

## 6.3 TCS CONNECTIONLESS MESSAGE FORMATS

### 6.3.1 CL INFO

This message is sent by either side to provide additional information in a connectionless manner.

Message Type: CL INFO

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Audio control	7.4.2	O	3-*
Company specific	7.4.9	O	3-*

Table 6.26: CL INFO message content



## 7 MESSAGE CODING

The figures and text in this section describe message contents. Within each octet, the bit designated 'bit 1' is transmitted first, followed by bit 2, 3, 4, etc. Similarly, the octet shown at the top of the figure is sent first.

Whenever a message is sent, according to the procedures of Sections 2, 3 and 4, it shall be coded as specified in this section.

### 7.1 OVERVIEW

The coding rules follow ITU-T Recommendation Q.931, but is tailored to the specific needs of TCS.

Every message consists of:

- a) Protocol discriminator
- b) Message type, and
- c) Other information elements, as required

The Protocol discriminator and Message type is part of every TCS message, while the other information elements are specific to each message type.

8	7	6	5	4	3	2	1	
Protocol discriminator				Message type				octet 1
Other information elements as required								octet 2

Table 7.1: General message format

A particular information element shall be present only once in a given message.

The term 'default' implies that the value defined shall be used in the absence of any assignment or negotiation of alternative values.

For notation purposes – when a field extends over more than one octet, the order of bit values progressively decreases as the octet number increases. The least significant bit of the field is represented by the lowest numbered bit of the highest-numbered octet of the field. In general, bit 1 of each octet contains the least significant bit of a field.

### 7.2 PROTOCOL DISCRIMINATOR

The purpose of the protocol discriminator is to distinguish the TCS messages into different functional groups. The protocol discriminator is the first part of every message.

The protocol discriminator is coded according to Figure 7.1 and Table 7.2.

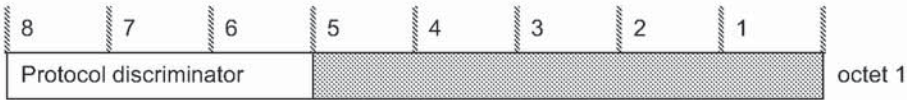


Figure 7.1: Protocol discriminator

Bits			
8	7	6	
0	0	0	Bluetooth TCS Call Control
0	0	1	Bluetooth TCS Group management
0	1	0	Bluetooth TCS Connectionless
			All other values reserved

Table 7.2: Protocol discriminator

### 7.3 MESSAGE TYPE

The purpose of the message type is to identify the function of the message being sent.

The Message type is the first part of every message and it is coded as shown in Figure 7.2 and Table 7.3.



Figure 7.2: Message type

Bits					
5	4	3	2	1	
					<i>Call Control messages</i>
					<i>Call Establishment</i>
0	0	0	0	0	ALERTING

Table 7.3: Message type

Bits					
5	4	3	2	1	
0	0	0	0	1	CALL PROCEEDING
0	0	0	1	0	CONNECT
0	0	0	1	1	CONNECT ACKNOWLEDGE
0	0	1	0	0	PROGRESS
0	0	1	0	1	SETUP
0	0	1	1	0	SETUP ACKNOWLEDGE
					<i>Call clearing</i>
0	0	1	1	1	DISCONNECT
0	1	0	0	0	RELEASE
0	1	0	0	1	RELEASE COMPLETE
					<i>Miscellaneous</i>
0	1	0	1	0	INFORMATION
1	0	0	0	0	START DTMF
1	0	0	0	1	START DTMF ACKNOWLEDGE
1	0	0	1	0	START DTMF REJECT
1	0	0	1	1	STOP DTMF
1	0	1	0	0	STOP DTMF ACKNOWLEDGE
					<b>Group management messages</b>
0	0	0	0	0	INFO SUGGEST
0	0	0	0	1	INFO ACCEPT
0	0	0	1	0	LISTEN REQUEST
0	0	0	1	1	LISTEN ACCEPT
0	0	1	0	0	LISTEN SUGGEST
0	0	1	0	1	LISTEN REJECT
0	0	1	1	0	ACCESS RIGHTS REQUEST
0	0	1	1	1	ACCESS RIGHTS ACCEPT
0	1	0	0	0	ACCESS RIGHTS REJECT
					<b>Connectionless messages</b>
0	0	0	0	0	CL INFO

