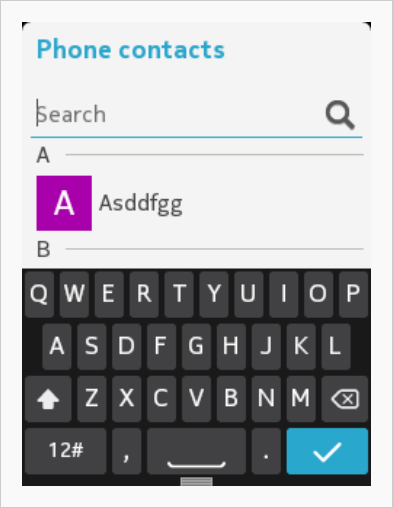
Creating LCDUI Custom Components: SearchBar

This article explains how to create a custom search bar component (and more generally any UI component) which mimics the look and feel of the native ones in Nokia Asha UI.

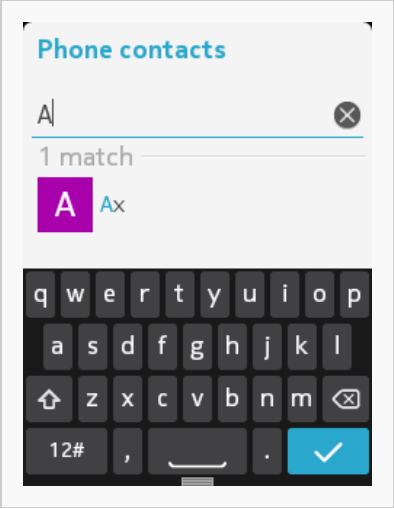
07 Jul 2013

Introduction

Although the LCDUI framework contains the essential UI components, it still lacks some of the items you see in the native apps. One such item is a search bar like the one in "Contacts". When you have several contacts, dragging the list of contacts down displays the search bar:



Nokia Asha native search bar focused



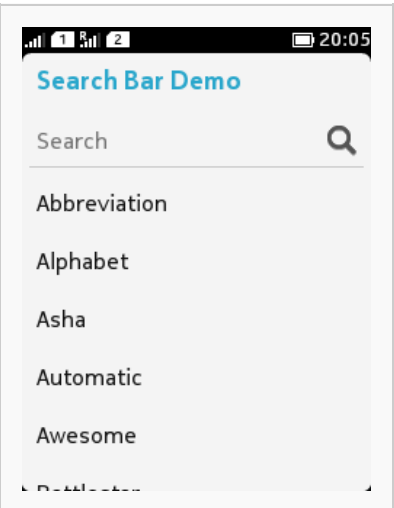
Nokia Asha native search bar with results line

The easiest way to implement the search bar with LCDUI framework is to use the provided TextBox class. However, the appearance of the `TextBox` is very plain and there's no possibility to add any icons on the same level vertically since items appended into the Form are positioned in a stack, one on top of the other. Going the distance and implementing components with enhanced look and feel may sometimes be worth it since it makes your app look more professional and appealing.

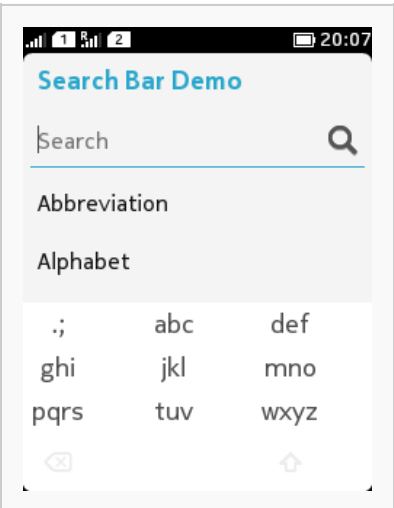
This article features a case study of creating a custom, `CustomItem`-based `SearchBar` for `Form` views.

Implementation

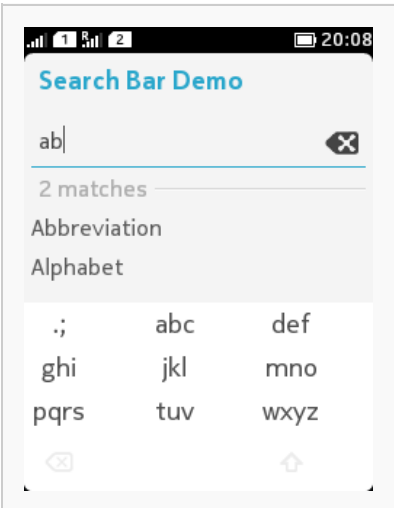
For comparison, it's best to start with the results of this case study; below are the screenshots of demo app (Media:SearchBarDemo.zip). You can immediately notice some major differences compared to the native component. Those are not due to the author exercising his artistic freedoms, but due to the limitations in the new platform. The problems are explained under Challenges and known issues.



