SyncML OBEX Binding, version 1.1

Abstract

This document describes how to use SyncML over OBEX. The document uses the primitives and methods defined in the OBEX specification V1.2 as defined in [1].

The document assumes a scenario consisting of a SyncML client (e.g., a mobile phone) and a server holding data. Within local area networks, the server could be a PIM application running on a PC.

SyncML Initiative

The following companies are Sponsors of the SyncML Initiative:

Ericsson
IBM
Lotus
Matsushita Communications Industrial Co., Ltd.
Motorola
Nokia
Openwave
Starfish Software
Symbian

Revision History

Revision	Date	Comments
1.0a	2000-08-29	SyncML Header Type Mime included. Minor editorial changes.
1.0b	2000-11-10	Minor changes to description text.
1.0	2000-12-07	The candidate version for the final release. Bluetooth UUID details included.
1.0.1a	2001-05-23	Errata rolled in.
1.0.1	2001-05-30	Cleaned up header
1.1	2002-02-15	Updated copyrights & version, added MIME types for SyncML DM.

Copyright Notice

Table of Contents

1 Introduction

This document describes how to use the SyncML over OBEX. The document uses the primitives and methods defined in the OBEX specification V1.2 [1].

The document assumes a scenario consisting of a SyncML client (e.g. a mobile phone) and a server holding data. The OBEX transport was originally used over short-range links like infrared. With short-range links, the SyncML server could be a local PC. With wide area networks, the SyncML server could be a remote WEB server.

2 Formatting Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [2].

The letter M is used in the tables to indicate MUST, and O to indicate OPTIONAL.

3 OBEX Introduction

OBEX [1] is a protocol for exchanging objects. It was initially designed for infrared, but it has been adopted by Bluetooth, and is also used over RS232, USB and WAP.

OBEX is a session-oriented protocol, which allows multiple request/response exchanges in one session. An OBEX session is initiated by an OBEX CONNECT request, and is established when the other device returns a success response. The connection is terminated by sending a DISCONNECT request.

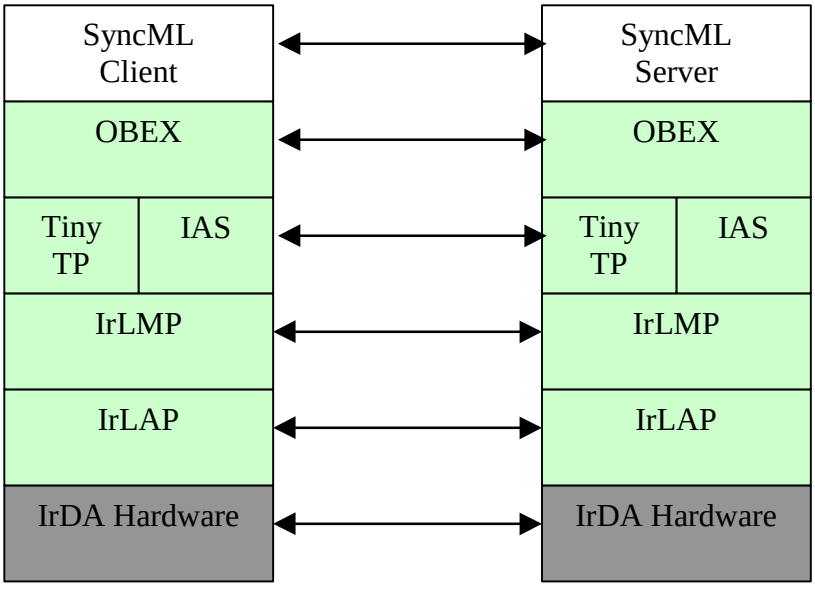
In this specification, the SyncML client can work either as an OBEX client or as an OBEX server at the OBEX protocol layer. In consequence, the SyncML server can work either as an OBEX client or as an OBEX server. The OBEX role depends on the fact which one, the SyncML client or the SyncML server, initiates sync. Thus the SyncML Client is not necessarily the OBEX Client.

When a session has been established, the data is transferred using the PUT request. The remote device acknowledges the data, by sending a response with a status code.

SyncML requires that an OBEX connection is established. Connectionless OBEX cannot be used with SyncML.

3.1 OBEX Over IrDA

The diagram below demonstrates the position of OBEX within the IrDA stack.



