

Write_and_read_Bluetooth_NDEF_tag

MIDlet to write and read **Bluetooth** NDEF tags. User can also search for **Bluetooth** devices MAC address and create a tag for it.

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

import java.io.IOException;
import java.util.Vector;

import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.ServiceRecord;
import javax.microedition.contactless.ContactlessException;
import javax.microedition.contactless.DiscoveryManager;
import javax.microedition.contactless.TargetListener;
import javax.microedition.contactless.TargetProperties;
import javax.microedition.contactless.TargetType;
import javax.microedition.contactless.ndef.NDEFMessage;
import javax.microedition.contactless.ndef.NDEFRecord;
import javax.microedition.contactless.ndef.NDEFRecordType;
import javax.microedition.contactless.ndef.NDEFTagConnection;
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.List;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

/**
 * MIDlet to write and read Bluetooth NDEF tags. User can also search for
 * Bluetooth devices MAC address and create a tag for it.
 */
public class BtNdefManager extends MIDlet implements TargetListener,
    CommandListener, DiscoveryListener {

    private Display display;
    private Form form;
    private TextField configTypeTf;
    private TextField btAddressTf;
    private TextField btClassTf;
    private TextField authInfoTf;
    private TextField shortNameTf;
    private Command writeCmd;
    private Command cancelCmd;
    private Command discoverCmd;
    private Command selectCmd;
    private Command exitCmd;

    private DiscoveryAgent discoveryAgent;
    private Vector devices;

    private List deviceList;
```

Write_and_read_Bluetooth_NDEF_tag

```
// Boolean to tell if MIDlet is on writing mode
private boolean write = false;

protected void destroyApp(boolean unconditional)
    throws MIDletStateChangeException {
}

protected void pauseApp() {
}

protected void startApp() throws MIDletStateChangeException {
    // Get instance of NFC Discovery Manager
    DiscoveryManager dm = DiscoveryManager.getInstance();

    // Register NDEF_TAG target to discovery
    try {
        dm.addTargetListener(this, TargetType.RFID_TAG);
    } catch (IllegalStateException e) {
        // Catch IllegalStateException
    } catch (ContactlessException e) {
        // Catch ContactlessException
    }

    // Initialize and show user interface elements
    display = Display.getDisplay(this);
    form = new Form("ReadSerial");

    configTypeTf = new TextField("configType:", "00", 2, TextField.ANY);
    btAddresTf = new TextField("btAddres:", "", 12, TextField.ANY);
    btClassTf = new TextField("btClass:", "000680", 6, TextField.ANY);
    authInfoTf = new TextField("authInfo:",
        "00000000000000000000000000000000", 32, TextField.ANY);
    shortNameTf = new TextField("shortName:", "BT", 255, TextField.ANY);

    form.append(configTypeTf);
    form.append(btAddresTf);
    form.append(btClassTf);
    form.append(authInfoTf);
    form.append(shortNameTf);

    writeCmd = new Command("Write", Command.OK, 1);
    cancelCmd = new Command("Cancel", Command.CANCEL, 1);
    discoverCmd = new Command("Discover", Command.ITEM, 1);
    selectCmd = new Command("Select", Command.OK, 1);
    exitCmd = new Command("Exit", Command.EXIT, 1);
    form.addCommand(writeCmd);
    form.addCommand(discoverCmd);
    form.addCommand(exitCmd);
    form.setCommandListener(this);
    display.setCurrent(form);
}

public void targetDetected(TargetProperties[] properties) {
    if (!write) {
        // Read
        NDEFTagConnection conn = null;
        try {
            conn = (NDEFTagConnection) Connector.open(properties[0]
                .getUrl());

            NDEFMessage message = conn.readNDEF();
            if (message != null) {

```

Write_and_read_Bluetooth_NDEF_tag

```
NDEFRecord[] records = message.getRecords();
for (int i = 0; i < records.length; i++) {
    byte[] record = records[i].toByteArray();

    byte[] typeName = new byte[12];
    System.arraycopy(record, 3, typeName, 0, 12);

    byte[] configType = new byte[] { record[15] };

    byte[] btAddress = new byte[6];

    System.arraycopy(record, 16, btAddress, 0, 6);

    byte[] btClass = new byte[3];
    System.arraycopy(record, 22, btClass, 0, 3);

    byte[] authInfo = new byte[16];
    System.arraycopy(record, 25, authInfo, 0, 16);

    byte[] shortName = new byte[record[41]];
    System.arraycopy(record, 42, shortName, 0, record[41]);

    configTypeTf.setString(toHexString(configType));
    btAddressTf.setString(toHexString(btAddress));
    btClassTf.setString(toHexString(btClass));
    authInfoTf.setString(toHexString(authInfo));
    shortNameTf.setString(new String(shortName));

}
} else {
    // No BT record found
}
} catch (Exception e) {
} finally {
    try {
        if (conn != null) {
            conn.close();
        }
    } catch (IOException e) {
    }
}
} else {
    // Write
    NDEFTagConnection conn = null;
    try {
        conn = (NDEFTagConnection) Connector.open(properties[0]
            .getUrl());
        byte payload[] = new byte[1 + 6 + 3 + 16 + 1
            + shortNameTf.size()];

