Low Level UI Canvas

Version 0.9, Draft

API GUIDE

COPYRIGHT

Trademarks and Service Marks

About this Document

This document describes how to implement low-level UI with some sample code snippets.

Scope

This document is intended for Java ME developers who wish to develop Java ME applications. It assumes good knowledge of java programming language.
To know about Java ME basics and Java programming language, refer to the Knowledge Base under Samsung Mobile Innovator (SMI).

http://innovator.samsungmobile.com/platform.main.do?platformId=3

Document History:

Date	Version	Comment
22/06/09	0.9	Draft

References:

User Interface Programming- http://developers.sun.com/mobility/midp/articles/ui/-

Abbreviations:

Java ME	Java Micro Edition
MIDP	Mobile Information Device Profile
API	Application Programming Interface

Table of Contents

Table of Figures

Introduction

In Java ME application, MIDlet interaction with the user is done through UI element. There are two ways to design UI elements as shown in Figure 1

- High Level API (Screen)
- Low Level API (Canvas)

High level API is used to achieve portability but one cannot get more control over its look and feel. E.g., Visual appearance of the components cannot be defined. These visual appearances may be shape, color or font etc.

Low-level API is used in applications with precise control over graphic elements and access to low level events. It provides methods to handle key events, game actions and pointer events (if supported by the device). Low-level API is used to draw images, perform animations, scrolling, updating the display etc. It is mainly used for games development and rich UI applications.

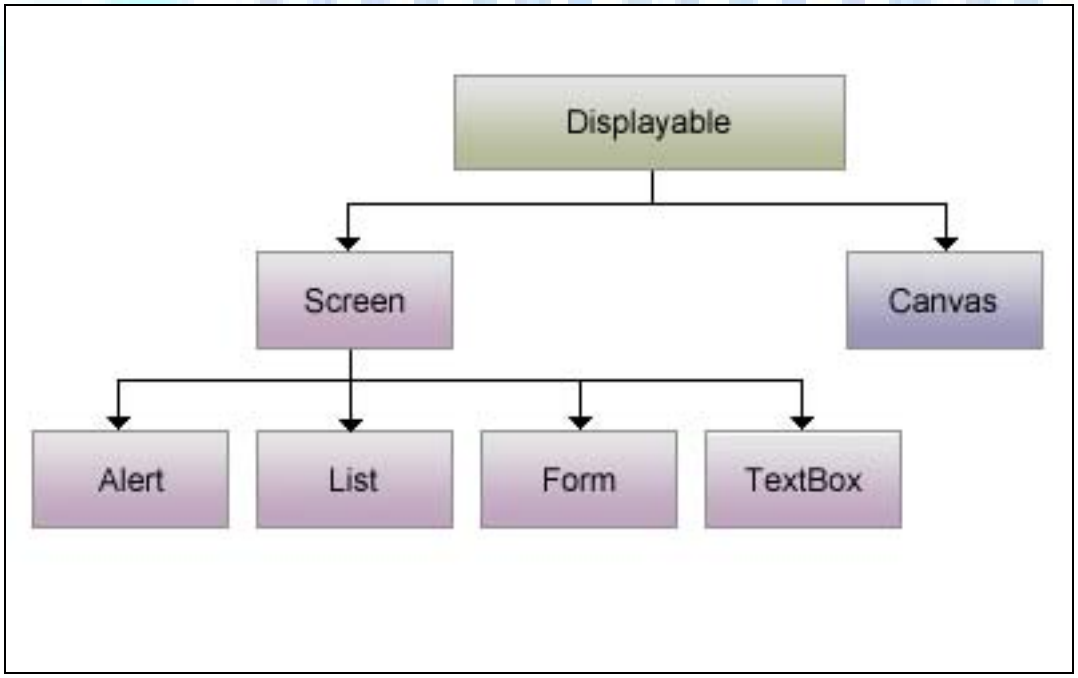

Figure 1: UI Elements

With the help of both APIs, Applications and Games for Samsung Handsets can be developed. To know more about High Level API, please refer to the Samsung Innovator > Java > Knowledge Base.

