

ID		Creation date	October 29, 2008
Platform	S60 3rd Edition, FP2	Tested on devices	
Category	Java ME	Subcategory	Bluetooth

Keywords (APIs, classes, methods, functions): javax.obex.SessionNotifier, javax.obex.RequestHandler, BtObexMessengerSrv.ObexServer, BtObexMessengerSrv.stopObexServer, BtObexMessengerSrv.run, javax.obex.SessionNotifier.acceptAndOpen, javax.obex.RequestHandler.onGet, javax.obex.RequestHandler.onPut, javax.obex.RequestHandler.onConnect, javax.obex.RequestHandler.onDisconnect

Overview

The code snippet demonstrates Bluetooth OBEX over RFCOMM server example. As an example this MIDlet start the OBEX server for single client. Server and client can exchange with messages.

Server gets started by pressing Start command. Service advertising starts from forming a String with connection URL. For more info on bluetooth connection URLs see JSR-82.

When we have a correct URL we call `open` method of the `Connector` class.

After that server can start waiting for incoming connections by calling `acceptAndOpen` method of `SessionNotifier` object instance.

When an incoming connection detected `RequestHandler.onConnect` gets called.

When a client initiates a GET request `RequestHandler.onGet` will be called on the server. This request will send messages from the message buffer to client in response.

When a client initiates a PUT request `RequestHandler.onPut` will be called. Server retrieves data from this request, constructs a string from them and prints them out on the form as a 'Message: ...' string.

When a client disconnects `RequestHandler.onDisconnect` gets called.

Source file: BtObexMessengerSrv.java

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
```

Using_Bluetooth_obex_server_in_Java_ME

```
import javax.microedition.lcdui.TextBox;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.io.StreamConnectionNotifier;
import javax.microedition.io.Connection;
import javax.obex.Operation;
import javax.obex.HeaderSet;
import javax.obex.ResponseCodes;
import javax.obex.ServerRequestHandler;
import javax.obex.SessionNotifier;

/**
 * BtObexMessengerSrv MIDlet is an Obex over RFCOMM server.
 * It acts as a message server.
 * This example shows usage of the OBEX API.
 * @see JSR82-specification for more info on server advertising.
 */
public class BtObexMessengerSrv extends MIDlet implements CommandListener,
    Runnable {
    private Display display;
    /**
     * Reference to a Form object.
     * Used for displaying current server status information.
     */
    private Form frmMain;
    /**
     * Reference to a text field. This field is opened when the user wishes
     * to send a message to connected client.
     */
    private TextBox txtSend;
    /**
     * Comand displayed on main form. When it is triggered application closes.
     */
    private Command cmdExit;
    /**
     * Command starts a server and initializes connecting waiting thread.
     */
    private Command cmdStart;
    /**
     * Command stops the server by closing connection and connection notifier.
     * Connection waiting thread gets interrupted when this command is being
     * handled.
     */
    private Command cmdStop;
    /**
     * Command is add to frmMain and to txtSend. When this command is triggered
     * by frmMain the txtSend is being opened. cmdSend triggered by txtSend
     * message that was typed in the textbox is added to msgSendBuffer.
     */
    private Command cmdSend;
    /**
     * Used by txtend
     * Closes txtSend and opens frmMain.
     */
    private Command cmdBack;
    /**
     * Protocol name entry from connection URL.
     * @see JSR82-specification.
     */
}
```

Using_Bluetooth_obex_server_in_Java_ME

```
private String connProtocol;
/**
 * Service class id entry from connection URL.
 * @see JSR82-specification.
 */
private String connServiceClassId;
/**
 * Target entry from connection URL.
 * @see JSR82-specification.
 */
private String connTarget;
/**
 * Name entry from connection URL.
 * @see JSR82-specification.
 */
private String connServiceName;
/**
 * Part of connection URL where security settings can be specified.
 * @see JSR82-specification.
 */
private String connSecurityStr;
/**
 * Stores a reference to incoming connection notifier.
 */
private SessionNotifier sessionNotifier;
/**
 * Stores a reference to a obex connection(session).
 * Connection waiting thread uses it for accepting conections.
 */
private Connection session;
/**
 * Used for handling client requests(GET and PUT).
 * @see RequestHandler
 * @see ServerRequestHandler
 */
private RequestHandler handler;
/**
 * Thread that listens for incomming connections.
 */
private Thread threadMain;
/**
 * Set to true if a server is started and waiting for incoming
 * connections. false - otherwise.
 */
private boolean serverStarted;
/**
 * String buffer for storing messages that will be send on GET request from
 * the client.
 */
private StringBuffer msgSendBuffer;
/**
 * Constructor. Constructs the object and initializes displayables.
 * Gets Display object reference.
 * Sets current server status to false(not running).
 * Initializes connection URL.
 */
public BtObexMessengerSrv() {
    InitializeComponents();
    //we create an instance of request handler
    handler = new RequestHandler();
    //we choosed an OBEX over RFCOMM protocol
    connProtocol = "btgoep";
}
```


Using_Bluetooth_obex_server_in_Java_ME

