SyncML OBEX Binding, version 1.1.2

Approved Version 12-June-2003

Open Mobile Alliance
OMA-SyncML-OBEXBinding-V1_1_2-20030612-A

Continues the Technical Activities
Originated in the SyncML Initiative

Contents

1. Scope

The SyncML Initiative, Ltd. was a not-for-profit corporation formed by a group of companies who co-operated to produce an open specification for data synchronization and device management. Prior to SyncML, data synchronization and device management had been based on a set of different, proprietary protocols, each functioning only with a very limited number of devices, systems and data types. These non-interoperable technologies have complicated the tasks of users, manufacturers, service providers, and developers. Further, a proliferation of different, proprietary data synchronization and device management protocols has placed barriers to the extended use of mobile devices, has restricted data access and delivery and limited the mobility of the users.

SyncML is a specification that contains the following main components:

- An XML-based representation protocol
- A synchronization protocol and a device management protocol
- Transport bindings for the protocol

The data representation specifies an XML DTD that allows the representation of all the information required to perform synchronization or device management, including data, metadata and commands. The synchronization and device management protocols specify how SyncML messages conforming to the DTD are exchanged in order to allow a SyncML client and server to exchange additions, deletes, updates and other status information.

There are also DTDs which define the representation of information about the device such as memory capacity, and the representation of various types of meta information such as security credentials.

Although the SyncML specification defines transport bindings that specify how to use a particular transport to exchange messages and responses, the SyncML representation, synchronization and device management protocols are transport-independent. Each SyncML package is completely self-contained, and could in principle be carried by any transport. The initial bindings specified are HTTP, WSP and OBEX, but there is no reason why SyncML could not be implemented using email or message queues, to list only two alternatives. Because SyncML messages are self-contained, multiple transports may be used without either the server or client devices having to be aware of the network topology. Thus, a short-range OBEX connection could be used for local connectivity, with the messages being passed on via HTTP to an Internet-hosted synchronization server.

To reduce the data size, a binary coding of SyncML based on the WAP Forum's WBXML is defined. Messages may also be passed in clear text if required. In this and other ways SyncML addresses the bandwidth and resource limitations imposed by mobile devices.

SyncML is both data type and data store independent. SyncML can carry any data type which can be represented as a MIME object. To promote interoperability between different implementations of SyncML, the specification includes the representation formats used for common PIM data.

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 [OBEX]

2. References

2.1 Normative References

[BTAN] "Bluetooth Assigned Numbers", Bluetooth SIG, URL: http://www.bluetoothsig.org/assigned-numbers/

[BTGOEP] "Bluetooth V1.1 Profile Specifications" – PartK:10 Generic Object Exchange Profile, Bluetooth SIG, URL: http://www.bluetooth.org/foundry/specification/document/Bluetooth_11_Profiles_Book/en/1/Bluetooth_11_Profiles_Book.pdf

[BTSDP] "Bluetooth V1.1 Core Specifications" - PartE: Service Discovery Protocol, Bluetooth SIG, URL: http://www.bluetooth.org/foundry/specification/document/Bluetooth_V1.1_Core_Specifications/en/1/Bluetooth_V1.1_Core_Specifications.pdf

[BTSEP] "Bluetooth V1.1 Profile Specifications" – PartK:5 Serial Port Profile, Bluetooth SIG, URL: http://www.bluetooth.org/foundry/specification/document/Bluetooth_11_Profiles_Book/en/1/Bluetooth_11_Profiles_Book.pdf

[OBEX] "IrDA Object Exchange Protocol (IrOBEX) Version 1.2", Infrared Data Association, URL: http://www.irda.org/standards/pubs/OBEX1p2_Plus.zip

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997, URL:http://www.ietf.org/rfc/rfc2119.txt

[RFC2396] "Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee, et al., August 1998, URL:http://www.ietf.org/rfc/rfc2396.txt

2.2 Informative References

None.

