Package
javax.obex

Class Summary	
Interfaces	
`Authenticator`	This interface provides a way to respond to authentication challenge and authentication response headers.
`ClientSession`	The `ClientSession` interface provides methods for OBEX requests.
`HeaderSet`	The `HeaderSet` interface defines the methods that set and get the values of OBEX headers.
`Operation`	The `Operation` interface provides ways to manipulate a single OBEX PUT or GET operation.
`SessionNotifier`	The `SessionNotifier` interface defines a connection notifier for server-side OBEX connections.
Classes	
`PasswordAuthentication`	This class holds user name and password combinations.
`ResponseCodes`	The `ResponseCodes` class contains the list of valid response codes a server may send to a client.
`ServerRequestHandler`	The `ServerRequestHandler` class defines an event listener that will respond to OBEX requests made to the server.

javax.obex
Authenticator

Declaration

```
public interface Authenticator
```

Description

This interface provides a way to respond to authentication challenge and authentication response headers. When a client or server receives an authentication challenge or authentication response header, the `onAuthenticationChallenge()` or `onAuthenticationResponse()` will be called, respectively, by the implementation.

For more information on how the authentication procedure works in OBEX, please review the IrOBEX specification at http://www.irda.org.

Authentication Challenges

When a client or server receives an authentication challenge header, the `onAuthenticationChallenge()` method will be invoked by the OBEX API implementation. The application will then return the user name (if needed) and password via a `PasswordAuthentication` object. The password in this object is not sent in the authentication response. Instead, the 16-byte challenge received in the authentication challenge is combined with the password returned from the `onAuthenticationChallenge()` method and passed through the MD5 hash algorithm. The resulting value is sent in the authentication response along with the user name if it was provided.

Authentication Responses

When a client or server receives an authentication response header, the `onAuthenticationResponse()` method is invoked by the API implementation with the user name received in the authentication response header. (The user name will be `null` if no user name was provided in the authentication response header.) The application must determine the correct password. This value should be returned from the `onAuthenticationResponse()` method. If the authentication request should fail without the implementation checking the password, `null` should be returned by the application. (This is needed for reasons like not recognizing the user name, etc.) If the returned value is not `null`, the OBEX API implementation will combine the password returned from the `onAuthenticationResponse()` method and challenge sent via the authentication challenge, apply the MD5 hash algorithm, and compare the result to the response hash received in the authentication response header. If the values are not equal, an `IOException` will be thrown if the client requested authentication. If the server requested authentication, the `onAuthenticationFailure()` method will be called on the `ServerRequestHandler` that failed authentication. The connection is **not** closed if authentication failed.

Member Summary

Methods

public PasswordAuthentication	onAuthenticationChallenge(String, boolean, boolean) Called when a client or a server receives an authentication challenge header.
public byte	onAuthenticationResponse(byte[]) Called when a client or server receives an authentication response header.

Methods

onAuthenticationChallenge(String, boolean, boolean)

```
public PasswordAuthentication onAuthenticationChallenge(java.lang.String description,
            boolean isUserIdRequired, boolean isFullAccess)
```

Called when a client or a server receives an authentication challenge header. It should respond to the challenge with a `PasswordAuthentication` that contains the correct user name and password for the challenge.

Parameters:

`description` - the description of which user name and password should be used; if no description is provided in the authentication challenge or the description is encoded in an encoding scheme that is not supported, an empty string will be provided

`isUserIdRequired` - `true` if the user ID is required; `false` if the user ID is not required

`isFullAccess` - `true` if full access to the server will be granted; `false` if read only access will be granted

Returns: a `PasswordAuthentication` object containing the user name and password used for authentication

onAuthenticationResponse(byte[])

```
public byte[] onAuthenticationResponse(byte[] userName)
```

Called when a client or server receives an authentication response header. This method will provide the user name and expect the correct password to be returned.

Parameters:

`userName` - the user name provided in the authentication response; may be `null`

Returns: the correct password for the user name provided; if `null` is returned then the authentication request failed

javax.obex
ClientSession

Declaration

```
public interface ClientSession extends javax.microedition.io.Connection
```

All Superinterfaces: javax.microedition.io.Connection

Description

The `ClientSession` interface provides methods for OBEX requests. This interface provides a way to define headers for any OBEX operation. OBEX operations are CONNECT, SETPATH, PUT, GET and DISCONNECT. For PUTs and GETs, this interface will return a `javax.obex.Operation` object to complete the operations. For CONNECT, DISCONNECT, and SETPATH operations, this interface will complete the operation and return the result in a `HeaderSet` object.

Connection ID and Target Headers

According to the IrOBEX specification, a packet may not contain a Connection ID and Target header. Since the Connection ID header is managed by the implementation, it will not send a Connection ID header if a Connection ID was specified in a packet that has a Target header. In other words, if an application adds a Target header to a `HeaderSet` object used in an OBEX operation and a Connection ID was specified, no Connection ID will be sent in the packet containing the Target header.

CREATE-EMPTY and PUT-DELETE Requests

To perform a CREATE-EMPTY request, the client must call the `put()` method. With the `Operation` object returned, the client must open the output stream by calling `openOutputStream()` and then close the stream by calling `close()` on the `OutputStream` without writing any data. Using the `DataOutputStream` returned from `openDataOutputStream()` works the same way.

There are two ways to perform a PUT-DELETE request. The `delete()` method is one way to perform a PUT-DELETE request. The second way to perform a PUT-DELETE request is by calling `put()` and never calling `openOutputStream()` or `openDataOutputStream()` on the `Operation` object returned from `put()`.

PUT example

```
    void putObjectViaOBEX(ClientSession conn, HeaderSet head, byte[] obj)
        throws IOException {

        // Include the length header
        head.setHeader(HeaderSet.LENGTH, new Long(obj.length));

        // Initiate the PUT request
        Operation op = conn.put(head);

        // Open the output stream to put the object to it
        OutputStream out = op.openOutputStream();

        // Send the object to the server
        out.write(obj);

        // End the transaction
        out.close();
        op.close();
    }
```

GET example

```
byte[] getObjectViaOBEX(ClientSession conn, HeaderSet head) throws IOException {

    // Send the initial GET request to the server
    Operation op = conn.get(head);

    // Get the object from the input stream
    InputStream in = op.openInputStream();
    ByteArrayOutputStream out = new ByteArrayOutputStream();

    int data = in.read();
    while (data != -1) {
        out.write((byte)data);
        data = in.read();
    }

    // End the transaction
    in.close();
    op.close();

    byte[] obj = out.toByteArray();
    out.close();

    return obj;
}
```

Member Summary

Methods

public HeaderSet	`connect(HeaderSet)` Completes an OBEX CONNECT operation.
public HeaderSet	`createHeaderSet()` Creates a `javax.obex.HeaderSet` object.
public HeaderSet	`delete(HeaderSet)` Performs an OBEX DELETE operation.
public HeaderSet	`disconnect(HeaderSet)` Completes an OBEX DISCONNECT operation.
public Operation	`get(HeaderSet)` Performs an OBEX GET operation.
public long	`getConnectionID()` Retrieves the connection ID that is being used in the present connection.
public Operation	`put(HeaderSet)` Performs an OBEX PUT operation.
public void	`setAuthenticator(Authenticator)` Sets the `Authenticator` to use with this connection.
public void	`setConnectionID(long)` Sets the connection ID header to include in the request packets.
public HeaderSet	`setPath(HeaderSet, boolean, boolean)` Completes an OBEX SETPATH operation.

