High Level UI using CustomItem

Version 0.2, Draft

INFORMATION GUIDE

COPYRIGHT

Trademarks and Service Marks

About This Document

This document gives an overview of High Level Item class *CustomItem* and provides a sample code snippet explaining the implementation of class *CustomItem* on Form.

Scope

This document is intended for Java ME developers wishing to develop Java ME applications. It assumes good knowledge of java programming language.

Document History:

Date	Version	Comment
04/02/09	0.2	Draft

Reference:

1. MIDP 2.0 Specification:

http://jcp.org/en/jsr/detail?id=118

Abbreviations:

MIDP	Mobile Information Device Profile
UI	User Interface

Table of Contents

List of Tables

Introduction

MIDP 2.0 has introduced a new user interface class *CustomItem* in *javax.microedition.lcdui* package that is specially intended to create your own Items (UI element). *CustomItem* extends *javax.microedition.lcdui.Item. CustomItem* is created by subclassing *javax.microedition.lcdui.CustomItem* class to create a new visual appearance and interactive element.
Visual appearance includes sizing, rendering, choosing the colors, fonts and graphics. User interaction is done by responding to events generated by keys, pointer actions and traversal actions.

Overview

All Item objects have a label field; a string that identifies the item. Label is displayed near the component on devices display. User can omit the label by passing null.

CustomItem (String label)

Subclass who extends the *CustomItem* has to implement the first five abstract methods shown in table 1.

Table 1: CustomItem Methods

Sr.No	Method	Description
1	getMinContentHeight()	Returns the minimal height of the content area.
2	getMinContentWidth()	Returns the minimal width of the content area.
3	getPrefContentHeight(int width)	Returns the preferred height of the content area. A tentative value for the opposite dimension “width” is passed to aid in the height calculation. The tentative value should be ignored if it is -1.
4	getPrefContentWidth(int height)	Returns the preferred width of the content area. A tentative value for the opposite dimension “height” is passed to aid in the width calculation. The tentative value should be ignored if it is -1.
5	paint(Graphics g, int width, int height)	Draws the item's content area, whose dimensions are given by the width and height parameters.
6	traverse(int dirn, int viewportWidth, int viewportHeight, int[] visRect_inout)	Called by the system when traversal has entered the item or has occurred within the item.

First, like other items, *CustomItem* has the concept of the **content size**. Content size is only the size of content area of *CustomItem*. Content area is a rectangular area within which the *CustomItem* subclass paints and receives input events. It does not include space consumed by labels and borders. Content area is a subset of Item's total area. The implementation is responsible for lying out, painting, and handling input events within the area of the Item that is outside the content area. Label string is drawn and positioned outside the content area by the system.

Second Items have the concept of **minimum** and **preferred sizes**. These include the total area of the item. The minimum size is the smallest size at which the Item can function and display its contents, though perhaps not optimally. The preferred size is generally a size based on the Item's contents and is the smallest size at which no information is clipped and text wrapping (if any) is kept to a tolerable minimum.

The first four methods shown in Table 1 deals with the minimum and preferred size of the Item.

Third custom items rendering is done by implementing *paint()* method shown in Table 1. The rendering part is very much the way canvases render. However, the height and width parameters are passed along with the Graphics object. The height and width are the current dimensions of the item's content area. All drawing is done on this dimensions and drawing origin is at the top left corner of the content area.

MIDP provides a mechanism for custom items to support traversal in a way that is consistent and portable. The *traverse()* method will return true if the traversal is internal and false if it should proceed out to another item in the form. The method's parameters indicate the direction of traversal, the size of the viewable area, and other information about the area. It will be up to the user to define when the *traverse()* method should return false, resulting in the activation of the next item on the form. The direction value of traversal will be one of the directional game actions defined in the Canvas class: *Canvas.UP*, *Canvas.DOWN*, *Canvas.LEFT*, and *Canvas.RIGHT*, or the value *CustomItem.NONE*. If the value is NONE, some platform-specific event like a resizing of the form caused item to gain the focus.

Sample code snippet for CustomItem

The sample code given below shows how to use *CustomItem* class.