3. Terminology and Conventions

3.1 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

Any reference to components of the SyncML DTD or XML snippets is specified in this `typeface`.

3.2 Definitions

Application - A SyncML application that supports the SyncML protocol. The application can either be the originator or recipient of the SyncML protocol commands. The application can act as a SyncML client or a SyncML server.

Capabilities exchange - The SyncML capability that allows a client and server to exchange what device, user and application features they each support.

Client - A SyncML Client refers to the protocol role when the application issues SyncML "request" messages. For example in data synchronization, the `Sync` SyncML Command in a SyncML Message.

Command - A SyncML Command is a protocol primitive. Each SyncML Command specifies to a recipient an individual operation that is to be performed. For example, the SyncML Commands supported by this specification include `Add`, `Alert`, `Atomic`, `Copy`, `Delete`, `Exec`, `Get`, `Map`, `Replace`, `Search`, `Sequence` and `Sync`.

Data - A unit of information exchange, encoded for transmission over a network.

Data collection - A data element which acts as a container of other data elements, (e.g., {c {{i_1, $data_1$}, ... {i_n, $data_n$}}}). In SyncML, data collections are synchronized with each other. See data element.

data element -A piece of data and an associated identifier for the data, (e.g., {i, data}).

Data element equivalence -When two data elements are synchronized. The exact semantics is defined by a given data synchronization model.

Data exchange - The act of sending, requesting or receiving a set of data elements.

Data format - The encoding used to format a data type. For example, characters or integers or character encoded binary data.

Data type - The schema used to represent a data object (e.g., text/calendar MIME content type for an iCalendar representation of calendar information or text/directory MIME content type for a vCard representation of contact information).

Data synchronization - The act of establishing an equivalence between two data collections, where each data element in one item maps to a data item in the other, and their data is equivalent.

Data synchronization protocol - The well-defined specification of the "handshaking" or workflow required to accomplish synchronization of data elements on an originator and recipient data collection. The SyncML specification forms the basis for specifying an open data synchronization protocol.

Message - A SyncML Message is the primary contents of a SyncML Package. It contains the SyncML Commands, as well as the related data and meta-information. The SyncML Message is an XML document.

OBEX – Objext Exchange Protocol that is defined in [OBEX] .

Operation - A SyncML Operation refers to the conceptual transaction achieved by the SyncML Commands specified by a SyncML Package. For example in the case of data synchronization, "synchronize my personal address book with a public address book".

Originator - The network device that creates a SyncML request.

Package - A SyncML Package is the complete set of commands and related data elements that are transferred between an originator and a recipient. The SyncML package can consist of one or more SyncML Messages.

Parser - Refers to an XML parser. An XML parser is not absolutely required to support SyncML. However, a SyncML implementation that integrates an XML parser may be easier to enhance.

This document assumes that the reader has some familiarity with XML syntax and terminology.

Recipient - The network device that receives a SyncML request, processes the request and sends any resultant SyncML response.

Representation protocol - A well-defined format for exchanging a particular form of information. SyncML is a representation protocol for conveying data synchronization and device management operations.

SyncML request message - An initial SyncML Message that is sent by an originator to a recipient network device.

SyncML response message - A reply SyncML Message that is sent by a recipient of a SyncML Request back to the originator of the SyncML Request.

Synchronization data - Refers to the data elements within a SyncML Command. In a general reference, can also refer to the sum of the data elements within a SyncML Message or SyncML Package.

Server - A SyncML Server refers to the protocol role when an application issues SyncML "response" messages. For example in the case of data synchronization, a Results Command in a SyncML Message.

3.3 Abbreviations

URI	Uniform Resource Identifier [RFC2396]
URL	Uniform Resource Locator [RFC2396]
WAP	Wireless Application Protocol
XML	Extensible Markup Language

4. 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 [OBEX] .

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.

5. OBEX Introduction

OBEX [OBEX] 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.