The custom search bar, not focused



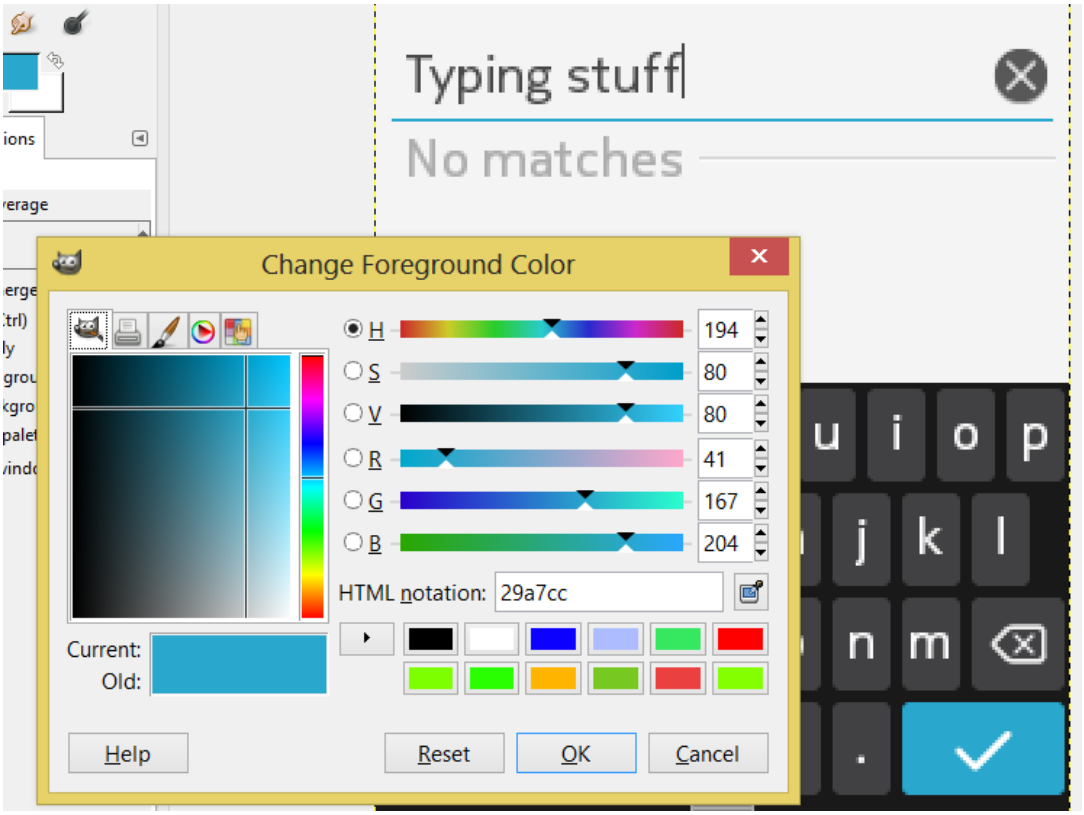
The custom search bar focused



The custom search bar with results line

Acquiring the look and feel

It may be possible to acquire some of the native colours using the existing APIs. Using the APIs to get properties for the components is more future proof, but sometimes not possible. In the latter cases the emulator is a good tool for reverse engineering the look and feel of the native components. You can get screenshots (to the clipboard) with CTRL-C key combination. Then paste the image to your image editing software. Here I've used Gimp which is feature-rich and free. Use the colour picker tool to get the platform colours (shown in the image below).



