

Simple_NFC_LLCP_peer-to-peer_Chat

Example of NFC LLCP communication in 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;

public class LLCPChat 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 e to open a Type 2 connection that uses service identifier
     * (SID) 7. Note that PID mentioned in the SDK javadocs is not valid, use SID
     * instead
     */
    private static final byte SID = 7;
    private static final byte TYPE = LLCPConnection.TYPE_2;

    /**
     * Connection data buffers
     */
}
```

Simple_NFC_LLCP_peer-to-peer_Chat

```
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 LLCPChat() {
    System.out.println("\n\n LLCPChat started");

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

    form = new Form("LLCPChat");

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

    addCommand(writeCmd);
    addCommand(cancelCmd);
    setCommandListener(this);

    setCurrentDisplay(form);
}

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

    stopListening(TYPE, SID, this);
    removeChangeListener(this);

    = closeConnection();
    = closeConnection();
}

/**
```

Simple_NFC_LLCP_peer-to-peer_Chat

```
* 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 {
            close();
            conn.
        } catch (IOException e) {
            System.out.println("Error closing connection : " + e.getMessage());
        }
    }
    return null;
}

/*
 * (non-Javadoc)
 * @see javax.microedition.lcdui.CommandListener#commandAction(javax.microedition.lcdui.C
 */
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.LL
 */
public void connectionOpened(LLCPConnection conn) {
    System.out.println("connectionOpened");
    if (inbound == null) {
        = conn; inbound
        (); handleIn
    } else {
        (conn); closeConn
    }
}

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

    = new TextFieldTextField("Text to write", null, 255, TextField.ANY);
    addCommand(writeCmdwriteCmd);
    addCommand(cancelCmdcancelCmd);
    setCommandListener(this);
}
```

Simple_NFC_LLCP_peer-to-peer_Chat

```
        setCurrentDisplay(textBox);
    }

    /*
     * (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() {
        System.out.println("handleIn");

        new Thread() {
            public void run() {

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

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

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

    /**
     *
     * @param uid
     */
    private final void handleOut(final String uid) {
        System.out.println("handleOut");

        new Thread() {
            public void run() {

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

                try {
                    = (LLCPConnection) Connector.open("bina:llcp;type=" + TYPE + ";sid=" + SID + ";uid=" + uid);
                    System.out.println("Opened outgoing connection");

                    if(reading) {
                        (outbound, "-1".getBytes());                sendData
                    }
                }
            }
        }
    }
}
```

Simple_NFC_LLCP_peer-to-peer_Chat

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

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

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

}
}.start();
}

/**
 *
 * @param data
 */
private synchronized void handleReceivedData(byte[] data) {
System.out.println("handleReceivedData");

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

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

/*
 * (non-Javadoc)
 * @see com.nokia.nfc.llcp.LLCPListener#linkEstablished(java.lang.String)
 */
public void linkEstablished(String uid) {
System.out.println("linkEstablished");

    = receiveBuffer
    (uid); handleOut
}

/*
 * (non-Javadoc)
 * @see com.nokia.nfc.llcp.LLCPListener#linkLost()
 */
public void linkLost() {
```

Simple_NFC_LLCP_peer-to-peer_Chat

```
System.out.println("linkLost");

        = closeConn(conn);
        = closeConn(conn);

    }

/**
     * This is called when MIDlet has started or resumed from paused state
     */
private void open() {
    System.out.println("open");

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

        addLLCPManager(this);
        startLLCPManager(TYPE, SID, this);

    }

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

        ();        close
    }

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

        = null; endBuffer
        = null; liveBuffer

    = true; reading
    setCurrentDevice();

}

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

    byte[] recv = conn.receive();
    System.out.println("Received " + new String(recv));
    return recv;
}
```

Simple_NFC_LLCP_peer-to-peer_Chat

```
}

/**
 *
 * @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();

    send(data);conn.
System.out.println("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");

    = false;reading
    = textArea.getText().getBytes();

// Show alert
    = new Alert("Waiting", "Touch to write", null, AlertType.INFO);
addCommand(cancelCmd);
setTimeout(Alert.FOREVER);
setCommandListener(this);
setCurrentDisplay.

}
}
```