3.3 OBEX Operations and Opcode definitions

OBEX operations consist of the following:

Opcode (w/high bit set)	Definition	Meaning
0x80 *high bit always set	Connect	choose your partner, negotiate capabilities
0x81 *high bit always set	Disconnect	signal the end of the session
0x02 (0x82)	Put	send an object
0x03 (0x83)	Get	get an object
0x04 (0x84)	Reserved	not to be used w/out extension to this specification
0x85 *high bit always set	SetPath	modifies the current path on the receiving side
0x06 (0x86)	Reserved	not to be used w/out extension to this specification
0x87 *high bit always set	Session	used for reliable session support
0xFF *high bit always set	Abort	abort the current operation
0x08 to 0x0F	Reserved	not to be used w/out extension to this specification
0x10 to 0x1F	User definable	use as you please with peer application
Bits 5 and 6 are reserved and should be set to zero.		
Bit 7 of the opcode means Final packet of request.		

The high bit of the opcode is used as a Final bit, described in the previous sections of this chapter. Bits 5 and 6 of the opcode are reserved for future use and should be set to zero by sender and ignored by receiver. However, one notable exemption from this rule is the **ABORT** opcode, which currently sets bits 5 and 6.

If a server receives an unrecognized opcode, it should return 0xD1 response code (Not Implemented, with Final bit set) and ignore the operation. It may send the request to the bit bucket, save it for later analysis, or whatever it chooses.

Each operation is described in detail in the following sections.

3.3.1 Connect

This operation initiates the connection and sets up the basic expectations of each side of the link. The request format is:

Byte 0	Bytes 1 and 2	Byte 3	Byte 4	Bytes 5 and 6	Byte 7 to n
0x80	connect packet length	OBEX version number	flags	maximum OBEX packet length	optional headers

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 **CONNECT** request and response must each fit in a single packet. Implementations are not *required* to recognize more than the first 7 bytes of these packets, though this may restrict their usage.

3.3.1.1 OBEX version number

The version number is the version of the OBEX protocol encoded with the major number in the high order 4 bits, and the minor version in the low order 4 bits. The protocol version is not always the same as the specification version. The current protocol version is 1.0. See the example later in this section.

3.3.1.2 Connect flags

The flags have the following meanings:

bit	Meaning
0	Response: Indicates support for multiple IrLMP connections to the same LSAP-SEL. Request: reserved
1	Reserved
2	Reserved
3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	Reserved

All request flags except bit 0 are currently reserved, and must be set to zero on the sending side and ignored by the receiving side. All reserved response bits must also be set to zero and ignored by the receiving side.

In a **CONNECT** Response packet bit 0 is used to indicate the ability of the OBEX server's transport to accept multiple IrLMP connections to the same LSAP-SEL. This capability, usually found in more robust IrDA stacks, allows the server to use the LM-MUX for multiplexing multiple simultaneous client requests. Conveying this information is necessary because the IrDA Lite specification does not support this type of multiplexing and attempts to connect to an already connected LSAP-SEL will result in an IrLAP disconnect.

Bit 0 should be used by the receiving client to decide how to multiplex operations to the server (should it desire to do so). If the bit is 0 the client should serialize the operations over a single TTP connection. If the bit is set the client is free to establish multiple TTP connections to the server and concurrently exchange its objects.

3.3.1.3 Maximum OBEX Packet Length

The maximum OBEX packet length is a two byte unsigned integer that indicates the maximum size OBEX packet that the device can receive. The largest acceptable value at this time is 64K bytes -1. However, even if a large packet size is negotiated, it is not required that large packets be sent - this just represents the maximum allowed by each participant. The client and server may have different maximum lengths.

This is one of the most important features of the **CONNECT** packet because it permits the application to increase the OBEX packet size used during an exchange. The default OBEX packet size of 255 bytes is inefficient for large object transfers and will impede transfer rates. Larger OBEX packet sizes such as 8k to 32k allow large amounts of data to be sent without acknowledgement. However, packet sizes should be intelligently limited on slower links to reduce abort request latency.

