Image processing using Java ME

This article explains how to make best use of the image processing capabilities in the new Asha Software Platform. Steps to overcome limitations present are also mentioned.

14 Jul 2013

Introduction

The new Asha Software Platform has a lot of new features to make the life of developers easier. For image processing, two new features are included. They are:

1. Image Scaler API
2. Advanced Multimedia Supplements (AMMS) - JSR-234.

Image Scaler API

Scaling an image is a major problem because of limited heap memory. The Image Scaler API helps in downscaling images that are present in either in the memory card or in the phone storage. It will process the image and then write it back to the location given, asynchronously. It will not return a reference to the scaled image as such. One has to manually open the file where it was written.

Scaling can be done in two ways. You can scale an image to a given width and height or you can scale it up to a given file size.

Note: This API can only downscale an Image. Use the Image Transform control, discussed below to upscale an Image.

The steps to make use of the image scaler API are very simple.

1. Create a new `ImageScaler` Object. Pass the location of source Image and the location where you want to save the scaled Image.

```
  ImageScaler imageScaler = new ImageScaler(sourceImageFilePath,
destinationImageFilePath);
```

2. Implement `ImageScalerListener` in the class. You need to define `scaleFinished` method. It will be called once the scale is done.

```
  imageScaler.addListener(this);
```

3. To start the scaling operation, call the `scaleImage()` method. There are two constructs available. You can set the `maxfilesize` it can have in KB, or you can specify the height and width to scale to. Set the final argument as true if you wish to maintain aspect ratio.

```
  imageScaler.scaleImage(newWidth, newHeight, true)
```

4. Each scaling operation returns an ID, which can be used to uniquely identify the image which has been scaled. The ID returned is 0 in case an error occurs.

```
  int id = imageScaler.scaleImage(newWidth, newHeight, true);
```

5. The `scaleFinished()` method will be called as soon as the scale is completed. Implement the business logic there on what to do with the scaled Image. The request ID can be used to find out which Image's scale operation is complete.

Overcoming Limitations

The Image Scaler API works only on files that are present either in the memory card or in the Phone Memory. It will not work on images that are there in memory (eg: Images taken using Camera, within the app). As a workaround, the AMMS API is used to store an image in memory and an input stream from it is returned.

(C) Copyright Nokia 2014. All rights reserved.

This method can be customized as required. This is also a part of the Java file which is included along with this wiki.

```
public void saveToStorage(Image img, final String path) {
  try {

    MediaProcessor mp = GlobalManager.createMediaProcessor("image/raw");
    mp.setInput(img);

    ImageFormatControl ifc = (ImageFormatControl)
mp.getControl("javax.microedition.amms.control.ImageFormatControl");

    ifc.setFormat("image/jpeg");
    ifc.setParameter("quality",75);

    final ByteArrayOutputStream bos = new ByteArrayOutputStream();
    mp.setOutput(bos);

    mp.addMediaProcessorListener(new MediaProcessorListener() {
        public void mediaProcessorUpdate(MediaProcessor processor, String event, Object
eventData) {

        if (event.equals(MediaProcessorListener.PROCESSING_COMPLETED)) {
                System.out.println("Processing Complete");

                //Writing the image to that path on a seperate thread.
                Thread t = new Thread() {
                    public void run() {
                        OutputStream os = null;
                        FileConnection fc = null;
                        try {
                            fc = (FileConnection) Connector.open(path,
Connector.READ_WRITE);
                            if (!fc.exists()) {
                                fc.create();
                            }
                            os = fc.openDataOutputStream();
                            os.write(bos.toByteArray());

                    } catch (Exception e) {
                        e.printStackTrace();
                        } finally {
                    try {
                        os.close();
                        fc.close();
                    } catch (IOException ex) {
                        ex.printStackTrace();
                    }
                }
            }
        };
        //Starting the thread
        t.start();
        }
        }
    });
    mp.start();
```


(C) Copyright Nokia 2014. All rights reserved.

```
    } catch (Exception e) {
  e.printStackTrace();
    }
```

This method basically takes an image in the memory, encodes it in JPEG format using the Image Format Encoder and saves it in the memory card with a predetermined name. The name and location of this temporary image can be changed as needed.

Now, this file can be sent to the image scaler API for scaling operations. Once the operation is done, the corresponding files can be deleted.

Mediaprocessor, Image Format Control used above, are explained in detail below.

