Facial Processing – User Guidelines

This guide is intended to help developers understand how to write an application from scratch using Snapdragon™ SDK Facial Processing.

This is a simple Android camera application that will start a camera preview and do two things:

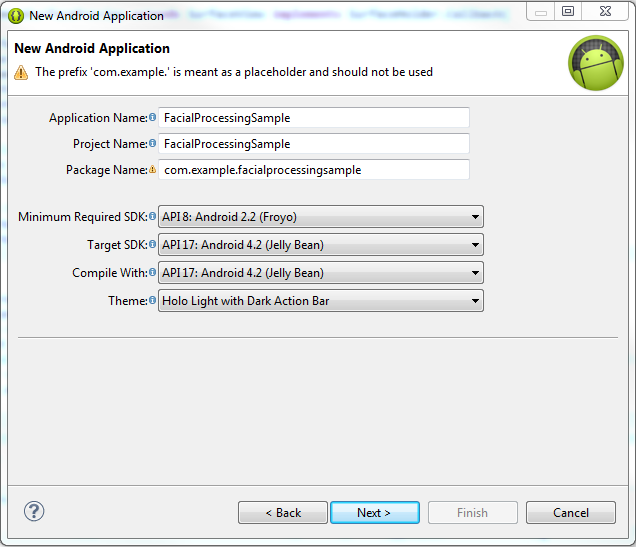
1. Track eyes, mouth and face movement by drawing points and a rectangle on the camera preview.

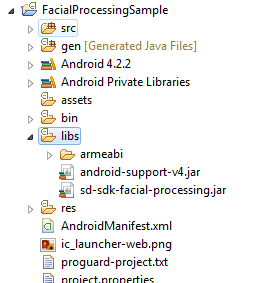
2. Change the color of the face rectangle based on the changing smile value of the person being photographed.

For more information about Facial Processing, [click here](https://developer.qualcomm.com/mobile-development/mobile-technologies/snapdragon-sdk-android/features/facial-processing). You can download the Snapdragon SDK bundle (which includes facial processing) [here](https://developer.qualcomm.com/mobile-development/mobile-technologies/snapdragon-sdk-android/tools-resources).

We will use Eclipse as our development platform. A few steps might differ based on your toolset.

**Setup:**

* Create an android project by selecting **File** -> **New** -> **Other** -> **Android Application Project**. Give an **Application Name** and a **Project Name** (e.g., FacialProcessingSample). **Finish** the rest of the steps with your desired settings.   
    
   
* After the wizard ends, you should see a new folder named “libs” in the project directory, as shown in the image below; if not, then create one.
* The downloaded Snapdragon SDK bundle should include the following files and directories:
  + Docs
  + Libs
  + Samples
  + Testapp
  + SDKLicense (pdf)
  + SnapdragonSDK\_ReleaseNotes\_2\_0
* From the SDK bundle, copy the .jar file named “sd-sdk-facial-processing” and the folder named “armeabi” into the newly created “libs” folder.



* Right-click on the project -> Properties -> Java Build Path -> “Go to Libraries tab” -> Add JARs -> “Add the sd-sdk-facial-processing.jar from the libs folder in the project.
* Go to ‘Order and Exports’ and check the .jar file that you just added and also the Android private libraries.
* Clean the project before proceeding.

**Flowchart:**

* Here is a flowchart of all the steps to create a Facial Processing application:

Decide on a GUI for your application

Create a surface for camera display.

Open device camera and display it on surface

Display facial data on preview frame

Use Snapdragon SDK Facial Processing

Impress your friends and colleagues

**Decide on a GUI:**

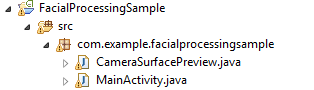
* You can design and decorate your application interface the way you want and like. We will keep it simple for the purpose of this application.
* Add a simple button to your layout:  
    
  activity\_main.xml  
  

**Create a camera surface for display:**

* In order to display the camera preview, make a Surface View to display the camera preview.
* Select the res/layout folder to view all the layout files. Open the .xml corresponding to the “MainActivity” and add the following FrameLayout in it:

activity\_main.xml



* Create a class named “CameraSurfacePreview” in your project. This class will take care of setting the camera preview display. It will be called whenever a new surface is created. It will have a constructor that takes in the context and the camera object. The three overridden methods ‘surfaceChanged’, ‘surfaceCreated’ and ‘surfaceDestroyed’ allow you to provide your implementation when the camera surface is changed, created or destroyed:  
     
   CameraSurfacePreview.java 

**Open device camera for display:**

* Once you have created the ‘CameraSurfacePreview’ class, go back to the ‘MainActivity’. Declare and initialize the objects required for starting a camera preview. Declare the following objects and variables:

MainActivity.java



* **startCamera( );**
  + This method will open the camera and display it on the available surface:

MainActivity.java

****

* + Call startCamera( ) as soon as the activity gets started. You will also notice that ‘cameraObj.setPreviewCallback(..)’ will show an error. In order to solve this, we want our class to “implement Camera.PreviewCallback” and add the unimplemented methods:  
      
    MainActivity.java  
    

* Before running the application on a device it is important to add the necessary permission in manifest file:

AndroidManifest.xml

****

Run the application on the device. You should be able to see a camera preview application. You will notice that the camera preview might be rotated by some degrees. This is because the camera sensor is mounted differently on different devices. We will take care of rotating the image at a later stage.