IrLAP is the link level protocol.
IrLMP is a multiplexing layer.
Tiny TP provides flow control.
IAS is the Information Access Service.
OBEX includes both a session level protocol and an application framework.

3.2 OBEX Over Bluetooth

The Bluetooth section is specified so that the SyncML client MUST be able to function as either an OBEX client, or an OBEX server, or both. The SyncML server MUST be able to function as both the OBEX server and client.

The figure below shows the protocols when SyncML and OBEX are run over the Bluetooth protocol stack.

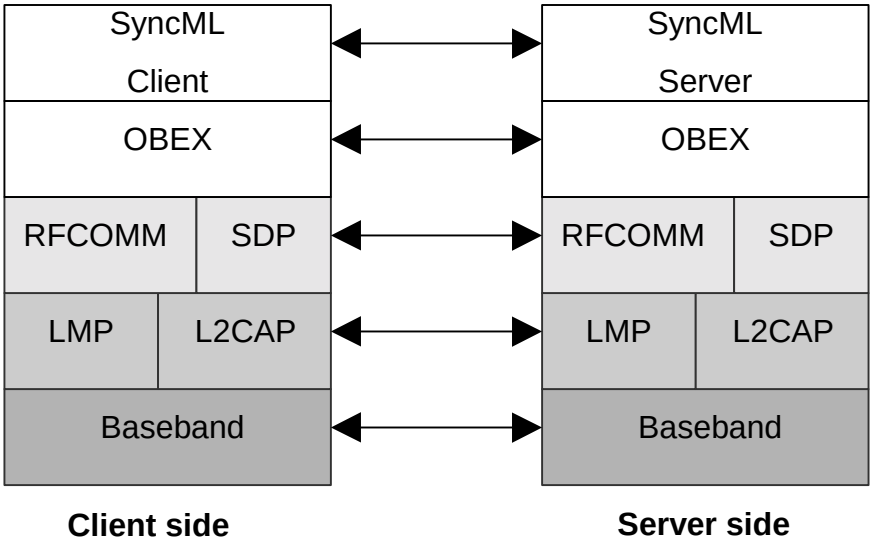


Figure 1 OBEX over Bluetooth

The Baseband, LMP, and L2CAP are the OSI layer 1 and 2 Bluetooth protocols. RFCOMM is the Bluetooth adaptation of GSM TS 07.10. SDP is the Bluetooth Service Discovery Protocol [3].

The SyncML Client layer shown in Figure 1 is the entity providing the sync client agent functionality. The SyncML Server is the SW providing the sync engine functionality.

In this specification, the SyncML client can work either as an OBEX client or as an OBEX server at the OBEX protocol layer. In consequence, the SyncML server can work either as an OBEX client or as an OBEX server. The OBEX role depends on the fact which one, the SyncML client or the SyncML server, initiates sync.

3.2.1 Bluetooth Service Discovery

To enable the OBEX connection over the Bluetooth protocol stack, the SyncML server MUST advertise and the SyncML client SHOULD advertise service records, which can be retrieved by a connecting device using the Bluetooth SDP [3].

In the case of the SyncML server, the following information, i.e., service records MUST be put into the SDDB (Service Discovery DataBase).

Item	Definition:	Type/ Size:	Value:	AttrID:	Status:	Default Value:
Service Class ID List			N/A	0x0001**	MUST	
Service Class #0	SyncMLServer	UUID	*	N/A	MUST	
Protocol Descriptor list			N/A	0x0004**	MUST	
Protocol ID #0	L2CAP	UUID	0x0100**	N/A	MUST	
Protocol ID #1	RFCOMM	UUID	0x0003**	N/A	MUST	
Param #0	CHANNEL	Uint8	Varies	N/A	MUST	
Protocol ID #2	OBEX	UUID	0x0008**	N/A	MUST	
Service name	Displayable Text name	String	Varies	0x0000+b***	MAY	“SyncML Server"

Table 1 SyncML Server Service Records

* The value 00000001-0000-1000-8000-0002EE000002 should be used in this place.

** The value or the attribute ID is specified in the Bluetooth Assigned Numbers specification [4].

*** ’b’ in this table represents a base offset as given by the LanguageBaseAttributeIDList attribute. For the principal language b must be equal to 0x0100 as described in the Bluetooth SDP specification [3].

