SMAATSDK

POST-PROCESSING MODULE ON ANDROID

REQUIREMENTS AND DOCUMENTATION

RELEASE v1.0

Serimag

Table of contents

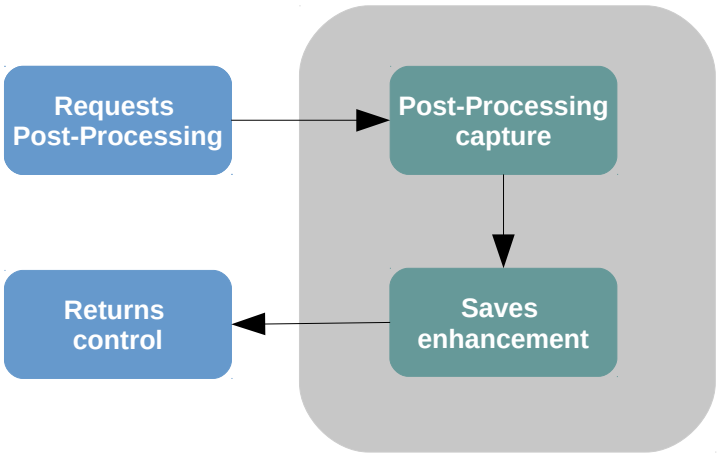
Scope

This document contains a detailed explanation of the functionalities and requirements of the Post-Processing Module. First, a general operating diagram is shown and the files required for the post-processing are explained, as well as all the steps that must be taken for their filing. Then, all the functions provided and their parameters are explained. Finally, a usage example is given.

Purpose

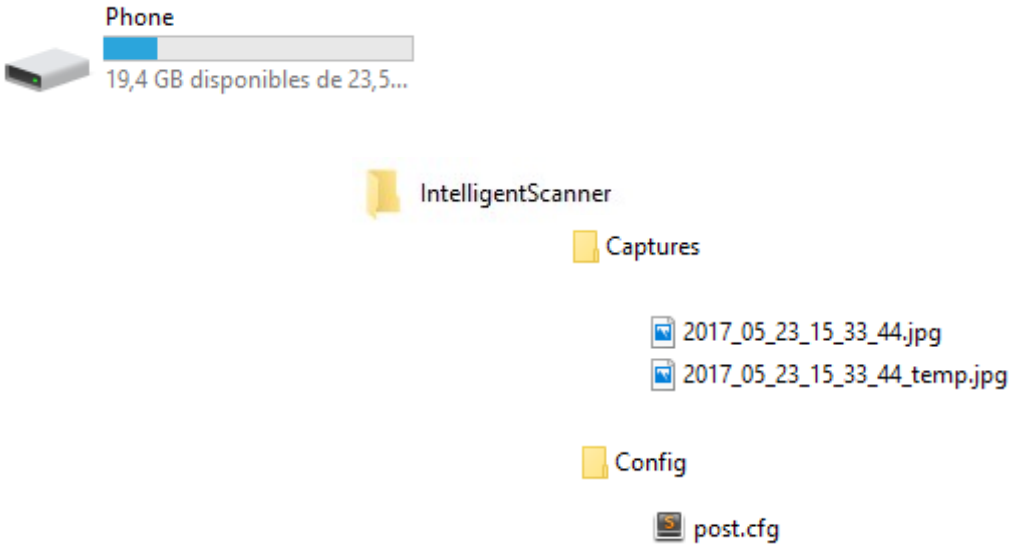
The purpose of this document is to explain the operation and requirements of the Post-Processing Module in detail, and to facilitate its integration into new or already existing projects.

General operating diagram



Example of folders and files necessary for document capture

The IntelligentScanner folder is created in the memory card (SD) storage, which contains the Captures folder where both the images for post-processing and the post-processed images will be saved. This module can post-process any image, but if you want to post-process an image generated by the engine, you must first enable the generatePostProcessImage() function in the IntelligentScanner Module, which will generate an image in the Captures folder. Once the image is post-processed, the input image is overwritten. A Config folder must also be created where the *post.cfg* file will be placed.