Advanced Multimedia Supplements (AMMS)

A subset of the Advanced Multimedia Supplements API (JSR-234) is implemented in Nokia Asha Platform 1.0. The main functionalities provided are Image Post Processing (Image Effects Control), Image Encoding (Image Format Control), Image Transformation (Transform Control) and Overlaying Images (Overlay Control).

If you are not interested in knowing how it works in detail, skip down to the bottom where I have mentioned how to use the Java files included with this wiki. It will be simple to understand and sufficient enough to get things done.

In order to use these features, we need to setup a Media Processor. It can be created as shown below

```
  MediaProcessor mp = GlobalManager.createMediaProcessor(inputType);
```

The `inputType` parameter can be set to one of a few MIME types. The Various MIME types that are supported can be obtained by using the following method

```
  GlobalManager.getSupportedMediaProcessorInputTypes();
```

This will return a string array with all the various MIME types that are supported In Asha 501. The supported types are:

- "image/raw"
- "image/jpeg"
- "image/png"

Use "image/raw" when you are supplying an Image from the memory as input. Use "image/jpeg" and "image/png" when you are supplying a JPEG or PNG Image from storage.

As an example, I will use an Image taken from the memory. Hence my `MediaProcessor` will be

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

The `MediaProcessor` goes through various stages in its life. For instance,as soon as the media processor is created. It will be in the UNREALIZED State.

Next step is to set the input and output parameters.Once they are set, the Media Processor will move into the REALIZED state.

The Input to a media processor can be any one of the following

1. An LCDUI Image
2. An InputStream

FileInputStream works well. I have noticed problems while using any other type of `InputStream`.

The input is set via

```
 mp.setInput()
```


(C) Copyright Nokia 2014. All rights reserved.

Note: If you are using LWUIT,an LCDUI Image can be easily converted to LWUIT Image by creating a new LWUIT Image and passing the LCDUI Image as parameter. In order to do the reverse, get the RGB data from the LWUIT Image and use it to create an LCDUI Image.

Similarly, an output stream has to be specified. The preferred OutputStream is a ByteArrayOutputStream.

```
mp.setOutput();
```

Only a REALIZED Media Processor can be started.

There are two ways to start the processing. Synchronously & asynchronously:

- Asynchronous - `mp.start();` - This is a non blocking call. Use the `MediaProcessorListener` to track progress. This is the recommended way of performing processing.
- Synchronous - `mp.complete()` - This is a blocking call. It will wait till either an error occurs or the processing is complete.

This will put the Media Processor in the STARTED state. You can pause the processing in the middle and continue again if you wish to. The `MediaProcessor` will be put into an interim state of STOPPED.

You can register a listener for finding out the state of the processing. This listener is called `MediaProcessorListener` and implements a function called `MediaProcessorUpdate`. The `MediaProcessorUpdate` method contains a parameter called event which can be used to find out the present state of the Media Processor.

```
mp.addMediaProcessorListener(new MediaProcessorListener() {
  public void mediaProcessorUpdate(MediaProcessor processor, String event, Object
eventData) {

   if (event.equals(MediaProcessorListener.PROCESSING_ABORTED)) {
    System.out.println("PROCESSING_ABORTED: "+ eventData.toString());
   } else if (event.equals(MediaProcessorListener.PROCESSING_ERROR)) {
    System.out.println("PROCESSING_ERROR: "+ eventData.toString() + event.toString());
   } else if (event.equals(MediaProcessorListener.PROCESSING_STARTED)) {
    System.out.println("PROCESSING_STARTED: "+ eventData.toString());
   } else if (event.equals(MediaProcessorListener.PROCESSING_STOPPED)) {
    System.out.println("PROCESSING_STOPPED: "+eventData.toString());
   } else if (event.equals(MediaProcessorListener.PROCESSOR_REALIZED)) {
    System.out.println("PROCESSOR_REALIZED: "+ eventData.toString());
   }
   else if (event.equals(MediaProcessorListener.PROCESSING_COMPLETED)) {
    System.out.println("PROCESSING_COMPLETED: "+ eventData.toString());
   }
  }
  });
```

We have not yet specified what type of processing has to be done! They are called as controls. As an example, let us look at the various controls that are available for Asha 501.

Image Effects Control

The Image Effects Control creates basic Image Filters.The Asha 501 supports four different types of filters namely Sepia, Monochrome, Negative and Solarize

First get an instance of the `ImageEffectControl` using the following

```
  ImageEffectControl iec = (ImageEffectControl)
mp.getControl("javax.microedition.amms.control.imageeffect.ImageEffectControl");
```

The specific effect to be applied is set via the `setPreset()` method.Supply the effect name as argument.