The service records, which the SyncML client SHOULD put into its SDDB, are listed below.

Item	Definition:	Type/ Size:	Value:	AttrID:	Status:	Default Value:
Service Class ID List			N/A	0x0001**	MUST	
Service Class #0	SyncMLClient	UUID	*	N/A	MUST	
Protocol Descriptor list			N/A	0x0004**	MUST	
Protocol ID #0	L2CAP	UUID	0x0100**	N/A	MUST	
Protocol ID #1	RFCOMM	UUID	0x0003**	N/A	MUST	
Param #0	CHANNEL	Uint8	Varies	N/A	MUST	
Protocol ID #2	OBEX	UUID	0x0008**	N/A	MUST	
Service name	Displayable Text name	String	Varies	0x0000+b***	MAY	“SyncML Client"

Table 2 SyncML Client Service Records

* The value 00000002-0000-1000-8000-0002EE000002 should be used in this place.

** The value or the attribute ID is specified in the Bluetooth Assigned Numbers specification [4].

*** ’b’ in this table represents a base offset as given by the LanguageBaseAttributeIDList attribute. For the principal language b must be equal to 0x0100 as described in the Bluetooth SDP specification [3].

3.2.1.1 SDP Protocol Data Units

Table 3 shows the specified SDP PDUs (Protocol Data Units), which are required.

PDU no.	SDP PDU	Ability to Send		Ability to Retrieve	
		SyncML Client	SyncML Server	SyncML Client	SyncML Server
1	SdpErrorResponse	MUST*	MUST	MUST**	MUST
2	SdpServiceSearchAttribute-Request	MUST**	MUST	MUST*	MUST
3	SdpServiceSearchAttribute-Response	MUST*	MUST	MUST**	MUST

Table 3 SDP PDUs

* This is only applicable if the SyncML client is able to function as the OBEX server.

** This is only applicable if the SyncML client is able to function as the OBEX client.

3.2.2 Other Bluetooth Protocol Requirements

This specification partially requires compliance to the Bluetooth Serial Port (SeP) Profile [5] if Bluetooth is used as a physical medium for OBEX. These are:

- The compliance is required to the RFCOMM requirements as defined in Chapter 4 in the SeP Profile.
- The compliance is required to the L2CAP requirements as defined in Chapter 5 in the SeP Profile.
- The compliance is required to the LM protocol requirements as defined in Chapter 7 in the SeP Profile.

The SDP requirements are defined by this specification and thus, any of the requirements defined in the SeP profile (Chapter 6 in the SeP profile) does not apply to this specification. The SyncML server MUST comply with both the Device 'A' and Device 'B' requirements as defined in the SeP Profile. The SyncML client MUST comply with either the Device 'A' requirements, or with the Device 'B' requirements, or both as defined in the SeP Profile.

The Bluetooth LC (Link Controller) capabilities and The Bluetooth Generic Access Profile (GAP) requirements for this specification are defined in Chapter 6.5 and Chapter 7 of the Bluetooth GOEP [6], respectively. The SyncML server MUST comply with both the client and server requirements as defined in Chapter 6.5 and Chapter 7 in the GOEP. The SyncML client MUST comply with either the client requirements, or the server requirements, or both as defined in Chapter 6.5 and Chapter 7 in the GOEP.

4 OBEX Mapping to SyncML

The following sections define the requirements for the binding of SyncML to OBEX.

In client initiated sync, the SyncML client initiates the OBEX link, so it is also the OBEX client. The SyncML client can disconnect the OBEX link when it has received the last sync message from the SyncML server.

With server alerted sync, the SyncML server initiates the OBEX link, so it is the OBEX client. The SyncML server cannot disconnect the OBEX link before it has received the SyncML response message for the last SyncML message including a Sync command that it sends.

4.1 OBEX Operations

The following OBEX operations are required for SyncML.

OBEX Operation	SyncML Server		SyncML Client	
	OBEX Client	OBEX Server	OBEX Client	OBEX Server
Connect	MAY	MUST	MUST	MAY
Disconnect	MAY	MUST	MUST	MAY
Put	MAY	MUST	MUST	MAY
Get	MAY	MUST	MUST	MAY
Abort	MAY	MUST	MAY	MAY

The OBEX layer must be disconnected using the *OBEX Disconnect* operation. The OBEX specification also allows the link to be disconnected by disconnecting the underlying transport layer.