Insert the acquired colors in the code as constants. Here are all the colors used for the search bar:

```
private static final int TEXT_COLOR = 0x464646;
private static final int HINT_TEXT_COLOR = 0x717171;
private static final int RESULTS_TEXT_COLOR = 0xacacac;
private static final int TEXT_LINE_COLOR = 0xc8c8c8;
private static final int HIGHLIGHT_COLOR = 0x29a7cc;
private static final int RESULTS_LINE_COLOR = 0xd8d8d8;
private static final int RESULTS_LINE_SHADOW_COLOR = 0xfcfcfc;
```

You can study the margins, font styles and sizes etc. in similar fashion. Finally, once you have all figured out, start implementing the `paint()` method of your CustomItem derived class. Here's the `paint()` method of the SearchBarDemo:

```
    /**
     * @see javax.microedition.lcdui.CustomItem#paint(
     * javax.microedition.lcdui.Graphics, int, int)
     */
    protected void paint(Graphics graphics, int width, int height) {
        if (this.width != width) {
            this.width = width;
        }

        String temp = searchTerm;
        final int clearIconImageWidth = clearIconImage != null ?
            clearIconImage.getWidth() : 0;

        if (searchTerm.length() > 0) {
            graphics.setColor(TEXT_COLOR);
            final int maxTextWidth = width - searchAndClearIconWidth - MARGIN * 3;
            boolean hasBeenCut = false;

            while (font.stringWidth(temp) > maxTextWidth) {
                temp = temp.substring(1);
                hasBeenCut = true;
            }

            graphics.drawString(temp, MARGIN, MARGIN, Graphics.TOP | Graphics.LEFT);

            if (hasBeenCut && textCutterImage != null) {
                graphics.drawImage(textCutterImage, MARGIN, MARGIN,
                    Graphics.TOP | Graphics.LEFT);
            }

            if (focused && clearIconImage != null) {
                graphics.drawImage(clearIconImage,
                    width - MARGIN - clearIconImageWidth,
                    (HEIGHT_WITHOUT_RESULTS - clearIconImage.getHeight()) / 2,
                    Graphics.TOP | Graphics.LEFT);
            }
        }
        else {
            if (hintText != null) {
                graphics.setColor(HINT_TEXT_COLOR);
                graphics.drawString(hintText, MARGIN, MARGIN, Graphics.TOP | Graphics.LEFT);
            }
```

```
        }

        if ((searchTerm.length() == 0 || !focused) && searchIconImage != null) {
            graphics.drawImage(searchIconImage,
                width - MARGIN - searchIconImage.getWidth(), MARGIN,
                Graphics.TOP | Graphics.LEFT);
        }

        if (focused) {
            graphics.setColor(HIGHLIGHT_COLOR);
        }
        else {
            graphics.setColor(TEXT_LINE_COLOR);
        }

        graphics.drawLine(0, fontHeight + MARGIN * 2, width, fontHeight + MARGIN * 2);

        if (focused) {
            graphics.setColor(TEXT_COLOR);
            cursorX = MARGIN + font.stringWidth(temp);
            graphics.drawLine(cursorX, MARGIN, cursorX, fontHeight);
        }

        if (matchCount < 0) {
            // No need to paint the results bit.
            return;
        }

        graphics.setColor(RESULTS_TEXT_COLOR);
        final String resultsText =
            matchCount == 0 ? NO_MATCHES_TEXT :
                matchCount == 1 ? matchCount + MATCH_TEXT
                    : matchCount + MATCH_TEXT + "es";
        graphics.drawString(resultsText, MARGIN, fontHeight + MARGIN * 3,
            Graphics.TOP | Graphics.LEFT);

        graphics.setColor(RESULTS_LINE_COLOR);
        final int lineStartX = MARGIN * 2 + font.stringWidth(resultsText);
        final int lineY = height * 8 / 10;
        graphics.drawLine(lineStartX, lineY, width, lineY);
        graphics.setColor(RESULTS_LINE_SHADOW_COLOR);
        graphics.drawLine(lineStartX, lineY + 1, width, lineY + 1);
    }
```

Granted, it may be a lot of work, but when done properly, one might not be able to recognize yours from a native component.

Implementing the virtual keyboard support

Because it is impossible to incorporate a text box (or any other Item-based component) into your `CustomItem`, we have to implement the support for the virtual keyboard ourselves. Doing that isn't very complicated but it takes time and code lines. Luckily, Nokia UI API provides the classes to interface with the keyboard. You can divide the virtual keyboard implementation roughly to three parts: First, get the control for the keyboard and start listening to keyboard events:

```
/**
 * A search bar component with close to native look and feel.
 */
public class SearchBar
    extends CustomItem
    implements KeyboardVisibilityListener
{
...
CustomKeyboardControl vkbControl = null;
...
        VirtualKeyboard.setVisibilityListener(this);
        vkbControl = VirtualKeyboard.getCustomKeyboardControl();
...
    /**
     * @see com.nokia.mid.ui.KeyboardVisibilityListener#hideNotify(int)
     */
    public void hideNotify(int keyboardCategory) {
        System.out.println("SearchBar.hideNotify()");
        setFocused(false);
    }

    /**
     * @see com.nokia.mid.ui.KeyboardVisibilityListener#showNotify(int)
     */
    public void showNotify(int keyboardCategory) {
        System.out.println("SearchBar.showNotify()");
    }
```