3.3.1.4 Minimum OBEX Packet Length

The minimum size of the OBEX Maximum packet length allowed for negotiation is 255 bytes. This is in order to provide a greater likelihood of meeting the requirement for single packet requests and responses in a broad range of cases. In addition, since an OBEX header must fit completely within one OBEX packet it is advantageous to mandate a minimum that allows for reasonable header sizes. Note that

OBEX does not exchange Minimum Packet Length values. This value is the minimum acceptable value for the OBEX Maximum Packet Length parameter exchanged in the **CONNECT** Operation.

3.3.1.5 Using Count and Length headers in Connect

The **Count** and **Length** headers, defined in the Object Model chapter, can be used in **CONNECT**. **Count** is used to indicate the number of objects that will be sent during this connection. **Length** is used to express the approximate total length of the bodies of all the objects in the transaction.

When used in the **CONNECT** Operation, the **Length** header contains the length in bytes of the bodies of all the objects that the sender plans to send. Note that this length cannot be guaranteed correct, so while the value may be useful for status indicators and resource reservations, the receiver should not die if the length is not exact. The receiver can depend on **Body** headers to accurately indicate the size of an object as it is delivered and the **End-of-Body** header to indicate when an object is complete.

3.3.1.6 Using Who and Target headers in Connect

Target and **Who** are used to hold a unique identifier, which allows applications to tell whether they are talking to a strict peer, or not. Typically, this is used to enable additional capabilities supplied only by an exact peer. If a **Who** header is used, it should be sent before any body headers.

On full-featured [PC] platforms, multiple OBEX applications may exist concurrently. This leads to the need for the client to be able to uniquely identify which server it wants to handle its request. The server is therefore identified with the OBEX **Target** header. If necessary, the client can also identify itself, using the OBEX **Who** header. The following text describes the exact uses of these headers.

To target a specific application with OBEX commands the client must set-up a connection to the application by using the OBEX **Target** header in a **CONNECT** request. This type of connection is called a directed connection and provides a virtual binding between the client and server. The **Target** header should specify the UUID of the desired application. The **Who** header can also be used when it is necessary to identify the client initiating the exchange. The **Who** header should be used in cases where the target server application supports different client applications and may care which one it is connecting to. It is unnecessary to send a **Who** header in the request if its only logical value is the same as the **Target** header.

The response to the targeted connect operation should contain a **Who** with the same UUID as sent in the request's matching **Target** header. If the **Who** header was present in the request, a **Target** header identifying the same client should be sent in the response. In addition, a unique connection identifier must be sent in a **Connection Id** header. This connection identifier is used by the client in all future operations. If the response does not contain the correct headers then it should be assumed that the connection has not been made to the specific application but to the inbox service. This will be the response when sending a directed connect to a system that does not parse these (Target) headers. In the event that a connection is made to the inbox service, it is the responsibility of the client application to determine whether to continue the exchange or disconnect.

3.3.1.7 Using Description headers in Connect

The **CONNECT** request or response may include a **Description** header with information about the device or service. It is recommended this information be presented through the user interface on the receiving side if possible.

3.3.1.8 The Connect response

The successful response to **CONNECT** is 0xA0 (Success, with the high bit set) in the response code, followed immediately by the required fields described above, and optionally by other OBEX headers as defined above. Any other response code indicates a failure to make an OBEX connection. A fail response

still includes the version, flags, and packet size information, and may include a **Description** header to expand on the meaning of the response code value.

3.3.1.9 Example

The following example shows a **CONNECT** request and response with comments explaining each component. The **CONNECT** sends two optional headers describing the number of objects and total length of the proposed transfer during the connection.

Client Request:	**bytes**	**Meaning**
Opcode	0x80	**CONNECT**, Final bit set
	0x0011	packet length = 17
	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
	0xC0	HI for **Count** header (optional header)
	0x00000004	four objects being sent
	0xC3	HI for **Length** header (optional header)
	0x0000F483	total length of hex F483 bytes
Server Response:		
response code	0xA0	SUCCESS, Final bit set
	0x0007	packet length of 7
	0x10	version 1.0 of OBEX
	0x00	Flags
	0x0400	1K max packet size

3.3.1.10 OBEX Operations without Connect

It is highly recommended that implementations assume default values for connection parameters (currently just a minimum OBEX packet size of 255 bytes) and accept operations such as **PUT** and **GET** without first requiring a **CONNECT** operation.

