A Java ME Canvas Qwerty Keyboard for Touch Devices

This article provides a virtual keyboard that you can use to enter any kind of text in a MIDlet (built with low-level UI classes) on touch devices with no physical keyboard like Nokia 5800 XpressMusic and Nokia 5530 XpressMusic phones.

Keyboard issues using low-level UI

When creating a low level (Canvas or GameCanvas) application using a device with a physical keyboard, it is possible to get events from keys being pressed and act accordingly. The ASCII code of every key is passed in the method `keyPressed(int)` of the `Canvas` class.

Alphanumeric keyboard: only one ASCII code is passed to the `keyPressed()` method, since every key references several characters it is not possible to know exactly which character the user wanted to enter. Usually the ASCII code passed to the method is the one corresponding to the numeric value.

Qwerty keyboard: if the keyboard is qwerty, it is possible to receive ASCII codes of all the char set. Device is in charge of changing the character set when keys like shift or numeric mode are pressed, so the right ASCII code is passed to the method.

According to the above, entering text in different Nokia phones is solved in qwerty keyboards, but it is present in alphanumeric and non physical keyboard phones.

There is a workaround to solve the basic problem. It involves combining the low level and high level display components. When a specific area of the low level application is created, then an idea is to invoke a `TextBox` component, retrieve the text inserted by the user and they display it in the low level component again.

This approach has the following disadvantages.

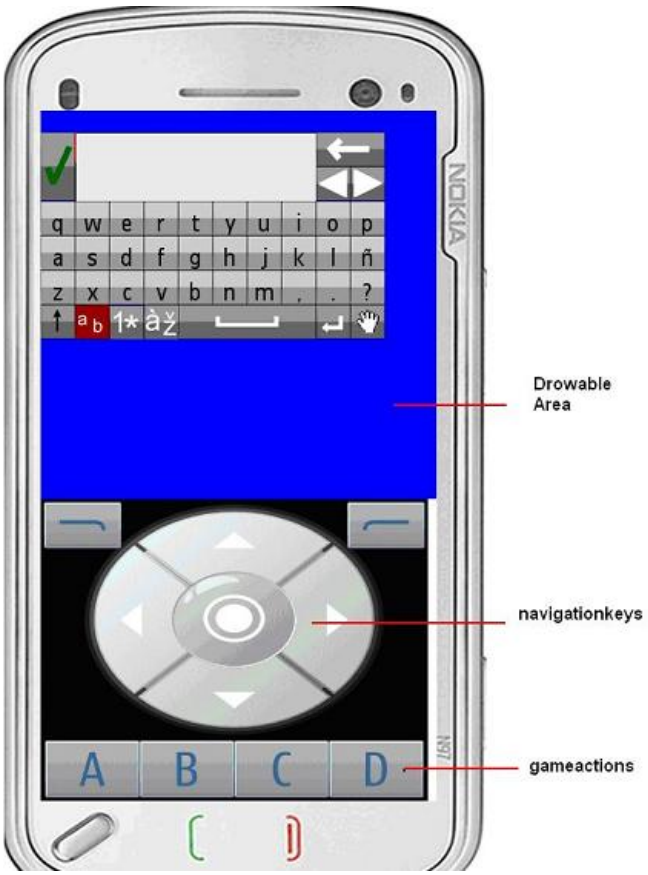
- The high level component will take the whole screen.
- The look and feel of the `TextBox` components will be according to the native look and feel and not to the one used in the application and this might disrupt the user experience. The look and feel of the application is blue entirely and the native component is black for instance.

In latest sw of S60 5th edition there is an additional component that does not take the entire screen, it requires only an additional attribute in the JAD file. You can see an example about how to use this mechanism in the following link: How to use pop-up TextBox in Java ME

DESIGN CONSIDERATIONS

Drawable Area

(C) Copyright Nokia 2014. All rights reserved.



The drawable area of a Canvas in S60 5th Edition can be changed with the attribute **Nokia-MIDlet-On-Screen-Keypad**. The different drawable sizes are shown in the table bellow.