```
* @param unconditional whether the MIDlet has to be unconditionally
* terminated
* @throws javax.microedition.midlet.MIDletStateChangeException
*/
public void destroyApp(boolean unconditional) throws MIDletStateChangeException {

    exitMIDlet();
}
/**
 * From CommandListener.
 * Called by the system to indicate that a command has been invoked on a
 * particular displayable.
 * @param command the command that was invoked
 * @param displayable the displayable where the command was invoked
 */
public void commandAction( Command command, Displayable displayable ) {

    if( command == cmdExit ) {
        exitMIDlet();
    }
    if( command == cmdStart ) {
        int err = 0;
        err = startObexServer();
        if( err != 0 ) {
            return;
        }
        serverStarted = true;
        //we do not need a start soft key when in descovery mode
        frmMain.removeCommand( cmdStart );
        //now we add stop soft key
        frmMain.addCommand( cmdStop );
    }
    if( command == cmdStop ) {
        stopObexServer();
        //we remove stop softkey
        frmMain.removeCommand( cmdStop );
        //adding start soft key
        frmMain.addCommand( cmdStart );
        frmMain.removeCommand( cmdSend );
        serverStarted = false;
    }
    if( command == cmdSend ) {
        if( displayable == frmMain ) {
            display.setCurrent( txtSend );
        }
        if( displayable == txtSend ) {
            putMsgToSendBuffer();
            display.setCurrent( frmMain );
        }
    }
    if( command == cmdBack ) {
        display.setCurrent( frmMain );
    }
}
/**
 * Method calls stopObexServer(if serverStarted == true ) and
 * notifyDestroyed after that.
 * @see BtObexMessengerSrv#serverStarted
 * @see BtObexMessengerSrv#stopObexServer()
 * @see MIDlet#notifyDestroyed()
 */
protected void exitMIDlet() {
```

Using_Bluetooth_obex_server_in_Java_ME

```
        if( serverStarted ) {
            stopObexServer();
        }
        notifyDestroyed();
    }
}
/**
 * Method creates a connection notifier and initializes a connection
 * waiting thread.
 * @return 0 if server startup was successfull. 1 if connection notifier
 * instantiation caused an exception. 2 if thread creating caused an
 * exception.
 * @see Connector#open(java.lang.String)
 * @see Thread#start()
 * @see BtObexMessengerSrv#run()
 */
private int startObexServer() {

    String strServerConnection = connProtocol +
        "://" +
        connTarget +
        ":" +
        connServiceClassId +
        ";" +
        "name=" + connServiceName +
        ";" +
        connSecurityStr;

    try {
        printToFrm("Starting server...");
        sessionNotifier = ( SessionNotifier ) Connector.open(
            strServerConnection );
        printToFrm( "Starting a listening thread..." );
        threadMain = new Thread( this );
        threadMain.start();
    } catch ( IllegalThreadStateException e ) {
        printToFrm( "Error starting a server thread!" );
        return 2;
    }

    } catch ( Exception e ) {
        printToFrm( "Error starting a server connection!" );
        return 1;
    }
    }
    return 0;
}
}
/**
 * @return 0 if the server shutdown was successfull. 1 if there was
 * an IOException exception during execution.
 * @see javax.microedition.io.StreamConnectionNotifier
 * @see javax.microedition.io.StreamConnection
 */
private int stopObexServer() {

    try {
        printToFrm("Stopping server...");
        sessionNotifier.close();
        sessionNotifier = null;
    }

    } catch ( IOException e ) {
        printToFrm( "Error closing server connection!" );
        return 1;
    }
    }
    return 0;
}
}
```

Using_Bluetooth_obex_server_in_Java_ME

