Using Bluetooth OBEX Client in Java ME

This code example describes how to use OBEX (Object Exchange Protocol) protocol service via Bluetooth as client-side chat device.

Overview

OBEX is a protocol developed by the Infrared Data Association for “pushing” or “pulling” objects to and from clients and servers through PUT and GET requests.

First of all we discovering devices (Discovering Devices snippet) then discovering service(Discovering Services snippet) with serviceClassID = 2222222222222222222222222222222 (Messenger class ID that unique identifying a service) on selected device.

If service found we get connection string which should be used for establishing the connection. Then getting `ClientSession` object provides methods for OBEX requests and establishing the connection using method `connectionSession.connect()` method. Note, that every request to OBEX Server can be cheked through `ResponseCodes()` method of `HeaderSet` object.

Thereafter we must start `Timer` object that uses `ObexClientTimerTask` (see code in ObexClientTimerTask.java) for periodically sending GET request to Server(Server can't send messages to Client if Client didn't sent GET request. That's why we are using Timer).

Message putting and getting we make using `PutMessage()` and `GetMessage()` methods. `Operation` object using in methods could be only one at once. Thats why we using synchronization of `ClientSession` object.

Preconditions

Bluetooth must be enabled on device before using.

It can't be used independently from server. For testing use Bluetooth OBEX server (look at "See also" part)

Source file: BtObexMessengerCliMidlet.java

```
import java.io.DataInputStream;
import java.io.IOException;
import java.io.OutputStream;

import java.util.Enumeration;
import java.util.Timer;
import java.util.TimerTask;
import java.util.Vector;

import javax.microedition.io.Connector;
import javax.microedition.lcdui.Alert;
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;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.TextBox;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.MIDlet;

import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DataElement;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.DiscoveryAgent;
```


(C) Copyright Nokia 2014. All rights reserved.

```
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.UUID;

import javax.obex.ClientSession;
import javax.obex.HeaderSet;
import javax.obex.Operation;
import javax.obex.ResponseCodes;

/**
 *
 */
public class BtObexMessengerCliMidlet extends MIDlet implements
        CommandListener, DiscoveryListener {
    /**
     * Variable which defined pause status
     */
    private boolean midletPaused = false;
    /**
     * creation PUT request to OBEX Server command
     */
    private Command cmdPut;
    /**
     * Send Message thru the PUT request command
     */
    private Command cmdSendMessage;
    /**
     * Discover devices command
     */
    private Command cmdDiscoverDevices;
    /**
    * Exit from midlet command
    */
    private Command cmdExit;
    /**
    * back to frmMain command
    */
    private Command cmdBack;
    /**
    * disconnect from OBEX server command
    */
    private Command cmdDisconnect;
    /**
    * A List object containing devices list of choices
    */
    private List devicesList;
    /**
    * Flag indicating us that server service is found
    */
    private boolean isServiceFound;
    /**
    * TextBox object containing message text for OBEX server
    */
    private TextBox txtMessageBox;
    /**
```


(C) Copyright Nokia 2014. All rights reserved.

```
    * Main form object for midlet
    */
    private Form frmMain;
    /**
     * Timer obect for periodic sending GET requests to OBEX server.
     */
    private Timer timer;
    /**
    * The DiscoveryAgent for the local Bluetooth device.
    */
    private DiscoveryAgent discoveryAgent;
    /**
    * Keeps track of the devices found
    */
    private Vector devicesVector;
    /**
    * Keeps track of the transaction ID returned from searchServices.
    */
    private int transactionID;
    /**
    * Describes ClassID of Messenger Service
    */
    private String serviceClassIDstring;
    /**
     * Provides methods for OBEX requests
     */
    ClientSession connectionSession;

    /**
     * The BtObexMessengerCliMidlet constructor.
     */
    public BtObexMessengerCliMidlet() {
        try {
            discoveryAgent = LocalDevice.getLocalDevice().getDiscoveryAgent();
        } catch (BluetoothStateException ex) {
            //Failed to get LocalDevice DiscoveryAgent Object.
            //It means that the Bluetooth system could not be initialized
            //TODO: write handler code
        }

        isServiceFound = false;
        serviceClassIDstring = "2222222222222222222222222222222";
        devicesVector = new Vector();
        connectionSession = null;
        cmdPut = new Command("Put", Command.SCREEN, 0);
        cmdDisconnect = new Command("Disconnect", Command.SCREEN, 0);
    }

    /**
     * Initilizes the application.
     * It is called only once when the MIDlet is started.
     */
    private void initialize() {
    }
    /**
     * Performs an action assigned to the Mobile Device - MIDlet Started point.
```