Class: CustomItemMIdlet.java

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Form;

public class CustomItemMidlet extends MIDlet {

   private Display display = null;
   private Form form = null;

   /*The non parameterized constructor */
   public CustomItemMidlet() {
      init();
   }

   public void startApp() {
      display.setCurrent(form);
   }

   /*The pauseApp() method signals the MIDlet that it has entered the Pause state */
   public void pauseApp() {
      /*all interupts can be handled here*/
   }

   public void destroyApp(boolean flag) {
   }

   /*The getWidth() method  returning the width in pixels of the displayable area available to the
     application */
   public int getWidth() {
      return form.getWidth();
   }

   /*The getHight() method  returning the height in pixels of the displayable area available to the
     application */
   public int getHight() {
      return form.getHeight();
   }

   /* The init()method  for to initializing the object */
   public void init() {
      display = Display.getDisplay(this);
      form = new Form("CustomItem Demo");
      form.append(new NewForm(this, display));
   }
}
```

Class: NewForm.java

```
import java.io.IOException;
import javax.microedition.lcdui.CustomItem;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.ItemCommandListener;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Font;

public class NewForm extends CustomItem implements ItemCommandListener {

   private CustomItemMidlet mainMidlet = null;
   private Display display = null;

   /*The class Command final Object CMD_EXIT to destroy the midlet*/
   private final Command CMD_EXIT = new Command("Exit", Command.EXIT, 1);

   /*The class String final Object samsungLogo to provide Image name*/
   private final String samsungLogo[] = {"image1.png",
      "image2.PNG",
      "image3.png",
      "image4.png",
      "image5.png",
      "image6.png"
   };
   private final String photoAlbum = "Photo Album";
   private final Font font = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN,
         Font.SIZE_MEDIUM);
   private Image imageArray[];
   private int displayableWidth;
   private int displayableHeight;
   private int initalIndex;
   private int finalIndex;
   private int ImageCounter;

   /*The parameterized constructor*/
   public NewForm(CustomItemMidlet mid, Display dis) {
      super(null);

      mainMidlet = mid;
      display = dis;

      displayableWidth = mainMidlet.getWidth();
```

```
    displayableHeight = mainMidlet.getHight();
    imageArray = new Image[samsungLogo.length];
    finalIndex = imageArray.length;

    for (int i = 0; i < samsungLogo.length; i++) {
        try {
            imageArray[i] = Image.createImage(
                    getClass().getResourceAsStream(samsungLogo[i]));
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
    addCommand(CMD_EXIT);
    setItemCommandListener(this);
}

/*The getMinContentWidth() Implemented by the subclass */
protected int getMinContentWidth() {
    return 0;
}

/*The getMinContentHeight() Implemented by the subclass */
protected int getMinContentHeight() {
    return 0;
}

/*The getPrefContentWidth() method Implemented by the subclass*/
protected int getPrefContentWidth(int width) {
    return displayableWidth;
}

/*The getPrefContentHeight() method Implemented by the subclass*/
protected int getPrefContentHeight(int height) {
    return displayableHeight;
}

/*The paint() method Implemented by the subclass*/
protected void paint(Graphics g, int width, int height) {
    g.setFont(font);
    g.setColor(157, 157, 255);
    g.fillRect(0, 0, width, height);
    g.setColor(0, 0, 0);
    g.drawString(photoAlbum, (width / 2) - (font.stringWidth(photoAlbum) / 2),
            height * 5 / 100, Graphics.TOP | Graphics.LEFT);
    g.drawImage(imageArray[ImageCounter], (width / 2) -
            (imageArray[ImageCounter].getWidth() / 2), (height * 5 / 100) +
            ((height / 2) - (imageArray[ImageCounter].getHeight() / 2)),
```

```
            Graphics.TOP | Graphics.LEFT);
    }

    protected void keyPressed(int keyCode) {
        switch (keyCode) {
            case Canvas.KEY_NUM2:
                ImageCounter--;
                if (ImageCounter < initalIndex) {
                    ImageCounter = finalIndex - 1;
                }
                break;
            case Canvas.KEY_NUM8:
                ImageCounter++;
                ImageCounter = ImageCounter % finalIndex;
                break;
        }
        repaint();
    }

    /* The traverse() method is called by the system when traversal has entered
     *the item or has occurred within the item */
    protected boolean traverse(int dir, int portWidth, int portHeight,
            int[] visRect_inout) {
        switch (dir) {
            case Canvas.DOWN:
                /*User can implements its code*/
                return true;
            case Canvas.UP:
                /*User can implements its code*/
                return true;
        }
        return true;
    }

    public void commandAction(Command cmd, Item item) {
        if (cmd == CMD_EXIT) {

            mainMidlet.destroyApp(true);
            mainMidlet.notifyDestroyed();
        }
    }
}
```