Facial Recognition – User Guidelines

This guide is intended to help developers understand how to write an Android application using Snapdragon™ SDK -Facial Recognition.

Before starting to write a Facial Recognition application it is recommended that you familiarize yourself with writing an application using the Snapdragon™ SDK -Facial Processing framework<@link to FP>, since Facial Recognition is an extension to the existing Facial Processing SDK with additional features.

Useful references for this guide:

* Facial Recognition FAQs<@link>
* Facial Processing User Guide<@link>
* Facial Processing FAQs<@link>

This guide will step through creating a simple Android Application that will start a camera preview and do the following

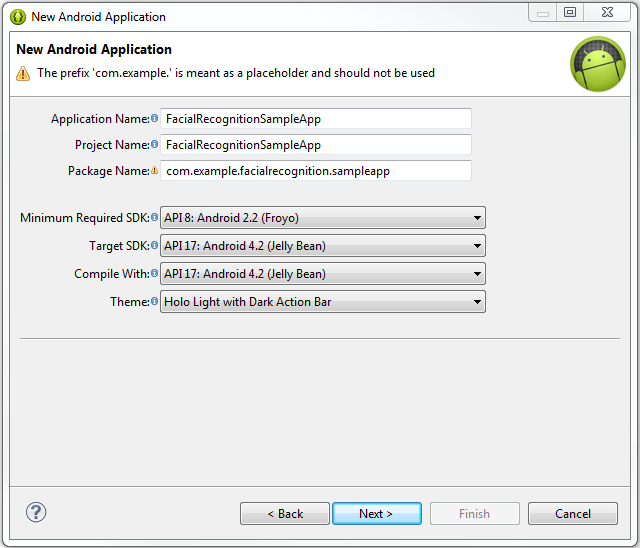
* Add a person
* Update a person
* Identify a person
* Delete a person
* Serialize the album
* De-serialize the album
* Reset an album

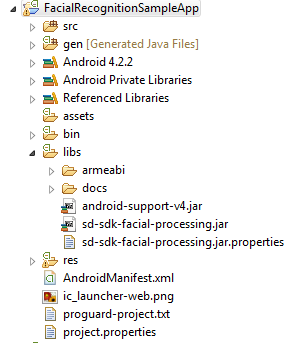
We will use Eclipse as our development platform. A few steps might be different depending on your toolset.

**Confirm your Environment:**

* Make sure you have a Snapdragon device which supports Facial Recognition feature
  + In order to know if your device is compatible you will have to run the Test App which comes with the SDK bundle.
* Make sure you have properly installed your Android Development Tools.

**Setup:**

* Create an android project by selecting **File** -> **New** -> **Other** -> **Android Application Project**. Give an **Application Name** and a **Project Name** (e.g., FacialRecognitionSampleApp). **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:
  1. Docs
  2. Libs
  3. Samples
  4. Testapp
  5. SDKLicense (pdf)
  6. SnapdragonSDK\_ReleaseNotes\_2\_1
* From the SDK bundle, copy all the contents from “libs (Facial\_Recognition)” 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.

**Setting up Facial Recognition:**

* Once the Activity is created, you will have to
  1. Check if the Facial Recognition feature is supported on the device.
  2. Create an instance of the Facial Recognition object

Sample code:

|  |
| --- |
| **public** **class** MainActivity **extends** Activity {  **public** **static** FacialProcessing *faceObj*;  @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_main*);  **boolean** isSupported = FacialProcessing.*isFeatureSupported*(FEATURE\_LIST.*FEATURE\_*  *FACIAL\_RECOGNITION*); // Check if Facial Recognition feature is supported on the device  **if** (isSupported)  {  Log.*d*(TAG, "Feature Facial Recognition is supported");  *faceObj* = (FacialProcessing) FacialProcessing.*getInstance*();  **if**(*faceObj*!=**null**)  {  *faceObj*.setRecognitionConfidence(58);  *faceObj*.setProcessingMode(FP\_MODES.*FP\_MODE\_STILL*);  }  }    }  } |

* If you are successfully able to get the instance of the Facial Processing object then it is very important to do the following steps
  1. Set the recognition confidence
     + Recognition confidence is the confidence value with which the Facial Recognition engine identifies a given face.
     + The setRecognitionConfidence method tells the face engine to ignore any face data matched with a confidence value below the set threshold.
  2. Set the processing mode
     + Use FP\_MODE\_STILL if you are using static images.
     + User FP\_MODE\_VIDEO if you are going to use live camera preview and will constantly process frames.