Overview

Low Level UI is mainly implemented by two classes present in *javax.microedition.lcdui* package. Those are:

- Canvas
- Graphics

Canvas

Canvas is a base class for developing Java ME applications that need to handle low-level events and to issue graphics call for drawing to the display. Different methods are provided by the Java ME developers to handle game actions, key events and pointer events (if supported by the device). Methods are also provided to identify the device's capabilities and mapping of keys to game actions.

Adding Commands in Canvas

Like other subclasses of Displayable, the Canvas class allows the application to register a listener for commands. Unlike other Displayables, however, the Canvas class requires applications to subclass it in order to use it.

The sample code snippet shows how to add commands in Canvas:

```
public class CommandCanvas extends Canvas implements CommandListener{

   private CanvasMidlet midlet;
   private int width;
   private int height;
  private static final String EXIT = "Exit";
  private static final String OK    = "Ok";
   private final Command CMD_EXIT=new Command(EXIT,Command.EXIT,0);
   private final Command CMD_OK=new Command(OK,Command.OK,1);

   public CommandCanvas(CanvasMidlet midlet){
      this.midlet=midlet;
      width=getWidth();
      height=getHeight();

   }

   public void paint(Graphics g){
      g.setColor(255,255,255);
      g.fillRect(0,0,width,height);
      this.addCommand(CMD_EXIT);
      this.setCommandListener(this);
```

```
  }

  public void commandAction(Command cmd,Displayable disp){
    if(cmd==cmdExit){
      midlet.notifyDestroyed();
      midlet.destroyApp(true);

    }
  }

}
```

Creating user interface using Low Level UI

Canvas is an abstract class. Application must be derived from the Canvas and implement the paint() method to create a user interface using the low-level APIs. paint() method accepts a Graphics object, which provides methods for rendering on the device screen.

The sample code snippet shows how to draw a string using paint() method:

```
public class MainCanvas extends Canvas{
    public void paint(Graphics g){
      g.setColor(255,255,255);
      g.drawString("SMI Java",0,0,Graphics.TOP|Graphics.LEFT);
    }
}
```

Full screen using Canvas

There are two modes in which Canvas can be displayed.

- Normal mode - In normal mode, space on the display may be occupied by command labels, a title, and a ticker.
- Full Screen mode - In Full Screen mode, the application is requesting that the Canvas occupies as much of the display space as is possible.

Canvas objects are in normal mode by default. The normal mode and full-screen mode is controlled using setFullScreenMode(boolean) method.

Calling setFullScreenMode(boolean) may result in sizeChanged() being called. The default implementation of this method does nothing. The application can override this method to handle changes in size of available drawing area.

Canvas width and height can be found through the use of getWidth() and getHeight() method. getWidth() returns the width of the canvas and getHeight() returns the height of the canvas.

The sample code snippet below shows how to set full screen in canvas:

```
public class FullCanvas extends Canvas{
    public FullCanvas(){
        setFullScreenMode(true);
    }

    public void paint(Graphics g){
        g.setColor(0,0,245);
        g.fillRect( 0, 0, getWidth(), getHeight());
    }

}
```

Low-Level Event Handling

To monitor key events, a subclass of canvas must override one of these methods as follows:

Method Name	Description
protected void keyPressed(int keycode)	This event is called when a physical key on the keypad is pressed.
protected void keyReleased(int keycode)	This event is called when a pressed key is released.
protected void keyRepeated(int KeyCode)	This event is called when a key is held down. This event may not be supported on all platforms. Support of this method can be verified with a call to hasRepeatEvents().
protected void pointerPressed(int x, int y)	This event is called when a pointer is pressed. This event may not be supported on all platforms. Support of this method can be verified with a call to hasPointerEvents().
protected void pointerDragged(int x, int y)	This event is called when a pointer is dragged. This event may not be supported on all platforms. Support of this method can be verified with a call to hasPointerMotionEvents().
protected void pointerReleased(int x, int y)	This event is called when a pointer is released. This event may not be supported on all platforms. Support of this method can be verified with a call to hasRepeatEvents().

keyRepeated is not necessarily available in all devices. The applications can check the availability of repeat actions by calling the hasRepeatEvents() method of the Canvas. hasRepeatEvents() returns true if the device supports keyRepeated functionality.