(C) Copyright Nokia 2014. All rights reserved.

```
     */
    public void startMIDlet() {
        switchDisplayable(null, getFormMain());
    }
    /**
     * Performs an action assigned to the Mobile Device - MIDlet Resumed point.
     */
    public void resumeMIDlet() {
    }
    /**
     * Switches a current displayable in a display. The Display instance is
     * taken from getDisplay() method. This method is used by all actions
     * in the design for switching displayable.
     * @param alert the Alert which is temporarily set to the display
     * @param nextDisplayable the Displayable to be set
     */
    public void switchDisplayable(Alert alert, Displayable nextDisplayable) {
        Display display = getDisplay();
        if (alert == null) {
            display.setCurrent(nextDisplayable);
        } else {
            display.setCurrent(alert, nextDisplayable);
        }
    }


    /**
     * From CommandListener.
     * Called by a system to indicated 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 (displayable == frmMain) {
            if (command == cmdExit) {
                exitMIDlet();
            } else if (command == cmdDiscoverDevices) {
                //clearing old devices data
                getDevicesList().deleteAll();
                devicesVector.removeAllElements();
                try {
                    //starting discovery devices process
                    discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
                    isServiceFound = false;
                    printToForm("Start devices search...");
                } catch (BluetoothStateException e) {
                    printToForm("Start device discovery error!");
                    //TODO: write handler code
                }
            } else if (command == cmdPut) {
                //Showing the TextBox for message which is going to be send to
                //OBEX server
                switchDisplayable(null, getTxtMessageBox());
            } else if (command == cmdDisconnect) {
                if(connectionSession != null) {
                    //closing connection with OBEX server
```


(C) Copyright Nokia 2014. All rights reserved.

```
                    HeaderSet header = connectionSession.createHeaderSet();
                    try{
                        //stoping the timer with GET request
                        timer.cancel();
                        //disconnecting
                        connectionSession.disconnect(header);
                        connectionSession.close();
                    } catch (IOException ex) {
                        printToForm(ex.toString());
                        //TODO: write handler code
                    }
                }
            }
        } else if (displayable == devicesList) {
            if (command == cmdBack) {
                //swithing back to frmMain
                switchDisplayable(null, getFormMain());
            } else if (command == List.SELECT_COMMAND) {
                //starting search for Messenger service on selected device
                RemoteDevice currentDevice =
                    (RemoteDevice) devicesVector.elementAt(
                                    devicesList.getSelectedIndex());
                startDiscoverService(currentDevice);
                switchDisplayable(null, getFormMain());
            }
        } else if (displayable == getTxtMessageBox()) {
            if (command == cmdBack) {
                //swithing back to frmMain
                switchDisplayable(null, getFormMain());
            } else if (command == cmdSendMessage) {
                //sending text message in PUT request to OBEX server
                PutMessage(txtMessageBox.getString());
                txtMessageBox.setString("");
                switchDisplayable(null, getFormMain());
            }
        }
    }

    /**
     * get Back command reference
     * @return the initialized component instance
     */
    public Command getBackCommand() {
        if (cmdBack == null) {
            cmdBack = new Command("Back to Form", Command.ITEM, 0);
        }
        return cmdBack;
    }

    /**
     * Returns an initiliazed instance of frmMain component.
     * @return the initialized component instance
     */
    public Form getFormMain() {
        if (frmMain == null) {
            frmMain = new Form("BtObexMessengerCli", new Item[] { });
```


