Simple NFC LLCP peer-to-peer Chat

Example of NFC LLCP communication in the Nokia 6212 classic. User can exchange messages with other phones by touching them. By default MIDlet is in reading state and when user wants to write something MIDlet goes to writing state.

Note that LLCPConnection class used in this example is proprietary implementation for Nokia 6212 classic so this example might not work in any other phone.

```
package com.nokia.nfc.sample.app;

import java.io.IOException;

import javax.microedition.io.Connector;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
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.TextBox;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

import com.nokia.nfc.llcp.LLCPConnection;
import com.nokia.nfc.llcp.LLCPConnectionListener;
import com.nokia.nfc.llcp.LLCPLinkListener;
import com.nokia.nfc.llcp.LLCPManager;
import javax.microedition.lcdui.StringItem;

public class NfcChat extends MIDlet implements CommandListener, LLCPConnectionListener, LLCPLinkListener {

    /**
     * LLCP manager
     */
    private LLCPManager llcpManager = null;
    /**
     * LLCP connections
     *
     * @note
     * Both inbound (connection received from connectionOpened() method) and
     * outbound connection (created after linkEstablished() method call)
     * needs to be created.
     */
    private LLCPConnection inbound;
    private LLCPConnection outbound;
    /**

    /**
     * Connection parameters
     *
     * @note
     * Connection URI to open a Type 2 connection that uses service identifier
     * (SID) 35. Note that PID mentioned in the SDK javadocs is not valid, use SID
     * instead.
     * Tested with SID 35 on the C7; Symbian supports SIDs from 32 - 63.
     * Use SIDs between LLCPConnection.MIN_SERVICE_ID and MAX_SERVICE_ID on Series 40.
     */
    private static final byte SID = 35;

    /**
     * Type 1 is connectionless, type 2 is connection-oriented.
     */
    private static final byte TYPE = LLCPConnection.TYPE_2;
    /**
     * Connection data buffers
     */
    private byte[] sendBuffer = null;
    private byte[] receiveBuffer = null;
    /**
     * UI elements
     */
    private Display display = null;
    private Form form = null;
    private TextBox textTextBox = null;
    /**
     * Commands
     */
    private Command writeCmd = null;
    private Command cancelCmd = null;
    private Command exitCmd = null;
    /**
     * "State machine" - is the MIDlet in reading or writing state
     */
    boolean reading = true;

    public NfcChat() {
        System.out.println("\n\n NfcChat started");

        // Initialize UI
        display = Display.getDisplay(this);

        form = new Form("NfcChat");

        writeCmd = new Command("Write", Command.ITEM, 1);
        cancelCmd = new Command("Cancel", Command.CANCEL, 1);
        exitCmd = new Command("Exit", Command.EXIT, 1);

        form.addCommand(writeCmd);
        form.addCommand(exitCmd);
        form.setCommandListener(this);

        display.setCurrent(form);
    }

    /**
```


(C) Copyright Nokia 2013. All rights reserved.

```
     * This is called when MIDlet is closing or entering to paused state
     */
    private void close() {
        System.out.println("close");

        llcpManager.stopListening(TYPE, SID, this);
        llcpManager.removeLinkListener(this);

        inbound = closeConn(inbound);
        outbound = closeConn(outbound);
    }

    /**
     * This is called by both inbound and outbound connection handlers to
     * close existing connections.
     *
     * @param conn
     * @return null when connection is closed
     */
    private final LLCPConnection closeConn(LLCPConnection conn) {
        System.out.println("closeConn");

        if (conn != null) {
            try {
                conn.close();
            } catch (IOException e) {
                logMessage("Error closing connection : " + e.getMessage());
            }
        }
        return null;
    }

    /*
     * (non-Javadoc)
     * @see javax.microedition.lcdui.CommandListener#commandAction(javax.microedition.lcdui.Command, javax.microedition.lcdui.Displa
     */
    public void commandAction(Command c, Displayable d) {
        System.out.println("commandAction");

        if (c == writeCmd && d == form) {
            displayWritingTextBox();
        } else if (c == writeCmd && d == textTextBox) {
            write();
        } else if (c == cancelCmd) {
            read();
        } else if (c == exitCmd) {
            notifyDestroyed();
        }
    }

    /*
     * (non-Javadoc)
     * @see com.nokia.nfc.llcp.LLCPConnectionListener#connectionOpened(com.nokia.nfc.llcp.LLCPConnection)
     */
    public void connectionOpened(LLCPConnection conn) {
        logMessage("Connection opened");
        if (inbound == null) {
            inbound = conn;
            handleIn();
        } else {
            closeConn(conn);
        }
    }

    /**
     * This is called to display the text writing box
     */
    private void displayWritingTextBox() {
        System.out.println("writeTb");

        textTextBox = new TextBox("Text to write", null, 255, TextField.ANY);
        textTextBox.addCommand(writeCmd);
        textTextBox.addCommand(cancelCmd);
        textTextBox.setCommandListener(this);

        display.setCurrent(textTextBox);

    }

    /*
     * (non-Javadoc)
     * @see javax.microedition.midlet.MIDlet#destroyApp(boolean)
     */
    protected void destroyApp(boolean unconditional) throws MIDletStateChangeException {
        System.out.println("destroyApp");
        close();
    }

    /**
     * Inbound connection handler
     */
    private final void handleIn() {
        logMessage("Inbound connection");

        new Thread() {

            public void run() {

                System.out.println("handleIn begin");

                try {
                    if (reading) {
                        sendData(inbound, "-1".getBytes());
                        byte[] data = receiveData(inbound);
                        handleReceivedData(data);
                    } else {
                        sendData(inbound, sendBuffer);
                        byte[] data = receiveData(inbound);
                        if (new String(data).equals("-1")) {
                            handleReceivedData(data);
                        }
                    }
                } catch (Exception e) {
                    logMessage("Error handling incoming connection : " + e.getMessage());
                }

                System.out.println("handleIn end");

            }
        }.start();
```


