Discovering Bluetooth services in Java ME

Overview

The following code snippet describes how to discover Bluetooth services on a selected remote device through the `DiscoveryAgent` object (example code uses predefined UUID for identifying service).

The `DiscoveryAgent` object can be retrieved from `LocalDevice` object instance.

The `deviceDiscovered()` function is called every time a new device has been discovered.

In `servicesDiscovered()`, which implements the `DiscoveryListener` interface, we get information about a found service through the `ServiceRecord[]` parameter.

When services discovering is finished, the `serviceSearchCompleted()` function will be called.

Preconditions

Bluetooth should be enabled on the device before starting the midlet. For best results use this example midlet with server midlet from Advertising a Bluetooth service in Java ME.

Source file: DiscoveringServicesMIDlet.java

```
import javax.microedition.midlet.MIDlet;

import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.ImageItem;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.Choice;
import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.ItemCommandListener;

import javax.bluetooth.DataElement;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.UUID;

import java.util.Vector;

import java.io.IOException;
import java.io.OutputStream;

public class DiscoveringServicesMIDlet extends MIDlet implements
        CommandListener, DiscoveryListener {
    /**
    * Variable which defines pause status
    */
    private boolean midletPaused = false;

    /**
    * Exit from midlet command
    */
    private Command exitCommand;

    /**
    * Start services discover command
    */
    private Command servicesDiscoverCommand;

    /**
    * Cancel devices discover command
    */
    private Command cancelDevicesDiscoverCommand;

    /**
    * Start devices discover command
    */
    private Command devicesDiscoveryCommand;

    /**
    * Cancel services discover command
    */
    private Command cancelServicesDiscoverCommand;

    /**
    * Main form object for midlet
    */
    private Form form;
```


(C) Copyright Nokia 2013. All rights reserved.

```
    /**
    * Choice Group for discovered devices
    */
    private ChoiceGroup deviceChoiceGroup;

    /**
    * The DiscoveryAgent for the local Bluetooth device.
    */
    private DiscoveryAgent bluetoothDiscoveryAgent;

    /**
    * Keeps track of the devices found
    */
    private Vector deviceList;

    /**
    * Keeps track of the transaction ID returned from searchServices.
    */
    private int transactionID;

    String connectionURL;

    /**
     * The DiscoveringServicesMIDlet constructor.
     * Member variables initializing
     */
    public DiscoveringServicesMIDlet() {
        //Retrieve the DiscoveryAgent object that allows us to perform device
        //and service discovery.
        try {
            bluetoothDiscoveryAgent =
                    LocalDevice.getLocalDevice().getDiscoveryAgent();
        } catch (BluetoothStateException ex) {
            //Failed to get Local Object.
            //It means that the Bluetooth system could not be initialized
            //TODO: write handler code
        }
        // Initialize the transactionID
        transactionID = -1;
        deviceList = new Vector();
    }

    /**
     * Performs an action assigned to the Mobile Device - MIDlet Started point.
     */
    public void startMIDlet() {
        switchDisplayable(null, getForm());
    }

    /**
     * Performs an action assigned to the Mobile Device - MIDlet Resumed point.
     */
    public void resumeMIDlet() {
        // No implementation required
    }

    /**
     * Switches a current displayable in a display. The display instance is
     * taken from the getDisplay method. This method is used by all actions in
     * the design for switching the displayable.
     * @param alert the Alert, which is temporarily set to the display;
     * if null, then nextDisplayable is set immediately
     * @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 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 (displayable == form) {
            if (command == cancelDevicesDiscoverCommand) {
                // cancel discovering for new bluetooth devices
                bluetoothDiscoveryAgent.cancelInquiry(this);
            } else if (command == cancelServicesDiscoverCommand) {
                // cancel discovering for services
                bluetoothDiscoveryAgent.cancelServiceSearch(transactionID);
            } else if (command == devicesDiscoveryCommand) {
                //Ui-components clearing from old data
                deviceList.removeAllElements();
                getDeviceChoiceGroup().deleteAll();
                //start discover for new bluetooth devices
                try {
                    bluetoothDiscoveryAgent.startInquiry(DiscoveryAgent.GIAC,
                            this);
                    printToForm("Start devices search...");
                    form.addCommand(cancelDevicesDiscoverCommand);
                } catch( BluetoothStateException e ) {
                    //Failed to start to discover devices
                    //TODO: write handler code
                }
            } else if (command == exitCommand) {
                exitMIDlet();
            } else if (command == servicesDiscoverCommand) {
                if(deviceList.size()==0) {
                    return;
                }
                //The class is used to represent a universally unique
                //identifier used widely as the value for a service attribute.
                //Add the UUID for L2CAP to make sure that the service record
                //found will support L2CAP.
                //You can change it according to The Bluetooth Assigned
                //Numbers document.
                UUID[] searchList = new UUID[1];
                searchList[0] = new UUID("11111111111111111111111111111111",false);

                //Initialization of service attributes whose values will be
                //retrieved on services which have the UUIDs specified
                //in searchList
```


