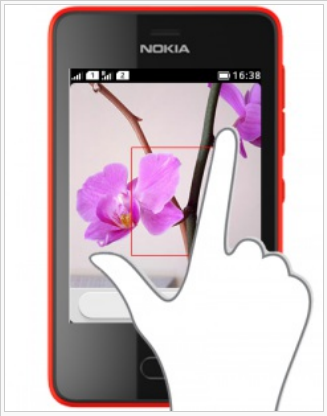
Handling multitouch in Java ME

This article explains how to handle multi-touch interactions in Java ME, using the Nokia UI API

21 Jul 2013

Introduction

The Nokia UI API is a Nokia-proprietary extension to MIDP which offers additional audio and graphics capabilities. One of the useful features it delivers is full support for multitouch interactions, which are available on Series 40 Touch UI and Nokia Asha devices.



Multitouch interactions on a Nokia 501

This article illustrated how these features can be used to handle multiple touches, and how those touches can be displayed on screen or used to perform specific tasks within a Java app.

The MultipointTouch API

The com.nokia.mid.ui.multipointtouch package contains the two objects that allow to easily implement multitouch handling capability in a Java app:

- the MultipointTouchListener interface is responsible for handling multipoint touch events
- the MultipointTouch class allows to add/remove listeners and to retrieve information about the touch events, such as their coordinates and status

The following sections show how multiple touches can be detected, handled and displayed on a LCDUI `Canvas`.

Handling multitouch events

Handling touches via tha *MultipointTouch API* requires the implementation of the `MultipointTouchListener` interface, that declares a single method:

- pointersChanged, that accepts as argument an `int array` containing the IDs of the touch events.

The following implementation of the `pointersChanged` event stores the pointer IDs array into an instance variable, and calls the `Canvas repaint` method to query a refresh of its user interface, so that the updated state of the touch events can be painted on the screen.

```
public class TouchTrackingCanvas extends Canvas implements MultipointTouchListener
{
 int[] currentPointers = null;

 public void pointersChanged(int[] pointerIds)
 {
  currentPointers = pointerIds;

  repaint();
 }
}
```

Registering the MultipointTouchListener

Once the `MultipointTouchListener` interface has been implemented, it must be set as listener via the MultipointTouch.addMultipointTouchListener method, as shown by the following code snippet:

(C) Copyright Nokia 2014. All rights reserved.

```
MultipointTouch multitouch = MultipointTouch.getInstance();

multitouch.addMultipointTouchListener(touchListener);
```

Displaying touch events

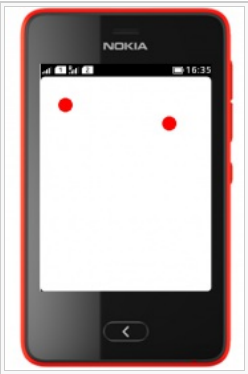
The `MultipointTouch` class defines 3 static method that can be used to retrieve the details for each of the notifies pointer touches:

- getX returns the X coordinate for the given pointer ID
- getY returns the Y coordinate for the given pointer ID
- getState return the touch state for the given pointer ID, where the state can assume one of the following values:
  - MultipointTouch.POINTER_PRESSED for press events
  - MultipointTouch.POINTER_DRAGGED for drag events
  - MultipointTouch.POINTER_RELEASED for release events

By using those methods, the `Canvas` instance defined above can paint the lasted notified touches on the screen as shown in the following code snippet:

```
protected void paint(Graphics g)
{
 g.setColor(0xffffff);
 g.fillRect(0, 0, getWidth(), getHeight());

 int pointSize = 20;

 if(currentPointers != null)
 {
  for(int i = 0; i < currentPointers.length; i++)
  {
   int touchState = MultipointTouch.getState(currentPointers[i]);

   if(touchState == MultipointTouch.POINTER_RELEASED)
    g.setColor(0xcccccc);
   else
    g.setColor(0xff0000);

   int x = MultipointTouch.getX(currentPointers[i]);
   int y = MultipointTouch.getY(currentPointers[i]);

   g.fillArc(x - pointSize / 2, y - pointSize / 2, pointSize, pointSize, 0, 360);
  }
 }
}
```

The above code displays a red dot for each active touch (so, with its state being one of `MultipointTouch.POINTER_PRESSED` and `MultipointTouch.POINTER_DRAGGED`), and one gray dot for an inactive touch (with its state being `MultipointTouch.POINTER_RELEASED`).



Two active touches displayed on a Nokia 501

Cropping an image with multitouch

This section describes how the multitouch features described above can be used to allow the user of a Java app to dynamically define an image area to be cropped, by adjusting the desired area with two fingers. The cropping feature will then be implemented

(C) Copyright Nokia 2014. All rights reserved.

by using the ImageTransformControl from the AMMS API.