Then, manage the launching and hiding of the keyboard:

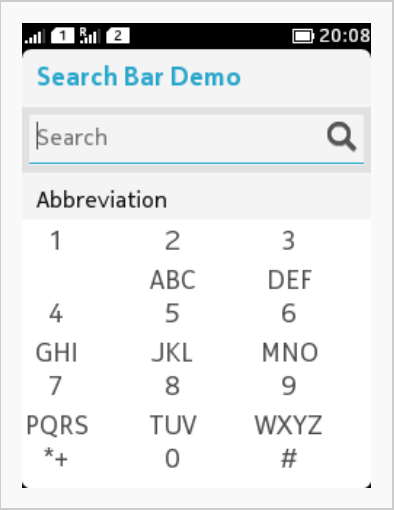
```
    /**
     * Sets the focus of this item.
     * @param focused If true, will launch the keyboard and change appearance.
     * If false, will hide the keyboard.
     */
    public void setFocused(boolean focused) {
        if (this.focused != focused) {
            System.out.println("SearchBar.setFocused(): " + focused);
            this.focused = focused;

            try {
                if (focused) {
                    vkbControl.launch(VirtualKeyboard.VKB_TYPE_ITUT,
                        VirtualKeyboard.VKB_MODE_ALPHA_LOWER_CASE);
                }
                else {
                    vkbControl.dismiss();
                }
            }
            catch (IllegalStateException e) {
            }

            repaint();
            listener.onSearchBarFocusedChanged(focused);
        }
    }
```

Finally, handle the keyboard events by overriding the `keyPressed()` method. This is the trickiest part, since we cannot get the native virtual keyboard but an ITUT keyboard. Thus, we need to implement rotating the characters associated with a single numeric key. The following code is also expected to work in Series 40 phones.

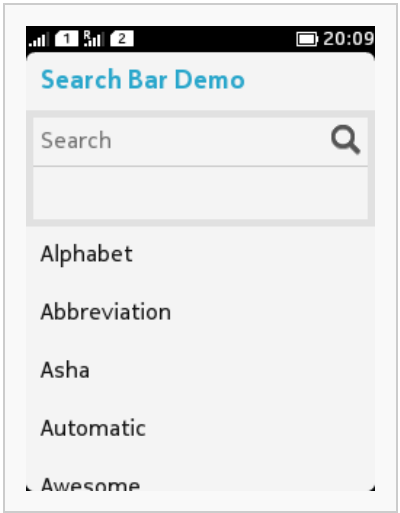
```
    private static final String[] ITUT_KEYS = {
        " 0", // 0
        ".;1", // 1
        "abc2", // 2
        "def3", // 3
        "ghi4", // 4
        "jkl5", // 5
        "mno6", // 6
        "pqrs7", // 7
        "tuv8", // 8
        "wxyz9" // 9
    };

...

    /**
     * @see javax.microedition.lcdui.CustomItem#keyPressed(int)
     */
    protected void keyPressed(int keyCode) {
        char key = (char)keyCode;
        System.out.println("SearchBar.keyPressed(): " + keyCode + " -> " + key);
        final int itutKeyIndex = keyCode - ZERO_KEY_CODE;

        if (keyCode == SPACE_KEY_CODE) {
            searchTerm += ' ';
            repaint();
            listener.onSearchTermChanged(searchTerm);
        }
        else if (keyCode == BACKSPACE_KEY_CODE) {
            if (searchTerm.length() > 0) {
                // Erase the last character
                searchTerm = searchTerm.substring(0, searchTerm.length() - 1);
                repaint();
                listener.onSearchTermChanged(searchTerm);
            }
        }
        else if (keyCode == TO_UPPER_KEY_CODE) {
            if (vkbControl.getKeyboardMode() == VirtualKeyboard.VKB_MODE_ALPHA_LOWER_CASE) {
                vkbControl.setKeyboardMode(VirtualKeyboard.VKB_MODE_ALPHA_UPPER_CASE);
            }
            else {
                vkbControl.setKeyboardMode(VirtualKeyboard.VKB_MODE_ALPHA_LOWER_CASE);
            }
        }
        else if (itutKeyIndex >= 0 && itutKeyIndex < ITUT_KEYS.length) {
            int index = 0;

            if (keyCode == lastKeyCode
                && lastKeyPressedTime > 0
                && System.currentTimeMillis() - lastKeyPressedTime
                    < ROTATE_CHARACTERS_THRESHOLD)
            {
                final int length = ITUT_KEYS[itutKeyIndex].length();

                for (int i = 0; i < length; ++i) {
                    if (searchTerm.charAt(searchTerm.length() - 1) ==
                        ITUT_KEYS[itutKeyIndex].charAt(i))
                    {
                        index = i;
                        break;
                    }
                }

                if (index + 1 == length) {
                   index = 0;
                }
                else {
                    index++;
                }

                searchTerm = searchTerm.substring(0, searchTerm.length() - 1);
            }

            if (vkbControl.getKeyboardMode() == VirtualKeyboard.VKB_MODE_ALPHA_LOWER_CASE) {
                searchTerm += ITUT_KEYS[itutKeyIndex].charAt(index);
            }
            else {
                String temp = new String();
                temp += ITUT_KEYS[itutKeyIndex].charAt(index);
                searchTerm += temp.toUpperCase();
            }

            repaint();
            listener.onSearchTermChanged(searchTerm);
            lastKeyPressedTime = System.currentTimeMillis();
        }
        else if (keyCode >= 0) {
            searchTerm += (char)keyCode;
            repaint();
            listener.onSearchTermChanged(searchTerm);
        }

        lastKeyCode = keyCode;
    }
```