(C) Copyright Nokia 2014. All rights reserved.

```
            cmdExit = new Command("Exit", Command.EXIT, 0);
            frmMain.addCommand(cmdExit);

            cmdDiscoverDevices = new Command("Devices", Command.SCREEN, 0);
            frmMain.addCommand(cmdDiscoverDevices);

            frmMain.setCommandListener(this);
            timer = new Timer();
        }
        return frmMain;
    }

    /**
     * Returns an initiliazed instance of List component.
     * @return the initialized component instance
     */
    public List getDevicesList() {
        if (devicesList == null) {
            devicesList = new List("OBEX Demo", List.IMPLICIT);
            devicesList.addCommand(getBackCommand());
            devicesList.setCommandListener(this);
        }
        return devicesList;
    }

    /**
     * Returns an initiliazed instance of TextBox component.
     * @return the initialized component instance
     */
    public TextBox getTxtMessageBox() {
        if (txtMessageBox == null) {
            txtMessageBox = new TextBox("Message", null, 128, TextField.ANY);
            cmdSendMessage = new Command("Send", Command.ITEM, 0);

            txtMessageBox.addCommand(cmdSendMessage);
            txtMessageBox.addCommand(getBackCommand());
            txtMessageBox.setCommandListener(this);
        }
        return txtMessageBox;
    }

    /**
     * Returns a display instance.
     * @return the display instance.
     */
    public Display getDisplay () {
        return Display.getDisplay(this);
    }

    /**
     * Exits MIDlet.
     */
    public void exitMIDlet() {
        switchDisplayable (null, null);
        destroyApp(true);
        notifyDestroyed();
```


(C) Copyright Nokia 2014. All rights reserved.

```
    }

    /**
     * From MIDlet.
     * Called when MIDlet is started.
     * Checks whether the MIDlet have been already started and
     * initializes/starts or resumes the MIDlet.
     */
    public void startApp() {
        if (midletPaused) {
            resumeMIDlet ();
        } else {
            initialize ();
            startMIDlet ();
        }
        midletPaused = false;
    }

    /**
     * From MIDlet.
     * Called when MIDlet is paused.
     */
    public void pauseApp() {
        midletPaused = true;
    }

    /**
     * From MIDlet.
     * Called to signal the MIDlet to terminate.
     * @param unconditional if true, then the MIDlet has to be unconditionally
     * terminated and all resources has to be released.
     */
    public void destroyApp(boolean unconditional) {
    }

    /**
    * From DiscoveryListener.
    * Called when a device was found during an inquiry. An inquiry
    * searches for devices that are discoverable. The same device may
    * be returned multiple times.
    * @see DiscoveryAgent#startInquiry
    * @param btDevice the device that was found during the inquiry
    * @param cod the service classes, major device class, and minor
    * device class of the remote device being returned
    */
    public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {
        //adding new device object to Vector object
        devicesVector.addElement(btDevice);
    }

    /**
    * From DiscoveryListener.
    * Called when service(s) are found during a service search.
    * This method provides the array of services that have been found.
    * Trying to connect to if Messenger service has been found.
    * @param transID the transaction ID of the service search that is
```