3.3.2 Disconnect

This opcode signals the end of the OBEX session. It may include a **Description** header for additional user readable information. The **DISCONNECT** request and response must each fit in one OBEX packet and have their Final bits set.

Byte 0	Bytes 1, 2	Bytes 3 to n
0x81	packet length	optional headers

The response to **DISCONNECT** is 0xA0 (Success), optionally followed with a **Description** header. A **DISCONNECT** may not be refused. However, if the disconnect packet contains invalid information, such as an invalid **Connection Id** header the response code may be “Service Unavailable” (0xD3). Server side handling of this case is not required.

Byte 0	Bytes 1, 2	Bytes 3 to n
0xA0 or 0xD3	response packet length	optional response headers

It is permissible for a connection to be terminated by closing the transport connection without issuing the OBEX **DISCONNECT** operation. Though, this precludes any opportunity to include descriptive information about the disconnection. Currently, this is common practice and **DISCONNECT** is infrequently used. However, it is good practice to send an OBEX **DISCONNECT** for each OBEX **CONNECT** sent but few applications track or care about such details.

3.3.3 Put

The **PUT** operation sends one object from the client to the server. The request will normally have at least the following headers: **Name** and **Length**. For files or other objects that may have several dated versions, the **Date/Time** header is also recommended, while the **Type** is very desirable for non-file object types. However, any of these may be omitted if the receiver can be expected to know the information by some other means. For instance, if the target device is so simple that it accepts only one object and prevents connections from unsuitable parties, all the headers may be omitted without harm. However, if a PC, PDA, or any other general-purpose device is the intended recipient, the headers are highly recommended.

A **PUT** request consists of one or more request packets, the last of which has the Final bit set in the opcode. The implementer may choose whether to include an object **Body** header in the initial packet, or wait until the response to a subsequent packet is received before sending any object body chunks.

Byte 0	Bytes 1, 2	Bytes 3 to n
0x02 (0x82 when Final bit set)	packet length	sequence of headers

Each packet is acknowledged by a response from the server as described in the general session model discussion above.

Byte 0	Bytes 1,2	Bytes 3 to n
Response code typical values: 0x90 for Continue 0xA0 for Success	Response packet length	optional response headers

3.3.3.1 Headers used in Put

Any of the headers defined in the Object model chapter can be used with **PUT**. These might include **Name**, **Type**, **Description**, **Length**, **Connection Id, HTTP** or other headers specifying compression, languages, character sets, and so on. It is strongly recommended that headers describing the object body precede the object **Body** headers for efficient handling on the receive side. If **Name** or **Type** headers are used, they must precede all object **Body** headers.

3.3.3.2 Put Response

The response for successfully received intermediate packets (request packets without the Final bit set) is 0x90 (Continue, with Final bit sent). The successful final response is 0xA0 (Success, with Final bit set). The response to any individual request packet must itself consist of just one packet with its Final bit set - multi-packet responses to **PUT** are not permitted.

Any other response code indicates failure. If the length field of the response is > 3 (the length of the response code and length bytes themselves), the response includes headers, such as a **Description** header to expand on the meaning of the response code value.

Here is a typical Final response:

Server Response:	**Bytes**	**Meaning**
response code	0xA0 0x0003	SUCCESS, Final bit set length of response packet

3.3.3.3 Put Example

For example, here is a **PUT** operation broken out with each component (opcode or header) on a separate line. We are sending a file called jumar.txt, and for ease of reading, the example file is 4K in length and is sent in 1K chunks.