(C) Copyright Nokia 2013. All rights reserved.

```
    }

    /**
     *
     * @param uid
     */
    private final void handleOut(final String uid) {
        logMessage("Outgoing connection");

        new Thread() {

            public void run() {

                System.out.println("handleOut begin");
                logMessage("nfc:llcp;type=" + TYPE + ";sid=" + SID + ";uid=" + uid);

                try {
                    outbound = (LLCPConnection) Connector.open("nfc:llcp;type=" + TYPE + ";sid=" + SID + ";uid=" + uid);
                    logMessage("Opened outgoing connection");

                    if (reading) {
                        sendData(outbound, "-1".getBytes());
                        byte[] data = receiveData(outbound);
                        handleReceivedData(data);
                    } else {
                        sendData(outbound, sendBuffer);
                        byte[] data = receiveData(outbound);
                        if (new String(data).equals("-1")) {
                            handleReceivedData(data);
                        }
                    }

                } catch (IOException e) {
                    logMessage("Error handling outgoing connection : " + e.getMessage());
                }

                System.out.println("handleOut end");

            }
        }.start();
    }

    private synchronized void handleReceivedData(byte[] data) {
        logMessage("Handle received data");

        /*
         * Handle only first arrived respose from either inbound or
         * outbound connection handlers
         */
        if (receiveBuffer == null) {
            receiveBuffer = data;
        } else {
            return;
        }

        if (reading) {
            if (!new String(receiveBuffer).equals("-1")) {
                logMessage(">> " + new String(receiveBuffer) + "\n");
            }
        } else {
            if (sendBuffer != null) {
                logMessage("<< " + new String(sendBuffer) + "\n");
                read();
            }
        }
    }

    /*
     * (non-Javadoc)
     * @see com.nokia.nfc.llcp.LLCPLinkListener#linkEstablished(java.lang.String)
     */
    public void linkEstablished(String uid) {
        logMessage("Link established");

        receiveBuffer = null;
        handleOut(uid);

    }

    /*
     * (non-Javadoc)
     * @see com.nokia.nfc.llcp.LLCPLinkListener#linkLost()
     */
    public void linkLost() {
        form.append("Link lost");

        outbound = closeConn(outbound);
        inbound = closeConn(inbound);

    }

    /**
     * This is called when MIDlet has started or resumed from paused state
     */
    private void open() {
        form.append("Open LLCP connection, SID " + SID);

        try {
            llcpManager = LLCPManager.getInstance();
        } catch (Exception e) {
            logMessage("Failed to get LLCPManager instance : " + e.getMessage());
            return;
        }

        llcpManager.addLinkListener(this);
        llcpManager.startListening(TYPE, SID, this);
    }

    /*
     * (non-Javadoc)
     * @see javax.microedition.midlet.MIDlet#pauseApp()
     */
    protected void pauseApp() {
        System.out.println("pauseApp");

        close();
    }

    private void logMessage(String msg) {
        StringItem si = new StringItem(null, msg + "\n");
```


(C) Copyright Nokia 2013. All rights reserved.

```
        form.append(si);
    }

    /**
     *
     */
    private void read() {
        System.out.println("read");

        sendBuffer = null;
        receiveBuffer = null;

        reading = true;
        display.setCurrent(form);
    }

    /**
     *
     * @param conn
     * @return
     * @throws IOException
     */
    private final byte[] receiveData(LLCPConnection conn) throws IOException {
        System.out.println("receiveData");

        byte[] recv = conn.receive();
        logMessage("Received " + new String(recv));
        return recv;
    }

    /**
     *
     * @param conn
     * @param data
     * @throws IOException
     */
    private final void sendData(LLCPConnection conn, byte[] data) throws IOException {
        System.out.println("sendData");

        if (data == null) {
            data = "\n".getBytes();
        }

        conn.send(data);
        logMessage("Sent " + new String(data));
    }

    /*
     * (non-Javadoc)
     * @see javax.microedition.midlet.MIDlet#startApp()
     */
    protected void startApp() throws MIDletStateChangeException {
        System.out.println("startApp");

        open();
    }

    private void write() {
        System.out.println("write");

        reading = false;
        sendBuffer = textTextBox.getString().getBytes();

        // Show alert
        Alert a = new Alert("Writing", "Touch to write", null, AlertType.INFO);
        a.addCommand(cancelCmd);
        a.setTimeout(Alert.FOREVER);
        a.setCommandListener(this);
        display.setCurrent(a);
    }
}
```


(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.