The base Canvas

The starting point is a `Canvas` instance, that defines the `Image` to be cropped, and the coordinates of the image mask. The `Canvas` instance also register itself as a `MultipointTouchListener`, in order to receive multitouch events.

```
public class ImageCanvas extends Canvas implements MultipointTouchListener
{
 Image image = null;

 int maskX = 0;
 int maskY = 0;
 int maskWidth = 0;
 int maskHeight = 0;

 public ImageCanvas()
 {
  try
  {
   image = Image.createImage("/photo.jpg");
  }
  catch(Exception e)
  {
  }
  maskWidth = image.getWidth();
  maskHeight = image.getHeight();

  MultipointTouch.getInstance().addMultipointTouchListener(this);
 }
}
```

The `Canvas` `paint` method displays on screen both the `Image` and the mask, displayed as a red rectangle drawn above the image, as shown by the code snippet below.

```
protected void paint(Graphics g)
{
 g.setColor(0xffffff);
 g.fillRect(0, 0, getWidth(), getHeight());

 g.drawImage(image, 0, 0, Graphics.TOP|Graphics.LEFT);

 g.setColor(0xff0000);
 g.drawRect(maskX, maskY, maskWidth - 1, maskHeight - 1);
}
```

Image and mask displayed on screen

Adjusting the image mask

The image mask can be adjusted by the user by dragging two fingers on the screen. The technique shown in the following code snippet looks for the minimum and maximum x/y coordinates of the first two active touches (so, excluding touches whose state is `POINTER_RELEASED`) and, if at least two of those are found, updates the mask coordinates with the new values, calling a `repaint` of

(C) Copyright Nokia 2014. All rights reserved.

the `Canvas` to show the updated mask on the screen.

```
public void pointersChanged(int[] pointerIds)
{
 int activeTouches = 0;

 // initialize maximum and minimum x/y coordinates
 int xMin = image.getWidth(), yMin = image.getHeight(), xMax = 0, yMax = 0;

 for(int i = 0; i < pointerIds.length; i++)
 {
  // check if this is an active touch
  if(MultipointTouch.getState(pointerIds[i]) != MultipointTouch.POINTER_RELEASED)
  {
   // increment number of found active touches
   activeTouches++;

   // retrieve x/y coordinates of the touch event
   int x = MultipointTouch.getX(pointerIds[i]);
   int y = MultipointTouch.getY(pointerIds[i]);

   // update maximum and minimum x/y coordinates
   xMin = Math.min(xMin, x);
   xMax = Math.max(xMax, x);
   yMin = Math.min(yMin, y);
   yMax = Math.max(yMax, y);

   // if 2 active touches were found, stop looking for more
   if(activeTouches == 2)
    break;
  }
 }
 // if 2 touches were found, update the image mask with the new values
 if(activeTouches == 2)
 {
  maskX = xMin;
  maskY = yMin;
  maskWidth = xMax - xMin;
  maskHeight = yMax - yMin;
 }
 repaint();

}
```

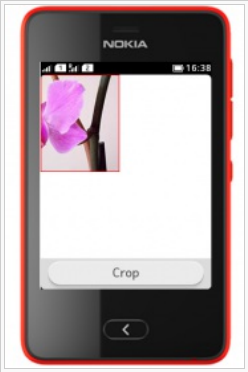
Cropping the image

The new features introduces by the *AMMS API* on the *Asha software platform* allow to perform the cropping of an image with few lines of code. The following code grabs a reference to the `ImageTransformControl` and appropriately sets the source rectangle and target size by using the mask coordinates and size. Done that, the MediaProcessor is initialized with the image input and a `ByteArrayOutputStream`, the output format is defined via the ImageFormatControl class, and the crop process is finally started.

```
MediaProcessor mp = GlobalManager.createMediaProcessor("image/raw");
mp.addMediaProcessorListener(this);

ImageTransformControl transformControl =
(ImageTransformControl)mp.getControl("javax.microedition.amms.control.imageeffect.ImageTransformControl");

transformControl.setSourceRect(maskX, maskY, maskWidth, maskHeight);
transformControl.setTargetSize(maskWidth, maskHeight, 0);
transformControl.setEnabled(true);

outputStream = new ByteArrayOutputStream();
mp.setInput(image);
mp.setOutput(outputStream);
```


(C) Copyright Nokia 2014. All rights reserved.

```
ImageFormatControl fc =
(ImageFormatControl)mp.getControl("javax.microedition.amms.control.ImageFormatControl");
fc.setFormat("image/jpeg");
fc.setParameter("quality", 100);

mp.start();
```



Cropped image shown on screen

Multitouch simulation

The Nokia Asha SDK 1.0 emulator offers support for simulating two-touches interactions, by accessing the *Tools* menu and selecting *Pinch-to-zoom simulation*.

Multitouch simulation on the Nokia Asha SDK 1.0 emulator

Once started, the multitouch tool allows to record two-fingers interactions directly on the simulator screen. Those interactions can then be played back so that multitouch can be properly simulated within the app.

Multitouch simulation tool

Summary

This article illustrates how multitouch interactions can be handled in a Java app by using the features offered by the *Nokia UI API*, and more specifically by its `com.nokia.mid.ui.multipointtouch` package.

A real world case is presented to show how those features can be used in real apps to provide a better user experience is various scenarios, if compared to single-touch interactions.

(C) Copyright Nokia 2014. All rights reserved.

Full source code of the Java app illustrated in this article is available: here: Media:WikiMultitouch.zip

(C) Copyright Nokia 2014. All rights reserved.