Client Request:	**Bytes**	**Meaning**
opcode	0x02	**PUT**, Final bit not set
	0x0422	1058 bytes is length of packet
	0x01	HI for **Name** header
	0x0017	Length of **Name** header (Unicode is 2 bytes per char)
	JUMAR.TXT	name of object, null terminated Unicode
	0xC3	HI for **Length** header
	0x00001000	Length of object is 4K bytes
	0x48	HI for Object **Body** chunk header
	0x0403	Length of **Body** header (1K) plus HI and header length
	0x.....	1K 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
	0x0406	1030 bytes is length of packet
	0x48	HI for Object **Body** chunk
	0x0403	Length of **Body** header (1K) plus HI and header length
	0x.....	next 1K bytes of body
Server Response:		
response code	0x90	CONTINUE, Final bit set
	0x0003	length of response packet

Another packet containing the next chunk of body is sent, and finally we arrive at the last packet, which has the Final bit set.

Client Request:		
opcode	0x82	**PUT**, Final bit set
	0x0406	1030 bytes is length of packet
	0x49	HI for **End-of-Body** chunk
	0x0403	Length of header (1K) plus HI and header length
	0x.....	last 1K bytes of body
Server Response:		
response code	0xA0	SUCCESS, Final bit sent
	0x0003	length of response packet

3.3.3.4 Server side handling of Put objects

Servers may do whatever they wish with an incoming object - the OBEX protocol does not require any particular treatment. The client may “suggest” a treatment for the object through the use of the **Target** and **Type** Headers, but this is not binding on the server. Some devices may wish to control the path at

which the object is stored (i.e. specify folder information such as C:\bin\pizza.txt rather than just pizza.txt). Path information is transferred using the **SETPATH** operation, but again, this is not binding on the server. It is important that clients, who have a particular purpose in mind when transferring an object, connect to a specific service that it knows can perform the desired behavior.

3.3.3.5 Sending objects that may have read errors

When sending the last portion of an object in an **End-of-Body** header, ambiguity arises if there is any chance that there are read errors in this last portion. This is because the **End-of-Body** normally triggers the receiving side to close the received object and indicate successful completion. If an **ABORT** packet subsequently arrives, it is "too late".

The recommended approach when sending objects which may have such errors is to send the **End-of-Body** header only when the sender knows that the entire object has been safely read, even if this means sending an empty **End-of-Body** header at the end of the object. This applies to both **GET** and **PUT** operations.

3.3.3.6 Put-Delete and Create-Empty Methods

A **PUT** operation with NO **Body** or **End-of-Body** headers whatsoever should be treated as a *delete* request. Similarly, a **PUT** operation with an empty **End-of-Body** header requests the recipient to create an *empty* object. This definition may not make sense or apply to every implementation (in other words devices are not required to support delete operations or empty objects),

A folder can be deleted using the same procedure used to delete a file. Deleting a non-empty folder will delete all its contents, including other folders. Some servers may not allow this operation and will return the "Precondition Failed" (0xCC) response code, indicating that the folder is not empty. In this case, the client will need to delete the contents before deleting the folder.

3.3.4 Get

The **GET** operation requests that the server return an object to the client. The request is normally formatted as follows:

Byte 0	Bytes 1, 2	Bytes 3 to n
0x03	Packet length	sequence of headers

The **Name** header can be omitted if the server knows what to deliver, as with a simple device that has only one object (e.g. a maintenance record for a machine). If the server has more than one object that fits the request, the behavior is system dependent, but it is recommended that the server return Success with the "default object" which should contain information of general interest about the device.

The final bit is used in a **GET** request to identify the last packet containing headers describing the item being requested, and the request phase of the **GET** is complete. The server device must not begin sending the object body chunks until the request phase is complete. Once a **GET** is sent with the final bit, all subsequent **GET** request packets must set the final bit until the operation is complete.

A successful response for an object that fits entirely in one response packet is 0xA0 (Success, with Final bit set) in the response code, followed by the object body. If the response is large enough to require multiple **GET** requests, only the last response is 0xA0, and the others are all 0x90 (Continue). The object is returned as a sequence of headers just as with **PUT**. Any other response code indicates failure. Common non-success responses include 0xC0 bad request, and 0xC3 forbidden. The response may include a **Description** header (before the returned object, if any) to expand on the meaning inherent in the response code value.

Byte 0	Bytes 1,2	Bytes 3 to n