BLUETOOTH® DOC	Date / Year-Month-Day 2010-08-26	Approved Adopted	Revision V20r00	Document No IrDA_SPEC
Prepared By OBEX WG	E-mail Address OBEX-feedback@bluetooth.org			N.B.

IrDA INTEROPERABILITY

Abstract:

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.

Disclaimer and Copyright Notice

Revision History

Revision	**Date**	**Comments**
Version 1.1		Version released with Bluetooth V1.1
D11r00	19 April 2005	Changed document numbering to conform with Bluetooth Document Naming Procedure V10r00.
D12r00	15 August 2005	Updated to indicate v1.2 or Later
D12r01	14 September 2004	Editorial updates
D12r02	31 October 2005	Editorial updates
D12r03	15 November 2005	Editorial updates
D12r04	30 November 2005	Editorial updates
D12r05	21 March 2006	Input reviewer's comments
D12r06	4 June 2010	Draft spec from TechEd, include LLO, TH review comments, remove red fonts
D20r01	26 June 2010	Updates from BARB review; correct version number for publication
D20r02	30 June 2010	Include tweaks to language suggested by TB
V20r00	26 August 2010	Adopted by the Bluetooth SIG Board of Directors

Contributors

Name	**Company**
Ole Heftholm-Jensen	CSR
Dave Suvak	Extended Systems
Chatschik Bisdikian	IBM Corporation
Brent Miller	IBM Corporation
Apratim Purakayastha	IBM Corporation
Aron Walker	IBM Corporation
Jon Inouye	Intel Corporation
Stephane Bouet (Section Owner)(PR15)	Nokia Mobile Phones(PR15)
Michael Camp	Nokia Mobile Phones
Riku Mettälä	Nokia Mobile Phones
Peter Ollikainen	Nokia Mobile Phones
James Scales	Nokia Mobile Phones
Tim Howes	Nokia
David Kammer	Palm(PR15)
Steve Rybicki	PumaTech(PR12)
John Stossel	PumaTech(PR12)
Greg Burns	Qualcomm
Mandar Gokhale	Qualcomm
Len Ott	Socket Mobile
Kevin Hendrix	Sybase
Christian Andersson	Telefonaktiebolaget LM Ericsson
Johannes Elg	Telefonaktiebolaget LM Ericsson
Patrik Olsson	Telefonaktiebolaget LM Ericsson
Johan Sörensen	Telefonaktiebolaget LM Ericsson

Contents

1 Introduction

The goal of this document is to enable the development of application programs that function 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 L2CAP and TCP/IP [2]. 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 L2CAP and TCP/IP. Note that previous versions of this specification provided means of running OBEX over RFCOMM. This version of the specification does not provide support for OBEX over RFCOMM. This allows new profile specifications to not use RFCOMM. Profiles that require the use of RFCOMM will reference versions of the IrDA Interoperability specification prior to v2.0. Note 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 the application profiles using OBEX and Bluetooth need 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 1.1depicts part of the hierarchy of the Bluetooth architecture and shows the placement of the OBEX protocol and the application profiles using it. The protocols can also communicate with the Service Discovery database even though the figure does not show it.