Inherited Member Summary

Methods inherited from interface javax.microedition.io.Connection

Inherited Member Summary
close

Methods

connect(HeaderSet)

```
public HeaderSet connect(HeaderSet headers)
            throws IOException
```

Completes an OBEX CONNECT operation. If the `headers` argument is `null`, no headers will be sent in the request. This method will never return `null`.

This method must be called and a successful response code of `OBEX_HTTP_OK` must be received before `put()`, `get()`, `setPath()`, `delete()`, or `disconnect()` may be called. Similarly, after a successful call to `disconnect()`, this method must be called before calling `put()`, `get()`, `setPath()`, `delete()`, or `disconnect()`.

Parameters:

`headers` - the headers to send in the CONNECT request

Returns: the headers that were returned from the server

Throws:

`IOException` - if an error occurred in the transport layer; if the client is already in an operation; if this method had already been called with a successful response code of `OBEX_HTTP_OK` and calls to `disconnect()` have not returned a response code of `OBEX_HTTP_OK`; if the headers defined in `headers` exceed the max packet length

`IllegalArgumentException` - if `headers` was not created by a call to `createHeaderSet()`

createHeaderSet()

```
public HeaderSet createHeaderSet()
```

Creates a `javax.obex.HeaderSet` object. This object can be used to define header values in a request.

Returns: a new `javax.obex.HeaderSet` object

See Also: `HeaderSet`

delete(HeaderSet)

```
public HeaderSet delete(HeaderSet headers)
            throws IOException
```

Performs an OBEX DELETE operation. This method will never return `null`.

Parameters:

`headers` - the header to send in the DELETE request

Returns: the headers returned by the server

Throws:

`IOException` - if an error occurred in the transport layer; if the client is already in an operation; if an OBEX connection does not exist because `connect()` has not been called; if `disconnect()` had been called and a response code of `OBEX_HTTP_OK` was received; if the headers defined in `headers` exceed the max packet length

`IllegalArgumentException` - if `headers` were not created by a call to `createHeaderSet()`

disconnect(HeaderSet)

```
public HeaderSet disconnect(HeaderSet headers)
            throws IOException
```

Completes an OBEX DISCONNECT operation. If the `headers` argument is `null`, no headers will be sent in the request. This method will end the session. A new session may be started by calling `connect()`. This method will never return `null`.

Parameters:

`headers` - the header to send in the DISCONNECT request

Returns: the headers returned by the server

Throws:

`IOException` - if an error occurred in the transport layer; if the client is already in an operation; if an OBEX connection does not exist because `connect()` has not been called; if `disconnect()` has been called and received a response code of `OBEX_HTTP_OK` after the last call to `connect()`; if the headers defined in `headers` exceed the max packet length

`IllegalArgumentException` - if `headers` were not created by a call to `createHeaderSet()`

get(HeaderSet)

```
public Operation get(HeaderSet headers)
            throws IOException
```

Performs an OBEX GET operation. This method will send the OBEX headers provided to the server and return an `Operation` object to continue with the operation. This method will never return `null`.

Parameters:

`headers` - the OBEX headers to send as part of the initial GET request

Returns: the OBEX operation that will complete the GET request

Throws:

`IOException` - if an error occurred in the transport layer; if an OBEX connection does not exist because `connect()` has not been called; if `disconnect()` had been called and a response code of `OBEX_HTTP_OK` was received; if `connect()` has not been called; if the client is already in an operation;

`IllegalArgumentException` - if `headers` were not created by a call to `createHeaderSet()`

See Also: `Operation`

getConnectionID()

```
public long getConnectionID()
```

Retrieves the connection ID that is being used in the present connection. This method will return -1 if no connection ID is being used.

Returns: the connection ID being used or -1 if no connection ID is being used

put(HeaderSet)

```
public Operation put(HeaderSet headers)
            throws IOException
```

Performs an OBEX PUT operation. This method will send the OBEX headers provided to the server and return an `Operation` object to continue with the PUT operation. This method will never return `null`.

Parameters:

`headers` - the OBEX headers to send in the initial PUT request

Returns: the operation object used to complete the PUT request

Throws:

`IOException` - if an error occurred in the transport layer; if an OBEX connection does not exist because `connect()` has not been called; if `disconnect()` had been called and a response code of `OBEX_HTTP_OK` was received; if `connect()` has not been called; if the client is already in an operation;

`IllegalArgumentException` - if `headers` were not created by a call to `createHeaderSet()`

See Also: `Operation`

setAuthenticator(Authenticator)

```
public void setAuthenticator(Authenticator auth)
```

Sets the `Authenticator` to use with this connection. The `Authenticator` allows an application to respond to authentication challenge and authentication response headers. If no `Authenticator` is set, the response to an authentication challenge or authentication response header is implementation dependent.

Parameters:

`auth` - the `Authenticator` to use for this connection

Throws:

`NullPointerException` - if `auth` is `null`

setConnectionID(long)

```
public void setConnectionID(long id)
```

Sets the connection ID header to include in the request packets. If a connection ID is set, it will be sent in each request to the server except for the CONNECT request. An application only needs to set the connection ID if it is trying to operate with different targets over the same transport layer connection. If a client receives a connection ID from the server, the implementation will continue to use that connection ID until the application changes it or until the connection is closed.

Parameters:

`id` - the connection ID to use

Throws:

`IllegalArgumentException` - if `id` is not in the range 0 to 2^{32}-1

setPath(HeaderSet, boolean, boolean)

```
public HeaderSet setPath(HeaderSet headers, boolean backup, boolean create)
            throws IOException
```

Completes an OBEX SETPATH operation. This method will never return `null`.

Parameters:

`backup` - if `true`, instructs the server to back up one directory before moving to the directory specified in name (similar to cd .. on PCs); if `false`, apply `name` to the current directory

`create` - if `true`, instructs the server to create the directory if it does not exist; if `false`, instruct the server to return an error code if the directory does not exist

`headers` - the headers to include in the SETPATH request

Returns: the headers that were returned from the server

Throws:

`IOException` - if an error occurred in the transport layer; if the client is already in an operation; if an OBEX connection does not exist because `connect()` has not been called; if `disconnect()` had been called and a response code of `OBEX_HTTP_OK` was received; if the headers defined in `headers` exceed the max packet length

`IllegalArgumentException` - if `headers` were not created by a call to `createHeaderSet()`

javax.obex
HeaderSet

Declaration

```
public interface HeaderSet
```

Description

The `HeaderSet` interface defines the methods that set and get the values of OBEX headers.

The following table describes how the headers specified in this interface are represented in OBEX and in Java. The Java types are used with the `setHeader()` and `getHeader()` methods and specify the type of object that must be provided and will be returned from these methods, respectively.