(C) Copyright Nokia 2014. All rights reserved.

```
    * posting the result
    * @param service a list of services found during the search request
    */
    public void servicesDiscovered(int transID, ServiceRecord[] serviceRecords){
        for(int i = 0; i < serviceRecords.length; i++) {
            DataElement rcvObject;

            //getting ServiceClassID as DataElement object
            rcvObject = serviceRecords[i].getAttributeValue(
                    ServiceRecord.ID_ServiceClassIDList);

            if(rcvObject!=null) {
                //rcvObject presents as enumeration of DataElement
                //getting enumeration...
                Enumeration enumeration = (Enumeration)rcvObject.getValue();

                for (Enumeration e = enumeration; e.hasMoreElements() ;) {
                    DataElement subDataElement = (DataElement)e.nextElement();
                    //...and converting every element of enumeration to UUID
                    UUID uuid = (UUID) subDataElement.getValue();

                    //comparing needed UUID with present
                    if(serviceClassIDstring.equals(uuid.toString())) {
                        printToForm(" Service have found!");
                        isServiceFound = true;
                        //if UUIDs are the same then trying to connect
                        //getting the connection string for connection
                        String connectionString =
                            serviceRecords[i].getConnectionURL(
                                ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
                        printToForm(connectionString);

                        try {
                            //getting the client connection object
                            connectionSession = (ClientSession) Connector.open(
                                                connectionString,
                                                Connector.READ_WRITE );

                            //trying to connect
                            printToForm(" connect...!");
                            HeaderSet headerSet =
                                    connectionSession.connect(null);

                            if (headerSet.getResponseCode() ==
                                    ResponseCodes.OBEX_HTTP_OK) {
                                printToForm("Request OK");
                            } else {
                                printToForm("Request Failed. code: " +
                                        headerSet.getResponseCode());
                                connectionSession.close();
                                return;
                            }

                            //if connection established we add some commands
                            //for sending message and disconnecting from
                            //server
                            frmMain.addCommand(cmdPut);
```


(C) Copyright Nokia 2014. All rights reserved.

```
                        frmMain.addCommand(cmdDisconnect);
                        //TimerTask object creating for periodic
                        //asynchronous sending GET requests to server
                        ObexClientTimerTask timerTask =
                                new ObexClientTimerTask(this);
                        //timer starting
                        timer.schedule((TimerTask)timerTask, 1000);
                        return;
                    } catch (IOException ex) {
                        printToForm(ex.toString());
                    }
                }
            }
        }
    }
}

/**
* From DiscoveryListener.
* The following method is called when a service search is completed or
* was terminated because of an error.
* @param transID the transaction ID identifying the request which
* initiated the service search
* @param respCode the response code which indicates the
* status of the transaction; guaranteed to be one of the
* aforementioned only
*/
public void serviceSearchCompleted(int transID, int respCode) {
    if(transactionID == transID) {
        switch(respCode) {
            case DiscoveryListener.SERVICE_SEARCH_COMPLETED:
                if(isServiceFound) {
                    printToForm("  Service discovering Complete!");
                } else {
                    printToForm("  Service discovering incomplete! " +
                            "Service not found!");
                }
                break;
            case DiscoveryListener.SERVICE_SEARCH_TERMINATED:
                printToForm("  Services discovering terminated!");
                break;
            case DiscoveryListener.SERVICE_SEARCH_ERROR:
                printToForm("  Sevice discovering error!");
                break;
            default:
                break;
        }
        transactionID = -1;
        return;
    } else {
        //if transactionID are not the same as transID
        //it's an error
        //TODO: write handler code
        printToForm("transactionID not right...");
    }
    printToForm("Discover Service Complete...");
```


