Session Capabilities in OBEX™

Version 0.14

July 16, 2002

Authors:

David Suvak Extended Systems

Contributors:

Kevin Hendrix Extended Systems

Revision History

Revision	Date	Comments
0.1	30-May-01	Initial Draft
0.11	12-July-01	Fixed document based on comments received. Added examples, flushed out algorithms, specified additions to Capability Object.
0.12	12-Sept-01	Change the multiple transport issue into a proposal that is not considered part of the errata proposal.
0.13	4-Oct-01	Changed the sequence number to wrap at 255 instead of 8
0.14	16-July-02	Moved the Session Opcode out of the OBEX Session command and into the Session Parameters header to assure interoperability with legacy OBEX devices that might not understand the new OBEX Session command format. Also changed the Session-Parameters header to 0x52 and the Session-Sequence-Number header to 0x93 to avoid conflict with the existing OBEX 1.2 errata.

Introduction

A number of applications and services built on OBEX™ desire the ability to recover an OBEX session that has been broken due to loss of the underlying transport (e.g. IrLAP link loss). Examples include restarting a file transfer and resuming a sync session. The Infrared Financial Messaging (IrFM) group also has determined the need to maintain an OBEX session when the IrLAP link is lost. This document contains a proposal for providing reliable session support in OBEX. This proposal is aimed at meeting the needs of IrFM but should provide enough capabilities to meet the needs of other services and applications. A list of desired features for an OBEX session layer is given below.

- Allow an OBEX session to be resumed without loss of data when the underlying transport layer is lost.
- Allow OBEX applications and services to suspend a session and later resume it.
- Provide indications to applications and services about loss of the underlying transport connection.
- Provide the ability to automatically resume the session including restarting the lower level transports without burdening the applications or services.
- Allow existing services such as the Inbox, File transfer, IrMC, etc. to have session support.
- Allow multiple directed connections to be started and stopped within a single session.

A lower priority feature is listed below.

- Ability to start a session on one transport and move it to another transport (e.g. start the OBEX session on IrDA and move it to Bluetooth).

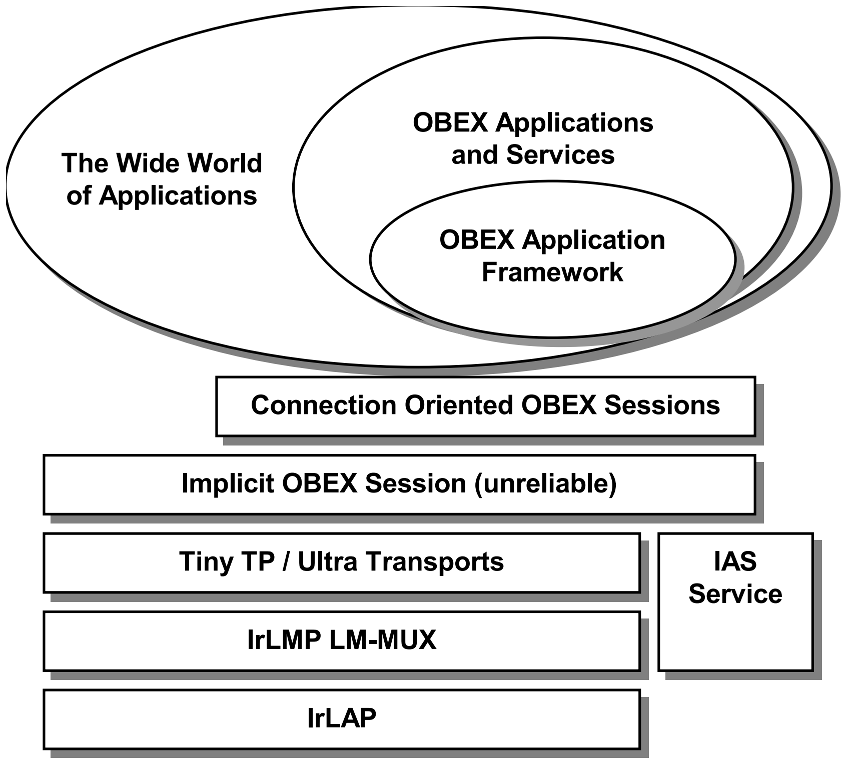
OBEX Session Overview

The OBEX specification divides OBEX into two parts: a model for representing objects and a session protocol. The session protocol uses a binary packet-based client/server request-response model. The session protocol defines the basic structure of an OBEX conversation and includes a set of commands to perform specific actions such as **PUT** and **GET**. The OBEX conversation occurs within the context of an OBEX connection. The OBEX session protocol runs on top of a transport layer such as TinyTP in IrDA, RFCOMM in Bluetooth or TCP. Currently, when the underlying transport is disconnected the OBEX session is lost. This document describes a solution, which allows OBEX to maintain the session when the underlying transport is lost. Before describing this solution it is necessary to review the current OBEX session protocol. The table below describes the elements of the OBEX session layer protocol and application framework.