**Things to take care of before proceeding:**

* Before we start using the Facial Processing framework, you must:
  + Write a method stopCamera( ), which will release the camera after preview so that other applications can use it:

MainActivity.java



* + Implement the onPause(), onResume() and onDestroy functions for the activity:  
    MainActivity.java



* + Implement an action listener for the switchCamera button that we created:

MainActivity.java  


**Use Facial Processing:**

* Snapdragon SDK Facial Processing makes it easy to obtain facial data and use it to create unique experiences.
* The most important thing is to first verify that the Snapdragon SDK Facial Processing feature is supported on the device.
* We verify that before we start the camera:

MainActivity.java****

* Once the camera stops or activity closes, it is important that we release the facial processing instance we created. We will do that in the stopCamera( ) method:  
    
  MainActivity.java  
  
* Now we’re ready to call the FacialProcessing.setFrame(..) method and obtain facial data. This can be done with the onPreviewFrame(byte[] data, Camera camera) method that we implemented previously.
* As noted above, the camera preview frame seems to be rotated by a certain amount (270 degrees in our case). This is a function of how the camera sensor is mounted. Similarly, if we try to obtain the facial data from the Facial Processing SDK, then it will be aligned with respect to the camera sensor. Therefore, we must rotate the camera preview as well as the facial data. The best way to do that is to check the screen orientation of the device and feed in the rotation values accordingly:  
    
  MainActivity.java  
  
* The rotation issue should now be fixed.
* Call the facial processing setFrame(..) method, which has the following parameters:
  + **byte[]** frameData : YUV frame data that we will get from onPreviewFrame(data, camera)
  + **int** frameWidth: This is the camera previewSize width.
  + **int** frameHeight: This is the camera previewSize height.
  + **boolean** isMirrored: True if camera is front-facing, and false if it is rear-facing.
  + **PREVIEW\_ROTATION\_ANGLE** rotationAngle: This is the rotation angle that we determined in the last step, depending on the screen orientation.

MainActivity.java



* See whether the number of faces in the preview is greater than zero. If so, then get their faceData by calling the method faceProc.getFaceData(), which will return a faceArray with all the stored facial data for each detected face:  
    
    
    
    
  MainActivity.java  
  

**Display the Facial Data:**

* Now we can take the facial data values and display them on the preview frame. For this we will create a new class named ‘DrawView’.
* The constructor of DrawView will have the following parameters -
  + **Context** context:
  + **FaceData[]** faceArray: Face array that we got from ‘getFaceData()’
  + **boolean** inFrame: See whether there is a face in the frame.
* An overridden method onDraw will be created for drawing on your surfaceView:

DrawView.java  


* Now we manipulate the onDraw call to draw what we need:  
    
    
  DrawView.java  
  
* setRectColor(), is a function that will set the color of the rectangle based on the changing smile value:

DrawView.java



* Now that the DrawView class is ready, it’s time to use it in the onPreviewFrame and see the results:  
    
  MainActivity.java  
  
* Run the app on your device. If all the preceding steps were followed correctly, then you should see a rectangle with a red dot in the left eye, a green dot in the right eye, and a white dot in the mouth.
* Note that these dots may not match up exactly with the eyes and mouth in the preview. This is because the Facial Processing SDK gives the eye and mouth coordinates with respect to the camera preview size, and the surface display size may differ from the camera preview size. Hence, the last important step is to normalize the coordinates with the actual surface display size.
* For this, use the ‘normalizeCoordinates(surfaceWidth, surfaceHeight)’ API of the Facial Processing framework:  
    
  MainActivity.java  
  

The next time you run your application, facial data should display as expected. Changing smile values will change the color of the rectangle.

Useful reference links:

* For more information on **Snapdragon SDK** visit our [**website**](https://developer.qualcomm.com/mobile-development/mobile-technologies/snapdragon-sdk-android/device-compatibility).
* Different **features** provided by the Snapdragon SDK can be found [**here**](https://developer.qualcomm.com/mobile-development/mobile-technologies/snapdragon-sdk-android/features).
* **API references** for our Snapdragon SDK can be found [**here**](https://developer.qualcomm.com/mobile-development/mobile-technologies/snapdragon-sdk/reference-api/overview-summary.html).
* To **discuss questions** relating to Snapdragon SDK visit our [**forum**](https://developer.qualcomm.com/forums/qdevnet-forums/snapdragon-sdk) page.
* Check out the <**FAQ’s for Facial Processing**> (Link still to be decided)