OBEX Errata

Following are a list of corrections/clarifications and suggested amendments or changes to the OBEX specification Version 1.3. Relevant section numbers for the OBEX v1.3 specification are shown in parenthesis next to the heading.

The points are classified according to the following scheme:

- CORRECTION: a change required to correct an error in the existing document. Corrections may change the specified behavior of the protocol to match that which was originally intended by the authors
- CLARIFICATION: textual enhancement that provides clearer explanation of a protocol element without changing any behavior
- MODIFICATION: a modification of the currently specified behavior that adds but does not delete any functionality from the protocol
- CHANGE: a modification of the currently specified protocol that may add and/or delete functionality from the protocol
- PROBLEM: a known problem for which an alteration to the document has yet to be proposed.

1 APPROVED ERRATA FOR APRIL 2003

1.1 Action operation for Copy, Move/Rename and Modifying Permissions (2.1, 2.2 & 3.3)

MODIFICATION

2.1 OBEX Headers

HI - identifier	Header name	Description
0x94	Action Id	Specifies the action to be performed (used in ACTION operation)
0x15	DestName	The destination object name (used in certain ACTION operations)
0xD6	Permissions	4 byte bit mask for setting permissions
0x17 to 0x2F	Reserved for future use	This range includes all combinations of the upper 2 bits

2.2.20 Action Identifier

Action Id is a 1-byte quantity for specifying the action of the **ACTION** operation. The **Action Id** header contains an Action Identifier, which defines what action is to be performed. The **Action Id** header is mandatory for the ACTION operation and restricted for any other OBEX operations (see Section 3.3.8 Action). When in use, the **Action Id** header must be the first header in the request unless the **Connection Id** header is in use, in which case the **Action Id** header must immediately follow the **Connection Id** header (see section 2.2.11 Connection Identifier).

Action Identifier	Action Name	Description
0x00	Copy Object	(See Section 3.3.8.1 Copy Object Action)

0x01	Move/Rename Object	(See Section 3.3.8.2 Move/Rename Object Action)
0x02	Set Object Permissions	(See Section 3.3.8.3 Set Object Permissions Action)
0x03 – 7F	Reserved for future use	Reserved for assignment by OBEX.
0x80 – FF	Reserved for use in vendor extensions	An area of the Action Identifier space, which is reserved for use in vendor specific applications.

2.2.21 DestName

DestName is a null terminated Unicode text string describing the destination name of the object. The **DestName** header is used with different **ACTION** operations and its usage is action specific (see for more details sections 3.3.8.1 Copy Object Action and 3.3.8.2 Move/Rename Object Action). It MUST NOT be used with the **PUT** and **GET** operations.

The schema used in the **DestName** can be either the same as the standard FTP (File Transfer Protocol) uses or UNC (Universal Naming Convention).

In the "FTP style" the object names can be relative to the current folder or absolute pathnames. The backslash "\"or slash "/" character must be used to indicate the folder hierarchy within the **DestName** headers i.e. "\blah\blahsubdir\file.txt", when referring locations outside the current folder.

Examples of valid **DestName** values using "FTP style":

Text.txt (relative to the current folder)
MyStuff\Text.txt (relative to the current folder)
MyStuff/Text.txt (relative to the current folder)
c:\MyStuff\Texts\Text.txt (absolute)
c:/MyStuff/Texts/Text.txt (absolute)
\MyStuff\Texts\Text.txt (relative to the root folder)
/MyStuff/Texts/Text.txt (relative to the root folder)

If UNC path names are used a full path name must be given starting with "\\". Backslash "\" is used to delimit path.

Example of valid UNC path name:

\\server1\storage1\MyStuff\Text.txt

2.2.22 Permissions

Permissions is a 4-byte unsigned integer where the 4 bytes describe bit masks representing the various permission values. It is used for setting "Read", "Write", "Delete" and "Modify Permissions" permissions for files and folders. The permissions are applied to three different permission levels, which are "User", "Group" and "Other". The **Permissions** header can be used with different **ACTION** operations or with the **PUT**, **GET** and **SETPATH** operations. In the case of a **PUT** and **SETPATH**, the object permissions should be set according to the **Permissions** header only when a new file/folder is created. When changing the permissions of an already existing file/folder the **ACTION** operation should be used (see section 3.3.8.3 Set Object Permissions Action). On a **GET**, the OBEX Server may return the permissions of the object in this header.