Element	Description
OBEX Client	An OBEX Client is the entity that initiates OBEX operations. The OBEX client also typically initiates the underlying transport connection to an OBEX server.
OBEX Server	An OBEX Server is the entity that responds to OBEX operations. The OBEX server typically waits for the OBEX client to initiate the underlying transport connection. However, in some cases the OBEX server may initiate the transport connection to an OBEX client (e.g. IrFM).
Default OBEX Server	The Default OBEX server is the server that resides at the LSAP-Sel specified in the OBEX IAS definition. Other OBEX servers can exist but the Default OBEX server is the "well known" server. This is analogous to the HTTP server located at TCP port number 80.
OBEX Transport Connection	The OBEX transport connection is the underlying transport connection, which carries the OBEX protocol.
OBEX Connection	An OBEX Connection is a virtual binding between two applications or services. An OBEX connection is initiated by sending an OBEX **CONNECT** packet. Once a connection is established all operations sent over the connection are interpreted in a continuous context.
Directed Connection	A directed connection is one where the OBEX **CONNECT** packet contains targeting information which the OBEX protocol uses to connect the client to its intended service or application.
The Inbox Connection	The inbox connection is the OBEX connection made to the default OBEX server, where the OBEX **CONNECT** packet does not contain targeting information. A number of services can be accessed via the inbox connection. These services are described later.
Inbox	The inbox is the intended recipient of a client push operation over the Inbox Connection. The inbox does not have to be an actual storage location. It is really a method for encapsulating the concept that the client pushes an object to a recipient without the need to understand the details of how the recipient stores the object.
Application	An OBEX application communicates using a proprietary method known only by the manufacturer. Such applications can only expect to be understood by exact peers. Alternatively, an application may be a service with proprietary extensions. In this case the application must know if it is communicating with a service or application peer.
Service	An OBEX service communicates using procedures specified in a publicly available standard, such as in IrMC or this specification.
Capability Service	The capability service is used to find information about the OBEX server including device information, types of objects supported, object profiles and supported applications.

When the OBEX client creates an OBEX Transport Connection to the OBEX Server, an implicit OBEX session is created. A connection-oriented session can be created by sending an OBEX **CONNECT** command. An implicit OBEX session exists because it is possible to send directed commands to a service without creating a directed connection to it first. Connection-oriented sessions created using the OBEX **CONNECT** command are also called OBEX Connections. In the case of the Default OBEX Server, sending an OBEX **CONNECT** command without a **Target** header creates a connection to the Inbox service. This connection is called the Inbox connection and there are a number of services, which utilize this connection. A Directed Connection is made by sending an OBEX **CONNECT** command with a **Target** header containing the ID of the desired service or application.

The Default OBEX Server resides at a well-known location. The method for finding the Default OBEX Server is dependent on the underlying transport being used. OBEX applications can use Directed Connections on the Default OBEX Server or they can use there own OBEX server which runs over a different OBEX transport connection from the one used by the Default OBEX Server.
Currently, the implicit OBEX session created when an OBEX Client initiates an OBEX Transport connection is only reliable as long as the OBEX Transport connection exists. When the underlying transport connection is disconnected the OBEX session fails. It is desired to have reliable OBEX sessions, which can be resumed when the underlying transport connection is broken. Given this context (broken OBEX Transport Connection), the implicit OBEX session described in the current OBEX specification is considered unreliable. The picture below shows the current architecture as it sits on IrDA.


The Wide World of Applications
OBEX Applications and Services
OBEX Application Framework
Connection Oriented OBEX Sessions
Implicit OBEX Session (unreliable)
Tiny TP / Ultra Transports
IAS Service
IrLMP LM-MUX
IrLAP