(C) Copyright Nokia 2014. All rights reserved.

```
    }

    /**
    * From DiscoveryListener.
    * Called when a device discovery transaction is
    * completed.
    * @param discType the type of request that was completed
    */
    public void inquiryCompleted(int discType){
        switch(discType) {
            case INQUIRY_COMPLETED:
                int devcesCount = devicesVector.size();
                if(devcesCount == 0) {
                    printToForm("There no devices! Trying do again later...");
                    return;
                }

                //before creating new elements in devicesList we must clear
                //old values
                getDevicesList().deleteAll();

                for(int i = 0; i < devcesCount; i++) {
                    String deviceName = "Noname";
                    try {
                        deviceName = ((RemoteDevice)
                            devicesVector.elementAt(i)).getFriendlyName(false);
                    } catch (IOException e) {
                        printToForm(e.toString());
                        return;
                    }
                    devicesList.append(deviceName, null);
                }
                printToForm("Discover devices complete!");

                //swithing to devicesList where we can choose the device
                //for service discover
                switchDisplayable(null, getDevicesList());
                break;
            case INQUIRY_TERMINATED:
                printToForm("Discover devices terminated!");
                break;
            case INQUIRY_ERROR:
                printToForm("Discover devices error!");
                break;
        }
    }

    /**
     * Adds a StringItem to the frmMain.
     * @param strPrint string to add to frmMain.
     * @see Form#append(java.lang.String)
     * @see Form
     */
    public void printToForm( String strPrint ) {
        frmMain.append( strPrint );
    }
```


(C) Copyright Nokia 2014. All rights reserved.

```
    /**
     * Used in commandAction()
     * Start nedeed OBEX service searching on current device
     * @param currentDevice the RemoteDevice. Current device
     * for searching
     */
    private void startDiscoverService(RemoteDevice currentDevice) {
        if(currentDevice == null) {
            throw new NullPointerException("Can't get RemoteDevice object!");
        }
        //UUID for OBEX protocol
        UUID[] searchList = new UUID[1];
        searchList[0] = new UUID(0x0008);

        //Initialization of service attributes whose values will be
        //retrieved on services which have the UUIDs specified
        //in searchList
        int[] attributesList = new int[2];
        attributesList[0] = ServiceRecord.ID_ServiceName;
        attributesList[1] = ServiceRecord.ID_ServiceClassIDList;

        //start searching services on current device
        //and get transaction ID of the service search
        try {
            transactionID = discoveryAgent.searchServices(
                attributesList, searchList, currentDevice, this);
            printToForm("Starting search service!");
        } catch (BluetoothStateException e) {
            //Failed to start the search on this device, try another
            //device.
            //TODO: write handler code
        }
    }
    /**
     * Used by ObexClientTimerTask for sending GET request to server
     * checking response and restarting the timer
     */
    public void GetMessageOnTimer() {
        GetMessage();
        ObexClientTimerTask timerTask = new ObexClientTimerTask(this);
        timer.schedule((TimerTask)timerTask, 1500);
    }
    /**
     * Used for creating PUT request to OBEX Server.
     * Sending text message to server.
     */
    private void PutMessage(String messageString) {
        //we can use only one Operation object at one time
        //it mean's that when we send PUT request we can't
        //send GET request
        synchronized(connectionSession) {
            //header creating for PUT request
            HeaderSet head = connectionSession.createHeaderSet();

            try {
                //setting up type and length of message inside header
```


(C) Copyright Nokia 2014. All rights reserved.

```
                head.setHeader(HeaderSet.TYPE, "BtObexMessage");
                head.setHeader(HeaderSet.LENGTH,
                        new Long(messageString.length()));
            } catch (IllegalArgumentException ex) {
                printToForm(ex.toString());
                //TODO: write handler code
                return;
            }

            Operation operation;
            try {
                //creating new Operation object for current connection
                operation = connectionSession.put(head);
            } catch (IOException ex) {
                printToForm(ex.toString());
                return;
            }

            try {
                // Issue the request
                // Open the output stream to put the object to it
                OutputStream out = operation.openOutputStream();

                // Send the object to the server
                printToForm("_" + messageString + "_");
                out.write(messageString.getBytes());
                // End the transaction
                out.flush();
                // Close the stream
                out.close();
            } catch(Exception ex) {
                //TODO: write handler code
            } finally {
                try {
                    //close the operation
                    operation.close();
                } catch (IOException ex) {
                    //TODO: write handler code
                }
            }
        }

    }
    /**
     * Used for creating GET request to OBEX Server.
     * Getting text message from server.
     */
    public void GetMessage() {
        //we can use only one Operation object at one time
        //it mean's that when we send GET request we can't
        //sending PUT request
        synchronized(connectionSession) {
            //header creating for GET request
            HeaderSet head = connectionSession.createHeaderSet();
            try {
                // Set the type of the object to retrieve
                head.setHeader(HeaderSet.TYPE, "BtObexMessage");
```