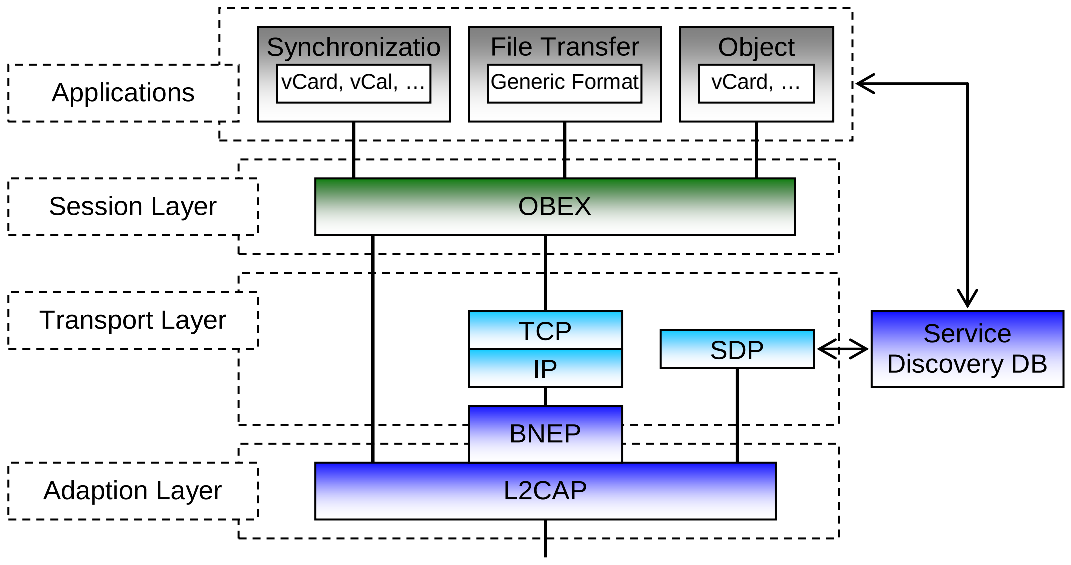

Figure 1.1: Part of Bluetooth Protocol Hierarchy

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 [3], vCalendar [4], vMessage [5], and vNotes [5] formats. The vCard, vCalendar, vMessage, and vNotes describe the formats for electronic business cards, electronic calendars and scheduling, electronic messages and mail, and electronic notes, respectively.

1.2 Bluetooth OBEX-Related Specifications

Bluetooth Specification includes a number of 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 L2CAP and TCP
   - Defines the application profiles using OBEX over Bluetooth
2. Bluetooth Generic Object Exchange Profile Specification [6]
   - Generic interoperability specification for the application profiles using OBEX
   - Defines the interoperability requirements of the lower protocol layers (e.g. Baseband, LMP, and L2CAP) for the application profiles
3. Bluetooth Application Profiles

- Defines the interoperability requirements for the applications
- Does not define the requirements for the Baseband, LMP, or L2CAP.

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 include the following:

- Count
- Name
- Type
- Length
- Time
- Description
- Target
- HTTP
- Body
- End of Body
- Who
- Connection ID
- Application Parameters
- Authenticate Challenge
- Authenticate Response
- Creator ID
- WAN UUID
- Object Class
- User-Defined

- Session Parameters
- Session Sequence Number
- Action Identifier
- DestName
- Permissions
- Single Response Mode
- Single Response Mode Parameters

These are explained in detail by Section 2.2 in the IrOBEX specification.

2.2 Session Protocol

The OBEX operations are formed by **request-response** 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. The exception to this rule occurs when performing Put or Get operations with OBEX Single Response Mode (SRM) enabled. In this case the standard request-response sequence is bypassed for the express purpose of increasing OBEX throughput, by eliminating unnecessary OBEX request/response packets. This mode is explained in detail in Section 3.3 of the IrOBEX specification [1]. Each request packet consists of a one-byte opcode (see Section 3.4 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 3.4.1 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. 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 [1]. 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 3.4.2 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, the server shall 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 3.4.3 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 the size of the transferred object, and the size of the OBEX packets. The server shall send a response for every **Put**-request packet, unless OBEX Single Response Mode (SRM) is enabled. If

SRM is enabled, only one response packet is expected for the entire Put operation, namely a single response after the last request has been issued, unless the server chooses to reject the operation early with an unsuccessful response code (excludes CONTINUE and SUCCESS). The exception to this case occurs if the OBEX Single Response Mode Parameters header is used during the PUT operation to trigger additional response packets. Otherwise, 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 3.4.4 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. The client shall send a request packet for every response packet, unless OBEX Single Response Mode (SRM) is used. If SRM is used, multiple unacknowledged request packets can be sent, but only one request including the Final Bit is expected for the entire Get operation, which is typically the first request to start the operation. The exception to this case occurs if the OBEX Single Response Mode Parameters header is used during the GET operation to trigger additional request packets.

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 **SetPath**, **Action**, **Session**, and **Abort** operations. These are defined in the Sections 3.4.5-8 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 L2CAP

This section specifies how OBEX is mapped over L2CAP, which is a multiplexing and transport protocol defined in the Bluetooth Core Specification. Bluetooth devices supporting the OBEX protocol operate as clients or servers or both, and shall satisfy the following requirements.

1. All OBEX clients and servers running simultaneously on a device shall each use separate L2CAP channels. Therefore, multiplexing at the OBEX level (as defined in IrOBEX 3.7.1) shall not be used.
2. Application profiles using OBEX shall be able to register service records into the service discovery database. The records for different application profiles are specified in the respective profile specifications.