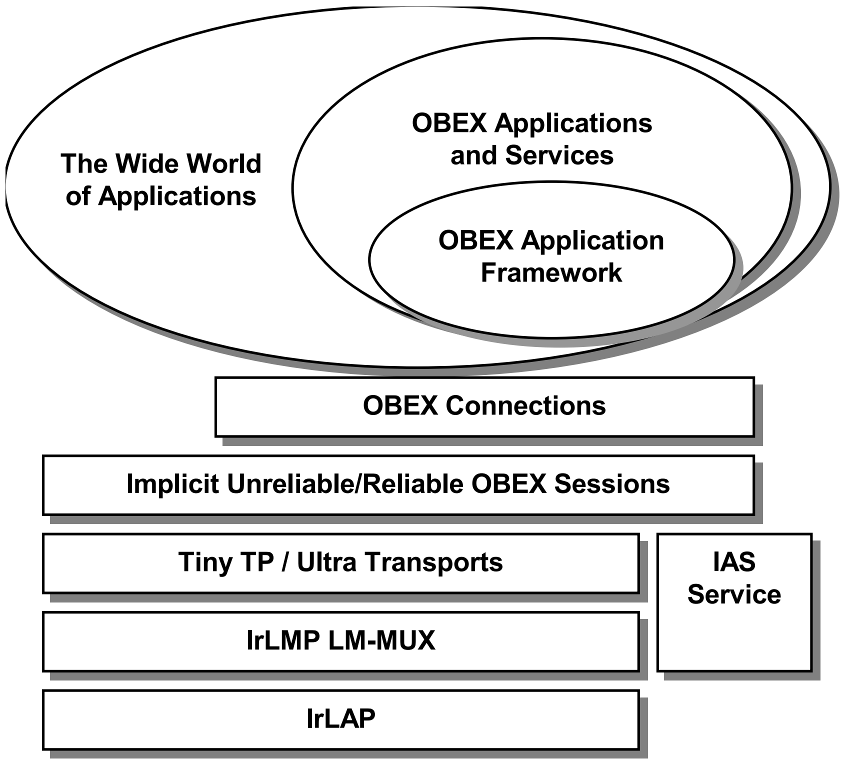
The next section describes a method for creating reliable OBEX sessions.

Reliable OBEX Session Overview

When the OBEX Client initiates the underlying transport connection to the OBEX Server it creates an unreliable implicit OBEX session. Any connection-oriented sessions made on top of this implicit session are also unreliable. This document proposes that reliable OBEX connections can be built by creating a reliable underlying session. The OBEX client can create a reliable session by sending a new OBEX command called **CREATESESSION**. In order to be able to reestablish the reliable session when the underlying transport breaks it is necessary for the session to have a Session ID that is known by both the OBEX Client and Server. This Session ID must be unique enough such that both the Client and the Server know they are resuming the same session. It is also necessary that the Client and Server know enough information about each other so that the underlying transport can be reestablished. In the case of IrDA this would be the 32-bit device address. Until a reliable session is established the unreliable implicit session is active. The list below outlines the concepts of a reliable OBEX Session.

- New OBEX commands are created for managing reliable sessions. These are **CREATESESSION**, **CLOSESESSION**, **SUSPENDSESSION, RESUMESESSION** and **SETTIMEOUT**.
- New OBEX headers are created to be used with the new commands and features. These are **Session-Parameters** and **Session-Sequence-Number**.
- When a reliable session is established a session ID is created that is known by both the Client and the Server. This ID must be unique enough such that the reliable session can be reestablished. When the underlying transport is broken under a reliable session, the session is considered suspended.
- The session context must be saved in order to resume a session when the underlying transport is broken.
- Applications/services must also maintain context in order to resume a session.
- Only one session can be active at a time per OBEX transport connection. When the OBEX transport connection is first established the unreliable implicit session is active. Multiple sessions can exist between devices but only one can be active at a time. The other sessions are considered suspended including the unreliable session. If the underlying transport is broken the unreliable session is broken as well.
- Any number of connection-oriented sessions (OBEX connections) can be made within a reliable session.
- The OBEX Capability Object can be accessed via any underlying session including the unreliable session.
- To facilitate reliability, the session layer must ensure reliable transfer of OBEX packets. To achieve this a header containing sequence numbers is added to each OBEX packet. If an OBEX packet is lost it will be retransmitted. This will occur only when a session is resumed since OBEX packets can only be lost when the underlying transport is lost (OBEX requires a reliable transport).
- Since OBEX packets are large, retransmission should only be used if the packet was actually lost. It is possible that the server received a command but the client did not receive the response. A method exists to allow the server to indicate the last packet received during session resume. The client can then send an empty packet (command without headers) to facilitate retransmission of the response.

The picture below shows how a reliable sessions fit into the architecture.


The Wide World of Applications
OBEX Applications and Services
OBEX Application Framework
OBEX Connections
Implicit Unreliable/Reliable OBEX Sessions
Tiny TP / Ultra Transports
IAS Service
IrLMP LM-MUX
IrLAP

New OBEX Headers

The two new headers being proposed are as follows:

HI – identifier	header name	Description
0x52	Session-Parameters	Parameters used in session commands/responses
0x93	Session-Sequence-Number	Sequence number used in each OBEX packet for reliability.

