Advertising a Bluetooth service in Java ME

This code example demonstrates the basic Bluetooth service advertising procedure with the Bluetooth API. In this example, the MIDlet starts the SPP server for a single client.

Overview

When a client is connecting, it sends a simple string, the server displays the message and closes the connection.

The server is started by pressing 'Start'. Service advertising starts by forming a String with a connection URL. For more information on bluetooth connection URLs, see JSR-82.

When there is a correct URL, the `open` method of the `Connector` class is called.

After this the server can start waiting for incoming connections by calling the `acceptAndOpen` method of the `StreamConnectionNotifier` object instance. Accordingly to JSR-82, the service record is being created in SDDB and `acceptAndOpen` is called.

When an incoming connection is detected, this method returns the `StreamConnection` object. `StreamConnection` used for communication with a connected client.

In this example, the `openInputStream` method of this object is being called to open an input stream. The server then reads the client's response string.

Next, the server closes the connection and stops the server by calling the `close` method of `StreamConnection` and `StreamConnectionNotifier`, respectively.

Accordingly to JSR-82, the `StreamConnectionNotifier.close` method deletes service record from SDDB.

Source file: BluetoothServiceAdvertising.java

```
import java.io.IOException;
import java.io.InputStream;

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.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.io.StreamConnectionNotifier;

/**
 * BluetoothServiceAdvertising MIDlet is an SPP server.
 * When user presses the  'Start' softkey, an SPP server is started,
 * waiting for incoming connection. When an incoming connection is accepted,
 * the server responds with a string defined in the strServerResp variable and closes
 * the connection.
 * @see JSR82-specification for more info on server advertising.
 */
public class BluetoothServiceAdvertising extends MIDlet implements CommandListener,
        Runnable {
    private Display display;
    /**
     * Reference to a Form object.
     * Used for displaying the current server status information.
     */
    private Form frmMain;
    private Command cmdExit;
    /**
     * Command starts a server and initializes connecting waiting thread.
     */
    private Command cmdStart;
    /**
     * Command stops the server by closing the connection and connection notifier.
     * Connection waiting thread is interrupted when this command is being
     * handled.
     */
    private Command cmdStop;
    /**
     * Protocol name entry from connection URL.
     * @see JSR82-specification.
     */
    private String connProtocol;
     /**
     * Service class id entry from connection URL.
     * @see JSR82-specification.
     */
    private String connServiceClassId;
     /**
     * Target entry from connection URL.
     * @see JSR82-specification.
     */
    private String connTarget;
    /**
     * Name entry from connection URL.
     * @see JSR82-specification.
     */
    private String connServiceName;
    /**
```


(C) Copyright Nokia 2013. All rights reserved.

```
     * Stores a reference to incoming connection notifier.
     */
    private StreamConnectionNotifier conNotifier;
    /**
     * Stores a reference to stream connection.
     * Connection waiting thread uses it for accepting conections.
     */
    private StreamConnection connection;
    /**
     * Thread that listens for incomming connections.
     */
    private Thread threadMain;
    /**
     * Set to true if a server is started and waiting for incoming
     * connections. false - otherwise.
     */
    private boolean serverStarted;

    /**
     * Constructor. Constructs the object and initializes displayables.
     * Gets Display object reference.
     * Sets current server status to false(not running).
     */
    public BluetoothServiceAdvertising() {
        InitializeComponents();
        //we choosed a RFCOMM as a protocol
        connProtocol = "btspp";
        //we set target to localhost to start a server
        connTarget = "localhost";
        //we set random 128-bit identificator
        connServiceClassId = "11111111111111111111111111111111";
        //we set some name to display for our service
        connServiceName = "My Spp-service";
        //we set server state flag to 'not running' e.g. false.
        serverStarted = false;
    }
    /**
     * Gets Display object reference.
     * Initializes a form.
     * Addes 'Start' and 'Exit' command to it.
     */
    protected void InitializeComponents() {
        display = Display.getDisplay( this );
        //initializing device list
        frmMain = new Form( "Server status" );

        //initializing commands
        cmdExit = new Command( "Exit", Command.EXIT, 1 );
        cmdStart = new Command( "Start", Command.SCREEN, 1);
        cmdStop = new Command( "Stop", Command.SCREEN, 1);

        frmMain.addCommand( cmdExit );
        frmMain.addCommand( cmdStart );
        frmMain.addCommand( cmdStop );
        frmMain.setCommandListener( this );
    }
    /**
     * From MIDlet.
     * When application is being started frmMain is set as default
     * displayable.
     * @see Displayable
     * @throws javax.microedition.midlet.MIDletStateChangeException
     */
    public void startApp() throws MIDletStateChangeException {

        display.setCurrent( frmMain );
    }
    /**
     * From MIDlet.
     * Called to signal the MIDlet to enter the Paused state.
     */
    public void pauseApp() {
        //No implementation required
    }
    /**
     * 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 == cmdStart ) {
            changeAppState( 1 );
        }
        if( command == cmdStop ) {
            changeAppState( 0 );
        }
    }
    /**
     * Method calls stopSerialPortServer(if serverStarted == true ) and
     * notifyDestroyed after that.
     * @see BluetoothServiceAdvertising#serverStarted
     * @see BluetoothServiceAdvertising#stopSerialPortServer()
     * @see MIDlet#notifyDestroyed()
     */
    protected void exitMIDlet() {
        if( serverStarted ) {
            stopSerialPortServer();
        }
        notifyDestroyed();
    }
    /**
     * Starts or stops bluetooth server depending on state parameter value.
     * @param aState determines wether the server should be started or shut down.
```