Nokia-MIDlet-On-Screen-Keypad

Value	Portrait	Landscape
*Default (gameactions & navigation keys)	360x360	320x360
navigationkeys	360x384	372x360
no	360x640	640x360

- Default means that the attribute is not written neither in JAD or manifest file.

For more information see Using on-screen keypad in MIDlets in S60 5th Edition devices

Orientation

There is a way to force the keyboard to keep a landscape orientation. This can be done by using the following attribute:

Nokia-MIDlet-App-Orientation

Its possible values are: portrait or landscape

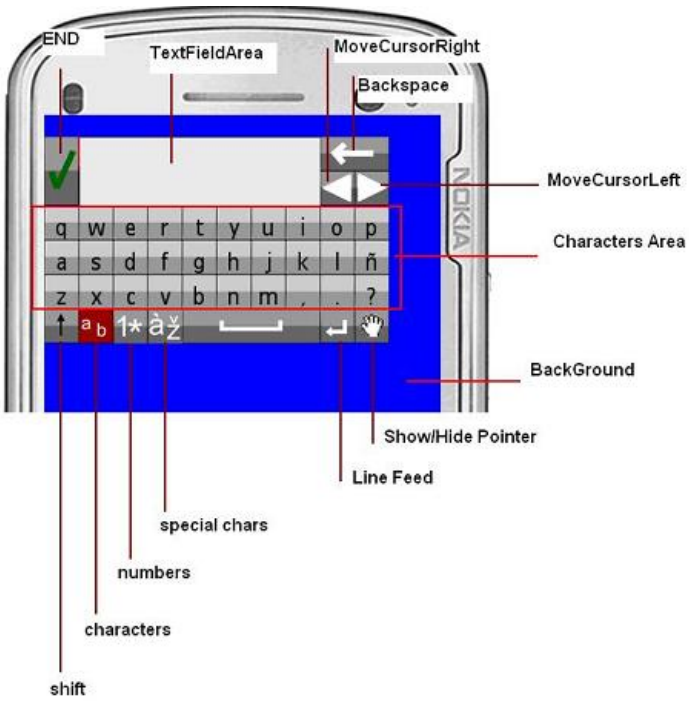
It is important to remark that in early software versions of 5800 this attribute causes no effect. The problem was corrected in early firmware versions. Archived:Fixed landscape mode cannot be set by using Java ME in Nokia 5800 XpressMusic (Known Issue)

Ideally, it would be possible to have a full screen virtual keyboard (640x360) with these two attributes either in JAD or manifest file:

- **Nokia-MIDlet-On-Screen-Keypad:** no
- **Nokia-MIDlet-App-Orientation:** landscape

Keyboard components

(C) Copyright Nokia 2014. All rights reserved.



TOUCHKEYBOARD

This class presents a virtual keyboard intended to be used in touch devices. The keyboard is composed of a `TextFieldArea` where the text entered is displayed, and a series of buttons representing either the available characters allowing to modify the behaviour of the virtual keyboard.

All the buttons extend from the `Button` class that declares the basic methods. The virtual keyboard contains three different kinds of buttons.

1. `MatrixButton`: is the part of the keyboard displaying the characters that will be part of the text. Every key has a preview and the action is taken when the button is released.
2. `PushButton`: are single buttons that are pressed but don't show a preview of the key. E.g. Ok, backspace, shift. Action is taken when the button is released
3. `ModeButton`: are buttons to change some behaviour in the keyboard, like the character set or to activate the pointer. Action is taken when button is pressed

Changing Look & Feel of the characters area

The `MatrixButton` known also as `charactersArea` is composed by a `TiledLayer`. The `TiledLayer` contains three different Tiles, the first one is used in the idle state of the key, the second one is used for the selected state of the key and the third one is used in the key preview appearing right above the key being pressed. The characters in the `MatrixButton` are painted directly on the Graphics of the `GameCanvas.0` Their Font and Color can be changed using the methods `setCharactersAreaFont(Font)` , `setCharactersAreaFontColor(int)` and `setCharactersAreaPreviewFontColor(int)`. This allows the `charactersArea` to adjust different sizes and have the same look & feel of the app.

Changing Look&Feel of the TextFieldArea

It is possible to change the look & Feel of the TextFieldArea according to the one used in the application.