Header Values	OBEX Representation	Java Type
COUNT	4 byte unsigned integer	`java.lang.Long` in the range 0 to 2^{32}-1
NAME	Unicode string	`java.lang.String`
TYPE	ASCII string	`java.lang.String`
LENGTH	4 byte unsigned integer	`java.lang.Long` in the range 0 to 2^{32}-1
TIME_ISO_8601	ASCII string of the form YYYYMMDDTHHMMSS[Z] where [Z] specifies Zulu time	`java.util.Calendar`
TIME_4_BYTE	4 byte unsigned integer	`java.util.Calendar`
DESCRIPTION	Unicode string	`java.lang.String`
TARGET	byte sequence	`byte[]`
HTTP	byte sequence	`byte[]`
WHO	byte sequence	`byte[]`
OBJECT_CLASS	byte sequence	`byte[]`
APPLICATION_PARAMETER	byte sequence	`byte[]`

The `APPLICATION_PARAMETER` header requires some additional explanation. The byte array provided with the `APPLICATION_PARAMETER` should be of the form Tag-Length-Value according to the OBEX specification where Tag is a byte long, Length is a byte long, and Value is up to 255 bytes long. Multiple Tag-Length-Value triples are allowed within a single `APPLICATION_PARAMETER` header. The implementation will NOT check this condition. It is mentioned only to allow for interoperability between OBEX implementations.

User Defined Headers

OBEX allows 64 user-defined header values. Depending on the header identifier provided, headers have different types. The table below defines the ranges and their types.

Header Identifier	Decimal Range	OBEX Type	Java Type
0x30 to 0x3F	48 to 63	Unicode String	`java.lang.String`
0x70 to 0x7F	112 to 127	byte sequence	`byte[]`
0xB0 to 0xBF	176 to 191	1 byte	`java.lang.Byte`
0xF0 to 0xFF	240 to 255	4 byte unsigned integer	`java.lang.Long` in the range 0 to 2^{32}-1

Member Summary

Fields

`public static final` APPLICATION_PARAMETER
Represents the OBEX Application Parameter header.

`public static final` COUNT
Represents the OBEX Count header.

`public static final` DESCRIPTION
Represents the OBEX Description header.

`public static final` HTTP
Represents the OBEX HTTP header.

`public static final` LENGTH
Represents the OBEX Length header.

`public static final` NAME
Represents the OBEX Name header.

`public static final` OBJECT_CLASS
Represents the OBEX Object Class header.

`public static final` TARGET
Represents the OBEX Target header.

`public static final` TIME_4_BYTE
Represents the OBEX Time header using the 4 byte representation.

`public static final` TIME_ISO_8601
Represents the OBEX Time header using the ISO 8601 standards.

`public static final` TYPE
Represents the OBEX Type header.

`public static final` WHO
Represents the OBEX Who header.

Methods

`public void` createAuthenticationChallenge(String, boolean, boolean)
Sets the authentication challenge header.

`public Object` getHeader(int)
Retrieves the value of the header identifier provided.

`public int` getHeaderList()
Retrieves the list of headers that may be retrieved via the `getHeader` method that will not return `null`.

`public int` getResponseCode()
Returns the response code received from the server.

`public void` setHeader(int, Object)
Sets the value of the header identifier to the value provided.

Fields

APPLICATION_PARAMETER

```
public static final int APPLICATION_PARAMETER
```

Represents the OBEX Application Parameter header. This header specifies additional application request and response information.

The value of `APPLICATION_PARAMETER` is 0x4C (76).

COUNT

```
public static final int COUNT
```

Represents the OBEX Count header. This allows the connection statement to tell the server how many objects it plans to send or retrieve.

The value of `COUNT` is 0xC0 (192).

DESCRIPTION

```
public static final int DESCRIPTION
```

Represents the OBEX Description header. This is a text description of the object.

The value of `DESCRIPTION` is 0x05 (5).

HTTP

```
public static final int HTTP
```

Represents the OBEX HTTP header. This allows an HTTP 1.X header to be included in a request or reply.

The value of `HTTP` is 0x47 (71).

LENGTH

```
public static final int LENGTH
```

Represents the OBEX Length header. This is the length of the object in bytes.

The value of `LENGTH` is 0xC3 (195).

NAME

```
public static final int NAME
```

Represents the OBEX Name header. This specifies the name of the object.

The value of `NAME` is 0x01 (1).

OBJECT_CLASS

```
public static final int OBJECT_CLASS
```

Represents the OBEX Object Class header. This header specifies the OBEX object class of the object.

The value of `OBJECT_CLASS` is 0x4F (79).

TARGET

```
public static final int TARGET
```

Represents the OBEX Target header. This is the name of the service an operation is targeted to.

The value of `TARGET` is 0x46 (70).

TIME_4_BYTE

```
public static final int TIME_4_BYTE
```

Represents the OBEX Time header using the 4 byte representation. This is only included for backwards compatibility. It represents the number of seconds since January 1, 1970.

The value of `TIME_4_BYTE` is 0xC4 (196).

TIME_ISO_8601

```
public static final int TIME_ISO_8601
```

Represents the OBEX Time header using the ISO 8601 standards. This is the preferred time header.

The value of `TIME_ISO_8601` is 0x44 (68).

TYPE

```
public static final int TYPE
```

Represents the OBEX Type header. This allows a request to specify the type of the object (e.g. text, html, binary, etc.).

The value of `TYPE` is 0x42 (66).

WHO

```
public static final int WHO
```

Represents the OBEX Who header. Identifies the OBEX application to determine if the two peers are talking to each other.

The value of `WHO` is 0x4A (74).

Methods

createAuthenticationChallenge(String, boolean, boolean)

```
public void createAuthenticationChallenge(java.lang.String realm, boolean userID,
            boolean access)
```

Sets the authentication challenge header. The `realm` will be encoded based upon the default encoding scheme used by the implementation to encode strings. Therefore, the encoding scheme used to encode the `realm` is application dependent.

Parameters:

`realm` - a short description that describes what password to use; if `null` no realm will be sent in the authentication challenge header

`userID` - if `true`, a user ID is required in the reply; if `false`, no user ID is required

`access` - if `true` then full access will be granted if successful; if `false` then read-only access will be granted if successful

getHeader(int)

```
public java.lang.Object getHeader(int headerID)
            throws IOException
```

Retrieves the value of the header identifier provided. The type of the Object returned is defined in the description of this interface.

Parameters:

`headerID` - the header identifier whose value is to be returned

Returns: the value of the header provided or `null` if the header identifier specified is not part of this `HeaderSet` object

Throws:

`IllegalArgumentException` - if the `headerID` is not one defined in this interface or any of the user-defined headers

`IOException` - if an error occurred in the transport layer during the operation or if the connection has been closed

getHeaderList()

```
public int[] getHeaderList()
            throws IOException
```

Retrieves the list of headers that may be retrieved via the `getHeader` method that will not return `null`. In other words, this method returns all the headers that are available in this object.