3.1 OBEX Server Start-up on L2CAP

A server shall perform the following before accepting connection requests from OBEX clients:

1. The server shall bind to an L2CAP PSM. Profiles shall specify either a fixed PSM (with a value determined by the Bluetooth SIG), or a dynamic PSM. The L2CAP section of the Core specification [8] provides rules regarding PSM values.
2. The server shall register its capabilities into the service discovery database using records defined by Application profiles. The L2CAP PSM shall be registered in the records as described by the application profiles.

After this, the server listens for an L2CAP connection and then OBEX requests from clients.

3.2 Receiving OBEX Packets from L2CAP

As discussed earlier, an object can be exchanged over one or more OBEX**_Put** requests or OBEX**_Get**-responses (i.e. the object is received in one or more packets).

OBEX receives packets from L2CAP. There shall be one OBEX packet contained in each L2CAP SDU. After the OBEX opcode or response code, the length of the OBEX packet is in the next two bytes. This should be compared with the length of the L2CAP SDU. All packets that are not recognized, or have invalid lengths, shall be discarded and the L2CAP channel may be disconnected if invalid SDUs are received.

3.3 Connection Establishment

The client shall perform the following to create an OBEX connection:

1. By using SDP (as defined in [1]), the client shall discover information (e.g. L2CAP PSM) associated with the server which is necessary for the OBEX connection to be established.

2. The client uses the L2CAP PSM to establish the L2CAP channel to the server.
3. The client sends the OBEX_Connect-request over the L2CAP channel to the server. The OBEX connection is established if the client receives a response from the server with the result code of “Success”; otherwise the OBEX connection is not established.

3.4 Disconnection

To disconnect an OBEX connection the client may send the OBEX_Disconnect-request (see section 2.2.2). If the client sends the OBEX_Disconnect-request and receives the response to that request, the client shall close the L2CAP channel used for the OBEX connection.

3.5 Pushing and Pulling OBEX Packets over L2CAP

Data is pushed in OBEX packets over L2CAP by using **Put**-requests (see section 2.2.3). In accordance with [1], if OBEX Single Response Mode is not used, after each request, a **Put**-response is required before the next **Put**-request with the data can be pushed from the OBEX layer.

Pulling data from a remote host is done by sending a **Get**-request (see section 2.2.4). The data arrives in OBEX **Get**-response packets. If OBEX Single Response Mode is not used, further **Get**-requests shall be sent in accordance with the IrOBEX specification [1], to pull more data after each **Get**-response.

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 [7]).
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 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 [1], 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 OBEX connection is established if the client receives a response from the server with the result code of “Success”; otherwise the OBEX connection is not established.

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 [1].

5 References

[1] IrOBEX v1.5 (http://www.irda.org)
[2] Internet Engineering Task Force, IETF Directory List of RFCs (http://www.ietf.org/rfc/)
[3] The Internet Mail Consortium, vCard - The Electronic Business Card Exchange Format, Version 2.1, September 1996.
[4] The Internet Mail Consortium, vCalendar - The Electronic Calendaring and Scheduling Exchange Format, Version 1.0, September 1996.
[5] Infrared Data Association, IrMC (Ir Mobile Communications) Specification, Version 1.1, February 1999.
[6] Bluetooth Generic Object Exchange Profile
[7] Internet Assigned Numbers Authority, IANA Protocol/Number Assignments Directory (http://www.iana.org/).
[8] Bluetooth Core Specification v3.0 or later

6 List of Acronyms and Abbreviations

Abbreviation or Acronym	Meaning
GOEP	Generic Object Exchange Profile
IANA	Internet Assigned Numbers Authority
IrDA	Infrared Data Association
IrMC	Ir Mobile Communications
L2CAP	Logical Link Control and Adaptation Protocol
LMP	Link Manager Protocol
LSB	Least Significant Byte
MSB	Most Significant Byte
OBEX	Object exchange protocol
PDU	Protocol Data Unit
PSM	Protocol Service Multiplexer
RFCOMM	Serial cable emulation protocol based on ETSI TS 07.10
SD	Service Discovery
SDDB	Service Discovery Database
SDP	Service Discovery Protocol
SDU	Service Data Unit
SRM	Single Response Mode
TCP/IP	Transport Control Protocol/Internet Protocol
Tiny TP	Tiny Transport Protocol