**Implementing an image source for processing:**

* There are two ways by which you can get an image
  1. Take a picture from the camera
  2. Choose a picture from the device photo Gallery’
* For ways to implement the Android Camera please refer to the Facial Processing User Guide <@link> until “Use Facial Processing” section.
* In order to take a picture, you will simply need to implement a button which will take a picture when clicked.
* Once the android camera is setup as per the above guideline, you can implement the following code in your activity

Sample code for taking a picture:

|  |
| --- |
| **public** **class** MainActivity **extends** Activity **implements** Camera.PreviewCallback{  // Variable declarations  @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_main*);  // Setup camera and do the necessary initializations  Button takePicture = (Button) findViewById(R.id.take\_picture);    takePicture.setOnClickListener(**new** View.OnClickListener() {    @Override  **public** **void** onClick(View v) {  shutterButtonClicked = **true**;    }  });  }  ShutterCallback shutterCallback = **new** ShutterCallback() {  **public** **void** onShutter() {  Log.*d*("TAG", "onShutter'd");  }  };    PictureCallback rawCallback = **new** PictureCallback() {  **public** **void** onPictureTaken(**byte**[] data, Camera camera) {  Log.*d*("TAG", "onPictureTaken - raw");  }  };  PictureCallback jpegCallback = **new** PictureCallback() {  **public** **void** onPictureTaken(**byte**[] data, Camera camera) {  usePicture(data); // Use this image data for implementing the Facial Recognition features.  }  };  /\*  \* Method to use the picture for post processing by converting byte [] into Bitmap  \*/  **public** **void** usePicture(**byte** [] data){  Bitmap storedBitmap = BitmapFactory.*decodeByteArray*(data, 0, data.length, **null**);  }  @Override  **public** **void** onPreviewFrame(**byte**[] data, Camera camera) {  **if**(shutterButtonClicked)  {  camera.takePicture(shutterCallback, rawCallback, jpegCallback);  shutterButtonClicked = **false**;  }  } |

* Once the picture is taken, you can use that image data for executing various Facial Recognition features on it.
* You can refer to the Facial Recognition sample app to see how the image data is handled.

**Facial Recognition Features:**

* So far we have covered how to
  1. Initialize the Facial Recognition object
  2. Setup the camera to take pictures
* Now you can implement the various Facial Recognition methods in order to perform facial recognition on the image data. The following sections include code snippets and descriptions to help you understand how to perform facial recognition.

**Add Person**

* We use this API to “add a new person” to the Facial Recognition album
* DO NOT use this API to add the same person. If you try to add the same person then addPerson will return error code (check FacialProcessingConstants <@link>).
* This API takes in a faceIndex of the person to be added.
  1. When the application calls getFaceData, the face engine will assign an index value to each face detected in the processed frame, an index value, which corresponds to the index value in the face data array returned by the getFaceData method.

|  |
| --- |
| 0 |

* 1. Example1:   
       
     Image1 =  FaceData [] faceArray =

The above image has only one face in it. Hence getFaceData on this image will return a FaceData array of size 1. The array index (0) is assigned to that face.