Returns: the array of headers that are set in this object or `null` if no headers are available

Throws:

`IOException` - if an error occurred in the transport layer during the operation or the connection has been closed

See Also: `getHeader(int)`

getResponseCode()

```
public int getResponseCode()
            throws IOException
```

Returns the response code received from the server. Response codes are defined in the `ResponseCodes` class.

Returns: the response code retrieved from the server

Throws:

`IOException` - if an error occurred in the transport layer during the transaction; if this method is called on a `HeaderSet` object created by calling `createHeaderSet()` in a `ClientSession` object; if an OBEX server created this object

See Also: `ResponseCodes`

setHeader(int, Object)

```
public void setHeader(int headerID, java.lang.Object headerValue)
```

Sets the value of the header identifier to the value provided. The type of object must correspond to the Java type defined in the description of this interface. If `null` is passed as the `headerValue` then the header will be removed from the set of headers to include in the next request.

Parameters:

`headerID` - the identifier to include in the message

`headerValue` - the value of the header identifier

Throws:

`IllegalArgumentException` - if the header identifier provided is not one defined in this interface or a user-defined header; if the type of `headerValue` is not the correct Java type as defined in the description of this interface

javax.obex
Operation

Declaration

```
public interface Operation extends javax.microedition.io.ContentConnection
```

All Superinterfaces: javax.microedition.io.Connection, javax.microedition.io.ContentConnection, javax.microedition.io.InputConnection, javax.microedition.io.OutputConnection, javax.microedition.io.StreamConnection

Description

The `Operation` interface provides ways to manipulate a single OBEX PUT or GET operation. The implementation of this interface sends OBEX packets as they are built. If during the operation the peer in the operation ends the operation, an `IOException` is thrown on the next read from the input stream, write to the output stream, or call to `sendHeaders()`.

Definition of methods inherited from `ContentConnection`

`getEncoding()` will always return `null`.
`getLength()` will return the length specified by the OBEX Length header or -1 if the OBEX Length header was not included.
`getType()` will return the value specified in the OBEX Type header or `null` if the OBEX Type header was not included.

How Headers are Handled

As headers are received, they may be retrieved through the `getReceivedHeaders()` method. If new headers are set during the operation, the new headers will be sent during the next packet exchange.

PUT example

```
    void putObjectViaOBEX(ClientSession conn, HeaderSet head, byte[] obj)
        throws IOException
{
        // Include the length header
        head.setHeader(head.LENGTH, new Long(obj.length));

        // Initiate the PUT request
        Operation op = conn.put(head);

        // Open the output stream to put the object to it
        DataOutputStream out = op.openDataOutputStream();

        // Send the object to the server
        out.write(obj);

        // End the transaction
        out.close();
        op.close();
    }
```

GET example

```
byte[] getObjectViaOBEX(ClientSession conn, HeaderSet head) throws IOException {

    // Send the initial GET request to the server
    Operation op = conn.get(head);

    // Retrieve the length of the object being sent back
    int length = op.getLength();

     // Create space for the object
     byte[] obj = new byte[length];

    // Get the object from the input stream
    DataInputStream in = trans.openDataInputStream();
    in.read(obj);

    // End the transaction
    in.close();
    op.close();

    return obj;
}
```

Client PUT Operation Flow

For PUT operations, a call to `close()` the `OutputStream` returned from `openOutputStream()` or `openDataOutputStream()` will signal that the request is done. (In OBEX terms, the End-Of-Body header should be sent and the final bit in the request will be set.) At this point, the reply from the server may begin to be processed. A call to `getResponseCode()` will do an implicit close on the `OutputStream` and therefore signal that the request is done.

Client GET Operation Flow

For GET operation, a call to `openInputStream()` or `openDataInputStream()` will signal that the request is done. (In OBEX terms, the final bit in the request will be set.) A call to `getResponseCode()` will cause an implicit close on the `InputStream`. No further data may be read at this point.

Member Summary

Methods

`public void`	`abort()` Sends an ABORT message to the server.
`public HeaderSet`	`getReceivedHeaders()` Returns the headers that have been received during the operation.
`public int`	`getResponseCode()` Returns the response code received from the server.
`public void`	`sendHeaders(HeaderSet)` Specifies the headers that should be sent in the next OBEX message that is sent.

Inherited Member Summary

Methods inherited from interface javax.microedition.io.Connection

Inherited Member Summary
`close`
Methods inherited from interface javax.microedition.io.ContentConnection
`getEncoding, getLength, getType`
Methods inherited from interface javax.microedition.io.InputConnection
`openDataInputStream, openInputStream`
Methods inherited from interface javax.microedition.io.OutputConnection
`openDataOutputStream, openOutputStream`

Methods

abort()

```
public void abort()
            throws IOException
```

Sends an ABORT message to the server. By calling this method, the corresponding input and output streams will be closed along with this object. No headers are sent in the abort request. This will end the operation since `close()` will be called by this method.

Throws:

`IOException` - if the transaction has already ended or if an OBEX server calls this method

getReceivedHeaders()

```
public HeaderSet getReceivedHeaders()
            throws IOException
```

Returns the headers that have been received during the operation. Modifying the object returned has no effect on the headers that are sent or retrieved.

Returns: the headers received during this `Operation`

Throws:

`IOException` - if this `Operation` has been closed

getResponseCode()

```
public int getResponseCode()
            throws IOException
```

Returns the response code received from the server. Response codes are defined in the `ResponseCodes` class.

Returns: the response code retrieved from the server

Throws:

`IOException` - if an error occurred in the transport layer during the transaction; if this object was created by an OBEX server

See Also: `ResponseCodes`

sendHeaders(HeaderSet)

```
public void sendHeaders(HeaderSet headers)
            throws IOException
```

Specifies the headers that should be sent in the next OBEX message that is sent.

Parameters:

`headers` - the headers to send in the next message

Throws:

`IOException` - if this `Operation` has been closed or the transaction has ended and no further messages will be exchanged

`IllegalArgumentException` - if `headers` was not created by a call to `ServerRequestHandler.createHeaderSet()` or `ClientSession.createHeaderSet()`

`NullPointerException` - if `headers` if `null`

javax.obex
PasswordAuthentication

Declaration
```
public class PasswordAuthentication

java.lang.Object
  |
  +--javax.obex.PasswordAuthentication
```

Description
This class holds user name and password combinations.

Member Summary	
Constructors	
public	`PasswordAuthentication(byte[], byte[])` Creates a new `PasswordAuthentication` with the user name and password provided.
Methods	
public byte	`getPassword()` Retrieves the password.
public byte	`getUserName()` Retrieves the user name that was specified in the constructor.

Inherited Member Summary
Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

PasswordAuthentication(byte[], byte[])

```
public PasswordAuthentication(byte[] userName, byte[] password)
```

Creates a new `PasswordAuthentication` with the user name and password provided.

Parameters:

`userName` - the user name to include; this may be `null`

`password` - the password to include in the response

Throws:

`NullPointerException` - if `password` is `null`

Methods

getPassword()

```
public byte[] getPassword()
```

Retrieves the password.