Session-Parameters

Session-Parameters is a byte sequence that is required for the **CREATESESSION**, **CLOSESESSION**, **SUSPENDSESSION**, **RESUMESESSION** and **SETTIMEOUT** commands and the responses to these commands. It contains a set of parameters, which are specific to the command in which the header is used. If required, the **Session-Parameters** must be the first header in a **SESSION** command or response. In addition, for **SESSION** commands only, the Session Opcode must be the first tag-length-value triplet within the **Session-Parameters** header.

A Tag-Length-Value encoding scheme is used to support a variety of parameters. A **Session-Parameters** header may contain more than one tag-length-value triplet. The header format is shown below:

Parameter Triplet 1			Parameter Triplet 2			Parameter Triplet . . .		
Tag1	Length	Value	Tag2	Length	Value	Tag	Length	Value

The tag and length fields are each one byte in length. The value field can be from zero to *n* bytes long. The value *n* is constrained by the maximum size of an OBEX Packet, which at this point is 255 bytes, the length field maximum of 255 bytes and the size of other TLV-triplets encoded in the header.

The following TAG values are defined.

Tag Value	Name	Meaning
0x00	Device Address	The *device address* of the device sending the header. If running over IrDA this is the 32-bit device address. For Bluetooth this is the 48-bit Bluetooth address. If Running over TCP/IP this is the IP address.
0x01	Nonce	The nonce is a value provided by the device, which will be used in the creation of the session ID. This number must be unique for each session created by the device. One method for creating the nonce is to start with a random number then increment the value each time a new session is created. The Nonce should be at least 4 bytes and at most 16 bytes in size.
0x02	Session ID	Session ID. This is a 16-byte value, which is formed by taking the device address and nonce from the client and server and running the MD5 algorithm over the resulting string. The Session ID is created as follows: MD5("*Client Device Address*" "*Client Nonce" "Server Device Address*" "*Server Nonce*")
0x03	Next Sequence Number	This is a one-byte value sent by the server, which indicates the next sequence number expected when the session is resumed.
0x04	Timeout	This is a 4-byte value that contains the number of seconds a session can be in suspend mode before it is considered closed. The value of 0xffffffff indicates a timeout of infinity. This is the default timeout. If a device does not send a timeout field then it can be assumed that the desired timeout is infinity. The timeout in affect is the smallest timeout sent by the client or server.
0x05	Session Opcode	The session opcode is a 1-byte value sent by the client to indicate the Session command that is to be performed. This tag-length-value is only sent in **SESSION** commands and must be the first tag in the **Session-Parameters** header. The session opcode definitions are defined in the bulleted list below.
0x06 - 0xff		Reserved

Session Opcode Definitions:

• 0x00	Create Session
• 0x01	Close Session
• 0x02	Suspend Session
• 0x03	Resume Session
• 0x04	Set Timeout
• 0x05 – 0xFF	Reserved

Generating a Session ID

The purpose of the *Session ID* is to uniquely identify the session. The *Session ID* is used when closing and resuming sessions. Both the client and the server want to be certain that the remote device is referring to the same session when issuing commands, especially when resuming a suspended session. Both the client and the server provide information used to generate the *Session ID*. This helps to make certain that the *Session ID* is unique. Each device provides two pieces of information. The first is its *Device Address*. In the case of IrDA this is the 32-bit device address, which is not guaranteed to be unique. The second is a *Nonce*, which must be unique for the given device. Given that the *Nonce* is unique as far as each device is concerned and that the device addresses are reasonably unique (even in the case of IrDA) the resulting *Session ID* has a high probability of being unique.

The Session ID is generated by applying the MD5 algorithm to the string created by concatenating the *Client Device Address*, *Client Nonce*, *Server Device Address* and *Server Nonce*. The reason for applying an algorithm such as MD5 is so that the size of *Session IDs* is fixed (16 bytes). Device Addresses and Nonces can very in size so the MD5 algorithm is used to produce a 16-byte quantity. The MD5 algorithm was designed to produce a unique number given a string or file. MD5 was chosen since it is used in the OBEX authentication procedure and was very likely to already be present in an OBEX implementation.

Negotiating Timeouts

All commands except **CLOSESESSION** involve timeout negotiation. Timeout negotiation is simple. Each time a Session Command exchange occurs (except **CLOSESESSION**) both sides take the smallest timeout values sent and use that as the new timeout for the session. If a timeout value is not explicitly sent it is assumed that the device is requesting an infinite timeout.

Session-Sequence-Number

Session-Sequence-Number is a 1-byte quantity containing the current sequence number. Even though OBEX is a command/response protocol and only one outstanding command can exist at any given time, the counting capacity of the sequence number is 256 using digits 0 – 255 (0xFF). At 255 the sequence number wraps around to 0. This adds another level of verification that the correct session is being resumed. If the sequence number is invalid then the session should be aborted. This procedure should be followed both when the session is being resumed and during an active session.