The OBEX connection can be authenticated as part of the OBEX CONNECT request/response messages, using the authenticate challenge and response headers

The client can send the OBEX ABORT request, to terminate a multi-packet operation (such as PUT) before it would normally end.

The PUT FINAL frame must be sent with an empty body.

4.2 OBEX Connection Overview

The OBEX connection is made at the start of the synchronisation, and remains open until the synchronisation has completed.

The following example shows the creation of an OBEX connection, the mapping of PUT and GET requests to the SyncML message transfers, and the OBEX disconnection.

This example is not intended to show a complete a SyncML Session but merely illustrates the use of PUT and GET within a SyncML OBEX binding implementation.

OBEX Client	OBEX Server	Message Direction
CONNECT Request		
	Success Response	
PUT Request		SyncML Message from
	Continue Response	OBEX Client to OBEX Server
PUT Request …		
	Continue Response …	
PUT Final Request		
	Success Response	
GET Final Request		SyncML Message from
	Continue Response	OBEX Server to OBEX Client
GET Final Request …		
	Continue Response …	
GET Final Request		
	Success Response	
DISCONNECT Request		
	Success Response	

4.2.1 Multiple Messages Per Package

Each SyncML message MUST be transferred as a SyncML MIME media type within the body of the OBEX request or response. However in order to transfer the message the OBEX / transport layer may split the message into many PUT requests, followed by a PUT Final Request. When there are multiple SyncML messages per SyncML package to transfer, each message is transferred in a separate 'set' of PUT/GET commands; depending on whether it is a SyncML request or response.

The recipient of a SyncML message can determine if there are more SyncML messages in the package by the absence of the Final element in the last received SyncML message. When the recipient receives a SyncML message with the Final element, it is the final message within that SyncML package.

Similarly if the PUT is not a PUT final then the recipient knows it is not the final part of the SyncML message, or if the response to the GET Final Request is not an OK/success then there is more data still to transfer.

4.2.2 MIME header type requirement

Data synchronization client implementations conforming to this specification MUST support this header with either the `"application/vnd.syncml+xml"` or `"application/vnd.syncml+wbxml"` media type values. Data synchronization server implementations conforming to this specification MUST support both `"application/vnd.syncml+xml"` and `"application/vnd.syncml+wbxml"` media type values, as requested by the SyncML data synchronization client.

Device Management client implementations conforming to this specification MUST support this header with either the `"application/vnd.syncml.dm+xml"` or `"application/vnd.syncml.dm+wbxml"` media type values. Device management server implementations conforming to this specification MUST support both `"application/vnd.syncml.dm+xml"` and `"application/vnd.syncml.dm+wbxml"` media type values, as requested by the SyncML device management client.

4.3 OBEX Connection Establishment

The OBEX connection is established by the SyncML application generating a Connect Request, and the remote device indicates that the connection has been established, by returning a Connect Response. For each SyncML session, a separate OBEX connection MUST be established.

The OBEX CONNECT request must contain the following fields.

Field/ Header	Name	Value	M/O	Explanation
Field	Opcode for CONNECT	0x80	M	
Field	Packet Length	Varies	M	
Field	OBEX Version Number	Varies	M	
Field	Flags	Varies	M	
Field	Max OBEX Packet Length	Varies	M	
Header	Target	Varies	M	The UUID to be used in data synchronization is SYNCML-SYNC and in device management is SYNCML-DM.

The OBEX CONNECT response must contain the following fields.

Field/ Header	Name	Value	M/O	Explanation
Field	Response code for CONNECT request	0x0A	M	0xA0 for success, otherwise fail
Field	Packet Length	Varies	M	
Field	OBEX Version Number	Varies	M	
Field	Flags	Varies	M	
Field	Max OBEX Packet Length	Varies	M	
Header	Connection ID	Varies	M	Connection ID is set by the Server during the OBEX Connect operation as a shorthand way for the client to direct the requests. **This must be the first header.**
Header	Who	Varies	M	The UUID returned is the same UUID that was sent in the connect request target header

4.4 Exchanging SyncML Data over the OBEX Connection

Once an OBEX connection has been established, SyncML data can be transferred over the link.

The PUT packet must include the following fields and headers.