(C) Copyright Nokia 2013. All rights reserved.

```
                //int[] attributesList = new int[2];
                //attributesList[0] = ServiceRecord.ID_ServiceID;
                //attributesList[1] = ServiceRecord.ID_ServiceName;

                //get current selected remote device by index from ChoiceGroup
                RemoteDevice currentDevice =
                    (RemoteDevice) deviceList.elementAt(
                        getDeviceChoiceGroup().getSelectedIndex());

                if(currentDevice == null) {
                    return;
                }
                //Start searching services on the current device
                //and get transaction ID of the service search.
                //Attribute 0x100 is used for retrieval of the Service's name
                try {
                    transactionID = bluetoothDiscoveryAgent.searchServices(
                            new int[] {0x100}, searchList, currentDevice, this);
                    printToForm("Start services under L2CAP searching...");
                    form.addCommand(cancelServicesDiscoverCommand);
                } catch (BluetoothStateException e) {
                    //Failed to start the search on this device, try another
                    //device.
                    //TODO: write handler code
                }
            }
        }
        // write post-action user code here
    }

    /**
     * Returns an initialized instance of the form component.
     * @return the initialized component instance
     */
    private Form getForm() {
        if (form == null) {
            form = new Form("Bluetooth Service Discovery",
                    new Item[] { getDeviceChoiceGroup() });

            exitCommand = new Command("Exit", Command.EXIT, 0);
            cancelDevicesDiscoverCommand = new Command(
                    "Cancel", Command.CANCEL, 0);
            devicesDiscoveryCommand = new Command(
                                    "Devices", Command.SCREEN, 0);
            cancelServicesDiscoverCommand = new Command(
                    "Cancel", Command.CANCEL, 0);
            servicesDiscoverCommand = new Command("Services",
                                                  Command.SCREEN, 0);
            form.addCommand(exitCommand);
            form.addCommand(devicesDiscoveryCommand);
            form.addCommand(servicesDiscoverCommand);

            form.setCommandListener(this);
        }

        return form;
    }

    /**
     * Returns an initialized instance of the deviceChoiceGroup component.
     * @return the initialized component instance
     */
    private ChoiceGroup getDeviceChoiceGroup() {
        if (deviceChoiceGroup == null) {
            deviceChoiceGroup = new ChoiceGroup("Devices", Choice.POPUP);

            deviceChoiceGroup.setDefaultCommand(servicesDiscoverCommand);
            deviceChoiceGroup.setLayout(ImageItem.LAYOUT_LEFT |
                    Item.LAYOUT_TOP | Item.LAYOUT_BOTTOM |
                    Item.LAYOUT_VCENTER | Item.LAYOUT_2);

            deviceChoiceGroup.setFitPolicy(Choice.TEXT_WRAP_ON);
            deviceChoiceGroup.setPreferredSize(-1, -1);
        }

        return deviceChoiceGroup;
    }

    /**
     * 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();
    }

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

    /**
     * From MIDlet.
     * Called to signal the MIDlet to enter the Paused state.
     */
    public void pauseApp() {
        midletPaused = true;
    }

    /**
     * From MIDlet.
```


