How to use basic image processing features of AMMS (JSR-234)

This article explains how to use basic image processing features of AMMS (JSR-234). Running the MIDlet requires the support of `imageencoding` and `imagepostprocessing` capabilities of AMMS API.

09 Mar 2014

Introduction

The Advanced Multimedia Supplements API (JSR-234) is implemented in Nokia Asha Platform 1.0, Series 40 devices from Series 40 5th Edition onwards and Symbian devices from S60 3rd Edition FP 1 onwards. The API consists of six separate optional capabilities: `music`, `audio3d`, `imageencoding`, `imagepostprocessing`, `camera` and `tuner`. Commonly Nokia devices have supported `music` and `audio3d` capabilities and some models have supported partly `camera` part. Asha Software Platform 1.0 devices like Asha 501 support now new capabilities: `imageencoding`, `imagepostprocessing` and `tuner`.

The API version and the supported capabilities can be queried by using the following system properties:

```
System.out.println("microedition.amms.version: " +
System.getProperty("microedition.amms.version")); // the API version
// In Nokia Asha 501: microedition.amms.version: 1.1
System.out.println("supports.mediacapabilities: " +
System.getProperty("supports.mediacapabilities")); // supported capabilities
// In Nokia Asha 501: supports.mediacapabilities: imageencoding imagepostprocessing
tuner
```

This article demonstrates, how to use `imageencoding` and `imagepostprocessing` capabilities. They offer basic image processing capabilities. When using these features, one should keep in mind, that processing images requires significant amount of heap memory and that in lower end devices heap memory amount is usually quite limited.