When the session is first created the **Session-Sequence-Number** number is set to 0. The first command packet sent by the client after the session is established will contain a **Session-Sequence-Number** header with value of 0. The response to this packet from the server contains the next packet number expected. If the packet was received successfully then the value of the **Session-Sequence-Number** header in the response would be 1.

The **Session-Sequence-Number** must be the first header. The OBEX specification indicates that the **Connection-ID** header must be the first header so there appears to be conflict. Since the new reliable session features sit below the standard OBEX parser, the reliable session could be considered a lower layer protocol. Thus, session information should be stripped off before being sent up to the OBEX parser. The standard OBEX parser will not see the **Session-Sequence-Number** header and the **Connection ID** header will appear to be the first header when it exists.

Session-Sequence-Number headers are not used in Session commands or responses. Thus, session numbers are not advanced when sending session commands/responses when a reliable session is active.

New OBEX Commands

The method proposed to add new session commands is to create one new OBEX command called **SESSION**. The **SESSION** request is formatted as follows:

Byte 0	Bytes 1, 2	Bytes 3 to n
0x87 (Final bit set)	Packet length	Sequence of headers. **Session-Parameters** must be the first header in the sequence.

The positive response to a **SESSION** request is 0xA0 (Success, with the high bit set. Any failure response code can be used to indicate a negative response. Certain types of failures and the corresponding failure codes are described in the sections corresponding to each session command. The format of the response is as follows:

Byte 0	Bytes 1, 2	Bytes 3 to n
response code	Response packet length	Optional sequence of response headers. If the **Session-Parameters** is used it must be the first header in the sequence.

CREATESESSION

The **CREATESESSION** command is sent by the Client to create a new session. This command must include a **Session-Parameters** header containing the *Session Opcode, Device Address* and *Nonce* fields. Optionally, a *Timeout* field can be included. The successful response to **CREATESESSION** is 0xA0 (Success, with the high bit set) in the response code followed by a **Session-Parameters** header containing *Device Address*, *Nonce*, and *Session ID* fields. The client should verify that it creates the same session ID as sent by the server. If the Session ID does not match then the client should close the session by using the **CLOSESESSION** command otherwise the session is considered active.

The **CREATESESSION** command and response must fit in a single OBEX packet of size 255 bytes (default maximum size) and have their Final bits set.

Only one session can be in the active state per transport connection. If a reliable session is already active a second request should be rejected using the 0xC3 (Forbidden) response. To switch sessions the current session must first be suspended. A server is not required to maintain multiple sessions at the same time. All servers will have some maximum number of suspended sessions they can maintain. At some point a server will receive a **CREATESESSION** request that will exceed this maximum number. The server can either reject the new request or close an existing session and accept the new request. The correct response depends on how long the existing sessions have been suspended. The server may want to prompt the user if it cannot decide on its own. A proposal for an algorithm is given in the next paragraph. If a **CREATESESSION** request is rejected because the maximum number of sessions already exists then the proper response is 0xD3 (Service Unavailable).

When a server receives a **CREATESESSION** request that exceeds the maximum number of sessions supported the following algorithm can be used.

1. Keep all suspended sessions with infinite timeouts on a sorted list by time suspended. The session with the longest time is at the front of the list.
2. Sessions with timeouts less then infinite should not be closed until the timeout occurs.
3. When a **CREATESESSION** request occurs exceeding the maximum number of sessions, the server should check the suspended session list. If the list is not empty the first session on the list is removed. This session is closed and its resources are used for the new session. If the list is empty the incoming **CREATESESSION** request is rejected.

CLOSESESSION

CLOSESESSION is used to gracefully close an existing session. This command must include a **Session-Parameters** header containing the *Session Opcode and Session ID* fields. The value of the *Session ID* field is the one sent by the server in the response to the **CREATESESSION** command corresponding to the session being closed. The successful response to **CLOSESESSION** is 0xA0 (Success, with the high bit set) in the response code. The **CLOSESESSION** command can be used to close the active session or any suspended sessions.

The **CLOSESESSION** command and response must fit in a single OBEX packet and have their Final bits set.

The CLOSESESSION command cannot be rejected if the *Session ID* corresponds to a valid session. If the *Session ID* does not correspond to a valid session then the proper response is 0xC3 (Forbidden). If the active session is closed the unreliable session becomes the active session.

SUSPENDSESSION

SUSPENDSESSION is used to gracefully suspend the active session. This command must include a **Session-Parameters** header containing the *Session Opcode* field. Optionally, a *Timeout* field can be included. The successful response to **SUSPENDESESSION** is 0xA0 (Success, with the high bit set) in the response code.