Returns: the password

getUserName()

```
public byte[] getUserName()
```

Retrieves the user name that was specified in the constructor. The user name may be `null`.

Returns: the user name

javax.obex
ResponseCodes

Declaration

```
public class ResponseCodes

java.lang.Object
  |
  +--javax.obex.ResponseCodes
```

Description

The `ResponseCodes` class contains the list of valid response codes a server may send to a client.

IMPORTANT NOTE

It is important to note that these values are different then those defined in `javax.microedition.io.HttpConnection`. The values in this interface represent the values defined in the IrOBEX specification. The values in `javax.microedition.io.HttpConnection` represent values defined in the HTTP specification.

`OBEX_DATABASE_FULL` and `OBEX_DATABASE_LOCKED` require further description since they are not defined in HTTP. The server will send an `OBEX_DATABASE_FULL` message when the client requests that something be placed into a database but the database is full (cannot take more data). `OBEX_DATABASE_LOCKED` will be returned when the client wishes to access a database, database table, or database record that has been locked.

Member Summary

Fields

`public static final`	`OBEX_DATABASE_FULL` Defines the OBEX DATABASE FULL response code.
`public static final`	`OBEX_DATABASE_LOCKED` Defines the OBEX DATABASE LOCKED response code.
`public static final`	`OBEX_HTTP_ACCEPTED` Defines the OBEX ACCEPTED response code.
`public static final`	`OBEX_HTTP_BAD_GATEWAY` Defines the OBEX BAD GATEWAY response code.
`public static final`	`OBEX_HTTP_BAD_METHOD` Defines the OBEX METHOD NOT ALLOWED response code.
`public static final`	`OBEX_HTTP_BAD_REQUEST` Defines the OBEX BAD REQUEST response code.
`public static final`	`OBEX_HTTP_CONFLICT` Defines the OBEX METHOD CONFLICT response code.
`public static final`	`OBEX_HTTP_CREATED` Defines the OBEX CREATED response code.
`public static final`	`OBEX_HTTP_ENTITY_TOO_LARGE` Defines the OBEX REQUESTED ENTITY TOO LARGE response code.
`public static final`	`OBEX_HTTP_FORBIDDEN` Defines the OBEX FORBIDDEN response code.
`public static final`	`OBEX_HTTP_GATEWAY_TIMEOUT` Defines the OBEX GATEWAY TIMEOUT response code.

Member Summary

`public static final`	`OBEX_HTTP_GONE` Defines the OBEX METHOD GONE response code.
`public static final`	`OBEX_HTTP_INTERNAL_ERROR` Defines the OBEX INTERNAL SERVER ERROR response code.
`public static final`	`OBEX_HTTP_LENGTH_REQUIRED` Defines the OBEX METHOD LENGTH REQUIRED response code.
`public static final`	`OBEX_HTTP_MOVED_PERM` Defines the OBEX MOVED PERMANENTLY response code.
`public static final`	`OBEX_HTTP_MOVED_TEMP` Defines the OBEX MOVED TEMPORARILY response code.
`public static final`	`OBEX_HTTP_MULT_CHOICE` Defines the OBEX MULTIPLE_CHOICES response code.
`public static final`	`OBEX_HTTP_NO_CONTENT` Defines the OBEX NO CONTENT response code.
`public static final`	`OBEX_HTTP_NOT_ACCEPTABLE` Defines the OBEX NOT ACCEPTABLE response code.
`public static final`	`OBEX_HTTP_NOT_AUTHORITATIVE` Defines the OBEX NON-AUTHORITATIVE INFORMATION response code.
`public static final`	`OBEX_HTTP_NOT_FOUND` Defines the OBEX NOT FOUND response code.
`public static final`	`OBEX_HTTP_NOT_IMPLEMENTED` Defines the OBEX NOT IMPLEMENTED response code.
`public static final`	`OBEX_HTTP_NOT_MODIFIED` Defines the OBEX NOT MODIFIED response code.
`public static final`	`OBEX_HTTP_OK` Defines the OBEX SUCCESS response code.
`public static final`	`OBEX_HTTP_PARTIAL` Defines the OBEX PARTIAL CONTENT response code.
`public static final`	`OBEX_HTTP_PAYMENT_REQUIRED` Defines the OBEX PAYMENT REQUIRED response code.
`public static final`	`OBEX_HTTP_PRECON_FAILED` Defines the OBEX PRECONDITION FAILED response code.
`public static final`	`OBEX_HTTP_PROXY_AUTH` Defines the OBEX PROXY AUTHENTICATION REQUIRED response code.
`public static final`	`OBEX_HTTP_REQ_TOO_LARGE` Defines the OBEX REQUESTED URL TOO LARGE response code.
`public static final`	`OBEX_HTTP_RESET` Defines the OBEX RESET CONTENT response code.
`public static final`	`OBEX_HTTP_SEE_OTHER` Defines the OBEX SEE OTHER response code.
`public static final`	`OBEX_HTTP_TIMEOUT` Defines the OBEX REQUEST TIME OUT response code.
`public static final`	`OBEX_HTTP_UNAUTHORIZED` Defines the OBEX UNAUTHORIZED response code.
`public static final`	`OBEX_HTTP_UNAVAILABLE` Defines the OBEX SERVICE UNAVAILABLE response code.
`public static final`	`OBEX_HTTP_UNSUPPORTED_TYPE` Defines the OBEX UNSUPPORTED MEDIA TYPE response code.
`public static final`	`OBEX_HTTP_USE_PROXY` Defines the OBEX USE PROXY response code.
`public static final`	`OBEX_HTTP_VERSION` Defines the OBEX HTTP VERSION NOT SUPPORTED response code.

Inherited Member Summary
Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Fields

OBEX_DATABASE_FULL

`public static final int` **`OBEX_DATABASE_FULL`**

Defines the OBEX DATABASE FULL response code.

The value of `OBEX_DATABASE_FULL` is 0xE0 (224).

OBEX_DATABASE_LOCKED

`public static final int` **`OBEX_DATABASE_LOCKED`**

Defines the OBEX DATABASE LOCKED response code.

The value of `OBEX_DATABASE_LOCKED` is 0xE1 (225).

OBEX_HTTP_ACCEPTED

`public static final int` **`OBEX_HTTP_ACCEPTED`**

Defines the OBEX ACCEPTED response code.

The value of `OBEX_HTTP_ACCEPTED` is 0xA2 (162).

OBEX_HTTP_BAD_GATEWAY

`public static final int` **`OBEX_HTTP_BAD_GATEWAY`**

Defines the OBEX BAD GATEWAY response code.

The value of `OBEX_HTTP_BAD_GATEWAY` is 0xD2 (210).

OBEX_HTTP_BAD_METHOD

`public static final int` **`OBEX_HTTP_BAD_METHOD`**

Defines the OBEX METHOD NOT ALLOWED response code.

The value of `OBEX_HTTP_BAD_METHOD` is 0xC5 (197).

OBEX_HTTP_BAD_REQUEST

`public static final int` **`OBEX_HTTP_BAD_REQUEST`**

Defines the OBEX BAD REQUEST response code.

