Part F:2

IrDA INTEROPERABILITY

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

1 Introduction ................................................................. 412
   1.1 OBEX and Bluetooth Architecture.............................. 413
   1.2 Bluetooth OBEX Related Specifications ....................... 413
   1.3 Other IrOBEX Implementations.................................... 414
2 OBEX Object and Protocol ............................................... 415
   2.1 Object......................................................................... 415
   2.2 Session Protocol ...................................................... 415
      2.2.1 Connect Operation ............................................ 416
      2.2.2 Disconnect Operation ......................................... 417
      2.2.3 Put Operation ................................................... 417
      2.2.4 Get Operation.................................................... 418
      2.2.5 Other Operations................................................. 418
3 OBEX over RFCOMM ....................................................... 419
   3.1 OBEX Server Start-up on RFCOMM............................... 419
   3.2 Receiving OBEX Packets from Serial Port..................... 419
   3.3 Connection Establishment .......................................... 420
   3.4 Disconnection .......................................................... 420
   3.5 Pushing and Pulling OBEX Packets over RFCOMM .......... 420
4 OBEX over TCP/IP ........................................................... 421
   4.1 OBEX Server Start-up on TCP/IP.................................. 421
   4.2 Connection Establishment .......................................... 421
   4.3 Disconnection .......................................................... 422
   4.4 Pushing and Pulling OBEX Packets over TCP................ 422
5 Bluetooth Application Profiles using OBEX ......................... 423
   5.1 Synchronization ....................................................... 423
   5.2 File Transfer ........................................................... 423
   5.3 Object Push ............................................................. 424
6 References ....................................................................... 425
7 List of Acronyms and Abbreviations ................................. 426
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. Now, this protocol is also utilized by the Bluetooth technology enabling the possibility 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 related to the lower layer protocols and so, IrOBEX will 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 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 423). The protocols can also communicate with the service discovery DB even the figure does not show it.

![Figure 1.1: Part of Bluetooth Protocol Hierarchy](image)

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 calendaring 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.
   • 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

   • 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.

   • 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.

   • 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:

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Bytes 1 and 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Bytes 5 and 6</th>
<th>Byte 7 to n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80 (opcode)</td>
<td>Connect</td>
<td>OBEX version</td>
<td>Flags</td>
<td>Maximum</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>request packet length</td>
<td>number</td>
<td></td>
<td>OBEX packet length</td>
<td>headers</td>
</tr>
</tbody>
</table>

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., non-successful one back to the client indicates a failure to make a connection. The response format is:

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Bytes 1 and 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Bytes 5 and 6</th>
<th>Byte 7 to n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response</td>
<td>Connect</td>
<td>OBEX version</td>
<td>Flags</td>
<td>Maximum</td>
<td>Optional</td>
</tr>
<tr>
<td>code</td>
<td>response packet length</td>
<td>number</td>
<td></td>
<td>OBEX packet length</td>
<td>headers</td>
</tr>
</tbody>
</table>

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.

When a connection is established, it is kept alive since the connection is disconnected by requests/responses or by failures. Thus, 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:

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Bytes 1 and 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x81</td>
<td>Packet length</td>
<td>Optional headers</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Bytes 1 and 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA0</td>
<td>Response packet length</td>
<td>Optional response headers</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Bytes 1 and 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x02 (0x82 when Final bit set)</td>
<td>Packet length</td>
<td>Sequence of headers</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Bytes 1 and 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response code</td>
<td>Response packet length</td>
<td>Optional response headers</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Bytes 1 and 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x03 (0x83 when Final bit set)</td>
<td>Packet length</td>
<td>Sequence of headers starting with Name</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Bytes 1 and 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response code</td>
<td>Response packet length</td>
<td>Optional response headers</td>
</tr>
</tbody>
</table>

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 in the middle of a request/response sequence. Thus, the whole OBEX object does not have to be received before sending Abort-request. In addition to these operations, the IrOBEX specification gives a possibility for user-defined operations but their use may not be necessary 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, or a server, or both.

2. All servers running simultaneously on a device must use separate RFCOMM server channels.

3. Application (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., running, there are few prerequisites that must be completed before the server can enter the listening mode.

1. The server must open an RFCOMM server channel.

2. The server registers its capabilities into the service discovery database.

After this, other hosts are able to find the server if needed and the server is listening for getting requests from clients.

3.2 RECEIVING OBEX PACKETS FROM SERIAL PORT

As it was discussed earlier, one object can be exchanged over one or more Put-requests or Get-responses and thus, the object is received in packet(s). However, if OBEX is directly running over the serial port, it does not receive packets from RFCOMM but a byte stream is received by OBEX from a serial port, which is 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 not any ‘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 be a problem over the emulated serial port of RFCOMM, which provides the reliable transport over Bluetooth.

3.3 CONNECTION ESTABLISHMENT

A client starts 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 to the server that the connection can be established.

2. The client uses the discovered RFCOMM channel for establishing 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 got the response, the next operation is to close the RFCOMM channel designated to the OBEX client.

3.5 PUSHING AND PULLING OBEX PACKETS OVER RFCOMM

Pushing data in OBEX packets over RFCOMM happens 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 servers, the TCP port number is 650 assigned by IANA or the port number can be a value above of 1023, if the assigned number is not desirable. 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 clients must use port numbers (on the client side), which are not within the 0-1023 range.
4. Application (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 for receiving requests. However, when the server is ready i.e., running, there are few prerequisites that must be completed before the server can enter 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 is listening for 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 to the server that 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 for establishing 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 got 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, and 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 synchronization functionality of the level 4 specified in the IrMC specification.

The actual logic of the synchronization engines, which processes the synchronization algorithm at the client device, is implementation specific. Thus, it is left for the implement SW vendors and it is not considered in the Bluetooth specifications.

The synchronization is not limited for one type of applications. 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 – Provides a means for a user to manage calendar items, which can also be used for the to-do or task lists.
3. Messaging – Provides a means for a user to manage messages (e.g., emails).
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, respectively. The File Transfer service also offers the functionality for browsing the folder of the remote Bluetooth device.

5.3 OBJECT PUSH

The Object Push profile is the special case of the file transfer profile for beam- ing objects and optionally pulling the default objects. This service is not limited for exchanging the business card but it offers this capability as a minimum.

6 REFERENCES


## Listing of Acronyms and Abbreviations

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