The **SUSPENDSESSION** command and response must fit in a single OBEX packet and have their Final bits set.

The **SUSPENDSESSION** command cannot be rejected if a reliable session is active. If the unreliable session is the current active session then the proper response is 0xC3 (Forbidden). The unreliable session becomes the active session when a reliable session is suspended.

RESUMESESSION

RESUMESESSION is used to resume a session that has been suspended. A session is suspended by using the **SUSPENDSESSION** command or when the underlying transport is broken. This command must include a **Session-Parameters** header containing the *Session Opcode, Device Address*, *Nonce*, and *Session ID* and fields. Optionally, a *Timeout* field can be included. The *Session ID* corresponds to the session being resumed and is the value returned by the server in its response to the **CREATESESSION** command. The *Device Address* and *Nonce* are the values originally used in the **CREATESESSION** command. This is true even if the device address has changed or the session is being resumed over a different transport. Matching a session requires that all values match not just the *Session ID* (*Device Address*, *Nonce* and *Session ID* must match).

The successful response to **RESUMESESSION** is 0xA0 (Success, with the high bit set) in the response code. A **Session-Parameters** header containing *Device Address*, *Nonce*, *Session ID* and *Next Sequence Number* fields must be sent. A *Timeout* field is optional. If the *Device Address*, *Nonce* and *Session ID* do not correspond to a valid session then 0xD3 (Service Unavailable) should be returned. If the information returned by the server does not match then the client should disconnect from the server. **CLOSESESSION** should not be used since this could be a valid session for another device.

The **RESUMESESSION** command and response must fit in a single OBEX packet and have their Final bits set.

If the session is resumed successfully then the client must determine if a packet should be retransmitted. If the session was suspended gracefully then no packets need to be retransmitted otherwise the client must follow the algorithm below:

```
If the Next Sequence Number field equals the sequence number of the last
packet sent then {
    Re-Send last packet
}else if the Next Sequence Number field equals the sequence number of
the next packet to send then {
    If the response to the last packet was not received then {
        Re-Send last packet minus all headers except
        Session-Sequence-Number;
    } else {
        Send next packet;
    }
} else {
    /* Sequence number is invalid */
    Disconnect underlying transport connection to OBEX server;
}
```

The server sets the *Next-Sequence Number* field in the **RESUMESESSION** response to the sequence number of the next OBEX packet it expects to receive. The server must behave according to the algorithm below upon receipt of the first request packet from the client after resuming the session.

```
If the sequence number of the request is equal to the (Next Sequence
Number field minus one) mod 256 then {
    /* The client did not receive the previous response */
    Re-send the previous response;
    Ignore the contents of the request since it was received properly
    before the link was suspended;
} else if the sequence number of the request is equal to the Next
Sequence Number field then {
    /* This is the next packet */
    Process the packet;
    Send the proper response;
} else {
    /* The sequence number is invalid */
    Send an Abort;
}
```

SETTIMEOUT

The **SETTIMEOUT** command is used to negotiate a new timeout specifying the number of seconds a session should be maintained while the session is suspended. This command must include a **Session-Parameters** header containing the *Session Opcode* field. Optionally, a *Timeout* field can be included. If the **Session-Parameters** header does not contain the *Timeout* field, the request is for an infinite timeout. The successful response to **SETTIMEOUT** is 0xA0 (Success, with the high bit set) in the response code. A **Session-Parameters** field may be sent with a *Timeout* field. The lowest value for the timeout is the value used by both sides. This command can only be used to change the timeout of the active connection. If the active connection is the unreliable connection then 0xC3 (Forbidden) is the proper response code.

The **SETTIMEOUT** command and response must fit in a single OBEX packet and have their Final bits set.

Session Context

In order to resume a suspended session a context must be saved by both the client and the server. This context includes both session and OBEX information. The context contains the following items.

Item	Description
Session ID	This is the session ID created for this session. The session ID is generated from the information sent by both the client and server when the session was created so it does not actually have to be saved since it can be re-calculated.
Client Device Address	The device address sent by the client in the **CREATESESSION** command.
Client Nonce	The nonce sent by the client in the **CREATESESSION** command.
Server Device Address	The device address sent by the server in its response to the **CREATESESSION** command.
Server Nonce	The nonce sent by the server in its response to the **CREATESESSION** command.
Session Sequence Number	For the client this is the sequence number of the last packet sent. For the server this is the next sequence number expected.
Last OBEX packet	If the session was suspended unexpectedly because of loss of the underlying transport the last packet sent must be saved in case it needs to be transmitted. Both the client and the server must save the last packet sent. Also if the OBEX implementation buffers packets for the application then these must also be saved. It is possible to create an API between the OBEX layer and application such that the OBEX layer does not actually need to save the data. The API can allow the OBEX layer to ask the application to provide the data to be retransmitted.
Target Strings and Connection IDs	For all directed connections, which were created within the session, the target string and the corresponding Connection ID must be saved. Along with this is the need to save enough information to map the target back to the corresponding application. Also if the inbox connection has been created then this knowledge must also be saved.
OBEX Packet sizes	For each OBEX connection the maximum OBEX packet size must be saved.