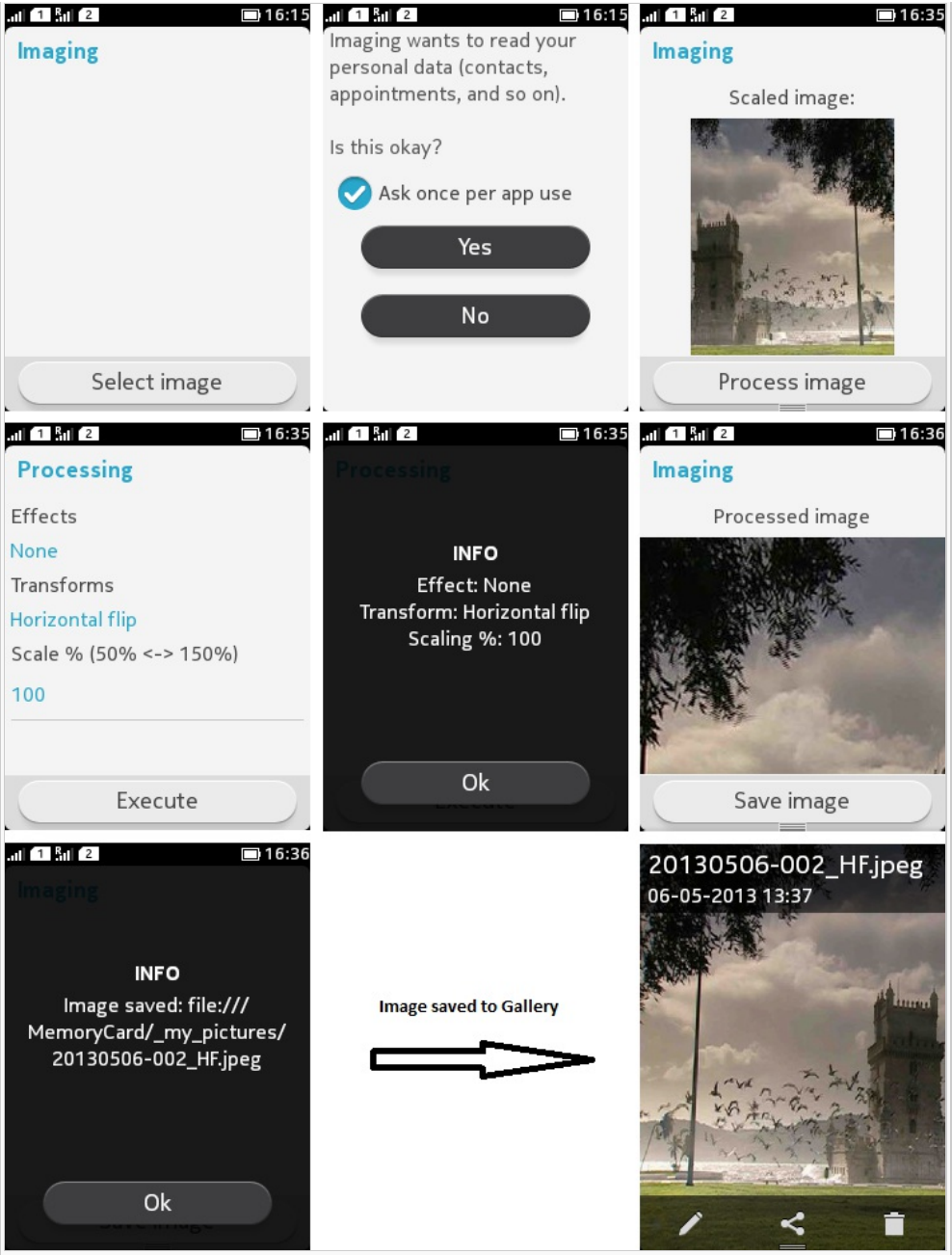
Imaging MIDlet

The Imaging MIDlet consists of three classes:

1. `Imaging.java`, which is the main MIDlet class and where constants for the MIDlet are defined.
2. `ImagingForm.java`, which is the main screen of the MIDlet, showing selected image and it as processed.
3. `ProcessForm.java`, which is used for selected the effects and transforms for the image

The image below shows the UI flow of the Imaging MIDlet.

(C) Copyright Nokia 2014. All rights reserved.



UI flow of Imaging MIDlet

First you need to select an image to be processed from the Gallery. It is not recommended to select too large images, because of the limited amount of heap memory. Images of VGA size (640x480 pixels) should be ok. The file selection is done by using the new File Select API:

```
FileSelectDetail [] selectedImage = FileSelect.launch(Imaging.IMAGEFOLDER,
FileSelect.MEDIA_TYPE_PICTURE, false);

// If a file was selected and returned
if (selectedImage != null) {
 imageName = selectedImage[0].displayName;
 imageUrl = selectedImage[0].url;
}
```

As result we get image name and image url (path). Then smaller image is scaled and shown on the Form. This is done by using the new ImageScaler API:

```
imageScaler = new ImageScaler(original, Imaging.IMAGEFOLDER + Imaging.SCALED_IMAGE);
imageScaler.addListener(this);
try {
 imageScaler.scaleImage(Imaging.SCALED_WIDTH, -1, true);
} catch (ImageScalerException ise) {
 System.out.println("ImageScalerException: " + ise.getMessage());
```


(C) Copyright Nokia 2014. All rights reserved.

```
}
```

The `ImageScaler` class usage is simple, just give original image file and destination image file paths as parameters. Then `ImageScalerListener` is used for tracking the progress of the scaling. Finally the scaled image is shown on the Form and we are ready for selecting the actual effects, transforms and scalings. The selected imaging tasks are done in `processImage(String path, int effect, int transform, int scale)` method:

```
/**
 * Reads image data from the given URL and process image based on the selected
 * effects, transformations and scaling.
 * @param path The path to the image
 * @param effect Selected effect
 * @param transform Selected transform
 * @param scale Selected scaling value
 */
public void processImage(String path, int effect, int transform, int scale) {
 this.transform = transform;
 this.scale = scale;
 this.effect = effect;
 InputStream is = null;
 try {
  MediaProcessor mp = GlobalManager.createMediaProcessor("image/jpeg");
  mp.addMediaProcessorListener(this);
  FileConnection fc = (FileConnection)Connector.open(path, Connector.READ);
  if(!fc.exists()) {
   append(path + " doesn't exist!");
  }
  else if (fc != null) {
   is = fc.openInputStream();
   mp.setInput(is, MediaProcessor.UNKNOWN);
  }
  baos = new ByteArrayOutputStream(); // create a OutputStream that will receive the
resulting image
  mp.setOutput(baos);
  ImageEffectControl imageEffect =

(ImageEffectControl)mp.getControl("javax.microedition.amms.control.imageeffect.ImageEffectControl");
  // two presets are supported: 'monochrome' and 'negative'
  if (effect == 1) {
   imageEffect.setPreset("monochrome");
  }
  if (effect == 2) {
   imageEffect.setPreset("negative");
  }
  if (effect != 0) imageEffect.setEnabled(true);
  ImageTransformControl imageTransform =

(ImageTransformControl)mp.getControl("javax.microedition.amms.control.imageeffect.ImageTransformControl");
  int sourceW = imageTransform.getSourceWidth();
  int sourceH = imageTransform.getSourceHeight();

  if (transform == 1) { // Horizontal flip
   imageTransform.setSourceRect(sourceW, 0, -sourceW, sourceH);
   imageTransform.setTargetSize(0, 0, 0);
  }

  if (transform == 2) { // Vertical flip
   imageTransform.setSourceRect(0, sourceH, sourceW, -sourceH);
   imageTransform.setTargetSize(0, 0, 0);
  }

  if (transform == 3) { // Rotate 90 degrees
   imageTransform.setSourceRect(0, 0, sourceW, sourceH);
   imageTransform.setTargetSize(sourceW, sourceH, 90);
  }
```


