Discovering Bluetooth device in Java ME

The following code snippet demonstrates how to initiate Bluetooth device discovery using the Bluetooth API.

Overview

To start device discovery, you need to get a reference to a `LocalDevice` object by using `LocalDevice.getLocalDevice`. With this reference, request for a `DiscoveryAgent`instance using the `getDiscoveryAgent` method of the the `LocalDevice` instance.

Next, implement a `DiscoveryListener` interface which is used by `DiscoveryAgent` for notifying about found devices and services.

This example uses the `deviceDiscovered` method which `DiscoveryAgent` calls every time a device has been discovered.

Finally, start the discovery procedure by calling the `startInquiry` method of the the `DiscoveryAgent` object instance. This method takes `DiscoveryAgent.GIAC` as the first parameter and a reference to `DiscoveryListener` implementor as the second parameter.

`DiscoveryAgent.GIAC` - inquiry access code for General/Unlimited Inquiry Access Code (GIAC).

To stop the inquiry procedure, call the `cancelInquiry` method of the `DiscoveryAgent` object instance.

Source file: BluetoothDeviceDiscovery.java

```
import java.io.IOException;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DiscoveryAgent;

public class BluetoothDeviceDiscovery extends MIDlet implements CommandListener,
        DiscoveryListener
{
    private Display display;
    /**
     * For displaying found devices.
     */
    private List listDev;
    private Command cmdExit;
    private Command cmdDiscover;
    private Command cmdStopDiscover;

    private LocalDevice devLocal;
    private DiscoveryAgent discoverAgent;


    /**
     * Constructor. Constructs the object and initializes displayables.
     */
    public BluetoothDeviceDiscovery() {
        InitializeComponents();
    }
    /**
     * Initializes a ListBox object and adds softkeys.
     */
    protected void InitializeComponents() {
        display = Display.getDisplay( this );
        //initializing device list
        listDev = new List( "Device list", List.IMPLICIT );

        cmdExit = new Command( "Exit", Command.EXIT, 1 );
        cmdDiscover = new Command( "Start", Command.SCREEN, 1);
        cmdStopDiscover = new Command( "Stop", Command.SCREEN, 1);

        listDev.addCommand( cmdExit );
        listDev.addCommand( cmdDiscover );
        listDev.setCommandListener( this );
    }

    /**
     * From MIDlet.
     * Called when MIDlet is started.
     * @throws javax.microedition.midlet.MIDletStateChangeException
     */
    public void startApp() throws MIDletStateChangeException {
        display.setCurrent( listDev );
    }
    /**
     * From MIDlet.
     * Called to signal the MIDlet to enter the Paused state.
     */
    public void pauseApp() {
        //No implementation required
    }
    /**
```


(C) Copyright Nokia 2013. All rights reserved.

```
     * From MIDlet.
     * Called to signal the MIDlet to terminate.
     * @param unconditional whether the MIDlet has to be unconditionally
     * terminated
     * @throws javax.microedition.midlet.MIDletStateChangeException
     */
    public void destroyApp(boolean unconditional) throws MIDletStateChangeException {
        exitMIDlet();
    }
    /**
     * From CommandListener.
     * Called by the 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( command == cmdExit ) {
            exitMIDlet();
        }
        if( command == cmdDiscover ) {
            int err = 0;
            err = initLocalDevice();
            if( err != 0 ){
                return;
            }
            err = startDeviceSearch();
            if( err != 0) {
                return;
            }
            //we do not need a start soft key when in descovery mode
            listDev.removeCommand( cmdDiscover );
            //now we add stop soft key
            listDev.addCommand( cmdStopDiscover );
        }
        if( command == cmdStopDiscover ) {
            stopDiscover();
            //we remove stop softkey
            listDev.removeCommand( cmdStopDiscover );
            //adding start soft key
            listDev.addCommand( cmdDiscover );
        }
    }


    /**
     * Method calls stopDiscover method and notifyDestroyed after that.
     */
    protected void exitMIDlet()
    {
        stopDiscover();
        notifyDestroyed();
    }
    /**
     * Gets a reference to LocalDevice and saves it to devLocal variable.
     * @return 0 if local device reference getting succeded. -1 on exception.
     */
    protected int initLocalDevice() {
        try {
            devLocal = LocalDevice.getLocalDevice();
        } catch (BluetoothStateException e) {
            showErrAlert( "Error getting local device!" );
            return -1;
        }
        return 0;
    }
    /**
     * Clears device listbox, retreves DiscoveryAgent instance and saves it to
     * discoveryAgent variable. Starts device inquiry mode with access code
     * General/Unlimited Inquiry Access Code (GIAC).
     * @return 0 function execution was successfull.
     * -1 if startInquiry threw an exception.
     * @see JSR82 for more info.
     */
    protected int startDeviceSearch() {
        discoverAgent = devLocal.getDiscoveryAgent();
        try {
            listDev.deleteAll();
            discoverAgent.startInquiry( DiscoveryAgent.GIAC,
                    this );
        } catch( BluetoothStateException e ) {
            showErrAlert( "Error starting device enquiry!" );
            return -1;
        }
        return 0;
    }

    /**
     * From DiscoveryListener.
     * Called when new bluetooth device discovered.
     * @param remoteDev reference to RemoteDevice object instance of
     * discovered device
     * @param devClass the service classes, major device class,
     * and minor device class of the remote device
     */
    public void deviceDiscovered(RemoteDevice remoteDev, DeviceClass devClass) {
        try {
            //add device "Friendly name" to a listbox
            listDev.append( "Device: " + remoteDev.getFriendlyName( false ),
                    null);
        } catch ( IOException e ) {
            //TODO: Handle error
        }
    }

    /**
     * From DiscoveryListener.
     * Called when service(s) are found during a service search.
     * @param transID the transaction ID of the service search that is
     * posting the result
     * @param servRecord a list of services found during the search request
     */
    public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {
        //No implementation required
    }

    /**
     * From DiscoveryListener.
     * Called when a service search is completed or was terminated
```


(C) Copyright Nokia 2013. All rights reserved.

```
     * because of an error
     * @param transID the transaction ID identifying the request which
     * initiated the service search
     * @param respCode the response code that indicates the status
     * of the transaction
     */
    public void serviceSearchCompleted(int transID, int respCode) {
        //No implementation required
    }

    /**
     * From DiscoveryListener.
     * Called when an inquiry is completed.
     * @param discType the type of request that was completed
     */
    public void inquiryCompleted(int discType) {
       // No implementation required
    }
    /**
     * Show error alert with a string specified in strError.
     * @param strError Error message show.
     */
    protected void showErrAlert( String strError ) {
        Alert alertErr = new Alert("Error", strError, null, AlertType.ERROR );
        display.setCurrent( alertErr );
    }

    /**
     * Stops device discovery mode.
     */
    protected void stopDiscover() {
       //we turning of the device discovery mode
       discoverAgent.cancelInquiry( this );
    }
}
```

Postconditions

Device discovery isn't started automatically when the app starts. To start the discovery, press 'Start'. This command clears the device list(listbox) and starts the device discovery procedure.

When a device is detected, its name is added to the list.

The Stop command cancels the discovering procedure.

Supplementary material

Look metadata-section for downloadable source files and application binaries.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.