The value of `OBEX_HTTP_BAD_REQUEST` is 0xC0 (192).

OBEX_HTTP_CONFLICT

`public static final int OBEX_HTTP_CONFLICT`

Defines the OBEX METHOD CONFLICT response code.

The value of `OBEX_HTTP_CONFLICT` is 0xC9 (201).

OBEX_HTTP_CREATED

`public static final int OBEX_HTTP_CREATED`

Defines the OBEX CREATED response code.

The value of `OBEX_HTTP_CREATED` is 0xA1 (161).

OBEX_HTTP_ENTITY_TOO_LARGE

`public static final int OBEX_HTTP_ENTITY_TOO_LARGE`

Defines the OBEX REQUESTED ENTITY TOO LARGE response code.

The value of `OBEX_HTTP_ENTITY_TOO_LARGE` is 0xCD (205).

OBEX_HTTP_FORBIDDEN

`public static final int OBEX_HTTP_FORBIDDEN`

Defines the OBEX FORBIDDEN response code.

The value of `OBEX_HTTP_FORBIDDEN` is 0xC3 (195).

OBEX_HTTP_GATEWAY_TIMEOUT

`public static final int OBEX_HTTP_GATEWAY_TIMEOUT`

Defines the OBEX GATEWAY TIMEOUT response code.

The value of `OBEX_HTTP_GATEWAY_TIMEOUT` is 0xD4 (212).

OBEX_HTTP_GONE

`public static final int OBEX_HTTP_GONE`

Defines the OBEX METHOD GONE response code.

The value of `OBEX_HTTP_GONE` is 0xCA (202).

OBEX_HTTP_INTERNAL_ERROR

`public static final int OBEX_HTTP_INTERNAL_ERROR`

Defines the OBEX INTERNAL SERVER ERROR response code.

The value of `OBEX_HTTP_INTERNAL_ERROR` is 0xD0 (208).

OBEX_HTTP_LENGTH_REQUIRED

`public static final int OBEX_HTTP_LENGTH_REQUIRED`

Defines the OBEX METHOD LENGTH REQUIRED response code.

The value of `OBEX_HTTP_LENGTH_REQUIRED` is 0xCB (203).

OBEX_HTTP_MOVED_PERM

`public static final int` **`OBEX_HTTP_MOVED_PERM`**

Defines the OBEX MOVED PERMANENTLY response code.

The value of `OBEX_HTTP_MOVED_PERM` is 0xB1 (177).

OBEX_HTTP_MOVED_TEMP

`public static final int` **`OBEX_HTTP_MOVED_TEMP`**

Defines the OBEX MOVED TEMPORARILY response code.

The value of `OBEX_HTTP_MOVED_TEMP` is 0xB2 (178).

OBEX_HTTP_MULT_CHOICE

`public static final int` **`OBEX_HTTP_MULT_CHOICE`**

Defines the OBEX MULTIPLE_CHOICES response code.

The value of `OBEX_HTTP_MULT_CHOICE` is 0xB0 (176).

OBEX_HTTP_NO_CONTENT

`public static final int` **`OBEX_HTTP_NO_CONTENT`**

Defines the OBEX NO CONTENT response code.

The value of `OBEX_HTTP_NO_CONTENT` is 0xA4 (164).

OBEX_HTTP_NOT_ACCEPTABLE

`public static final int` **`OBEX_HTTP_NOT_ACCEPTABLE`**

Defines the OBEX NOT ACCEPTABLE response code.

The value of `OBEX_HTTP_NOT_ACCEPTABLE` is 0xC6 (198).

OBEX_HTTP_NOT_AUTHORITATIVE

`public static final int` **`OBEX_HTTP_NOT_AUTHORITATIVE`**

Defines the OBEX NON-AUTHORITATIVE INFORMATION response code.

The value of `OBEX_HTTP_NOT_AUTHORITATIVE` is 0xA3 (163).

OBEX_HTTP_NOT_FOUND

`public static final int` **`OBEX_HTTP_NOT_FOUND`**

Defines the OBEX NOT FOUND response code.

The value of `OBEX_HTTP_NOT_FOUND` is 0xC4 (196).

OBEX_HTTP_NOT_IMPLEMENTED

`public static final int` **`OBEX_HTTP_NOT_IMPLEMENTED`**

Defines the OBEX NOT IMPLEMENTED response code.

The value of `OBEX_HTTP_NOT_IMPLEMENTED` is 0xD1 (209).

OBEX_HTTP_NOT_MODIFIED

`public static final int OBEX_HTTP_NOT_MODIFIED`

Defines the OBEX NOT MODIFIED response code.

The value of `OBEX_HTTP_NOT_MODIFIED` is 0xB4 (180).

OBEX_HTTP_OK

`public static final int OBEX_HTTP_OK`

Defines the OBEX SUCCESS response code.

The value of `OBEX_HTTP_OK` is 0xA0 (160).

OBEX_HTTP_PARTIAL

`public static final int OBEX_HTTP_PARTIAL`

Defines the OBEX PARTIAL CONTENT response code.

The value of `OBEX_HTTP_PARTIAL` is 0xA6 (166).

OBEX_HTTP_PAYMENT_REQUIRED

`public static final int OBEX_HTTP_PAYMENT_REQUIRED`

Defines the OBEX PAYMENT REQUIRED response code.

The value of `OBEX_HTTP_PAYMENT_REQUIRED` is 0xC2 (194).

OBEX_HTTP_PRECON_FAILED

`public static final int OBEX_HTTP_PRECON_FAILED`

Defines the OBEX PRECONDITION FAILED response code.

The value of `OBEX_HTTP_PRECON_FAILED` is 0xCC (204).

OBEX_HTTP_PROXY_AUTH

`public static final int OBEX_HTTP_PROXY_AUTH`

Defines the OBEX PROXY AUTHENTICATION REQUIRED response code.

The value of `OBEX_HTTP_PROXY_AUTH` is 0xC7 (199).

OBEX_HTTP_REQ_TOO_LARGE

`public static final int OBEX_HTTP_REQ_TOO_LARGE`

Defines the OBEX REQUESTED URL TOO LARGE response code.

The value of `OBEX_HTTP_REQ_TOO_LARGE` is 0xCE (206).

OBEX_HTTP_RESET

`public static final int OBEX_HTTP_RESET`

Defines the OBEX RESET CONTENT response code.

The value of `OBEX_HTTP_RESET` is 0xA5 (165).

OBEX_HTTP_SEE_OTHER

```
public static final int OBEX_HTTP_SEE_OTHER
```

Defines the OBEX SEE OTHER response code.

The value of `OBEX_HTTP_SEE_OTHER` is 0xB3 (179).

OBEX_HTTP_TIMEOUT

```
public static final int OBEX_HTTP_TIMEOUT
```

Defines the OBEX REQUEST TIME OUT response code.

The value of `OBEX_HTTP_TIMEOUT` is 0xC8 (200).

OBEX_HTTP_UNAUTHORIZED

```
public static final int OBEX_HTTP_UNAUTHORIZED
```

Defines the OBEX UNAUTHORIZED response code.

