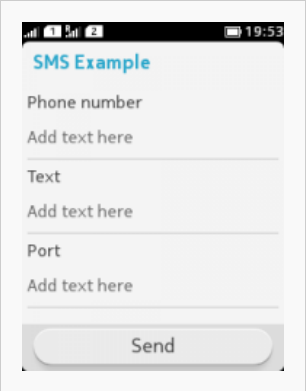
How to send a message to a given port as SMS with Java ME

This code example demonstrates how to use the Wireless Messaging API in order to send a text message to a specific port. Specifying the port number is of high importance when you use this application in combination with another application that listens to the same port for incoming messages. The device's AMS (Application Management Software), can also assign a port to a given application and launch that application via PushRegistry when a message arrives.

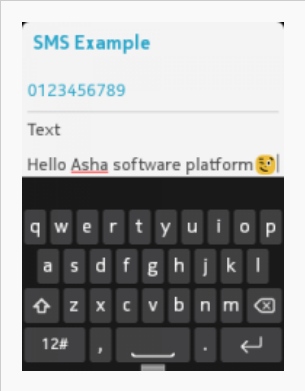
10 Nov 2013

Overview

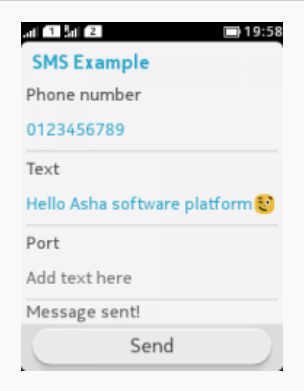
High-level LCDUI Components are used for creating the application's UI, which consists of a numeric field for typing the recipient's number, a message body, a field for typing the port number (optional), the **Send** button and an **Exit** button.



A simple UI for text messaging



Vitual Keyboard supports rendering of emoticons



Info message acknowledges task completion

The User Interface

Both `TextFields` are empty by default. The `TextField` at the top of the form becomes of type `PHONENUMBER` and can take up to 20 digits, while the `TextField` beneath it, becomes of type `PLAIN` and takes up to 160 characters, which is the character limit for an SMS transmission without splitting it up to more than one piece. The port number field can only receive up to 4 digits.

```
        mainForm = new Form("SMS Example");

        smsNumber = new TextField("Phone number", null, 20,
            TextField.PHONENUMBER);
        mainForm.append(smsNumber);

        smsText = new TextField("Text", null, 160, TextField.PLAIN);
        mainForm.append(smsText);

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

        sendCommand = new Command("Send", Command.OK, 0);
        mainForm.addCommand(sendCommand);

        exitCommand = new Command("Exit", Command.EXIT, 0);
        mainForm.addCommand(exitCommand);

        mainForm.setCommandListener(this);
```

Sending the text message

The method `handleSendCommand` is called via the `CommandListener` when the user selects the Send action. The `TextMessage` instance is being given an address in the form sms:// [recipient number] : [port number]. The port number is only appended to the address if the user has provided an non empty numeric entry. A connection is opened to the newly created address and a message of type `TEXT_MESSAGE` is associated with the address. A payload is added to the message by calling the `setPayloadText` method and passing as argument the content of the second TextField. Finally the method `sent()` is called, which receives as argument the `TextMessage` instance. Appropriate handling of Exceptions follows, along with a print screen message, confirming the completion of the operation. The sending operation takes place inside a thread, because the `connector.open()` call is protected and Nokia Asha software platform, does not allow protected calls directly initiated from a CommandAction, without being wrapped in a Thread.

```
private void handleSendCommand() {
        // Send the message on its own thread of execution
        Thread smsThread = new Thread() {
         public void run() {
          try {
           String number;
           //Obtain the destination address
           if(port.getString().equals("")) {
```


(C) Copyright Nokia 2013. All rights reserved.

```
            number = "sms://" + smsNumber.getString();
           }
           else {
            number = "sms://" + smsNumber.getString() + ":" + port.getString();
           }

           // Obtain the specified text
           String text = smsText.getString();
           connection = (MessageConnection)Connector.open(number);
                         message = (TextMessage)connection.newMessage(MessageConnection.TEXT_MESSAGE);
                         message.setPayloadText(text);
           connection.send(message);
           mainForm.append("Message sent!\n");
          }
          catch (InterruptedIOException ex) {
           mainForm.append("Interrupted Exception!\n");
          }
          catch (IOException ex) {
           mainForm.append("IOException!\n");
          }
          catch (IllegalArgumentException ex) {
           mainForm.append("Illegal Argument!\n");
          }
          catch (SecurityException ex) {
           mainForm.append("Security Exception!\n");
          }
         }
        };
        smsThread.start();
```

Sending Binary Messages

Binary messages can also be sent with the Wireless Messaging API. The process is very similar to sending TextMessages with the following differences:

- The `TextMessage` instance should be replaced by a `BinaryMessage` instance that uses a `MessageConnection.BINARY_MESSAGE`
- The payload for a binary message cannot be a `String`, but a byte array instead.

For more information, see also Fragmenting a binary message for sending over SMS using Java ME.

See also

- Listening asynchronously for incoming SMS messages in Java ME
- Listening synchronously for incoming SMS messages in Java ME
- How to launch a MIDlet remotely via SMS or locally via an Alarm with PushRegistry in Java ME
- The Wireless Messaging API
- Fragmenting a binary message for sending over SMS using Java ME

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.