```
/**
 * From Runnable.
 * The method is being run when a connection listening thread was created
 * and started(in startObexServer). It runs acceptAndOpen method.
 * When a connection accepted doConnReplay is being executed. After that a
 * cmdStop command is trigger to stop the server.
 * @see StreamConnectionNotifier
 * @see BtObexMessengerSrv#doConnReply()
 */
public void run() {
    try {
        printToFrm( "Waiting for connections..." );
        sessionNotifier.acceptAndOpen( handler );
        printToFrm( "Incomming connection..." );
        //waiting for messages

    } catch ( Exception e ) {
        printToFrm( "Connection waiting thread interropted!" );
        printToFrm( e.toString() );
    }
}
/**
 * Adds a StringItem to the frmMain.
 * @param strPrint string to add to frmMain.
 * @see Form#append(java.lang.String)
 * @see Form
 */
protected void printToFrm( String strPrint ) {
    frmMain.append( strPrint );
}
/**
 * Copies message from txtSend to msgSendBuffer.
 * Clears txtSend.
 * @see BtObexMessengerSrv#txtSend
 * @see BtObexMessengerSrv#msgSendBuffer
 */
private void putMsgToSendBuffer() {
    printToFrm( "Adding message to the send buffer ..." );
    msgSendBuffer.append( txtSend.getString() );
    txtSend.setString( "" );
}
/**
 * Gets bytes from msgSendBuffer. Method cleans buffer data after execution
 * if the buffer was not empty.
 * @return array of bytes retrived from msgSendBuffer
 */
private byte[] popMsgFromSendBuf() {
    //getting data from mesasge buffer
    byte[] arrMsgBytes = null;
    synchronized ( msgSendBuffer ) {
        arrMsgBytes = msgSendBuffer.toString().getBytes();
        if(arrMsgBytes.length > 0) {
            msgSendBuffer.delete( 0, msgSendBuffer.length() );
        }
    }
    return arrMsgBytes;
}
/**
 * Defines an event listener that will respond to OBEX requests made to
 * the server.
 * @see ServerRequestHandler

```

Using_Bluetooth_obex_server_in_Java_ME

```
*/
private class RequestHandler extends ServerRequestHandler {
    /**
     * Default constructor.
     */
    public RequestHandler() {
        //No implementation required.
    }
    /**
     * From ServerRequestHandler.
     * Checks whether msgSendBuffer is empty right now.
     * If the buffer is empty than OBEX_HTTP_NO_CONTENT will be returned to
     * the client that initiated GET request.
     * If the buffer contains some data than this method preforms a folowing
     * procedure:
     * - current msgSendBuffer content is converted to byte[];
     * - gets current operation header(HeaderSet);
     * - sets header TYPE to BtObexMessage;
     * - sets header LENGTH equal to length of the byte array containing
     * msgSendBuffer content;
     * - Opens openDataOutputStream for the current operation and writes
     * the byte array there.
     * - Finally OBEX_HTTP_ACCEPTED is being returned.
     * @param op contains the headers sent by the client.
     * @return OBEX_HTTP_OK - if the request handling procedure executed.
     * OBEX_HTTP_NO_CONTENT - if no mesasge found in the msgSendBuffer.
     */
    public int onGet( Operation op ) {
        OutputStream out = null;
        HeaderSet header = null;
        byte[] arrMsgBytes = null;
        int[] arrHeader = null;

        arrMsgBytes = popMsgFromSendBuf();
        if( arrMsgBytes.length == 0 ) {
            return ResponseCodes.OBEX_HTTP_NO_CONTENT;
        }
        //creating a header to send back to the client
        header = createHeaderSet();
        //adding our headers
        //TYPE - represents obex message
        header.setHeader( HeaderSet.TYPE, "BtObexMessage" );
        //LENGTH - holds message length
        header.setHeader( HeaderSet.LENGTH,
            new Long ( arrMsgBytes.length ) );
        try {
            op.sendHeaders( header );
        } catch ( Exception ex ) {
            printToFrm( "Error setting up header!" );
        }
        try {
            // Open the output stream to put the object to it
            out = op.openDataOutputStream();

            // Sending the object to client
            out.write( arrMsgBytes );
            out.flush();