Table 7.3: Message type



## 7.4 OTHER INFORMATION ELEMENTS

### 7.4.1 Coding rules

The coding of other information elements follows the coding rules described below.

Three categories of information elements are defined:

- a) single octet information elements (see Figure 7.3 on page 474)
- b) double octet information element (see Figure 7.4 on page 474)
- c) variable length information elements (see Figure 7.5 on page 474).

Table 7.4 on page 474 summarizes the coding of the information element identified bits for those information elements used in this specification.

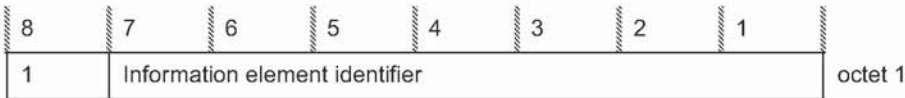


Figure 7.3: Single octet information element format

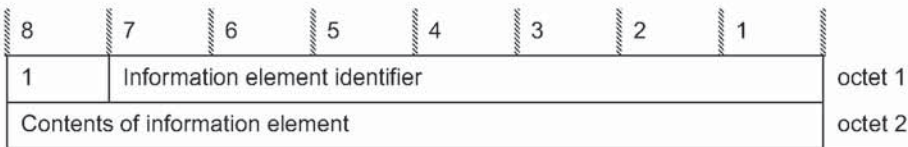


Figure 7.4: Double octet information element format

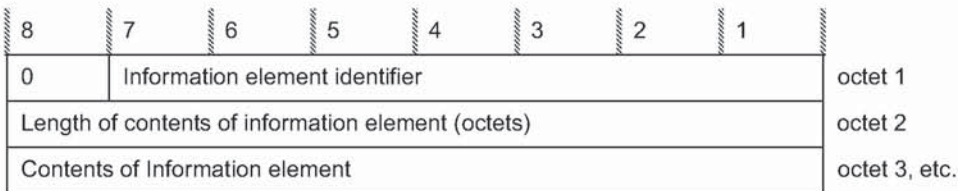


Figure 7.5: Variable length information element format

Coding								Ref.	Max Length (octets)
8	7	6	5	4	3	2	1		
1								7.4.15	1
	0	1	0	0	0	0	1		
1									

Table 7.4: Information element identifier coding

8	Coding							Ref.	Max Length (octets)	
	7	6	5	4	3	2	1			
	1	0	0	0	0	0	0	Call class	7.4.4	2
	1	0	0	0	0	0	1	Cause	7.4.7	2
	1	0	0	0	0	1	0	Progress indicator	7.4.13	2
	1	0	0	0	0	1	1	Signal	7.4.16	2
	1	0	0	0	1	0	0	Keypad facility	7.4.12	2
	1	0	0	0	1	0	1	SCO handle	7.4.14	2
0								<i>Variable length information elements</i>		
	0	0	0	0	0	0	0	Clock offset	7.4.8	4
	0	0	0	0	0	0	1	Configuration data	7.4.2	*
	0	0	0	0	0	1	0	Bearer capability	7.4.3	4(26)
	0	0	0	0	0	1	1	Destination CID	7.4.11	4
	0	0	0	0	1	0	0	Calling party number	7.4.6	*
	0	0	0	0	1	0	1	Called party number	7.4.5	*
	0	0	0	0	1	1	0	Audio control	7.4.2	*
	0	0	0	0	1	1	1	Company specific	7.4.9	*

Table 7.4: Information element identifier coding

The descriptions of the information elements below are organized in alphabetical order. However, there is a particular order of appearance for each information element in a message. The code values of the information element identifier for the variable length formats are assigned in ascending numerical order, according to the actual order of appearance of each information element in a message. This allows the receiving devices to detect the presence or absence of a particular information element without scanning through an entire message.