Applications and services must also save context in order to resume a session. The OBEX layer does not know how commands such as PUT, GET, SETPATH, etc are used by the application/service so it cannot save context for them.

Persistence

If the OBEX layer and/or applications are shut down while a session is suspended then the context must be saved in a persistent data store if resuming the session later is desired.

Examples

Create Session Followed by a Directed Connection

Client Request:	bytes	Meaning
opcode	0x87	**SESSION**, Final bit set
	0x0015	packet length = 21
	0x52	**Session-Parameters** HI
	0x0012	Length of **Session-Parameters** header (18 bytes)

	0x05, 0x01,	Client *Session Opcode* field (1-byte)
	0x00	Create Session opcode
	0x00, 0x04,	Client *Device Address* field (IrDA size)
	0xXXXXXXXX	
	0x01, 0x04,	Client *Nonce* field (4 bytes)
	0xYYYYYYYY	
Server Response:		
response code	0xA0	SUCCESS, Final bit set
	0x0024	packet length of 36
	0x52	**Session-Parameters** HI
	0x0021	Length of **Session-Parameters** header (33 bytes)
	0x00, 0x04,	Server *Device Address* field (IrDA size)
	0xXXXXXXXX	
	0x01, 0x04,	Server *Nonce* field (4 bytes)
	0xYYYYYYYY	
	0x02, 0x10,	*Session ID* field (16 bytes)
	0xZZZZZZZZZZZZZZZZ	
	ZZZZZZZZZZZZZZZZ	
Client Request:		
opcode	0x80	**CONNECT**, Final bit set
	0x001C	packet length = 28
	0x10	version 1.0 of OBEX
	0x00	flags, all zero for this version of OBEX
	0x2000	8K is the max OBEX packet size client can accept
	0x93	**Session-Sequence-Number** HI
	0x00	First packet sent by Client has sequence number of 0
	0x46	**Target** HI
	0x0013	Length of **Target** Header
		UUID for desired service/application
	0x382D2BD03C3911D1A	{382D2BD0-3C39-11d1-AADC-0040F614953A}
	ADC0040F614953A	
Server Response:		
response code	0xA0	SUCCESS, Final bit set
	0x0021	packet length of 33
	0x10	version 1.0 of OBEX
	0x00	flags (LSAP-SEL multiplexing not supported)
	0x0800	2K max packet size
	0x93	**Session-Sequence-Number** HI
	0x01	Next OBEX packet expected by Server is 1
	0xCB	**Connection Id** HI
	0x00000001	ConnId = 1
	0x4A	**Who** HI
	0x0013	Length of **Who** Header
		UUID of responding application (same value as **Target**
	0x382D2BD03C3911D1A	header in request){382D2BD0-3C39-11d1-AADC-
	ADC0040F614953A	0040F614953A}

Suspend a Session

Note that **Session-Sequence-Number** headers are not used when sending Session commands in an active session.

Client Request:	**Bytes**	**Meaning**
opcode	0x87	**SESSION**, Final bit set
	0x0009	packet length = 9

	0x52	**Session-Parameters** HI
	0x0006	Length of **Session-Parameters** header (6 bytes)
	0x05, 0x01,	Client *Session Opcode* field (1-byte)
	0x02	Suspend Session opcode
Server Response:		
response code	0xA0	SUCCESS, Final bit set
	0x000C	packet length of 12
	0x52	**Session-Parameters** HI
	0x0009	Length of **Session-Parameters** header (9 bytes)
	0x04, 0x04,	*Timeout* Field
	0x000004B0	Requesting timeout of 20 minutes (1200 seconds)

Resume a Session

If a session was suspended using the SUSPENDSESSION command then resuming the session does not require retransmission of OBEX packets. If the session was suspended because the underlying transport was lost it might be necessary to retransmit packets. Three cases exist. The first case is when the last client request is lost. The second case is when the last server response is lost. The third case is when the client receives the last response but the server does not know if the client has received it (packets do not need to be retransmitted in this case). In the first case the client knows the server did not receive the request because the *Next-Sequence-Number* field in the response to the RESUMESESSION command indicates this, so the client will retransmit the command. In the second and third cases the server knows what to do based on the value of the *Session-Sequence-Number* field in the first packet sent by the client.

The example below shows the sequence needed to resume the session. Retransmission of packets is not shown.