The value of `OBEX_HTTP_UNAUTHORIZED` is 0xC1 (193).

OBEX_HTTP_UNAVAILABLE

```
public static final int OBEX_HTTP_UNAVAILABLE
```

Defines the OBEX SERVICE UNAVAILABLE response code.

The value of `OBEX_HTTP_UNAVAILABLE` is 0xD3 (211).

OBEX_HTTP_UNSUPPORTED_TYPE

```
public static final int OBEX_HTTP_UNSUPPORTED_TYPE
```

Defines the OBEX UNSUPPORTED MEDIA TYPE response code.

The value of `OBEX_HTTP_UNSUPPORTED_TYPE` is 0xCF (207).

OBEX_HTTP_USE_PROXY

```
public static final int OBEX_HTTP_USE_PROXY
```

Defines the OBEX USE PROXY response code.

The value of `OBEX_HTTP_USE_PROXY` is 0xB5 (181).

OBEX_HTTP_VERSION

```
public static final int OBEX_HTTP_VERSION
```

Defines the OBEX HTTP VERSION NOT SUPPORTED response code.

The value of `OBEX_HTTP_VERSION` is 0xD5 (213).

javax.obex
ServerRequestHandler

Declaration

```
public class ServerRequestHandler

java.lang.Object
  |
  +--javax.obex.ServerRequestHandler
```

Description

The `ServerRequestHandler` class defines an event listener that will respond to OBEX requests made to the server.

The `onConnect()`, `onSetPath()`, `onDelete()`, `onGet()`, and `onPut()` methods may return any response code defined in the `ResponseCodes` class except for `OBEX_HTTP_CONTINUE`. If `OBEX_HTTP_CONTINUE` or a value not defined in the `ResponseCodes` class is returned, the server implementation will send an `OBEX_HTTP_INTERNAL_ERROR` response to the client.

Connection ID and Target Headers

According to the IrOBEX specification, a packet may not contain a Connection ID and Target header. Since the Connection ID header is managed by the implementation, it will not send a Connection ID header, if a Connection ID was specified, in a packet that has a Target header. In other words, if an application adds a Target header to a `HeaderSet` object used in an OBEX operation and a Connection ID was specified, no Connection ID will be sent in the packet containing the Target header.

CREATE-EMPTY Requests

A CREATE-EMPTY request allows clients to create empty objects on the server. When a CREATE-EMPTY request is received, the `onPut()` method will be called by the implementation. To differentiate between a normal PUT request and a CREATE-EMPTY request, an application must open the `InputStream` from the `Operation` object passed to the `onPut()` method. For a PUT request, the application will be able to read Body data from this `InputStream`. For a CREATE-EMPTY request, there will be no Body data to read. Therefore, a call to `InputStream.read()` will return -1.

Member Summary

Constructors

protected	`ServerRequestHandler()` Creates a ServerRequestHandler.

Methods

public final HeaderSet	`createHeaderSet()` Creates a `HeaderSet` object that may be used in put and get operations.
public long	`getConnectionID()` Retrieves the connection ID that is being used in the present connection.
public void	`onAuthenticationFailure(byte[])` Called when this object attempts to authenticate a client and the authentication request fails because the response digest in the authentication response header was wrong.
public int	`onConnect(HeaderSet, HeaderSet)` Called when a CONNECT request is received.

Member Summary	
public int	onDelete(HeaderSet, HeaderSet) Called when a DELETE request is received.
public void	onDisconnect(HeaderSet, HeaderSet) Called when a DISCONNECT request is received.
public int	onGet(Operation) Called when a GET request is received.
public int	onPut(Operation) Called when a PUT request is received.
public int	onSetPath(HeaderSet, HeaderSet, boolean, boolean) Called when a SETPATH request is received.
public void	setConnectionID(long) Sets the connection ID header to include in the reply packets.

Inherited Member Summary
Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

ServerRequestHandler()

```
protected ServerRequestHandler()
```

Creates a ServerRequestHandler.

Methods

createHeaderSet()

```
public final HeaderSet createHeaderSet()
```

Creates a `HeaderSet` object that may be used in put and get operations.

Returns: the `HeaderSet` object to use in put and get operations

getConnectionID()

```
public long getConnectionID()
```

Retrieves the connection ID that is being used in the present connection. This method will return -1 if no connection ID is being used.

Returns: the connection id being used or -1 if no connection ID is being used

onAuthenticationFailure(byte[])

```
public void onAuthenticationFailure(byte[] userName)
```

Called when this object attempts to authenticate a client and the authentication request fails because the response digest in the authentication response header was wrong.

If this method is not implemented by the class that extends this class, this method will do nothing.

Parameters:

`userName` - the user name returned in the authentication response; `null` if no user name was provided in the response

onConnect(HeaderSet, HeaderSet)

```
public int onConnect(HeaderSet request, HeaderSet reply)
```

Called when a CONNECT request is received.

If this method is not implemented by the class that extends this class, `onConnect()` will always return an `OBEX_HTTP_OK` response code.

The headers received in the request can be retrieved from the `request` argument. The headers that should be sent in the reply must be specified in the `reply` argument.

Parameters:

`request` - contains the headers sent by the client; `request` will never be `null`

`reply` - the headers that should be sent in the reply; `reply` will never be `null`

Returns: a response code defined in `ResponseCodes` that will be returned to the client; if an invalid response code is provided, the `OBEX_HTTP_INTERNAL_ERROR` response code will be used

onDelete(HeaderSet, HeaderSet)

```
public int onDelete(HeaderSet request, HeaderSet reply)
```

Called when a DELETE request is received.

If this method is not implemented by the class that extends this class, `onDelete()` will always return an `OBEX_HTTP_NOT_IMPLEMENTED` response code.

The headers received in the request can be retrieved from the `request` argument. The headers that should be sent in the reply must be specified in the `reply` argument.

Parameters:

`request` - contains the headers sent by the client; `request` will never be `null`

`reply` - the headers that should be sent in the reply; `reply` will never be `null`

Returns: a response code defined in `ResponseCodes` that will be returned to the client; if an invalid response code is provided, the `OBEX_HTTP_INTERNAL_ERROR` response code will be used

onDisconnect(HeaderSet, HeaderSet)

```
public void onDisconnect(HeaderSet request, HeaderSet reply)
```

Called when a DISCONNECT request is received.

The headers received in the request can be retrieved from the `request` argument. The headers that should be sent in the reply must be specified in the `reply` argument.

Parameters:

`request` - contains the headers sent by the client; `request` will never be `null`

`reply` - the headers that should be sent in the reply; `reply` will never be `null`

onGet(Operation)

```
public int onGet(Operation op)
```

Called when a GET request is received.

If this method is not implemented by the class that extends this class, `onGet()` will always return an `OBEX_HTTP_NOT_IMPLEMENTED` response code.

If an ABORT request is received during the processing of a GET request, `op` will be closed by the implementation.

Parameters:

`op` - contains the headers sent by the client and allows new headers to be sent in the reply; `op` will never be `null`

Returns: a response code defined in `ResponseCodes` that will be returned to the client; if an invalid response code is provided, the `OBEX_HTTP_INTERNAL_ERROR` response code will be used

onPut(Operation)

```
public int onPut(Operation op)
```

Called when a PUT request is received.