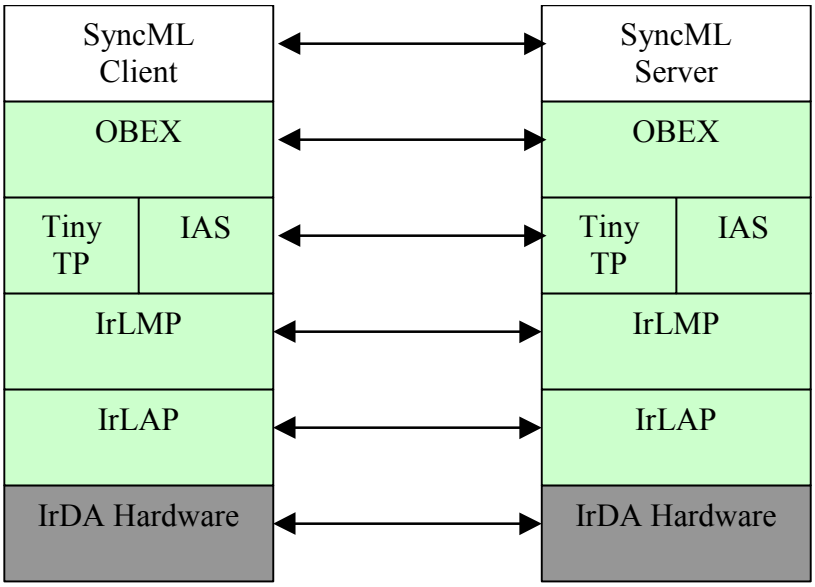
In this specification, the SyncML client can work either as an OBEX client or as an OBEX server at the OBEX protocol layer. In consequence, the SyncML server can work either as an OBEX client or as an OBEX server. The OBEX role depends on the fact which one, the SyncML client or the SyncML server, initiates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

5.1 OBEX Over IrDA

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



e.g. Phone **e.g. PC**

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.

5.1.1 IAS Entry

To enable an OBEX connection over IrDA, the OBEX protocol stack needs to provide IAS setting information to the IAS protocol stack. The SyncML server and SyncML client SHOULD use the following IAS entry settings for SyncML communication via OBEX over IrDA.

5.1.1.1 IAS Entry Settings for SyncML Server

SyncML server SHOULD use the following IAS entry settings.

Class		OBEX:SyncML-Server
Attribute	Name	IrDA:TinyTP:LsapSel
	Type	Integer
	Description	IrLMP LSAP selector for SyncML over IrOBEX, legal values from 0x01 to 0x6F

5.1.1.2 IAS Entry Settings for SyncML Client

SyncML client SHOULD use the following IAS entry settings.

Class		OBEX:SyncML-Client
Attribute	Name	IrDA:TinyTP:LsapSel
	Type	Integer
	Description	IrLMP LSAP selector for SyncML over IrOBEX, legal values from 0x01 to 0x6F

5.2 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.