Byte 0	Byte 1	Byte 2	Byte 3
Reserved (Should be set to zero)	User permissions	Group permissions	Other permissions

Within the context of OBEX, a "User" refers to permissions for the OBEX Client's current authenticated user (see section 3.5 Authentication Procedure). If no user is authenticated, permissions are relative to a "guest" user. "Group" refers to the group or groups to which the OBEX Server has assigned the current authenticated user. "Other" refers to permissions for a user who is neither the current user nor the member of any of the user's groups.

Permissions flags

The bits in each permissions byte have the following meanings:

bit	Meaning
0	Read. When this bit is set to 1, reading permission is granted. For a file object, the OBEX Client may use ACTION/Copy or GET on the file. For a folder object, the OBEX Client may use ACTION/Copy or SETPATH on the folder. To do ACTION/Move client needs to have both "Read" and "Delete" permission to the source file/folder.
1	Write. When this bit is set to 1, writing permission is granted. For a file object, the OBEX Client may use PUT on the file. To do ACTION/Move client needs to have both "Read" and "Delete" permission to the source file/folder.
2	Delete. When this bit is set to 1, deletion permission is granted. For a file/folder object, the OBEX Client may use PUT-delete.
3	Reserved for future use
4	Reserved for future use
5	Reserved for future use
6	Reserved for future use
7	Modify Permissions. When this bit is set to 1 the file access permissions can be changed. For a file/folder object, the OBEX Client may use ACTION/Set Object Permissions

For devices that do not discriminate between permission groups, the default is "User" permissions. That is, the "Group" permissions byte should match the "User" permissions byte.

3.3 OBEX Operations and Opcode definitions

Opcode (w/high bit set)	Definition	Meaning
0x06 (0x86)	Action	common actions to be performed by the target
0x08to 0x0F	Reserved for future use	not to be used w/out extension to this specification

3.3.8 Action

The ACTION operation is defined to cover the needs of common actions. The **Action Id** header is used in the **ACTION** operation and contains an action identifier, which defines what action is to be performed. All actions are optional and devices may chose to implement all, some or none of the actions.

The **Action Id** header is described in section 2.2.20 Action Identifier.

Byte 0	Bytes 1, 2	Bytes 3,4	Bytes 5 to n
0x06 (0x86)	Packet length	Action Identifier Header	Sequence of headers

The positive responses to an **ACTION** request are either SUCCESS or CONTINUE depending on the presence of the FINAL bit in the request. Any failure response code can be used to indicate a negative response.

3.3.8.1 Copy Object Action

This action copies an object from one location to another. The **Name** header specifies the source file name and the **DestName** header specifies the destination file name. These two headers are mandatory for this action.

The Action Identifier for the Copy Object Action is specified in section 2.2.20 Action Identifier.

Byte 0	Bytes 1, 2	Bytes 3, 4		Bytes 5 to n			Bytes n to m
Opcode	Packet length	Action Identifier Header for Copy		Name Header (Source Filename)	DestName Header (Destination Filename)		Optional headers
0x06 (0x86)		0x94	0x00		0x15		

Response Codes:

- Success 0x20 (0xA0)
- Not Found 0x44 (0xC4) – source object or destination folder does not exist
- Forbidden 0x43 (0xC3) – cannot read source object or create object in destination folder, permission denied
- Database Full 0x60 (0xE0) - cannot create object in destination folder, out of memory
- Conflict 0x49 (0xC9) - cannot create object in destination folder, sharing violation, object or Copy Object Action reserved/busy
- Not Implemented 0x51 (0xD1) – Copy Object Action not supported
- Not modified 0x34 (0xB4) - cannot create folder/file, destination folder/file already exits

Comment: Clearer usage of error codes helps the UI design.

If the **Permissions** header is used with the Copy Object Action, then the permissions of the destination object should be set as specified in the **Permissions** header. If the header is not used, the permissions should be set to be the same as the source.

3.3.8.2 Move/Rename Object Action

This action moves an object from one location to another. It can also be used to rename an object. The **Name** header specifies the source file name and the **DestName** header specifies the destination file name. These two headers are mandatory for this action.

