Listening synchronously for incoming SMS messages in Java ME

Overview

This code snippet demonstrates how to implement a MIDlet that listens for incoming SMS messages, as a synchronous implementation. The flow of operations is as follows:

1. The user installs the SMSSyncListener to a device and starts it. Let's call this device Device A. The MIDlet doesn't listen for incoming messages yet, so if the user now sends a message from another device – let's call this Device B – they wouldn't be intercepted by the MIDlet. Instead, they would arrive in the inbox of Device A.
2. The user tells the MIDlet to start listening for incoming messages by selecting `Options` > `Start listening`. Inside the MIDlet, the following happens:
   1. The `startListening()` method is called. It opens the connection to the specified port, creates a new thread, and kicks it running.
   2. In the `run()` method, the synchronous `receive()` method is called, which blocks until there is a message available.
3. From Device B, the user may now send a message, which will get grabbed by the MIDlet in Device A. The following happens now in the MIDlet:
   1. When a message arrives, the `receive()` method returns and the `processMessage()` method is called.
   2. The `processMessage()` method checks the type of the message (text, binary, or multipart message) and acts accordingly. In practice, it displays the payload text of the message on the screen. Only handling of text messages is implemented in this snippet.
4. Note that messages sent to the specified port will never arrive in the inbox of Device A as long as the MIDlet is listening for them. If the user now stopped the listener (by selecting `Options` > `Stop listening`), subsequent messages to the specified port would end up in the inbox of Device A instead of the MIDlet.

Source

```
import java.io.IOException;
import javax.microedition.io.Connector;
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.StringItem;
import javax.microedition.midlet.MIDlet;
import javax.wireless.messaging.BinaryMessage;
import javax.wireless.messaging.Message;
import javax.wireless.messaging.MessageConnection;
import javax.wireless.messaging.MultipartMessage;
import javax.wireless.messaging.TextMessage;
```

```
public class SMSSyncListener
        extends MIDlet
        implements CommandListener, Runnable {

    // The port which is listened for incoming messages
    private final String PORT = "6553";

    private Form mainForm;
    private Command startCommand;
    private Command stopCommand;
    private Command exitCommand;
    private MessageConnection connection;
    private boolean listening;

    /**
     * Constructor. Constructs the object and initializes displayables.
     */
    public SMSSyncListener() {
        mainForm = new Form("SMS Listener");

        startCommand = new Command("Start listening", Command.ITEM, 0);
        mainForm.addCommand(startCommand);

        stopCommand = new Command("Stop listening", Command.ITEM, 1);
        mainForm.addCommand(stopCommand);

        exitCommand = new Command("Exit", Command.EXIT, 1);
        mainForm.addCommand(exitCommand);

        mainForm.setCommandListener(this);
    }

    /**
     * From MIDlet.
     * Called when the MIDlet is started.
     */
    public void startApp() {
        // The initial display is the main form
        Display.getDisplay(this).setCurrent(mainForm);
    }

    /**
     * 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
```


(C) Copyright Nokia 2013. All rights reserved.

```
     */
    public void destroyApp(boolean unconditional) {
        // Stop listening
        stopListening();
    }

    /**
     * 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 == exitCommand) {
            // Exit the MIDlet
            destroyApp(true);
            notifyDestroyed();
        } else if (command == startCommand) {
            startListening();
        } else if (command == stopCommand) {
            stopListening();
        }
    }

    /**
     * Starts listening for incoming messages.
     */
    private void startListening() {
        // If we are already listening, no need to start again
        if (listening) {
            return;
        }

        try {
            // Open the connection to the specified port
            connection = (MessageConnection)Connector.open("sms://:" + PORT);
        } catch (IOException ex) {
            return;
        }

        // Create a listener thread and start listening
        Thread listenerThread = new Thread(this);
        listening = true;
        listenerThread.start();

        mainForm.append("\nListener started.\n");
    }

    /**
     * Stops listening for incoming messages.
     */
    private void stopListening() {
        // If we are not listening, no need to do anything
        if (!listening) {
            return;
        }

        if (connection != null) {
            try {
                // Close the message connection
                connection.close();
                connection = null;
            } catch (IOException ex) {
                // TODO: Exception handling
            }
        }

        listening = false;

        mainForm.append("\nListener stopped.\n");
    }

    /**
     * The main listening loop.
     */
    public void run() {
        while (listening) {
            try {
                // Receive all incoming messages to the specified port. The
                // receive() method will block until there is a message
                // available.
                Message message = connection.receive();
                if (message != null) {
                    mainForm.append("Message received.\n");
                    processMessage(message);
                }
            } catch (IOException ex) {
                // Stop listening
                stopListening();
            }
        }
    }

    /**
     * Processes the received message according to its type.
     * @param message the received message
     */
    private void processMessage(Message message) {
        if (message instanceof TextMessage) {
            processTextMessage((TextMessage)message);
        } else if (message instanceof BinaryMessage) {
            processBinaryMessage((BinaryMessage)message);
        } else if (message instanceof MultipartMessage) {
            processMultipartMessage((MultipartMessage)message);
        }
    }

    /**
     * Processes a text message.
     */
    private void processTextMessage(TextMessage message) {
        String text = message.getPayloadText();
        StringItem textItem = new StringItem("Text", text);
        mainForm.append(textItem);
    }

    /**
     * Processes a binary message.
     */
```


(C) Copyright Nokia 2013. All rights reserved.

```
    private void processBinaryMessage(BinaryMessage binaryMessage) {
        // Not implemented
    }

    /**
     * Processes a multipart message.
     */
    private void processMultipartMessage(MultipartMessage multipartMessage) {
        // Not implemented
    }
}
```

Note: When you send the message, make sure the port number is the same as in the listener. Otherwise, the listener thread won't notice the message. Here's an example of preparing a text SMS for sending:

```
// Prepare the text message
TextMessage message = (TextMessage)connection.newMessage(
    MessageConnection.TEXT_MESSAGE);

// Obtain the address from a text field, append a port number to it, and set the
// whole thing as the destination address for the message. The port number is
// the same as above.
String address = "sms://" + smsAddress.getString() + ":" + PORT;
message.setAddress(address);

// Obtain the specified text and set it as the payload
String text = smsText.getString();
message.setPayloadText(text);
```

For a complete Code example please check How to send a text message to a given port as SMS with Java ME

See also

- Listening asynchronously for incoming SMS messages in Java ME
- Fragmenting a binary message for sending over SMS using Java ME
- How to send a message to a given port as SMS with Java ME
- How to launch a MIDlet remotely via SMS or locally via an Alarm with PushRegistry in Java ME

(C) Copyright Nokia 2013. All rights reserved.

 (C) Copyright Nokia 2013. All rights reserved.