        System.arraycopy(toByteArray(configTypeTf.getString()), 0,
            payload, 0, 1);
        System.arraycopy(toByteArray(btAddressTf.getString()), 0,
            payload, 1, 6);
        System.arraycopy(toByteArray(btClassTf.getString()), 0,
            payload, 7, 3);
        System.arraycopy(toByteArray(authInfoTf.getString()), 0,
            payload, 10, 16);
        payload[26] = (byte) shortNameTf.size();
        System.arraycopy(shortNameTf.getString().getBytes(), 0,
            payload, 27, shortNameTf.size());
    }
}
```

Write_and_read_Bluetooth_NDEF_tag

```

NDEFRecord[] records = new NDEFRecord[] { new NDEFRecord(
    new NDEFRecordType(NDEFRecordType.EXTERNAL_RTD,
        "urn:nfc:ext:nokia.com:bt"), null, payload) };
NDEFMessage message = new NDEFMessage(records);
conn.writeNDEF(message);
form.append("Written");

form.removeCommand(cancelCmd);
form.addCommand(writeCmd);
write = false;
display.setCurrent(new Alert("Alert", "Written", null,
    AlertType.INFO), form);
} catch (ContactlessException e) {
    form.append(e.toString());
} catch (IOException e) {
    form.append(e.toString());
} finally {
    if (conn != null) {
        try {
            conn.close();
        } catch (IOException e) {
        }
    }
}
}

private static String toHexString(byte[] value) {
    if (value == null)
        return "null";

    StringBuffer result = new StringBuffer();

    for (int i = 0; i < value.length; i++) {
        result.append(Integer.toHexString(value[i] >>> 4 & 0x0F));
        result.append(Integer.toHexString(value[i] & 0xF));
    }
    return result.toString();
}

private static byte[] toByteArray(String string) {
    if (string.length() % 2 != 1) {
        byte[] result = new byte[string.length() / 2];
        for (int i = 0; i < result.length; i++) {
            result[i] = (byte) Integer.parseInt(string.substring(2 * i,
                2 * i + 2), 16);
        }
        return result;
    } else {
        return null;
    }
}

public void commandAction(Command c, Displayable d) {

    if (d == form) {
        if (c == cancelCmd || c == writeCmd) {
            if (write) {
                form.removeCommand(cancelCmd);
                form.addCommand(writeCmd);
            }
        }
    }
}

```

Write_and_read_Bluetooth_NDEF_tag

```
        } else {
            form.removeCommand(writeCmd);
            form.addCommand(cancelCmd);
        }
        write = !write;
    } else if (c == discoverCmd) {
        write = false;
        form.removeCommand(cancelCmd);
        form.removeCommand(writeCmd);
        form.removeCommand(discoverCmd);
        btSearch();
    } else if (c == exitCmd) {
        notifyDestroyed();
    }
} else if (d == deviceList) {
    if (c == selectCmd) {
        RemoteDevice btDevice = (RemoteDevice) devices
            .elementAt(deviceList.getSelectedIndex());
        btAddressTf.setString(btDevice.getBluetoothAddress());
        form.removeCommand(cancelCmd);
        form.removeCommand(writeCmd);
        form.addCommand(writeCmd);
        form.addCommand(discoverCmd);
        display.setCurrent(form);
    } else if (c == cancelCmd) {
        write = false;
        discoveryAgent.cancelInquiry(this);
        form.removeCommand(cancelCmd);
        form.addCommand(writeCmd);
        form.addCommand(discoverCmd);
        display.setCurrent(form);
    }
}
}

// Starts the search for the Bluetooth devices
private void btSearch() {

    // Set the right UI
    deviceList = new List("Searching devices", List.IMPLICIT);
    deviceList.setSelectCommand(selectCmd);
    deviceList.addCommand(cancelCmd);
    deviceList.setCommandListener(this);
    display.setCurrent(deviceList);

    devices = new Vector();
    // Start an inquiry to find Bluetooth devices
    try {
        LocalDevice local = LocalDevice.getLocalDevice();
        discoveryAgent = local.getDiscoveryAgent();
        discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
    } catch (BluetoothStateException e) {
    }
}

public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {
    // When device has been found add it to devices Vector..
    devices.addElement(btDevice);

    // .. and to the deviceList List
}
```

Write_and_read_Bluetooth_NDEF_tag

```
String devName;
try {
    devName = btDevice.getFriendlyName(false);

    if (devName == null || devName.compareTo("") == 0) {
        devName = btDevice.getBluetoothAddress();
    }
    deviceList.append(devName, null);
} catch (IOException e) {
}
}

public void inquiryCompleted(int arg0) {
    // When searching of the devices has completed set title to indicate
    // that
    deviceList.setTitle("Inquiry completed");
}

public void serviceSearchCompleted(int arg0, int arg1) {
}

public void servicesDiscovered(int arg0, ServiceRecord[] arg1) {
}
}
```