If this method is not implemented by the class that extends this class, `onPut()` will always return an `OBEX_HTTP_NOT_IMPLEMENTED` response code.

If an ABORT request is received during the processing of a PUT request, `op` will be closed by the implementation.

Parameters:

`op` - contains the headers sent by the client and allows new headers to be sent in the reply; `op` will never be `null`

Returns: a response code defined in `ResponseCodes` that will be returned to the client; if an invalid response code is provided, the `OBEX_HTTP_INTERNAL_ERROR` response code will be used

onSetPath(HeaderSet, HeaderSet, boolean, boolean)

```
public int onSetPath(HeaderSet request, HeaderSet reply, boolean backup, boolean create)
```

Called when a SETPATH request is received.

If this method is not implemented by the class that extends this class, `onSetPath()` will always return an `OBEX_HTTP_NOT_IMPLEMENTED` response code.

The headers received in the request can be retrieved from the `request` argument. The headers that should be sent in the reply must be specified in the `reply` argument.

Parameters:

`request` - contains the headers sent by the client; `request` will never be `null`

`reply` - the headers that should be sent in the reply; `reply` will never be `null`

`backup` - `true` if the client requests that the server back up one directory before changing to the path described by `name`; `false` to apply the request to the present path

`create` - `true` if the path should be created if it does not already exist; `false` if the path should not be created if it does not exist and an error code should be returned

Returns: a response code defined in `ResponseCodes` that will be returned to the client; if an invalid response code is provided, the `OBEX_HTTP_INTERNAL_ERROR` response code will be used

setConnectionID(long)

```
public void setConnectionID(long id)
```

Sets the connection ID header to include in the reply packets.

Parameters:

`id` - the connection ID to use; -1 if no connection ID should be sent

Throws:

`IllegalArgumentException` - if `id` is not in the range -1 to 2^{32}-1

javax.obex
SessionNotifier

Declaration
```
public interface SessionNotifier extends javax.microedition.io.Connection
```

All Superinterfaces: javax.microedition.io.Connection

Description
The `SessionNotifier` interface defines a connection notifier for server-side OBEX connections. When a `SessionNotifier` is created and calls `acceptAndOpen()`, it will begin listening for clients to create a connection at the transport layer. When the transport layer connection is received, the `acceptAndOpen()` method will return a `javax.microedition.io.Connection` that is the connection to the client. The `acceptAndOpen()` method also takes a `ServerRequestHandler` argument that will process the requests from the client that connects to the server.

Member Summary	
Methods	
`public Connection`	`acceptAndOpen(ServerRequestHandler)` Waits for a transport layer connection to be established and specifies the handler to handle the requests from the client.
`public Connection`	`acceptAndOpen(ServerRequestHandler, Authenticator)` Waits for a transport layer connection to be established and specifies the handler to handle the requests from the client and the `Authenticator` to use to respond to authentication challenge and authentication response headers.

Inherited Member Summary
Methods inherited from interface javax.microedition.io.Connection
`close`

Methods

acceptAndOpen(ServerRequestHandler)
```
public javax.microedition.io.Connection acceptAndOpen(ServerRequestHandler handler)
            throws IOException
```

Waits for a transport layer connection to be established and specifies the handler to handle the requests from the client. No authenticator is associated with this connection, therefore, it is implementation dependent as to how an authentication challenge and authentication response header will be received and processed.

Additional Note for OBEX over Bluetooth
If this method is called on a `SessionNotifier` object that does not have a `ServiceRecord` in the SDDB, the `ServiceRecord` for this object will be added to the SDDB. This method requests the BCC to put the local device in connectable mode so that it will respond to connection attempts by clients.

The following checks are done to verify that the service record provided is valid. If any of these checks fail, then a `ServiceRegistrationException` is thrown.

- ServiceClassIDList and ProtocolDescriptorList, the mandatory service attributes for a `btgoep` service record, must be present in the `ServiceRecord` associated with this notifier.
- L2CAP, RFCOMM and OBEX must all be in the ProtocolDescriptorList
- The `ServiceRecord` associated with this notifier must not have changed the RFCOMM server channel number

This method will not ensure that `ServiceRecord` associated with this notifier is a completely valid service record. It is the responsibility of the application to ensure that the service record follows all of the applicable syntactic and semantic rules for service record correctness.

Parameters:
`handler` - the request handler that will respond to OBEX requests

Returns: the connection to the client

Throws:
`IOException` - if an error occurs in the transport layer

`NullPointerException` - if `handler` is `null`

`ServiceRegistrationException` - if the structure of the associated service record is invalid or if the service record could not be added successfully to the local SDDB. The structure of service record is invalid if the service record is missing any mandatory service attributes, or has changed any of the values described above which are fixed and cannot be changed. Failures to add the record to the SDDB could be due to insufficient disk space, database locks, etc.

`BluetoothStateException` - if the server device could not be placed in connectable mode because the device user has configured the device to be non-connectable

acceptAndOpen(ServerRequestHandler, Authenticator)

```
public javax.microedition.io.Connection acceptAndOpen(ServerRequestHandler handler,
            Authenticator auth)
            throws IOException
```

Waits for a transport layer connection to be established and specifies the handler to handle the requests from the client and the `Authenticator` to use to respond to authentication challenge and authentication response headers.

Additional Note for OBEX over Bluetooth
If this method is called on a `SessionNotifier` object that does not have a `ServiceRecord` in the SDDB, the `ServiceRecord` for this object will be added to the SDDB. This method requests the BCC to put the local device in connectable mode so that it will respond to connection attempts by clients.

The following checks are done to verify that the service record provided is valid. If any of these checks fail, then a `ServiceRegistrationException` is thrown.

- ServiceClassIDList and ProtocolDescriptorList, the mandatory service attributes for a `btgoep` service record, must be present in the `ServiceRecord` associated with this notifier.

- L2CAP, RFCOMM and OBEX must all be in the ProtocolDescriptorList
- The `ServiceRecord` associated with this notifier must not have changed the RFCOMM server channel number

This method will not ensure that `ServiceRecord` associated with this notifier is a completely valid service record. It is the responsibility of the application to ensure that the service record follows all of the applicable syntactic and semantic rules for service record correctness.

Parameters:

`handler` - the request handler that will respond to OBEX requests

`auth` - the `Authenticator` to use with this connection; if `null` then no `Authenticator` will be used

Returns: the connection to the client

Throws:

`IOException` - if an error occurs in the transport layer

`NullPointerException` - if `handler` is `null`

`ServiceRegistrationException` - if the structure of the associated service record is invalid or if the service record could not be added successfully to the local SDDB. The structure of service record is invalid if the service record is missing any mandatory service attributes, or has changed any of the values described above which are fixed and cannot be changed. Failures to add the record to the SDDB could be due to insufficient disk space, database locks, etc.

`BluetoothStateException` - if the server device could not be placed in connectable mode because the device user has configured the device to be non-connectable