Client Request:	**Bytes**	**Meaning**
Opcode	0x87	**SESSION**, Final bit set
	0x0027	packet length = 39
	0x52	**Session-Parameters** HI
	0x0024	Length of **Session-Parameters** header (36 bytes)
	0x05, 0x01,	Client *Session Opcode* field (1-byte)
	0x03	Resume Session opcode
	0x00, 0x04,	Client *Device Address* field (IrDA size)
	0xXXXXXXXX	
	0x01, 0x04,	Client *Nonce* field (4 bytes)
	0xYYYYYYYY	
	0x02, 0x10,	*Session ID* field (16 bytes)
	0xZZZZZZZZZZZZZZZZ	
	ZZZZZZZZZZZZZZZZ	
Server Response:		
response code	0xA0	SUCCESS, Final bit set
	0x0027	packet length of 39
	0x52	**Session-Parameters** HI
	0x0024	Length of **Session-Parameters** header (36 bytes)
	0x00, 0x04,	Server *Device Address* field (IrDA size)
	0xXXXXXXXX	
	0x01, 0x04,	Server *Nonce* field (4 bytes)
	0xYYYYYYYY	
	0x02, 0x10,	*Session ID* field (16 bytes)
	0xZZZZZZZZZZZZZZZZ	
	ZZZZZZZZZZZZZZZZ	
	0x03, 0x01,	*Next Sequence Number* field (1-byte).
	0x05	As an example, the server expects 0x05

Close a Session

Client Request:	**bytes**	**Meaning**
opcode	0x87	**SESSION**, Final bit set
	0x001B	packet length = 27
	0x52	**Session-Parameters** HI
	0x0018	Length of **Session-Parameters** header (24 bytes)
	0x05, 0x01,	Client *Session Opcode* field (1-byte)
	0x01	Close Session opcode
	0x02, 0x10,	*Session ID* field (16 bytes)
	0xZZZZZZZZZZZZZZZZ	
	ZZZZZZZZZZZZZZZZ	
Server Response:		
response code	0xA0	SUCCESS, Final bit set
	0x0003	packet length of 3

Negotiate a Timeout

Client Request:	bytes	Meaning
opcode	0x87	**SESSION**, Final bit set
	0x000F	packet length = 15
	0x52	**Session-Parameters** HI
	0x000C	Length of **Session-Parameters** header (12 bytes)
	0x05, 0x01,	Client *Session Opcode* field
	0x04	Set Timeout opcode
	0x04, 0x04,	*Timeout* field
	0x000004B0	Requesting timeout of 20 minutes (1200 seconds)
Server Response:		
response code	0xA0	SUCCESS, Final bit set
	0x000C	packet length of 12
	0x52	**Session-Parameters** HI
	0x0009	Length of **Session-Parameters** header (9 bytes)
	0x04, 0x04,	*Timeout* field
	0x00000384	Requesting timeout of 15 minutes (900 seconds)

Multiple Transport Support Proposal

This section describes a proposal for supporting sessions over different transports. This is not part of the errata but really just a method for capturing the idea. The actual method will be proposed as errata at a later date.

A possible method for supporting a session over different transports utilizes the OBEX Capability Object. The OBEX capability object must be accessible from any session via a connection to the Inbox service or the service that is accessed via the non-targeted OBEX connection. OBEX implementations that support sessions running over multiple transports will list the transport information in the Capability Object. This includes the name of the transport and the device address used for that transport (e.g. IrDA is the 32-bit device address, Bluetooth is the 48-bit address, IP is the IP address, etc).

The proposal is to add a new section to the Capability Object listing the transports in which OBEX currently runs over. These are transports that allow an OBEX reliable session to be resumed independent of the transport the session was originally created or suspended. The section contains a list of transports where each transport contains the following attributes:

Protocols	The pertinent list of protocols on which OBEX resides. Items in the list are separated by commas. The first item in the list should be the main physical/link layer in the protocol stack (e.g. IrDA, Bluetooth, USB, Ethernet, Serial, etc). Typically this is the only protocol required. Other protocols are listed if needed.
Addresses	The list of addresses or port numbers needed to access the OBEX layer. Items in the list are separated by commas. There should be an item in the **Address List** corresponding to each item in the **Protocol List**. The last address in **the Address List** is the value used as the *Device Address* field in Session commands.

Both attributes are required. Below is an example Transports section with two entries.

```
<!—Transports Section -->
<Transports>
    <Transport Protocols =”IrDA” Addresses=”442356AE” />
    <Transport Protocols=”Bluetooth” Addresses=”00F349125E01” />
</Transports>
```

The method for moving a session to a different transport is to first suspend the session then resume the session over the new transport.