(C) Copyright Nokia 2014. All rights reserved.

```
  iec.setPreset("sepia");
```

The most important step is to enable the effect.

```
  iec.setEnabled(true);
```

Now, once the media processing is started, the image supplied is converted to Sepia.

Chaining Multiple Effects

If you wish to chain multiple effects together (eg: sepia+solarize), you need to set the input and output of the media processor once again, even if they are the same from last time. This is because a Media Processor enters the UNREALIZED state as soon as processing is over. An UNREALIZED Media Processor cannot be started.

Only when both of them are set again, does the `MediaProcessor` enter REALIZED state. Make sure you clear the `oupututStream` as well if you are using the same.

Image Format Control

The Image Format Control can be used to encode an Image in either JPEG or PNG format. You can set parameters like quality of the image as well.

```
  ImageFormatControl ifc = (ImageFormatControl)
mp.getControl("javax.microedition.amms.control.ImageFormatControl");
```

The Image format and quality can be set as shown below

```
 ifc.setFormat("image/jpeg");
 ifc.setParameter("quality", 75);
```

Like the Image effect control, it must also be enabled in order to function.

Image Transform Control

The Image Transform Control can be used to perform operations like rotation, shrinking, scaling. Rotation can be done only in multiples of +90 or -90 degrees.

This control can be used for Upscaling an image. Together with the Image Scaler API, they form a formidable unit for performing scaling operations in the Asha Software Platform.

First create an instance as always

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

The next step is to set the Source Rectangle. The order in which this rectangle is defined matters. If a positive value is given, that corresponding axis is treated in the forward direction. If a negative value is given,the direction is reversed.

In case any rotation is performed, the size of the resultant image is changed. This has to specified using the `setTargetSize()` method.

eg : Performing a Horizontal Flip

```
  // In the X Axis,we are starting at the end of the image and going backwards a
distance of sourceW. This basically flips the image horizontally.
  imageTransform.setSourceRect(sourceW, 0, -sourceW, sourceH);
```


(C) Copyright Nokia 2014. All rights reserved.

```
  //No change in target Size as we are not rotating.
  imageTransform.setTargetSize(0, 0, 0);
```

Do not forget to enable the effect before starting the `MediaProcessor`!

Overlay Control

The Overlay Control can be used to draw an Image on top of another. The basic usage is similar to the rest.

```
  OverlayControl oc = (OverlayControl)
mp.getControl("javax.microedition.amms.control.imageeffect.OverlayControl");
```

The insertImage function is used to insert a particular Image at a particular order in a given x,y location.

```
  oc.insertImage(img2, x, y, order);
```

Making things simple!

If all this is too overwhelming for you, you can use the Java file I have attached along with this wiki. You can tweak and use it if you want. It is fairly simple to use. It is mainly designed for Images stored in memory.

Methods Available :

1. `performEffect(Image source,String effectName)` - This function takes an Image as input and performs the desired effect and returns the final Image.
2. `transformImage(Image source,int transform)` - This function takes in an Image, applies a particular pre-defined transformation and returns the final Image.
3. `overlayImage(Image source,Image Destination,int x,int y,int order,int transparen_colour)` - This function overlays the destination image on the source image and returns the final image.
4. `saveToStorage(Image Source,String path)` - This function will encode an Image in JPEG format and save it in the particular location and return an inputstream to that file.
5. `downscale(Image source,int newWidth,int newHeight)` - This function will downscale an Image based on new width and height provided. It will also delete the temporary files created. The business logic of what to do with the scaled Image has to be written in the scaleFinished method.
6. `upscale(Image source,int newWidth,int newHeight)`

This function will upscale an image to the new width and height provided and returns the final Image.

Files

File:AmmsEffectsUpdated.zip - The Java file is included in the zip.

This class has been written with the intention of just achieving what it is meant to do. I give no assurance that it is the very optimized. Any suggestions for improving the same are welcome.

Reference

My initial guidance was provided by this article. How to use basic image processing features of AMMS (JSR-234).

The code used for Image Transform Control is based on the code provided in that article.

Summary

The new Asha Software Platform contains many new features that can be utilized for image processing. However, they are not optimized for Images that are in memory (like Image obtained from a camera snapshot). This article describes in details the image processing capabilities introduced and how to make use of them effectively.

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.