The Action Identifier for the Move/Rename Object Action is specified in section 2.2.20 Action Identifier.

Byte 0	Bytes 1, 2	Bytes 3, 4		Bytes 5 to n			Bytes n to m
Opcode	Packet length	Action Identifier header for Move/Rename		Name header (Source filename)	DestName header (Destination filename)		Optional headers
0x06 (0x86)		0x94	0x01		0x15		

Response Codes:

- Success 0x20 (0xA0)
- Not Found 0x44 (0xC4) – source object or destination folder does not exist
- Forbidden 0x43 (0xC3) – cannot read source object or create object in destination folder, permission denied
- Database Full 0x60 (0xE0) - cannot create object in destination folder, out of memory
- Conflict 0x49 (0xC9) - cannot create object in destination folder, sharing violation, object or Move/Rename Object Action busy
- Not Implemented 0x51 (0xD1) - Move/Rename Object Action not supported
- Not modified 0x34 (0xB4) - cannot create folder/file, destination folder/file already exits

If the **Permissions** header is used with the Move/Rename Object Action, then the permissions of the destination object should be set as specified in the **Permissions** header. If the header is not used, the permissions should be set to be the same as the source.

3.3.8.3 Set Object Permissions Action

This action sets the access permissions of an object or folder. The **Name** header specifies the object and the **Permissions** header specifies the new permissions for this object. These two headers are mandatory for this action.. When using the Set Object Permissions Action for folders, it will set the permissions only for the folder; it doesn't affect the permissions of the contents of the folder.

The Action Identifier for the Set Object Permissions Action is specified in section 2.2.20 Action Identifier.

Byte 0	Bytes 1, 2	Bytes 3, 4		Bytes 5 to n			Bytes n to m
Opcode	Packet length	Action Identifier header for Set Object Permissions		Name header	Permissions header		Optional headers
0x06 (0x86)		0x94	0x02		0xD6		

Response Codes:

- Success 0x20 (0xA0)
- Not Found 0x44 (0xC4) – source object or destination folder does not exist
- Forbidden 0x43 (0xC3) – cannot modify the permissions of the destination object/folder, permission denied
- Not Implemented 0x51 (0xD1) – Set Object Permissions Action not supported
- Conflict 0x49 (0xC9) - cannot modify the permissions, sharing violation, object or Set Object Permissions Action busy

3.3.8.4 Common Issues For Move/Rename, Copy and Set Object Permissions Actions

The operations and actions on files within a file structure are complicated when some of the files are protected. Ideally the command should succeed or fail completely. Unfortunately, some devices may not be able to restore deleted files if an error occurs in a multi-file delete operation. The following defines what should happen in the event that the entire operation cannot be completed or failed as a single operation.

When a **PUT** operation is used to delete a folder, all of the subfolders beneath it should be deleted. If a file or folder within the structure is protected (deleting forbidden), then the file/folder itself and the folders in the tree above the file/folder should not be deleted. All of the other files and folders in the subfolders that are not protected should be deleted. The error response FORBIDDEN should be returned in this case.

A Move Action of a folder should also move the contents of all the subfolders beneath it. If a file or folder within the structure is protected (deleting forbidden), then the file/folder itself and the folders in the tree above the file/folder should not be deleted. Instead, new versions of the protected files/folders should be created in the destination folder. All of the other files in the folders that are not protected should be moved. The error response FORBIDDEN should be returned in this case.

A Copy Action of a folder should also copy the contents of all the subfolders beneath it. If a file or folder within the structure is protected (reading forbidden), then the folder itself and the files/folders beneath it will not be copied. The same principle applies to Move action when reading of the source file/folder is forbidden. The error response FORBIDDEN should be returned.

When a folder is moved or copied, the current folder is not changed, even if this folder is no longer valid. For example if you are in the folder “\blah” and you move the folder “\blah” to “\blah2”, the current folder is still “\blah”.

Files and folders at the same folder level MUST not have the same name, i.e. it must be obvious by the name that they are different, and not implicitly known by the operating system of the file store.

The client must preserve case between the folder listings returned and the names sent in the Name and **DestName** headers, as the remote file system may be case sensitive.