An integer code corresponding to the key to which the event pertains is passed as a parameter in keyPressed, keyRepeated and keyReleased methods. It is important to understand that key codes are likely to change between different manufacturer devices, making key event handling another sensitive area of low-level UI portability.

E.g., On device A, pressing LeftSoftKey on the keypad may come through as key code 21 but on device B the same key might come through as key code -6. Luckily, MIDP provides tools to assist in normalizing key press events.

To make key handling generic across the devices, applications should use standard keycodes defined by MIDP as shown in below code snippet.

```
public void keyPressed(int keycode)
{
    if (keycode == KEY_NUM0)
    {
      System.out.println("Pressed Key 0");
    } else if(keycode == KEY_NUM1){
      System.out.println("Pressed Key 1");
    } else if(keycode == KEY_NUM2){
      System.out.println("Pressed Key 2");
    }

}
```

There are portable applications, which need arrow key events and gaming-related events. These applications should use game actions in preference to key codes and key names.
MIDP defines the following set of abstract key events: UP, DOWN, LEFT, RIGHT, FIRE, GAME_A, GAME_B, GAME_C and GAME_D.

An application can get the mapping of the key events to abstract key events by calling Canvas.getGameAction(int keycode) method as shown in sample code snippet below:

```
public void keyPressed(int key)
{
    int action = getGameAction(key);
    switch (action)
    {
        case UP:

            form.append("Pressed UP arrow key");
```

```
            break;

        case LEFT:

           form.append ("Pressed LEFT arrow key");
            break;

        case GAME_B:

           form.append ("Pressed GAME_B key");
            break;


    }
}
```

Low Level External Interrupts

Interruptions such as incoming calls to a Java ME application in a device invokes hideNotify() and showNotify() on any displaying canvas.

The showNotify() method is called prior to the Canvas actually being made visible on the display, and the hideNotify() method is called after the Canvas has been removed from the display.

```
/*called when an interrupt such as incoming call is received*/

protected void hideNotify() {

}

/*called when an interrupt such as incoming call is ended*/

protected void showNotify() {

}
```

Graphics

Graphics object is used by the Canvas to do all 2D geometric rendering capability. Rendering of Graphics can be done directly to the display or to an off-screen image buffer.

- Graphics object aids to draw text, images, lines etc. Rectangles and arcs may also be filled with a solid color. Rectangles may also be specified with rounded corners.

- A 24-bit color model is provided with 8 bits for each of red, green, and blue components of a color. Not all devices support a full 24 bits worth of color and thus they will map colors requested by the application into colors available on the device.

- In the Display class, facilities are provided for obtaining device characteristics, such as whether color is available and how many distinct gray levels are available.

- With the help of Graphics object, you can draw images and create animation by manipulating the pixel level. Rendering can be done directly to a screen or to an off screen image buffer.

- Methods required for drawing content are: drawImage() for drawing images, drawLine() for drawing lines, drawString() for drawing strings.

- Co-ordinates represent pixel locations. The default coordinate system's origin is at the upper left-hand corner of the destination. The X-axis direction is positive towards the right, and the Y-axis direction is positive downwards. See figure 2.