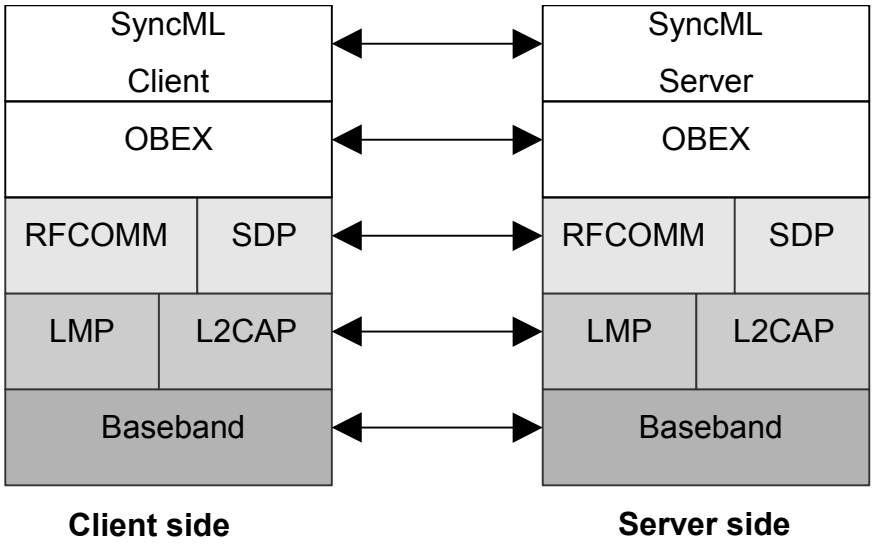


Figure 1 OBEX over Bluetooth

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 [BTSDP].

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.

5.2.1 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 [BTSDP].

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 [BTAN].

*** ’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 [BTSDP].

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 [BTAN].

*** '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 [BTSDP].

5.2.1.1 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.

5.2.2 Other Bluetooth Protocol Requirements

This specification partially requires compliance to the Bluetooth Serial Port (SeP) Profile [BTSEP] 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 [BTGOEP], 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.

6. 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.

6.1 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.

6.2 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	

6.2.1 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.

6.2.2 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.

6.3 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	0x10	M	In [OBEX] it's stated that current version is 1.0
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	0x10	M	In [OBEX] it's stated that current version is 1.0
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

6.4 Exchanging SyncML Data over the OBEX Connection

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

The PUT request packets have 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. Sent only once in first packet in the request sequence.
Header	Type	Varies	M	The MIME type of the object. This MUST contain the SyncML MIME type declaration. Sent only once in first packet in the request sequence, MUST precede object Body headers.
Header	Length	Varies	O	Length of the object. This header is optional but highly recommended. Sent only once in the request sequence.

Header	Body/End of Body	Varies	M	End of Body identifies the last chunk of the object body. End of Body header (PUT final packet) SHOULD be sent with an empty 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 [OBEX]

The GET packets have 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 packet containing headers. 0x83 (which is 0x03 with the final bit set) is used for the last packet containing headers and for subsequent get packets in the current Get request sequence without headers. No headers can be sent in Get request packets once the 0c83 has been sent in a previous 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. Sent only once in first packet in the request sequence.
Header	Type	0x42, …	M	The MIME type of the object. This MUST contain the SyncML MIME type declaration. Sent only once in first packet in the request sequence.

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. Sent only once in the request sequence.
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 [OBEX]

6.5 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 [OBEX] .

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	

6.6 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 [OBEX] .

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 [OBEX] .

7. Examples

The following examples are formatted as the examples in [OBEX] .

7.1 OBEX Connect Example

Client Request:	**Bytes**	**Meaning**
Opcode	0x80	**CONNECT**, Final bit set
	0x0015	packet length = 21
	0x10	version 1.0 of OBEX
	0x00	flags, all zero for this version of OBEX
	0x0200	512 bytes is the max OBEX packet size client can accept
	0x46	HI for **Target** header
	0x000E	Length of **Target** header
	0x53594E434D4C	UUID for SyncML DS (“SYNCML-SYNC”)
	2D53594E43	
Server Response:		
response code	0xA0	SUCCESS, Final bit set
	0x001A	packet length of 26
	0x10	version 1.0 of OBEX
	0x00	Flags
	0x0200	1K max packet size
	0xCB	HI for **Connection Id** header
	0x00000001	ConnId = 1
	0x4A	**Who** HI
	0x000E	Length of **Who** Header
	0x53594E434D4C	UUID of responding application (same value as Target header in request (“SYNCML-SYNC”)
	2D53594E43	

7.2 OBEX Disconnect Example

Client Request:	**bytes**	**Meaning**
opcode	0x81	**DISCONNECT**, Final bit set
	0x0008	packet length = 8
	0xCB	HI for **Connection Id** header
	0x00000001	ConnId = 1
Server Response:		
response code	0xA0	SUCCESS, Final bit set
	0x0003	packet length = 3