Where the description of information elements in this specification contains spare bits, these bits are indicated as being set to '0'. In order to allow compatibility with future implementation, messages should not be rejected simply because a spare bit is set to '1'.

The second octet of a variable length information element indicates the total length of the contents of that information element regardless of the coding of the first octet (i.e. the length is calculated starting from octet 3). It is the binary coding of the number of octets of the contents, with bit 1 as the least significant bit (2°).

An optional variable-length information element may be present, but empty (zero length). The receiver should interpret this as if that information element



was absent. Similarly, an absent information element should be interpreted by the receiver as if that information element was empty.

**7.4.2 Audio control**

The purpose of the Audio control information elements is to indicate information relating to the control of audio.

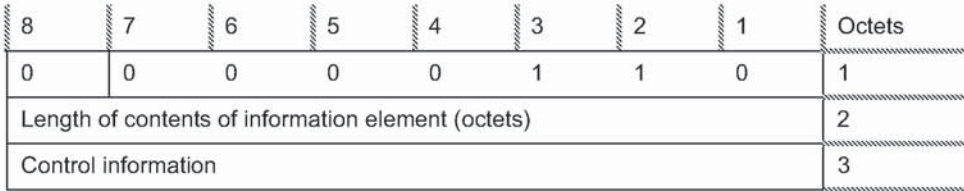


Figure 7.6:

Control information (octet 3)								
Bits								
	7	6	5	4	3	2	1	
	0	0	0	0	0	0	0	Volume increase
	0	0	0	0	0	0	1	Volume decrease
	0	0	0	0	0	1	0	Microphone gain increase
	0	0	0	0	0	1	1	Microphone gain decrease
	0	X	X	X	X	X	X	Reserved for Bluetooth standardization
	1	X	X	X	X	X	X	Company specific

Table 7.5: Audio Control information element coding

**7.4.3 Bearer capability**

The purpose of the Bearer capability information elements is to indicate a requested or available bearer service.

If this information element is absent, the default Bearer capability is Link type Synchronous Connection-Oriented with packet type HV3, using CVSD coding for the User information layer 1.

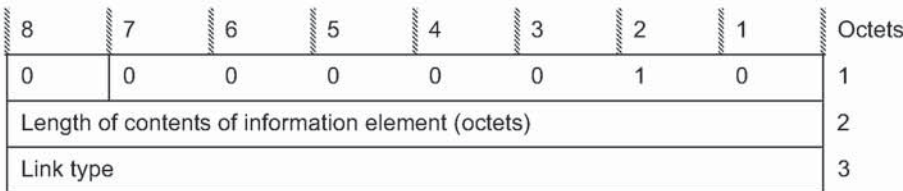


Figure 7.7:

Link type element coding = 00000000 (SCO)



User information layer 1	Packet type	4
--------------------------	-------------	---

Figure 7.8:

Link type element coding = 00000001 (ACL)

Flags	4	
Service type	5	
Token Rate	6	
	7	
	8	
	9	
Token Bucket Size (bytes)	10	
	11	
	12	
	13	
Peak Bandwidth (bytes/second)	14	
	15	
	16	
	17	
Latency (microseconds)	18	
	19	
	20	
	21	
Delay Variation (microseconds)	22	
	23	
	24	
	25	
User information layer 3	User information layer 2	26

Figure 7.9:

Note: the Quality of Service is repeated at TCS level, as only TCS has the knowledge of end-to-end Quality of Service requirements.

