High Level UI using Gauge

Version 0.9, Draft

INFORMATION GUIDE

COPYRIGHT

Samsung Electronics Co. Ltd.
This material is copyrighted by Samsung Electronics. Any unauthorized reproductions, use or disclosure of this material, or any part thereof, is strictly prohibited and is a violation under the Copyright Law Samsung Electronics reserves the right to make changes in specifications at any time and without notice. The information furnished by Samsung Electronics in this material is believed to be accurate and reliable, but is not warranted true in all cases.

Trademarks and Service Marks

The Samsung Logo is the trademark of Samsung Electronics. Java is the trademark of Sun Microsystems.
All other company and product names may be trademarks of the respective companies with which they are associated.

About This Document

This document gives an overview of High Level Item class Gauge: Interactive and Non Interactive and provides a sample code snippet.

Scope:

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

Document History:

Date	Version	Comment
04/02/09	0.9	Draft

Reference:

1. Gauge article:

   http://java.sun.com/developer/J2METechTips/2001/tt0917.html

Abbreviations:

MIDP	Mobile Information Device Profile
UI	User Interface

Table of contents

Table of Figures

Introduction

Mobile Information Device Profile (MIDP) package *javax.microedition.lcdui* has a number of High Level User Interface (UI) components. These UI components are referred to as Item since they extend *javax.microedition.lcdui.Item* class.

One of the UI components in MIDP package is *Gauge*. This document describes about *Gauge* and its importance in developing a MIDlet application.

Overview

Gauge is an Item, which represents a bar graph (Progress bar or Activity bar) display that can be used within a form. Gauge is used to display an integer value that lies between the ranges of specified values (initial to maximum). User can specify maximum & initial value.

Types of Gauge

- Interactive Gauge
- Non Interactive Gauge

Figure 1 shows two different types of Gauge:

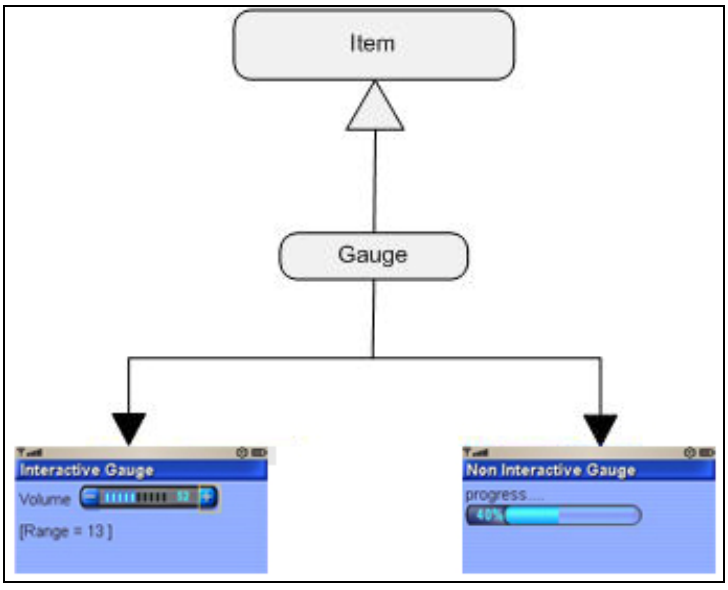

Figure 1: Different types of Gauge

Interactive Gauge:

In Interactive Gauge, user can modify the gauge value. User can change the gauge value from initial to Max or vice versa i.e. increment or decrement by some factor. Interactive Gauge does not allow the user to move the value outside the established range (Initial to Max).

Non interactive Gauge:

Non Interactive Gauge does not allow the user to modify the value. Non Interactive gauge gives feedback to the user on the state of long running operation. **Progress indicator** and **Activity indicator** are the two expected behaviors in this type, which give feedback during long running operation. The Non Interactive gauge can have definite or indefinite range.

Creating Gauge:

Gauge is derived from Item class, creating Interactive Gauge or Non Interactive Gauge object is done by invoking Gauge constructor:

```
Gauge (String label, boolean interactive, int maxValue, int initialValue)
```

Where
label – indicates the Gauge name.
interactive – setting this variable to:

true: makes interactive Gauge.
false: makes non interactive Gauge.

maxValue, initialValue – are the specified range between minimum and maximum value set by the user.

Sample code snippet for Interactive Gauge & Non Interactive gauge:

Given below are the examples for Interactive and Non Interactive Gauge.