(C) Copyright Nokia 2013. All rights reserved.

```
     * '1' - to start the server and '0' to stop it.
     */
    private void changeAppState( int state ) {
        switch( state ) {
            case 0:
                if( !serverStarted ) {
                    return;
                }
                if( stopSerialPortServer() != 0 ) {
                    return;
                }
                //we remove stop softkey
                frmMain.removeCommand( cmdStop );
                //adding start soft key
                frmMain.addCommand( cmdStart );
                serverStarted = false;
                break;
            case 1:
                if( serverStarted ) {
                    return;
                }
                if( startSerialPortServer() != 0) {
                    return;
                }
                serverStarted = true;
                //we do not need a start soft key when in descovery mode
                frmMain.removeCommand( cmdStart );
                //now we add stop soft key
                frmMain.addCommand( cmdStop );
        }
    }
    /**
     * Method creates a connection notifier and initializes a connection
     * waiting thread.
     * @return 0 if server startup was successfull. 1 if connection notifier
     * instantiation caused an exception. 2 if thread creating caused an
     * exception.
     * @see Connector#open(java.lang.String)
     * @see Thread#start()
     * @see BluetoothServiceAdvertising#run()
     */
    private int startSerialPortServer() {

        String strServerConnection = connProtocol +
                "://" +
                connTarget +
                ":" +
                connServiceClassId +
                ";" +
                "name=" + connServiceName;
        try {
            printToFrm("Starting server...");
            conNotifier = ( StreamConnectionNotifier ) Connector.open(
                    strServerConnection );
            printToFrm( "Starting a listening thread..." );
            threadMain = new Thread( this );
            threadMain.start();
        } catch ( IllegalThreadStateException e ) {
            printToFrm( "Error starting a server thread!" );
            return 2;

        } catch ( Exception e ) {
            printToFrm( "Error starting a server connection!" );
            return 1;
        }
        return 0;
    }
    /**
     * @return 0 if the server shutdown was successfull. 1 if there was
     * an IOException exception during execution.
     * @see javax.microedition.io.StreamConnectionNotifier
     * @see javax.microedition.io.StreamConnection
     */
    private int stopSerialPortServer() {

        try {
            printToFrm("Disconnected");
            if( connection != null ) {
                connection.close();
            }
            conNotifier.close();
        } catch ( IOException e ) {
            printToFrm( "Error closing server connection!" );
            return 1;
        }
        return 0;
    }
    /**
     * From Runnable.
     * The method is being run when a connection listening thread was created
     * and started(in startSerialPortServer). It runs acceptAndOpen method.
     * When a connection accepted doConnReplay is being executed. After that a
     * cmdStop command is trigger to stop the server.
     * @see StreamConnectionNotifier
     * @see BluetoothServiceAdvertising#doConnReply()
     */
    public void run() {
        try {
            connection = (StreamConnection) conNotifier.acceptAndOpen();
            printToFrm( "Incomming connection..." );
            doConnReply();
        } catch ( IOException e ) {
            printToFrm( "Connection waiting thread interropted!" );
        }
        changeAppState( 0 );
    }
    /**
     * Opens an InputStream via connection member variable.
     * Reads a string from the stream and closes the connection.
     * @throws java.io.IOException
     * @see InputStream
     * @see StreamConnection
     */
    protected void doConnReply() throws IOException {
        printToFrm( "Receiving client response..." );
        byte data[]=null;
        try {


          InputStream input   = connection.openInputStream();
```


(C) Copyright Nokia 2013. All rights reserved.

```
            int length = input.read();
            data       = new byte[length];
            length     = 0;

            while (length != data.length)
            {
               int ch = input.read(data, length, data.length - length);

               if (ch == -1)
               {
                printToFrm( "Can't read data" );
               }
               length += ch;
            }
            input.close();
            connection.close();
            String incoming=new String(data);
            printToFrm(incoming);
         } catch ( IOException e) {
             printToFrm( "Error writing data output stream!" );
             return;
         }
     }
     /**
      * Adds a StringItem to the frmMain.
      * @param strPrint string to add to frmMain.
      * @see Form#append(java.lang.String)
      * @see Form
      */
     protected void printToFrm( String strPrint ) {
         frmMain.append( strPrint );
     }
 }
```

Postconditions

When 'Start' is pressed, the server is started and at that moment, this service can be detected from any Bluetooth device. To connect to it, you should use appropriate client. A client example can be downloaded from Discovering Bluetooth services in Java ME.

When a client is connecting, the server prints the "Incoming connection" string to the form, responds with a string, closes the connection, and stops the server.

The server can be stopped while waiting for incoming connections if user presses 'Stop'.

Supplementary material

The source file and executable application are available for download at Media:Advertising a Bluetooth service in Java ME.zip.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.