Challenges and known issues

There were many challenges creating this specific component. The problems were caused by the limitations and few bugs of this new but rapidly maturing developer platform. The problems are likely to be fixed by the upcoming updates. The only major problem, making the component almost useless, is that a long press on the `CustomItem` is required in order for the keyboard to come visible.



Shadow-like highlight and wrong keyboard layout.



The height of the item does not return to its correct size after the "matches" line has once become visible.

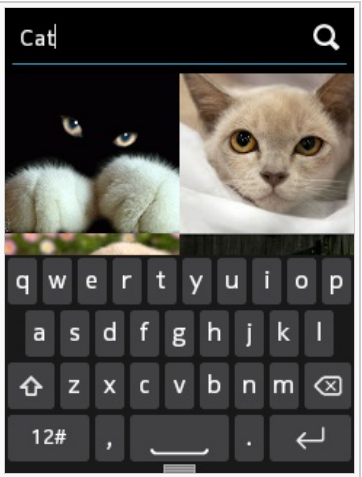
All the known issues are:

- A long press on the `CustomItem` is required in order for the keyboard to come visible.
- A long press on the `CustomItem` displays a shadow-like highlight, which cannot be removed, around the item.
- There is no API to open the native virtual keyboard. Instead, you're stuck with the ITUT keyboard.
- The initial mode of the virtual keyboard is wrong. Hiding the the keyboard using the back key and tapping on the search bar to bring it back is required to show the keyboard in the proper mode.
- After the search bar is extended to show the results line, the size of the component is not minimised even when the results line is hidden.
- Sometimes the items in the form are not appended in the proper order.

`Form`-based views set some additional limitations when creating custom components e.g. you cannot freely position elements since items are always positioned in a stack. However, unlike with Canvas, you get ready-made features such as view scrolling with scroll bar element.

SearchBar - Canvas edition

Using this `CustomItem`-based solution as the basis, a new `Canvas`-based search bar was implemented in the version 1.1 of PicasaViewer example. With `Canvas` there are much less problems and by integrating the `TextEditor` inside the search bar, a native virtual keyboard could be used. See the implementation of the search bar here: SearchBar.java. Get the full source code of the example here: https://github.com/nokia-developer/picasa-viewer



The search bar in PicasaViewer example forced to find fluffy animals.

Summary

Although the LCDUI framework provides APIs for all the essential UI components, there are still few components you can find from the native apps that are not available for developers. The LWUIT framework provides a more extensive set of components, but if neither that nor `Canvas` suits your purposes, using `CustomItem` is the way to go. It is easy to design reusable components, but since the Nokia Asha software platform is still fairly young, there are some gaps in the developer offering. These obstacles are bound to be removed in the future as the platform matures.

See also

- Creating LCDUI Custom Components: CategoryBar
- PicasaViewer example