* 1. Example2:

|  |  |
| --- | --- |
| 0 | 1 |

Image2 =  FaceData [] faceArray =

The above image has 2 faces in it. Hence getFaceData on this image will return a FaceData array of size 2. The array index (0) corresponds to **Face1** and array index (1) corresponds to **Face2**

* If the face was added successfully then addPerson will return a unique PersonId for that face.
* **NOTE: The face engine processes faces in the order of how close they are to the camera sensor. This ordering can change from frame to frame as the distance from the face to the sensor changes. Your application will need to employ an external method to determine which face in the array you want to act upon if you do not wish to act on all of them.**
* **VARIOUS WAYS TO GET USER INPUT: (This completely depends on your application design)**
  1. Draw a face rect and a corresponding index (text) around each face and let the user input the face index that they want to act on.
  2. Take touch input coordinates from the screen and check if it lies within the face rect
  3. Add all the faces in the frame.

Sample code to add **Face2** to the recognition album from Example2 above:

|  |
| --- |
| **public** **void** addPerson(){  **if**(faceObj!=**null**)  {  **boolean** result = faceObj.setBitmap(storedBitmap);  **if**(result)  {  FaceData [] faceArray = faceObj.getFaceData();  **if**(faceArray!=**null**) // If the faceArray is NULL that means no faces were  detected in the frame.  {  **int** personId = faceObj.addPerson(1); // Since we want to add Face2,  the corresponding face  index for it is "1".  /\* Store this personId and its corresponding name somewhere for  future reference \*/  }  **else**  {  Log.*e*(TAG, "No Face Detected");  }  }  **else**  {  Log.*e*(TAG, "setBitmap failed");  }  }  **else**  {  Log.*e*(TAG, "Facial Recognition object is NULL");  }  } |

**Update Person:**

* We use this API to ‘update an existing person’ in the Facial Recognition Album
* This API takes in two paramters
  1. PersonId: This is the unique id returned by a successful addPerson method, executed previously when you added that face.
* FaceIndex**:** Same concept as explained above in ‘addPerson’

Example:

* 1. We will update **Face2** that was previously added in the Example2 above, with a different face of the female.

|  |
| --- |
| 0 |

New Image =  FaceData [] faceArray =

* If the person was updated successfully then updatePerson API will return 0 (SUCCESS).
* **Note: maximum of ‘10’ faces can updated per person. Anything above that will result in an internal error.**

Sample code demostrating updating an existing person(**Face2)** in the Facial Recognition Albumwith new face:

|  |
| --- |
| **public** **void** updatePerson(){  **if**(faceObj!=**null**)  {  **boolean** result = faceObj.setBitmap(newBitmap);  **if**(result)  {  FaceData [] faceArray = faceObj.getFaceData();  **if**(faceArray!=**null**) // If the faceArray is NULL that means no faces were  detected in the frame.  {  **int** newResult = faceObj.updatePerson(personId, 0); // personId is the  same number that was returned when adding  the person originally. FaceIndex is the  corresponding index value of that face.  }  **else**  {  Log.*e*(TAG, "No Face Detected");  }  }  **else**  {  Log.*e*(TAG, "setBitmap failed");  }  }  **else**  {  Log.*e*(TAG, "Facial Recognition object is NULL");  }  } |

**Identify Person:**

* We use this API to identify a person that may or may not have been added to the facial recognition album.
* In order to identify a person you will -
  1. Either have to specify FACE\_IDENTIFICATION Enum in getFaceData.
  2. Or do a getFaceData call with no parameters which will cause the face engine to process the image for all facial data.
* Example:   
  In this example we will assume that **Face2** was previously added and **Face1** has not been added to the Facial Recognition album.

|  |  |
| --- | --- |
| 0 | 1 |

Image =  FaceData [] faceArray =

**Face1:**

**faceArray[0].getPersonId()** returns **-111  
faceArray[0].getRecognitionConfidence** returns **-111**  
  
Because the confidence value with which the person was identified was less than the initially set confidence value.

**Face2:**

**faceArray[1].getPersonId()** returns **“Same personId that was returned previously in AddPerson. Number >= 0”  
faceArray[1].getRecognitionConfidence** returns **“Confidence with which this face was identified as someone”**  
  
Because the confidence value with which the person was identified was greater than or equal to the initially set confidence value.