(C) Copyright Nokia 2014. All rights reserved.

```
  if (transform == 4) { // Rotate -90 degrees
   imageTransform.setSourceRect(0, 0, sourceW, sourceH);
   imageTransform.setTargetSize(sourceW, sourceH, -90);
  }

  if (scale != 100) {  // Scaling
   imageTransform.setSourceRect( 0, 0, sourceW, sourceH);
   double factor = (double)scale / (double)100;
   int targetW = (int)(factor * (double)sourceW);
   int targetH = (int)(factor * (double)sourceH);
   imageTransform.setTargetSize(targetW, targetH, 0);
  }

  if (transform != 0) {
   imageTransform.setEnabled(true);
  }

  mp.start();         // Use with MediaProcessorListener
  fc.close();
  is.close();
 }
 catch (IOException ioe) {
 }
 catch (IllegalArgumentException iae) {
 }
 catch (IllegalStateException ise) {
 }
 catch (MediaException me) {
 }
 catch (ArrayIndexOutOfBoundsException aioobe) {
 }
 catch (SecurityException se) {
  // If "No" is selected in the security prompt
 }
}

/**
 * This method is called to deliver an event to a registered listener when a
MediaProcessor event is observed.
 * @param processor the MediaProcessor which generated the event
 * @param event the generated event
 * @param eventData the associated event data
 */
public void mediaProcessorUpdate(MediaProcessor processor, String event, Object
eventData) {
 // Called, when MediaProcessor.start() and MediaProcessorListener is used
 if (event.equals(MediaProcessorListener.PROCESSING_ERROR)) {
  System.out.println("PROCESSING_ERROR: " + eventData.toString());
  this.append("Processing error!");
 }
 if (event.equals(MediaProcessorListener.PROCESSING_COMPLETED)) {
  System.out.println("PROCESSING_COMPLETED: " + eventData.toString());
  imageData = baos.toByteArray();
  Image result = Image.createImage(imageData, 0, baos.size());
  if (result != null) {
   imageItem.setImage(result);
   imageItem.setLabel("Processed image");
   removeCommand(processCommand);
   addCommand(saveCommand);
  }
  else imageItem.setAltText("Image processing failed!");
 }
}
```

The effects are done by using `ImageEffectControl` and the transforms by using `ImageTransformControl`. The image is read from

(C) Copyright Nokia 2014. All rights reserved.

Gallery by using FileConnection API.

Processing an image might take some time. AMMS API has a convenient way of tracking the process, `MediaProcessor` and `MediaProcessorListener`. When the listener observes an `MediaProcessor` event, `mediaProcessorUpdate(MediaProcessor processor, String event, Object eventData)` method is called. When `MediaProcessorListener.PROCESSING_COMPLETED` event occurs, the processing is done. Note, that selecting both transform and scaling doesn't work, because both are done by using `ImageTransformControl` and only one tranformation can be done at a time.

When the processed image is shown on the screen, it is possible to save it to Gallery. The file name is extended with supplements, based on the selected effects and transformations.

Example application

Imaging.zip containing sources, **Imaging.jad** and **Imaging.jar**

(C) Copyright Nokia 2014. All rights reserved.