To do that use the following methods: `setTextFieldFont(Font)`, `setTextFieldBkgColor(int)`, `setTextFieldBorderColor(int)`, `setCursorColor(int)`, `setTextFieldFontColor(int)`

Changing Look&Feel of The Rest Of The Buttons

The rest of the buttons of the virtual keyboard are a set of `PushButton` and `ModeButton` that are basically composed by a `Sprite` object. To change the look&feel of any of them it is necessary to modify the corresponding image in the resources folder. Each

(C) Copyright Nokia 2014. All rights reserved.

image is referenced by a private constant, so if the image name changes it is important to make sure that the private constant is also changed.

Limiting the number of characters

It is possible to limit the number of characters that can be entered in the keyboard using the function `setMaxSize(int)`

TextFieldArea

This is a component able to hold and display text. It is created with a specific size and this size remains for the entire life of the component. The `TextFieldArea` resembles the behaviour of the native text input area of S60 that you can see for instance when writing a text message.

Changing Look&Feel

It is possible to change the look & Feel of the `TextFieldArea` according to the one used in the application.

To do that use the following methods **setTextFieldFont(Font) ,setTextFieldBkgColor(int), setTextFieldBorderColor(int), setCursorColor(int), setTextFieldFontColor(int)**

Arranging words

The component is in charge of arranging the words that it holds. A word is identified by the space character following it. If the word is too long to fit the remaining empty space of the line, it will be pushed to the next line. If a word alone is bigger than the horizontal space of the `TextFieldArea`, then the word is broken into several pieces to fit the width of the component. Every time a new character is inserted or deleted the whole `textArea` must be reorganized. This is because text can be entered in any place.

About The Cursor

The cursor marks the place where a new character can be inserted. When this happens, the cursor moves to the next position. Cursor color can be modified using the method `setCursorColor(int)` or also hidden with `setCursorVisible(boolean)`. Cursor can be moved to the left or right using the methods `moveCursorLeft()` or **moveCursorRight().** When the cursor reaches the final position of the last line, the next character entered will make the component to shift one line downwards so the cursor is always visible and remains in the last line. In the same way if cursor is taken to the upper right of the component and there are text lines hidden above, then it will make the component to shift one line upwards so the cursor remains visible. In few words the cursor controls the lines of text displayed in the `TextFieldArea`.

Painting the TextFieldArea

The component requires a `Graphics` object to be painted. In order to do so, an explicit call to the method `paint(Graphics, int, int)` must be done. The two last parameters indicate the x and y position in the Graphics where the component will be painted. The big advantage of this mechanism is that the same instance of the component can be painted in different Graphics and different positions of it. The component breaks the text in different lines, only the lines that fit the screen will be painted, the rest of then will be ignored. The line where the cursor is present always remains visible.

Adding New Characters And Getting the Text

The `TextFieldArea` maximum number of characters can be delimited using the method `setMaxSize(int)`. If this value is not specified, there is no limit of characters that can be entered. Adding a new character in the cursor position can be done by using the method `insertCharacterInCursorPosition(char)`. Deleting a character in the cursor position can be done by using the method `deleteLastCharBeforeCursor()`. It is also possible to set the text at any time using the method `setText(String)` this will replace the existing text and will place the cursor at the end of the line. To get the current text of the `TextFieldArea` use the `getText()` method.

To take advantage of the text arrangement functionality in devices with a keyboard, the component provides the `insertAscii(int)` method. When Canvas or `GameCanvas` is used, the keyPressed method provides the ascii code of the button pressed. For normal keypad phones this is the ascii of the number pressed, for qwerty devices it will be the value of the key pressed. The device changes automatically the character set if the shift or numbers key is pressed and so the ascii received is different and corresponds to the intended key. The backspace character is also received as an ascii code, so it is not necessary to call in explicit way the delete function.

(C) Copyright Nokia 2014. All rights reserved.