7.3 OBEX Abort Example

Client Request:	**bytes**	**Meaning**
opcode	0xFF	**ABORT**, Final bit set
	0x0008	packet length = 8
	0xCB	HI for **Connection Id** header
	0x00000001	ConnId = 1
Server Response:		
response code	0xA0	SUCCESS, Final bit set
	0x0003	packet length = 3

7.4 OBEX Put Example

Client Request:	**Bytes**	**Meaning**
opcode	0x02	**PUT**, Final bit not set
	0x0200	512 bytes is length of packet
	0xCB	HI for **Connection Id** header
	0x00000001	ConnId = 1
	0x42	HI for **Type** header
	0x0020	Length of **Type** header

	0x6170706C6963	Type of object (application/vnd.syncml+wbxml)
	6174696F6E2F76	(null terminated ASCII text)
	6E642E73796E63	
	6D6C2B7762786D6C00	
	0xC3	HI for **Length** header
	0x000004FE	Length of object is 1278 bytes
	0x48	HI for Object **Body** chunk header
	0x01D3	Length of **Body** header = 467. 464 plus HI and header length
	0x.....	464 bytes of body
Server Response:		
response code	0x90	CONTINUE, Final bit set
	0x0003	length of response packet
Client Request:		
opcode	0x02	**PUT**, Final bit not set
	0x0200	512 bytes is length of packet
	0x48	HI for Object **Body** chunk
	0x01FD	Length of **Body** header = 509. 506 plus HI and header length
	0x.....	next 506 bytes of body
Server Response:		
response code	0x90	CONTINUE, Final bit set
	0x0003	length of response packet
Client Request:		
opcode	0x02	**PUT**, Final bit not set
	0x013A	314 bytes is length of packet
	0x48	HI for Object **Body** chunk
	0x0137	Length of header = 311. 308 plus HI and header length
	0x.....	last 308 bytes of body
Server Response:		

response code	0x90	CONTINUE, Final bit set
	0x0003	length of response packet
Client Request:		
opcode	0x82	**PUT**, Final bit set
	0x0006	6 bytes is length of packet
	0x49	HI for **End-of-Body** chunk
	0x0003	Length of header = 3. 0 plus HI and header length
Server Response:		
response code	0xA0	SUCCESS, Final bit sent
	0x0003	length of response packet

7.5 OBEX Get Example

Client Request:	**bytes**	**Meaning**
opcode	0x83	**GET**, Final bit set
	0x0028	40 bytes is length of **GET** packet
	0xCB	HI for **Connection Id** header
	0x00000001	ConnId = 1
	0x42	HI for **Type** header
	0x0020	Length of **Type** header
	0x6170706C6963	Type of object (application/vnd.syncml+wbxml)
	6174696F6E2F76	(null terminated ASCII text)
	6E642E73796E63	
	6D6C2B7762786D 6C00	
Server Response:		
Response code	0x90	CONTINUE, Final bit set
	0x0200	length of response packet
	0xC3	HI for **Length** header
	0x000002BE	Length of object is 702 bytes
	0x48	HI for Object **Body** chunk

	0x01F8	Length of header = 504. 501 plus HI and header length
	0x.....	501 bytes of body
Client Request:	**bytes**	**Meaning**
opcode	0x83	**GET**, Final bit set
	0x0003	3 bytes is length of **GET** packet
Server Response:		
Response code	0xA0	SUCCESS, Final bit set
	0x0038	length of response packet
	0x49	HI for **End-of-Body** chunk
	0x00CC	Length of header = 204. 201 plus HI and header length
	0x.....	last 201 bytes of body.

Appendix A. Static Conformance Requirements (Normative)

The static conformance requirements for this specification are specified in section 6 “OBEX Mapping to SyncML” of this document.

Appendix B. Change History (Informative)

B.1 Approved Version History

Reference	Date	Description
OMA-SyncML-OBEXBinding-V1_1_2-20030612-A	12 June 2003	Approved by TP. TP ref# OMA-TP-2003-0265R1