(C) Copyright Nokia 2013. All rights reserved.

```
     * Called to signal the MIDlet to terminate.
     * @param unconditional whether the MIDlet has to be unconditionally
     * terminated
     */
    public void destroyApp(boolean unconditional) {
        // No implementation required
    }

    /**
    * 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 ChoiceGroup and to Vector objects
        String friendlyName = null;
        try {
            friendlyName = btDevice.getFriendlyName(false);
        } catch(IOException e) {
            //Failed to get device name
            //TODO: write handler code
        }

        String choiceElementText =
                (friendlyName == null) ?
                    btDevice.getBluetoothAddress() : friendlyName;
        getDeviceChoiceGroup().append(choiceElementText, null);
        deviceList.addElement(btDevice);
    }

    /**
    * 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) {
        switch(respCode) {
            case DiscoveryListener.SERVICE_SEARCH_COMPLETED:
                printToForm("Services discovering Complete!");

                StreamConnection strcon;
                try {
                    strcon = (StreamConnection) Connector.open(connectionURL);

                    OutputStream outp   = strcon.openOutputStream();
                     printToForm("Sending...\n");
                     String message="Just a message\n";
                     outp.write(message.length());
                     outp.write(message.getBytes());
                     outp.flush();
                     outp.close();
                     printToForm("Sent!\n");
                } catch (IOException e) {
                    e.printStackTrace();
                }
                break;
            case DiscoveryListener.SERVICE_SEARCH_TERMINATED:
                printToForm("Services discovering terminated!");
                break;
            case DiscoveryListener.SERVICE_SEARCH_ERROR:
                printToForm("Sevice discovering error!");
                break;
            default:
                break;
        }
        form.removeCommand(cancelServicesDiscoverCommand);
        transactionID = -1;
    }

    /**
    * From DiscoveryListener.
    * Called when service(s) are found during a service search.
    * This method provides the array of services that have been found.
    * @param transID the transaction ID of the service search that is
    * posting the result
    * @param service a list of services found during the search request
    */
    public void servicesDiscovered(int transID, ServiceRecord[] serviceRecords){
        if (serviceRecords.length>0 && serviceRecords!=null)
        {
            connectionURL=serviceRecords[0].getConnectionURL(0, false);
            //The Service's name can be retrieved as follows
            //All the attributes are retrieved
            int[] ids=serviceRecords[0].getAttributeIDs();

            //For each attribute
            for(int j=0;j<ids.length;j++)
            {
                DataElement ServiceName;
                try{
                    ServiceName=serviceRecords[0].getAttributeValue(ids[j]);
                    //If this is a String Attribute
                    if(ServiceName.getDataType()==DataElement.STRING)
                    {
                        printToForm("The Service name is: "+ServiceName.getValue());
                    }
                }
                catch(Exception e){}
            }

        }
    }

    /**
    * From DiscoveryListener.
    * Called when a device discovery transaction is
    * completed.
    * @param discType the type of request that was completed
    */
    public void inquiryCompleted(int discType) {
```


(C) Copyright Nokia 2013. All rights reserved.

```
        switch(discType) {
            case DiscoveryListener.INQUIRY_COMPLETED:
                printToForm("Device discovering Complete! Select device " +
                    "for services searching.");
                break;
            case DiscoveryListener.INQUIRY_TERMINATED:
                printToForm("Device discovering terminated!");
                getDeviceChoiceGroup().deleteAll();
                break;
            case DiscoveryListener.INQUIRY_ERROR:
                printToForm("Device discovering error!");
                getDeviceChoiceGroup().deleteAll();
                break;
            default:
                throw new IllegalArgumentException("Unknown type of message!");
        }
        form.removeCommand(cancelDevicesDiscoverCommand);
    }

    /**
     * Adds a StringItem to the main form.
     * @param strPrint string to add to main form.
     */
    private void printToForm(String message) {
        form.append(message + "\n");
    }
}
```

Postconditions

The Midlet implements the service searching process. If services have been found, it connects to the service and sends a simple String message. A working server that can do a Bluetooth handshake with this client can be found Advertising a Bluetooth service in Java ME.

`ServiceRecord`, is an object which describes service information found during discovering operation.

See also

- Discovering Bluetooth device in Java ME

Supplementary material

The source code and binaries are available for download at Media:DiscoveringServicesMIDlet.zip and Media:DiscoveringServicesBinaries.zip.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.