DOWNLOADING THE SOURCE CODE AND BINARIES

Here you can find the complete NetBeans project including source code and binaries. File:TouchKeyboard.zip

DOWNLOADING THE JAVADOCS

Here you will find the API reference. File:Javadocs.zip

DOWNLOADING THE EXAMPLE

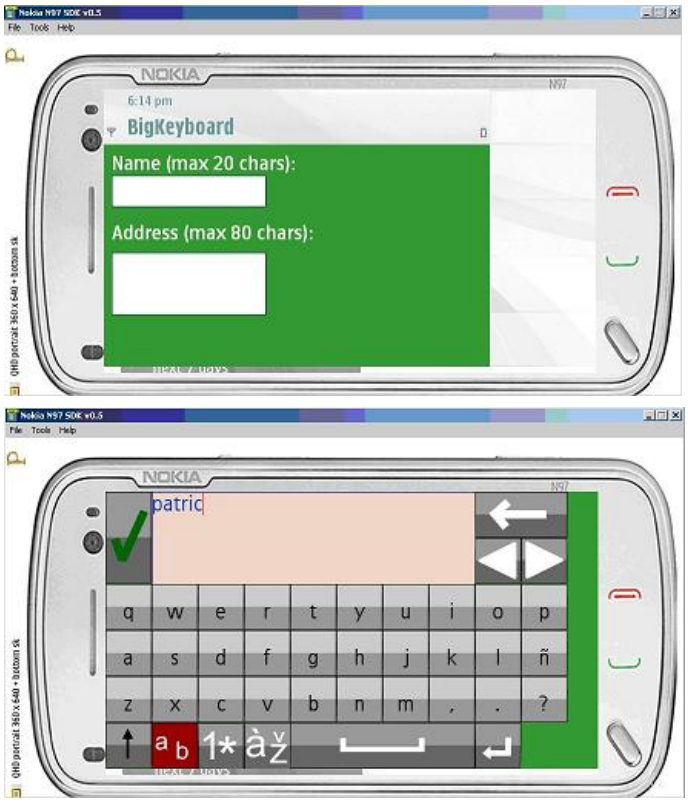
Bellow you will find an example using the component. You can download the complete example here. Media:KeyBoardExample.zip

Using the component

The following example shows how to use either the TouchKeyboard and TextFieldArea clases. The example consists in a main canvas with two TextFieldAreas. The responsibility of detecting which area was pointed by the user belongs to the application. Once the application detects which was the TextFieldArea touched then it displays the virtual keyboard. When user selects the OK button in the virtual keyboard, the text entered is stored in a variable according to the data (name or address) and this value is passed to the correct TextFieldArea. Next time the keyboard is invoked, it is fed with the text stored in the TextFieldArea.

Remember that TextFieldArea will adjust the text it holds to its defined own size, that's why the text inside the TouchKeyboard and the two TextFieldAreas might appear differently.

The picture bellow shows a couple screen shots of the application



JAD & Manifest Attributes

```
Nokia-MIDlet-On-Screen-Keypad: no
Nokia-MIDlet-App-Orientation: landscape
```

BigKeyboard.java

(C) Copyright Nokia 2014. All rights reserved.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import keyboard.TextFieldArea;
import keyboard.TouchKeyboard;
import keyboard.VirtualKeyboardListener;


public class BigKeyboard extends MIDlet
        implements VirtualKeyboardListener {

    public static BigKeyboard instance;     //static instance for the MIDlet

    private TouchKeyboard tkb;              //Virtual keyboard

    private Display display;
    private CanvasScreen canvas;            //Main application canvas

    int nameTransId;                        //transaction id to identify name

    int addressTransId;                     //transaction id to identify address


    public BigKeyboard() {
        instance = this;
        canvas = new CanvasScreen();
        display = Display.getDisplay(this);
    }

    public void startApp() {
        showCanvas();
    }

    /**
     * Shows the main application Canvas
     */
    public void showCanvas() {
        display.setCurrent(canvas);
    }

    /**
     * Shows the virtual keyboard in order to retrieve the name.
     * The name will be limited to 20 characters.
     * @param name
     */
    public void getName(String name) {
        if (tkb == null) {
            createKeyboard();            //create keyboard if it does not exists
        }
        //when keyboard is reseted, it returns a transactionId
        nameTransId = tkb.resetKeyBoard();
        tkb.setMaxSize(20);
        tkb.setText(name);
        display.setCurrent(tkb);
    }
```


(C) Copyright Nokia 2014. All rights reserved.