Sample code for identifying faces in a given image:

|  |
| --- |
| **public** **void** identifyPerson(){  **if**(faceObj!=**null**)  {  **boolean** result = faceObj.setBitmap(storedBitmap);  **if**(result)  {  FaceData [] faceArray = faceObj.getFaceData(EnumSet.*of*(FP\_  DATA.*FACE\_IDENTIFICATION*));  **if**(faceArray!=**null**) // If the faceArray is NULL that means no faces were  detected in the frame.  {  **int** personId\_face1 = faceArray[0].getPersonId();  **int** recog\_confidence\_face1 = faceArray[0].getRecognitionConfidence();    **int** personId\_face2 = faceArray[1].getPersonId();  **int** recog\_confidence\_face2 = faceArray[1].getRecognitionConfidence();    // Use these values in your android application where ever necessary.  }  **else**  {  Log.*e*(TAG, "No Face Detected");  }  }  **else**  {  Log.*e*(TAG, "setBitmap failed");  }  }  **else**  {  Log.*e*(TAG, "Facial Recognition object is NULL");  }  } |

**Delete Person:**

* Use this method to delete a specific person from the facial recognition album.
* Deleting a person will clear all the previously stored image data for that person.
* This API takes a valid PersonId of the person to be deleted.

Sample Code to delete previously added **Face2**:

|  |
| --- |
| **public** **void** deletePerson(){  **if**(faceObj!=**null**)  {  **boolean** result = faceObj.deletePerson(0);  **if**(result)  {  Log.e(TAG, "Delete Successful");  }  **else**  {  Log.e(TAG, "Delete Failed");  }  }  } |

* If the delete person operation was successful then it will return TRUE or else FALSE.

**ResetAlbum**:

* Use this method to delete and clear an entire recognition album.
* **Note: This is an irreversible process.**

Sample Code to Reset entire album:

|  |
| --- |
| **public** **void** resetAlbum(){  **if**(faceObj!=**null**)  {  **boolean** result = faceObj.resetAlbum();  **if**(result)  {  Log.e(TAG, "Reset Album Successful");  }  **else**  {  Log.e(TAG, "Reset Album Failed");  }  }  } |

* If Resetting the album was successful then it will return TRUE else FALSE.

**Serialize and De-serialize an Album:**

* While an instance of the Facial Processing object is in scope, the Facial Recognition album will be stored in a temporary memory buffer by the face engine. To re-use the facial recognition album it is necessary to serialize the album and save it to a permanent memory store.
* Once the Facial Processing object is back in scope, you can load an existing (saved) album into memory by calling the deserializeRecognitionAlbum API.
* The serializeRecognitionAlbum API returns a byte array that can be stored to a permanent memory store.
* There are many ways to store this byte array. One way is to use Android Shared Preferences<@link to Android Shared Preferences>
* **Tip: It is always a good idea to serialize and store the album whenever there are any changes made to the album. This will help prevent user data loss should the application be removed from the foreground unexpectedly.**

Sample code to serialize an album and store it using SharedPreferences:

|  |
| --- |
| **private** **void** saveAlbum(){  **byte** [] albumBuffer = *faceObj*.serializeRecogntionAlbum();  SharedPreferences settings = getSharedPreferences(“*ALBUM\_NAME”*, 0);  SharedPreferences.Editor editor = settings.edit();  editor.putString("myByteArray", Arrays.*toString*(albumBuffer));  editor.commit();  } |

* Once the application returns back, we can de-serialize the album for re-use.

Sample code to de-serialize an album for re-use:

|  |
| --- |
| **private** **void** loadAlbum(){  SharedPreferences settings = getSharedPreferences(“*ALBUM\_NAME”*, 0);  String stringArray = settings.getString("myByteArray", **null**);    **byte**[] array=**null**;  **if** (stringArray != **null**)  {  String[] split = stringArray.substring(1, stringArray.length()-1).split(", ");  array = **new** **byte**[split.length];  **for** (**int** i = 0; i < split.length; i++) {  array[i] = Byte.*parseByte*(split[i]);  }  *faceObj*.deserializeRecognitionAlbum(array);  Log.*e*("TAG", "De-Serialized my album");  }  } |