Functions provided and parameters to use them

Necessary functions These functions are defined in the PostProcessInterface.java file	
PostProcessInterface postProcessInterface = PostProcessInterface.getInstance()	Assigns the instance of the post-processing engine. The instance of this object follows the Singleton pattern.
postProcessInterface.setConfigPath(String configPath)	Sets the path of the Config folder, where the "post.cfg" configuration file is located.
postProcessInterface.setInputPath(String inputPath)	Sets the path of the folder that contains the image to be post-processed.
postProcessInterface.setOutputPath(String outputPath)	Sets the path of the folder where the post-processed image will be saved.

*Ver funciones opcionales	
String postProcessInterface.run(String documentType, String imageName)	Start the post-processing of the image by specifying the type of document to be post-processed. Exceptions: • ArgumentException • ConfigFileNotFound • ConfigFileMalformatted • Exception

***Optional functions** These functions can be used to change the capture properties.	
void postProcessInterface.setImageQuality(int width, int height, int quality)	Set properties (width, height, quality) of the post-processed image. If this function is not called, the default parameters will be applied. Exceptions: • IllegalArgumentException

Parameters to use Parameters to use in the above functions	
String configPath	This parameter specifies the path to the "Config" folder. This is an ESSENTIAL parameter for the operation of the SDK.
String inputPath	This parameter specifies the path to the folder that contains the image to be post-processed.
String outputPath	This parameter specifies the path to the folder where the post-processed image will be saved.
String documentType	This parameter can take six values that correspond to the names of the document models to be captured: • ES_ID_FRONT • ES_ED_BACK • ES_RESIDENCE_FRONT • ES_RESIDENCE_BACK • INT_PASSPORT • EU_CHECK_FRONT • STRANDARD_CARD

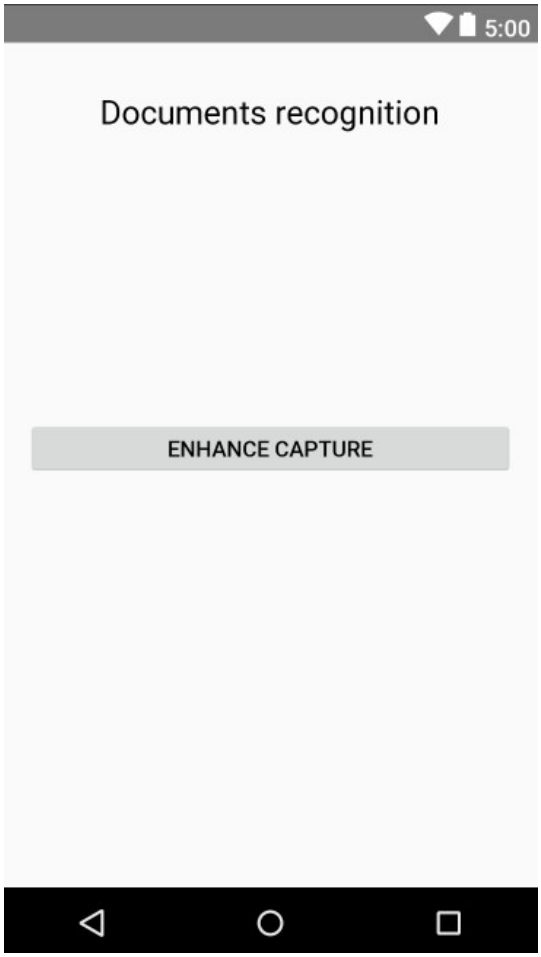
String imageName	This parameter contains the name of the image to be post-processed that is saved in the Captures folder stored in the SD memory. This parameter must be the output of the getPostProcessImageName() function.
int width	This parameter specifies the capture area width in pixels. • Minimum size permitted: 1 pixel • Maximum size permitted: ∞ pixels • Default size on ID/RESIDENCE/STANDARD_CARD: 670 pixels • Recommended size on ID/RESIDENCE/STANDARD_CARD: 670 pixels • Default size on INT_PASSPORT: 475 pixels • Recommended size on INT_PASSPORT: 475 pixels • Default size on EU_CHECK: 692 pixels • Recommended size on EU_CHECK: 692 pixels
int height	This parameter specifies the capture area width in pixels. • Minimum size permitted: 1 pixel • Maximum size permitted: ∞ pixels • Default size on ID/RESIDENCE/STANDARD_CARD: 425 pixels • Recommended size on ID/RESIDENCE/STANDARD_CARD: 425 pixels • Default size on INT_PASSPORT: 475 pixels • Recommended size on INT_PASSPORT: 475 pixels • Default size on EU_CHECK: 315 pixels • Recommended size on EU_CHECK: 315 pixels