```
    /**
     * Shows the virtual keyboard in order to retrieve the name.
     * The name will be limited to 80 characters.
     * @param address
     */
    public void getAddress(String address) {
        if (tkb == null) {              //create keyboard if it does not exists
            createKeyboard();
        }
        //when keyboard is reseted, it returns a transactionId
        addressTransId = tkb.resetKeyBoard();
        tkb.setMaxSize(80);
        tkb.setText(address);
        display.setCurrent(tkb);
    }

    /**
     * Creates a new instance of the keyboard and sets it with the right
     * customization parameters
     */
    private void createKeyboard() {
        tkb = new TouchKeyboard(TouchKeyboard.KEYBOARD_BIG, 0, 0, false);
        tkb.setTextFieldBkgColor(0xE0DBCA);
        tkb.setTextFieldBorderColor(0x040477);
        tkb.setTextFieldFontColor(0x0033CC);
        tkb.setBackgroundColor(0x44A51C);
        tkb.togglePointer();
        tkb.setVirtualKeyboardListener(this);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }



    /**
     * Method of the interface VirtualKeyboardListener.
     * This method is invoked when the OK button is pressed at the moment
     * the keyboard is displayed on the screen.
     * @param transactionId
     * @param text
     */
    public void okPressed(int transactionId, String text) {
        //use the transactionId to identify the purpose of the call
        if (transactionId == nameTransId) {
            canvas.setName(text);
        } else if (transactionId == addressTransId) {
            canvas.setAddress(text);
        }
        showCanvas();
    }
}
```


(C) Copyright Nokia 2014. All rights reserved.

CanvasScreen.java

```
class CanvasScreen extends Canvas {

    /**
     * TextFieldArea containing the data of the name.  This textfield is only
     * to show information and so cursor should not be visible.
     */
    private TextFieldArea tfName;
    /**
     * TextFieldArea containing the data of the address. This textfield is only
     * to show information and so cursor should not be visible.
     */
    private TextFieldArea tfAddress;



    public CanvasScreen() {
        tfName = new TextFieldArea(200, 40);
        tfName.setCursorVisible(false);
        tfName.setTextFieldBorderColor(0xE024C1C);
        tfName.setTextFieldBkgColor(0xffffff);
        tfName.setTextFieldFontColor(0x0033CC);

        tfAddress = new TextFieldArea(200, 80);
        tfAddress.setCursorVisible(false);
        tfAddress.setTextFieldBorderColor(0xE024C1C);
        tfAddress.setTextFieldBkgColor(0xffffff);
        tfAddress.setTextFieldFontColor(0x0033CC);
    }

    /**
     * paints two rectangles in the screen
     * @param g
     */
    protected void paint(Graphics g) {
        g.setColor(68, 165, 28);
        g.fillRect(0, 0, getWidth(), getHeight());
        g.setColor(255, 255, 255);
        g.drawString("Name (max 20 chars):", 10, 10, 0);
        tfName.paint(g, 10, 40);
        g.setColor(255, 255, 255);
        g.drawString("Address (max 80 chars):", 10, 100, 0);
        tfAddress.paint(g, 10, 140);
    }

    /**
     * sets the text of the TextFieldArea destined to name
     * @param name
     */
    public void setName(String name) {
        tfName.setText(name);
    }
```


(C) Copyright Nokia 2014. All rights reserved.

```
    /**
     * sets the text of the TextFieldArea destined to address
     * @param address
     */
    public void setAddress(String address) {
        tfAddress.setText(address);
    }



    /**
     * Detects if user points the first or the second textFieldArea in the
     * touch device
     * @param x
     * @param y
     */
    protected void pointerPressed(int x, int y) {
        if (x > 10 && x < 210) {
            if ((y > 40) && (y < 80)) {       //Pointer hits the first textfield

                BigKeyboard.instance.getName(tfName.getText());
            } else if ((y > 140) && (y < 220)) {  //pointer hits the second textfield

                BigKeyboard.instance.getAddress(tfAddress.getText());
            }
        }
    }
}
```


(C) Copyright Nokia 2014. All rights reserved.