<i>Link type (octet 3)</i>	
Bits	
8   7   6   5   4   3   2   1	
0 0 0 0 0 0 0 0	Synchronous Connection-Oriented
0 0 0 0 0 0 0 1	Asynchronous Connection-Less
0 0 0 0 0 0 1 0	None
All other values are reserved	
<i>Octet 4 coding (Link type element coding = 000000000)</i>	
<i>Packet type (octet 4)</i>	
Bits	
5   4   3   2   1	
0 0 1 0 1	HV1
0 0 1 1 0	HV2
0 0 1 1 1	HV3
0 1 0 0 0	DV
All other values are reserved	
<i>User information layer 1 (octet 4)</i>	
Bits	
8   7   6	
0 0 1	CVSD
0 1 0	PCM A-law
0 1 1	PCM $\mu$ -law
All other values reserved	
<i>Octets 4-26 coding (Link type element coding = 000000001)</i>	
The details of the coding Octets 4-25 can be found in L2CAP, see L2CAP, Section 6 on page 289	
<i>User information layer 2 (octet 26)</i>	
Bits	
4   3   2   1	
0 0 0 0	RFCOMM over L2CAP
All other values are reserved	
<i>User information layer 3 (octet 26)</i>	
Bits	
8   7   6   5	
0 0 0 0	Not specified
0 0 0 1	PPP
0 0 1 0	IP
All other values reserved	
<i>Octet 4 coding (Link type element coding = 000000010)</i>	
Octet 4 is absent	

Table 7.6: Bearer capability information element coding

**7.4.4 Call class**

The purpose of the Call class is to indicate the basic aspects of the service requested. This element allows the user to indicate the use of default attributes, thereby reducing the length of the set-up message.

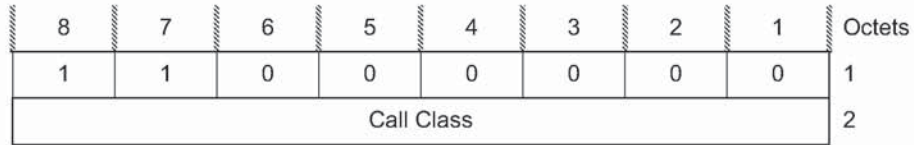


Figure 7.10:

Call class (octet 2)								
	Bits							
8	7	6	5	4	3	2	1	
0	0	0	0	0	0	0	0	External call
0	0	0	0	0	0	0	1	Intercom call
0	0	0	0	0	0	1	0	Service call
0	0	0	0	0	0	1	1	Emergency call
All other values reserved								

Table 7.7: Call class information element coding

**Note**

- An external call is a call to/from an external network; e.g. the PSTN.
- An intercom call is a call between Bluetooth devices.
- A service call is a call for configuration purposes.
- An emergency call is an external call using a dedicated emergency call number, using specific properties.



**7.4.5 Called party number**

The purpose of the Called party number information element is to identify the called party of a call.

8	7	6	5	4	3	2	1	Octets
0	0	0	0	0	1	0	1	1
Length of contents of information element (octets)								2
0	Type of number			Numbering plan identification				3
0	Number digits (IA5 characters) (Note)							4 etc.

Note – The number digits appear in multiple octet 4's in the same order in which they would be entered, that is, the number digit which would be entered first is located in the first octet 4.

Figure 7.11:

<i>Type of number (octet 3)</i>				
Bits				
7	6	5		
0	0	0	Unknown	
0	0	1	International number	
0	1	0	National number	
0	1	1	Network specific number	
1	0	0	Subscriber number	
1	1	0	Abbreviated number	
1	1	1	Reserved for extension	
All other values are reserved				
<i>Numbering plan identification (octet 3)</i>				
Bits				
4	3	2	1	
0	0	0	0	Unknown
0	0	0	1	ISDN/telephony numbering plan E.164
0	0	1	1	Data numbering plan Rec. X.121
0	1	0	0	Reserved
1	0	0	0	National standard numbering plan
1	0	0	1	Private numbering plan
All other values are reserved				

Table 7.8: Called party information element coding