int quality	This parameter specifies the capture quality percentage: • Minimum percentage permitted: 0 • Maximum percentage permitted: 100 • Default percentage: 100 • Recommended percentage: 100

Usage example

Once the AAR ImagePostProcessor is installed, we can start to use it.

Assuming that we have a button that allows us to post process an image asynchronously, the implemented code is shown in the following image. Firstly, on line 37 an object of the class is created that allows the post-processing to be executed. On the following three lines, the paths are set to the folders that contain the "*post.cfg*" configuration file, and the input and output folders. In this case, both the input and output are located in the same folder called Captures. Below, on line 83, the height, width and quality of the image are set. Finally, on line 84, the post-processing is run asynchronously, and awaits the name of the improved image.



```
01:     package serimagmedia.capturaautomatica;
02:
03:     import android.graphics.Color;
04:     import android.os.Bundle;
05:     import android.os.Environment;
06:     import android.support.v7.app.AppCompatActivity;
07:     import android.view.View;
08:     import android.widget.TextView;
09:     import java.io.File;
10:
11:     import automaticdocumentcapturesdk.PostProcessInterface;
12:
13:
14:     public class StartMenu extends AppCompatActivity {
15:
16:         private File path_sd = Environment.getExternalStorageDirectory();
17:         private File path_captures = new File(path_sd.getAbsolutePath() +
18:     "//IntelligentScanner//Captures");
19:         private File path_config = new File(path_sd.getAbsolutePath() +
20:     "//IntelligentScanner//Config");
21:         public static PostProcessInterface postProcessInterface;
22:         static final int PERMISSION_REQUEST = 1;
23:
24:         public StartMenu(){
25:         }
26:
27:         @Override
28:         protected void onCreate(Bundle savedInstanceState) {
29:             super.onCreate(savedInstanceState);
30:             requestPermissions();
31:             setContentView(R.layout.activity_start_menu);
32:
33:     }
```

```
34:
35:     public void initSMAAT_SDK(){
36:
37:         postProcessInterface = PostProcessInterface.getInstance();
38:         postProcessInterface.setConfigPath(path_config.getPath());
39:         postProcessInterface.setInputPath(path_captures.getPath());
40:         postProcessInterface.setOutputPath(path_captures.getPath());
41:
42:     }
43:
44:     public void requestPermissions(){
45:         ActivityCompat.requestPermissions(this, new String[]{
46:
47:                 // Only if you implement the example using external
48:                    //storage (SD)
49:                 Manifest.permission.WRITE_EXTERNAL_STORAGE,
50:
51:         },PERMISSION_REQUEST);
52:     }
53:
54:
55:     public void onRequestPermissionsResult(int requestCode,
56:                            String permissions[], int[] grantResults) {
57:         switch (requestCode) {
58:             case PERMISSION_REQUEST: {
59:           // If request is cancelled, the result arrays are empty.
60:                 if (grantResults.length > 0
61:                         && grantResults[0] ==
62:                                PackageManager.PERMISSION_GRANTED) {
63:
64:          // Initialize SDK only if permissions are granted
65:                     initSMAAT_SDK();
66:
67:                 }
68:                 return;
69:             }
70:         }
71:     }
72:
73:
74:
75:
76:
```

```
77:   public void onClickPost(View v){
78:
79:           Runnable runnable = new Runnable() {
80:               @Override
81:               public void run() {
82:
83:                   postProcessInterface.setImageQuality(670,425,100);
84:                   postProcessInterface.run("ES_ID_BACK",
85:   "2018_04_24_04_34_45_temp.jpg");
86:
87:               }
88:           };
89:           new Thread(runnable).start();
90:
91:       }
92:   }
```