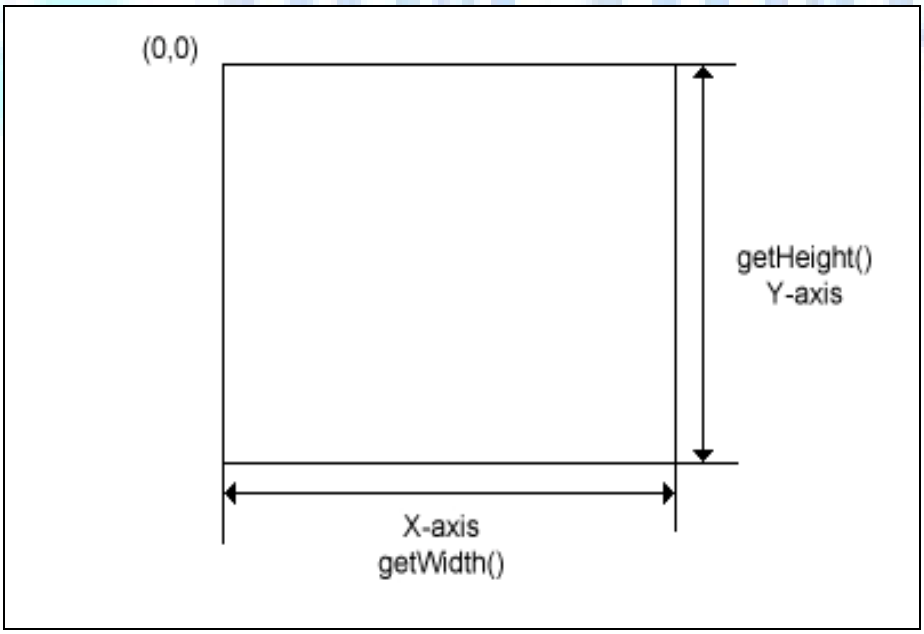

Figure 2: X-Y Co-ordinates

- The translate() method is used to change the origin of the coordinate system.

- The region of the screen where drawing takes effect can be further limited to a rectangular area by the clipRect() method. Drawing outside the clip area will have no effect. See figure 3.

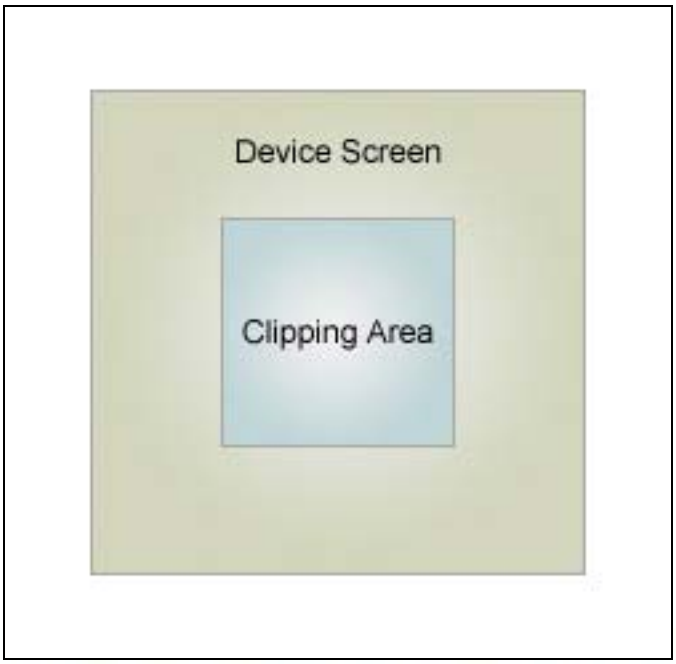

Figure 3: Clipping Region

With the help of the anchor points, you can draw the text. These anchor points are used to minimize the amount of computation required when placing text. E.g., In order to center a piece of text, an application needs to call stringWidth() or charWidth() to get the width and perform a combination of subtraction and division to compute the proper location.

How to call paint?

Java ME developer can create instance of Canvas class to call **paint()** method. We can also call **paint()** asynchronously by **repaint()** method.

The call of the **paint()** method using **repaint()** is not performed immediately. It may be delayed until the control flow returns from the current event handling method. Here delay is not a problem, but while doing animation, the safest way to trigger repaints is to use **Display.callSerially()** or to request the repaint from a separate **Thread** or **TimerTask**. Alternatively, the application can force an immediate repaint by calling **serviceRepaints()**.

Applications can use either **Display.callSerially()** or **serviceRepaints()**to synchronize with its **paint()** routine or they can code explicit synchronization into their **paint()** routine.

Sample source code showing how to display a splash screen using Canvas

This sample source code shows how to display a splash screen using Canvas:

Class: SplashMidlet.java

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

public class SplashMidlet extends MIDlet {

   /*current display object*/
   private Display display;

   /*object for displaying canvas screen*/
   private SplashCanvas canvas;

   /* midlet constructor*/
   public SplashMidlet() {
      display = Display.getDisplay(this);
      /*initializing the SplashCanvas class*/
      canvas = new SplashCanvas(this);
   }

   /*called when the midlet is called for first time*/
   public void startApp() {
      display.setCurrent(canvas);
   }

   /*called when any interrupt occurs and can handle the player*/
   public void pauseApp() {
      /*can write code here for handling all interrupts
      and for unhandled garbage collections*/
   }