(C) Copyright Nokia 2014. All rights reserved.

```
            head.setHeader(HeaderSet.LENGTH, new Long(0));
        } catch (IllegalArgumentException ex) {
            printToForm(ex.toString());
            //TODO: write handler code
            return;
        }

        Operation operation;
        try {
            //creating new Operation object for current connection
            operation = connectionSession.get(head);
        } catch (IOException ex) {
            printToForm(ex.toString());
            return;
        }

        try {
            // getting header...
            HeaderSet getHeader = operation.getReceivedHeaders();
            if(getHeader == null) {
                printToForm(" getHeader == null ");
                operation.close();
                return;
            }

            // ...and checking the response code
            while(true) {
                int responseCode = getHeader.getResponseCode();
                if(responseCode == ResponseCodes.OBEX_HTTP_OK) {
                    //it's Ok! getting the message
                    break;
                }
                if(responseCode == ResponseCodes.OBEX_HTTP_NO_CONTENT) {
                    //it's no content! wait for next message
                    //printToForm("OBEX_HTTP_NO_CONTENT");
                    operation.close();
                    return;
                }
                try {
                    //waiting two seconds for next attemp to getting
                    //response code
                    Thread.sleep(2000);
                } catch (InterruptedException ex) {
                    printToForm(ex.toString());
                    //TODO: write handler code
                    }
            }


            //getting the headerlist where we have additional information
            //about message, like length
            int[] fieldsIDs = getHeader.getHeaderList();
            if(fieldsIDs == null) {
                printToForm(" header on GET response is empty");
                operation.close();
                return;
```


(C) Copyright Nokia 2014. All rights reserved.

```
                }

                // Get the object from the input stream
                DataInputStream in = operation.openDataInputStream();
                if(in == null) {
                    printToForm(" error input stream ");
                    operation.close();
                    return;
                }

                //getting lenght of message
                Long stringLength =
                        (Long) getHeader.getHeader(HeaderSet.LENGTH);
                //creating string object
                byte[] object = new byte[ (int)stringLength.longValue() ];
                in.read(object);
                String objectString = new String(object);
                printToForm( objectString );
                //closing input stream
                in.close();
                //closing current operation
                operation.close();
            } catch (IOException ex) {
                printToForm(ex.toString());
                //TODO: write handler code
            }
        }
    }
}
```

Source file: ObexClientTimerTask.java

```
import java.util.TimerTask;

/**
 * A task object that can be scheduled for one-time
 * or repeated execution by a Timer
 * Used in BtObexMessengerCliMidlet object for sending GET requests
 * to OBEX Server
 */
public class ObexClientTimerTask extends TimerTask {
    /**
     * This midlet used for calling GetMessageOnTimer() which create requests
     * to OBEX server
     */
    BtObexMessengerCliMidlet parentMidlet;

    /**
     * Constructor
     * @param parent is BtObexMessengerCliMidlet. Used for calling
     * GetMessageOnTimer() function in run().
     */
    public ObexClientTimerTask(BtObexMessengerCliMidlet parent) {
        parentMidlet = parent;
```


(C) Copyright Nokia 2014. All rights reserved.

```
    }

    /**
     * From TimerTask.
     * The action to be performed by this timer task.
     */
    public void run() {
        //calling function that create GET reqest to OBEX Server
        parentMidlet.GetMessageOnTimer();
        }
}
```

Postconditions

Midlet implements client-side of Obex chat.

See also

- Java ME Bluetooth OBEX Messenger Server

(C) Copyright Nokia 2014. All rights reserved.