Example 1: Gauge Interactive

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Gauge;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.ItemStateListener;
import javax.microedition.lcdui.StringItem;

public class GaugeMidlet extends MIDlet implements CommandListener, ItemStateListener {

   private Display display = null;
   private Form form = null;
   private Gauge gauge = null;
   private static final String EXIT = "Exit";
   private final int GAUGE_INITIALVALUE = 1;
   private final int GAUGE_MAXVALUE = 25;
   private final Command CMD_EXIT = new Command(EXIT, Command.EXIT, 1);
   private StringItem stringItem = null;

   public GaugeMidlet() {
      init();
   }

   public void startApp() {
      form.append(gauge);
      form.append(stringItem);
      form.setItemStateListener(this);
      form.addCommand(CMD_EXIT);
      form.setCommandListener(this);
      display.setCurrent(form);
   }

   public void pauseApp() {
      /*User can handle pauseApp state as per requirement */
   }
```

```
    public void destroyApp(boolean flag) {
        /*User can handle destroyApp state as per requirement */
    }

    /*
     * The function is used for to initializing the object
     */
    public void init() {
        display = Display.getDisplay(this);
        form = new Form("Interactive Gauge");
        gauge = new Gauge("Volume", true, GAUGE_MAXVALUE, GAUGE_INITIALVALUE);
        stringItem = new StringItem(null, "[ Range = " + GAUGE_INITIALVALUE + " ]",
            Item.PLAIN);
    }

    public void commandAction(Command cmd, Displayable dis) {

        if (cmd == CMD_EXIT) {
            destroyApp(true);
            notifyDestroyed();
        }

    }

    public void itemStateChanged(Item item) {
        if (item == gauge) {

            stringItem.setText("[Range = " + gauge.getValue() + " ]");

        }
    }
}
```

Example 2: Gauge Non Interactive

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Gauge;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;

public class GaugeMidlet extends MIDlet implements CommandListener, Runnable {

    private Display display = null;
```

```
private static final String EXIT = "Exit";
private static final String START = "Start";
private static final String STOP = "Stop";
private final Command CMD_EXIT = new Command(EXIT, Command.EXIT, 1);
private final Command CMD_START = new Command(START, Command.SCREEN, 2);
private final Command CMD_STOP = new Command(STOP, Command.SCREEN, 3);
private Gauge gauge = null;
private final int GAUGE_MAXVALUE = 200;
private final int GAUGE_INITIALVALUE = 1;
private final int GAUGE_PROGRESS = 10;
private final int SLEEP_TIME = 200;
private final int MAXCHOICE = 2;
private Form form = null;
private boolean on_off_Flag;
private int switchCommand;

public GaugeMidlet() {
   init();
}

public void startApp() {
   form.addCommand(CMD_EXIT);
   form.addCommand(CMD_START);
   form.append(gauge);
   form.setCommandListener(this);
   display.setCurrent(form);
}

public void pauseApp() {
   /*User can handle pauseApp state as per requirement */
}

/*
 *The destroyApp() method signals the MIDlet to terminate and enter the Destroyed state
 */
public void destroyApp(boolean unconditional) {
   /*User can handle destroyApp state as per requirement */
}

public void commandAction(Command cmd, Displayable dis) {
   if (cmd == CMD_EXIT) {
      destroyApp(true);
      notifyDestroyed();
   } else {
      switchCommand++;
      switchCommand = switchCommand % MAXCHOICE;
      on_off_Flag = (switchCommand > 0) ? false : true;
```

```
            if (!on_off_Flag) {
                form.removeCommand(CMD_START);
                form.addCommand(CMD_STOP);
                new Thread(this).start();
            } else {
                form.removeCommand(CMD_STOP);
                form.addCommand(CMD_START);
            }


        }
    }

    /*the function is used for to initializing the object */
    public void init() {
        display = Display.getDisplay(this);
        form = new Form("Non Interactive Gauge");
        gauge = new Gauge("progress....", false, GAUGE_MAXVALUE, GAUGE_INITIALVALUE);
    }

    public void run() {
        while (!on_off_Flag) {
            try {
                Thread.sleep(SLEEP_TIME);
                if ((gauge.getValue()) >= gauge.getMaxValue()) {
                    gauge.setValue(0);
                }
                gauge.setValue(gauge.getValue() + GAUGE_PROGRESS);
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```