   /*is used to destroy the Midlet cleanup all that are not
   handled by garbage collection*/
   public void destroyApp(boolean unconditional) {
   }
   /*This method is used to exit the midlet*/
   public void exitMidlet() {
      destroyApp(true);
      notifyDestroyed();
   }
}
```

Class SplashCanvas.java

```
import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Font;
import javax.microedition.lcdui.Graphics;

public class SplashCanvas extends Canvas implements Runnable, CommandListener {

   /*to create midlet object*/
   private SplashMidlet midlet;
   /*int object to get width of the screen*/
   private int width;
   /*int object to get width of the screen*/
   private int height;
   /*boolean object*/
   private boolean remSplash = false;
   /*this command is used to exit from the current application*/
   private final Command cmdExit = new Command("Exit", Command.EXIT,
     2);
   /*font object*/
   private Font font = null;

   public SplashCanvas(SplashMidlet midlet) {
      this.midlet = midlet;
      /*initializes the font to be displayed*/
      font = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN,
        Font.SIZE_MEDIUM);
      /*need to start the thread or timer to make splash screen remain
      for some time*/
      new Thread(this).start();
      setCommandListener(this);
   }

   public void paint(Graphics g) {
      /*gets the width of the screen*/
      width = getWidth();
      /*gets the height of the screen*/
      height = getHeight();
      /*sets the font property*/
      g.setFont(font);
      g.setColor(255, 255, 255);
      g.fillRect(0, 0, width, height);
      g.setColor(0, 0, 255);
      g.drawString("Splash Screen over", width / 2, height / 2,
```

```
        Graphics.HCENTER | Graphics.BASELINE);
    /*code to show splash screen*/
    if (!remSplash) {
      g.setColor(0, 0, 255);
      g.fillRect(0, 0, width, height);
      g.setColor(255, 255, 255);
      g.drawString("Welcome to ", width / 2, height / 2,
        Graphics.HCENTER | Graphics.BASELINE);
      g.drawString("Samsung Mobile Innovator", width / 2, (height / 2) +
        font.getHeight(), Graphics.HCENTER | Graphics.BASELINE);
    }
  }

  public void run() {
    synchronized (this) {
      try {
        /*Spash Screen appears for 4 sec*/
        wait(4000L);

      } catch (InterruptedException ie) {
        ie.printStackTrace();
      }
    }
    exitSplash();
  }

  public void commandAction(Command cmd, Displayable disp) {
    if (cmd == cmdExit) {
      midlet.exitMidlet();
    }
  }

  /*to exit from the splash screen*/
  public void exitSplash() {
    remSplash = true;
    addCommand(cmdExit);
    repaint();
  }

  public void keyPressed(int keycode) {
    if (!remSplash) {
      exitSplash();
    }
  }
}
```

See figure 4. for the output of the above sample code once launched.

Figure 4: Splash Screen

Click any key or after 4 seconds, output of the above sample code is seen as in figure 5.

Figure 5: Splash Screen over

Sample source code using Low Level API

Sample code given below demonstrates on how to use full Canvas and handle events using low level API.

Class: CanvasMidlet.java

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

public class CanvasMidlet extends MIDlet {
   /*current display object*/
   Display display;
   /*object for displaying canvas screen*/
   MainCanvas canvas;

   public CanvasMidlet(){
      display=Display.getDisplay(this);
      /*initializing the MainCanvas class*/
      canvas=new MainCanvas(this);
   }

   /*called when the midlet is called for first time*/
   public void startApp() {
      display.setCurrent(canvas);
   }

   /*called when any interrupt occurs and can handle the player*/
   public void pauseApp() {
      /*can write code here for handling all interrupts
        and for unhandled garbage collections*/
   }

    /*is used to destroy the Midlet cleanup all that are not
    handled by garbage collection*/
   public void destroyApp(boolean unconditional) {
   }

   /*This method is used to exit the midlet*/
   public void exitMidlet(){
      destroyApp(true);
      notifyDestroyed();
   }
}
```

