Listening asynchronously for incoming SMS messages in Java ME

Overview

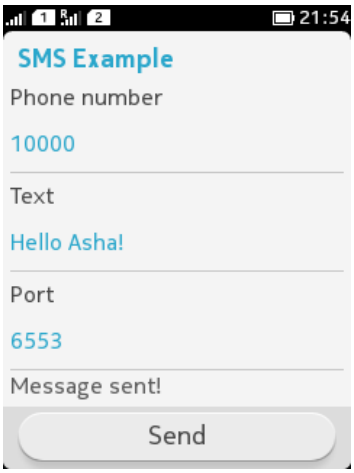
01 Jul 2012

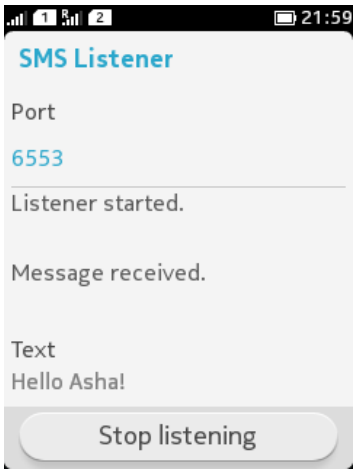
This code snippet demonstrates how to implement a MIDlet that listens for incoming SMS messages sent to a specific port, as an asynchronous implementation. Messages that are not intercepted (because the MIDlet is not running or is not listening on the port) are delivered to the inbox.

The asynchronous implementation provided has the following flow of operations:

1. The user installs the SMSListenerMIDlet 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 sent 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.
   2. The method opens the connection to the specified port and registers the MIDlet as the listener for incoming messages.
3. From Device B, the user may now send a message to the specified port in Device A, which will cause the following to happen in the MIDlet:
   1. The `notifyIncomingMessage()` method gets called by the WMA implementation.
   2. To minimize the time spent in the `notifyIncomingMessage()` method, a separate thread is created to fetch the message.
   3. Once the thread is started, it will block in the `receive()` method until there is a message available. This shouldn't take long, since the MIDlet has already been notified that there is a message waiting. Nevertheless, it is good practice to ensure that the `notifyIncomingMessage()` system method returns quickly, hence the thread creation.
   4. When a message has been received, the `processMessage()` method is called. It 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. Terminating the listening MIDlet would be the same as stop listening for incoming SMSes, i.e. the message would end up in the Inbox. As soon as a message is delivered there, it cannot become available to the listening MIDlet.

Note: On Asha software platform and Series 40 DP 2.0, a text message with defined port number in its address, will not be received at the inbox of the recipient device, even if there is no MIDlet listening to the given port at the recipient's end.





Note: While this example establishes a TEXT_MESSAGE connection, the logic for establishing a BINARY_MESSAGE connection is the same

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.lcdui.TextField;
import javax.microedition.midlet.MIDlet;
import javax.wireless.messaging.BinaryMessage;
import javax.wireless.messaging.Message;
import javax.wireless.messaging.MessageConnection;
import javax.wireless.messaging.MessageListener;
import javax.wireless.messaging.MultipartMessage;
import javax.wireless.messaging.TextMessage;

public class SMSListenerMIDlet
        extends MIDlet
        implements CommandListener, MessageListener {
```


(C) Copyright Nokia 2013. All rights reserved.

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

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

        port = new TextField("Port", "6553", 4, TextField.NUMERIC);
        mainForm.append(port);

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

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

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

        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
     */
    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.getString());

            // Register this MIDlet as the listener that should be notified
            // whenever messages arrive
            connection.setMessageListener(this);
        } catch (IOException ex) {
            return;
        }

        listening = true;
        mainForm.removeCommand(startCommand);
        mainForm.addCommand(stopCommand);
        mainForm.append("Listener 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 {
                // Deregister the message listener and close the connection
                connection.setMessageListener(null);
                connection.close();
                connection = null;
            } catch (IOException ex) {
                // TODO: Exception handling
            }
```

 (C) Copyright Nokia 2013. All rights reserved.

```
        }

        listening = false;
        mainForm.removeCommand(stopCommand);
        mainForm.addCommand(startCommand);
        mainForm.append("Listener stopped.\n");
    }

    /**
     * Asynchronous callback method for receiving incoming messages.
     * Called by the WMA implementation when a new message is ready.
     * @param connection the message connection with incoming messages
     */
    public void notifyIncomingMessage(final MessageConnection conn) {
        // Because this method is called by the platform, it is good practice to
        // minimize the processing done here, on the system thread. Therefore,
        // let's create a new thread for reading the message.
        Thread smsThread = new Thread() {
            public void run() {
                try {
                    // Receive all incoming messages to the specified
                    // connection. The receive() method will block until there
                    // is a message available.
                    Message message = conn.receive();
                    if (message != null) {
                        mainForm.append("Message received.\n");
                        processMessage(message);
                    }
                } catch (IOException ex) {
                    // Stop listening
                    stopListening();
                }
            }
        };
        smsThread.start();
    }

    /**
     * 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.
     */
    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 synchronously 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.