        } catch ( Exception e ) {
            printToFrm( "Error sending a message!" );
        } finally {
            // End the transaction

```

Using_Bluetooth_obex_server_in_Java_ME

```
//closing data stream
if( out != null ) {
    try {
        out.close();
    } catch (Exception e) {
        printToFrm( "Error closing data stream!" );
    }
}
}
return ResponseCodes.OBEX_HTTP_OK;
}
/**
 * From ServerRequestHandler.
 * Checks whether header(HeaderSet) is present in the operation.
 * If it doesn't than OBEX_HTTP_BAD_REQUEST will be send back to
 * the client that initiated a PUT request.
 * If a header detected than funciton performs the following procedure:
 * - LENGTH is being extracted from the header(HeaderSet);
 * - openDataInputStream is opened for current operation;
 * - function reads number of butes specified in LENGTH header.
 * - sequeunce of bytes is converted to String object and printed out
 * to frmMain.
 * - Finally OBEX_HTTP_OK is being returned.
 * @param op contains the headers sent by the client.
 * @return OBEX_HTTP_BAD_REQUEST - if no header(HeaderSet) detected and
 * OBEX_HTTP_OK - if the request handling procedure executed.
 */
public int onPut( Operation op ) {
    HeaderSet header = checkHeader( op );
    if( header == null ) {
        return ResponseCodes.OBEX_HTTP_BAD_REQUEST;
    }
    InputStream in = null;
    int data;
    Long dataLength = null;
    byte[] msg = null;

    // Get the object from the input stream
    try {
        //opening stream to read the incoming message
        in = op.openDataInputStream();
        //retrieving message length
        dataLength = (Long) header.getHeader( HeaderSet.LENGTH );
        msg = new byte[ (int) dataLength.longValue() ];
        //reading data
        data = in.read( msg, 0, (int) dataLength.longValue() );
        if( data > 0 ) {
            printToFrm( "Message: " +
                new String( msg ) );
        }
    } catch (Exception e) {
        printToFrm( "Error reading received data!" );
    } finally {
        // End the transaction
        //closing stream
        try {
            in.close();
        } catch (Exception e) {
            printToFrm( "Error closing data stream!" );
        }
    }
    return ResponseCodes.OBEX_HTTP_OK;
}
```

Using_Bluetooth_obex_server_in_Java_ME

```
    }
    /**
     * Gets header(HeaderSet) from the operation and returns a pointer to
     * it. If there was an error function returns null;
     * @param op contains the headers sent by the client.
     * @return null if header getting procedure threw and exception.
     * A pointer to HeaderSet object will be returned in case of success.
     */
    protected HeaderSet checkHeader(Operation op) {
        HeaderSet header = null;
        try {
            header = op.getReceivedHeaders();
        } catch (Exception e) {
            printToFrm( "Error getting request header!" );
            return null;
        }
        //TODO: add any header checking procedure here

        //header checking procedure
        return header;
    }
    /**
     * From ServerRequestHandler.
     * Called when a DISCONNECT request is received.
     * @param request- contains the headers sent by the client; request
     * will never be null
     * @param reply - the headers that should be sent in the reply;
     * reply will never be null
     */
    public void onDisconnect(HeaderSet request, HeaderSet reply) {
        printToFrm( "Disconnecting..." );
        commandAction( cmdStop, frmMain );
    }
    /**
     * From ServerRequestHandler.
     * Called when a CONNECT request is received.
     * @param request- contains the headers sent by the client;
     * request will never be null
     * @param reply - the headers that should be sent in the reply;
     * reply will never be null
     * @return
     */
    public int onConnect(HeaderSet request, HeaderSet reply) {
        printToFrm( "Client is connecting..." );
        frmMain.addCommand( cmdSend );
        return ResponseCodes.OBEX_HTTP_OK;
    }
}
}
```

Postconditions

After Start command is pressed server is being started. In that moment we can detect this service from any bluetooth device. To connect to the server you should use OBEX client from BtObexMessengerCli snippet.

When user presses 'Send Message' a text box opens. after typing a message 'Send Message' command should be pressed again. By pressing the command user adds the message to the 'send buffer'.

Using_Bluetooth_obex_server_in_Java_ME

This buffer contents will be send on next incoming GET request. Server can be stoped while waiting for incoming connections if user presses Stop command.

See also

[BtObexMessengerCli snippet](#)