Field/ Header	Name	Value	M/O	Explanation
Field	Opcode for PUT	0x02 or 0x82	M	0x02 is used for packets previous to the last put packet. 0x82 (which is 0x02 with the high bit set) is used for the last put packet.
Field	Packet Length	Varies	M	
Header	Connection ID	Varies	M	Connection ID is set to the value returned by the Server during the OBEX Connect operation. This must be the first header.
Header	Type	Varies	M	The MIME type of the object. This should contain the SyncML MIME type declaration.
Header	Length	Varies	O	Length of the object. This header is optional but highly recommended.
Header	Body/End of Body	Varies	M	End of Body identifies the last chunk of the object body.

The response to the PUT request has the following fields and headers.

Field/ Header	Name	Value	M/O	Explanation
Field	Response code for PUT	0x90, 0xAO, 0xCD, 0xCF, …	M	0x90 for continue 0xA0 for success 0xCD if the object is too large 0xCF if the object type is not supported
Field	Packet Length	Varies	M	

Other headers, which can be optionally used, are found in [1]

The GET packet must include the following fields and headers.

Field/ Header	Name	Value	M/O	Explanation
Field	Opcode for GET	0x03 or 0x83	M	0x03 is used for packets previous to the last get packet. 0x83 (which is 0x03 with the high bit set) is used for the last get packet. Note, in most cases the GET fits within a single packet.
Field	Packet Length	Varies	M	
Header	Connection ID	Varies	M	Connection ID is set to the value returned by the Server during the OBEX Connect operation. This must be the first header.
Header	Type	0x42, …	M	The MIME type of the object. This should contain the SyncML MIME type declaration.

The response to the GET request has the following fields and headers.

Field/ Header	Name	Value	M/O	Explanation
Field	Response code for GET	0x90, 0xAO, 0xC0, 0xC3, …	M	0x90 for continue 0xA0 for success 0xC0 bad request 0xC3 forbidden
Field	Packet Length	Varies	M	
Header	Length	0xC3, …	O	Length of the object. This header is optional but highly recommended.
Header	Body/End of Body	0x48/0x49, …	M	End of Body identifies the last chunk of the object body.

Other headers, which can be optionally used, are found in [1]

4.5 OBEX Disconnection

The OBEX connection is disconnected by the SyncML application, generating a Disconnect Request, and the remote device indicates that the connection has been terminated, by returning a success Response.

The OBEX DISCONNECT request must contain the following fields.

Field/ Header	Name	Value	M/O	Explanation
Field	Opcode for DISCONNECT	0x81	M	
Field	Packet Length	Varies	M	
Header	Connection ID	Varies	M	Connection ID is set to the value returned by the Server during the OBEX Connect operation. This must be the first header.

Other headers (such as Description) which can be optionally used are found in [1].

The response to an OBEX DISCONNECT request must contain the following fields.

Field/ Header	Name	Value	M/O	Explanation
Field	Response code for DISCONNECT	0xAO	M	0xA0 for success, otherwise fail
Field	Packet Length	Varies	M	

4.6 OBEX ABORT

The client can send an OBEX abort request to terminate a multi-packet operation (such as PUT) before it would normally end. The ABORT request and response always fit in one OBEX packet, and they always have the Final bit set.

The OBEX ABORT request must contain the following fields.

Field/ Header	Name	Value	M/O	Explanation
Field	Opcode for ABORT	0xFF	M	
Field	Packet Length	Varies	M	
Header	Connection ID	Varies	M	Connection ID is set to the value returned by the Server during the OBEX Connect operation. This must be the first header.

Other headers (such as Description) which can be optionally used are found in [1].

The response to an OBEX ABORT request must contain the following fields.

Field/ Header	Name	Value	M/O	Explanation
Field	Response code for ABORT	0xAO	M	0xA0 for success, otherwise fail and the client should disconnect the OBEX connection.
Field	Packet Length	Varies	M	

Other headers (such as Description) which can be optionally used are found in [1].

5 References

[1] IrDA Object Exchange Protocol (IrOBEX), Version 1.2, IrDA

[2] Key words for use in RFCs to Indicate Requirement Levels, IETF

[3] Bluetooth Service Discovery Protocol, Bluetooth

[4] Bluetooth Assigned Numbers specification, Bluetooth

[5] Bluetooth Serial Port (SeP) Profile, Bluetooth

[6] Bluetooth GOEP, Bluetooth