Class: MainCanvas.java

```
import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Font;
import javax.microedition.lcdui.Graphics;

public class MainCanvas extends Canvas implements CommandListener {

   /*to create midlet object*/
   CanvasMidlet midlet;
   /*int object to get width of the screen*/
   int width;
   /*int object to get width of the screen*/
   int height;
   /*this command is used to exit from the current application*/
   private final Command cmdExit = new Command("Exit", Command.EXIT,
     2);
   /*this command is used to select an event*/
   private final Command cmdSelect = new Command("Select",
    Command.OK, 2);
   /*left command object added at left side*/
   private Command leftCommand = null;
   /*right command object added at right side*/
   private Command rightCommand = null;
   /*font object*/
   private Font font = null;
   /*boolean object*/
   private boolean selected;
   /*object to get keyvalue*/
   private String keyvalue = null;
   /*object to get keyname*/
   private String keyname = null;

   public MainCanvas(CanvasMidlet midlet) {
      this.midlet = midlet;
      /*to use full screen of the device for displaying*/
      setFullScreenMode(true);
      /*gets the width of the screen*/
      width = getWidth();
      /*gets the height of the screen*/
      height = getHeight();
      /*initializes the font to be displayed*/
      font = Font.getDefaultFont();
      addCommand(cmdExit, 1);
```

```
    addCommand(cmdSelect, 0);
}

public void showNotify() {
    /*can write code to resume the application*/
}

public void hideNotify() {
    /*can write code here for handling all interrupts
    and for unhandled garbage collections*/
}

/*to draw Commands on the screen*/
public void drawCommands(Graphics g) {
    g.setColor(0, 0, 0);
    if (leftCommand != null && leftCommand.getLabel().equals("Select")) {
        g.drawString("Select", 0, (height - font.getHeight() - 1), 0);
    }

    if (rightCommand != null && rightCommand.getLabel().equals("Exit"))
    {
        g.drawString("Exit", (width – font.stringWidth("Exit")  – 1), (height –
        font.getHeight()– 1), 0);
     }
}

/*to add commands*/
public void addCommand(Command cmd, int pos) {
    if (pos == 0) {
        leftCommand = cmd;
    } else {
        rightCommand = cmd;
    }
}

public void paint(Graphics g) {
    g.setColor(138, 237, 244);
    g.fillRect(0, 0, width, height);
    /*sets the font property*/
    g.setFont(font);
    /*condition to display font color on clicking Select button*/
    if (selected) {
        g.setColor(255, 255, 255);
    } else {
        g.setColor(0, 0, 255);
    }
    g.drawString("Welcome to ", width / 2,0, Graphics.HCENTER |
      Graphics.TOP);
```

```
    g.drawString("Samsung Mobile Innovator", width /
     2,font.getHeight(),Graphics.HCENTER | Graphics.TOP);
    g.setColor(0,0,0);
    g.drawString("KeyPressed", width / 2, font.getHeight()*2,
     Graphics.HCENTER | Graphics.TOP);

    if (keyvalue != null) {
       g.drawString("keyvalue="+keyvalue, width /
        2,font.getHeight()*3,Graphics.HCENTER | Graphics.TOP);
    }
    if (keyname != null) {
       g.drawString("keyname="+keyname, width /
        2,font.getHeight()*4,Graphics.HCENTER | Graphics.TOP);
    }

    drawCommands(g);
}

public void commandAction(Command cmd, Displayable disp) {
    if (cmd == cmdExit) {
       midlet.exitMidlet();
    } else if (cmd == cmdSelect) {
       if (selected) {
          selected = false;
       } else {
          selected = true;
       }
       repaint();
    }
}

public void keyPressed(int keycode) {
    /*get keycode value to a string variable*/
    keyvalue = Integer.toString(keycode);
    /*get key names*/
    keyname=getKeyName(keycode);

    switch (keycode) {
       case -7:
          /*to handle the event of right softkey*/
          if (rightCommand != null) {
             commandAction(rightCommand, this);
          }
          break;
       case -6:
          /*to handle the event of left softkey*/
          if (leftCommand != null) {
             commandAction(leftCommand, this);
```

```
            }
            break;
        }
        repaint();
    }
}
```

Output of above sample code is seen in figure 6, with full screen and commands drawn using Canvas.

Figure 6: Initial Screen

Select soft Command text "Welcome to Samsung Mobile Innovator" is converted to white color, key value and key name of the soft key is displayed as seen in figure 7.

Figure 7: Key Events display