

Acuant Android SDK API Documentation

Last updated on – 04/25/2016

Contents

1	Introduction	3
2	Requirements	3
3	Integration.....	4
4	Activating the license key.....	5
5	Initialize and create the SDK's instance.....	6
6	Capturing and cropping a card.....	7
7	Processing a card	10
8	Errors handling	16
9	Error Types	17
10	Change Log.....	18

1 Introduction

The AcuantAndroidMobileSDK is designed to simplify your development efforts. The processing of the captured images takes place via Acuant's Web Services. Our Web Services offer fast data extraction and zero maintenance as software is looked after by Acuant on our optimized cloud infrastructure.

Benefits:

- ❖ Process Enhancement: Faster data extraction and process images via Acuant's Web Services.
- ❖ Easy to set up and deploy.
- ❖ No maintenance and support: All maintenance and updates are done on Acuant servers.
- ❖ Secured Connection: Secured via SSL and HTTPS AES 256-bit encryption.

Acuant Web Services supports processing of drivers licenses, state IDs, other govt issued IDs, custom IDs, driver's license barcodes, passports, medical insurance cards etc. It also supports address verification, identity verification and personal verification.

For IDs from Asia, Australia, Europe, South America, Africa – we are support dd-mm-yyyy date format.

For IDs from Canada, USA – we are support mm-dd-yyyy date format.

For a complete list of regions, states, and countries supported for ID processing, please see Appendix F of ScanW document - <http://www.id-reader.com/ftp/applications/sdk/docs/ScanW.pdf>

To execute any Acuant Android Mobile SDK method, a valid license key is required. Please contact sales@acuantcorp.com to obtain a license key.

This Acuant Android Mobile SDK API documentation document has the detailed description of all the important functions a developer would need to write integration with Acuant Android Mobile SDK.

2 Requirements

- AndroidSDK Version 17 or later.
- 5 MP camera resolution or higher.
- The card image must be taken in an acceptable light conditions to avoid glare and overhead lights for example.
- The card must preferably be fitted with in the brackets on the camera screen, to allow the picture to be taken at a maximum resolution.

3 Integration

A Add AcuantAndroidMobileSDK SDK

a Using Gradle

In order to add the framework to your project, add the AcuantAndroidMobileSDK.aar dependencies

a.1 Local file

Add the following code in your build.gradle to avoid file collision.

```
dependencies {
    configurations.create("default")
    artifacts.add("default", file('acuantMobileSDK.aar'))
}
android {
    packagingOptions {
        exclude 'META-INF/NOTICE'
        exclude 'META-INF/LICENSE'
        exclude 'META-INF/DEPENDENCIES'
        exclude 'META-INF/DEPENDENCIES.txt'
        exclude 'META-INF/LICENSE.txt'
        exclude 'META-INF/NOTICE.txt'
    }
}
dependencies {
    compile 'com.microblink:pdf417.mobi:5.4.1@aar'
    compile ('com.android.support:appcompat-v7:23.1.1')
    compile ('com.google.code.gson:gson:2.5')
    compile ('com.squareup.okhttp3:okhttp:3.2.0')
}
```

a.2 JCenter repositories

In order to add the framework to your project, add the AcuantAndroidMobileSDK dependencies from JCenter

```
repositories {
    jcenter ()
}
dependencies {
    compile 'com.microblink:pdf417.mobi:5.4.1@aar'
    compile ('com.android.support:appcompat-v7:23.1.1')
    compile ('com.google.code.gson:gson:2.5')
    compile ('com.squareup.okhttp3:okhttp:3.2.0')
```

```
}
```

Add the following code in your build.gradle to avoid some file collision

```
android{
    packagingOptions {
        exclude 'META-INF/NOTICE'
        exclude 'META-INF/LICENSE'
        exclude 'META-INF/DEPENDENCIES'
        exclude 'META-INF/DEPENDENCIES.txt'
        exclude 'META-INF/LICENSE.txt'
        exclude 'META-INF/NOTICE.txt'
    }
}
```

B Add views into manifest

Add the followings activities into manifest.xml file:

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>

<activity
    android:name="com.acuant.mobilesdk.detect.CameraCardDetectManual">
</activity>
<activity
    android:name="com.acuant.mobilesdk.detect.PDF417.CameraPDF417">
</activity>
<activity
    android:name="com.acuant.mobilesdk.detect.Camera2CardDetectManual">
</activity>
```

4 Activating the license key

In order to activate the license key, use the following method:

```
AcuantAndroidMobileSDKControllerInstance.setWebServiceListener(this);
```

then, call the web service:

```
AcuantAndroidMobileSDKControllerInstance.callActivateLicenseKeySe
```

```
rvise(key);
```

the callback method `activateLicenseKeyCompleted` in the listener will be called when the activation finishes.

Note: The license key only needs to be activated once. Execute this method only one time. Some licensees are issued by Acuant pre-activated and don't need further actions.

5 Initialize and create the SDK's instance

A With activity and license key.

Pass an activity to initialize the `AcuantAndroidMobileSDKController` class, and the license key.

```
AcuantAndroidMobileSDKController.getInstance(activity,  
licenseKey);
```

B With activity, cloud address and license Key

Pass an activity to initialize the `AcuantAndroidMobileSDKController` class, the cloud address and the license key. The cloud Address must not contain "https://". Ex: "https://cloud.myAddress.com/" must be written "cloud.myAddress.com". Note: Only set cloud address if you are hosting Acuant web services in your own data center. By default, Android MobileSDK communicates with the Acuant data center.

```
AcuantAndroidMobileSDKController.getInstance(activity,  
"cloud.myAddress.com", licenseKey);
```

C With activity.

Pass an activity to initialize the `AcuantAndroidMobileSDKController` class, the entry point to the library:

```
AcuantAndroidMobileSDKController.getInstance(activity);
```

D If your instance was created previously.

Once the controller was created, you can obtain it through:

```
AcuantAndroidMobileSDKController.getInstance();
```

6 Capturing and cropping a card

A Add the card capture method.

In order to show the camera interface, you need to know the card type that you want to capture.

If you need to capture driver's license card or medical card or passport you will need to use the manual camera interfaces.

If you need to capture Driver's License, you need to call 2 times: for the front side card and for the back side card.

a Validating a license key and show the camera interface

```
AcuantAndroidMobileSDKControllerInstance.setWidth(myWidth);
```

```
AcuantAndroidMobileSDKControllerInstance.setPdf417BarcodeImageDrawable(getResources().getDrawable(R.drawable.barcode));
```

```
AcuantAndroidMobileSDKControllerInstance.  
getInstanceAndShowCameraInterface(contextActivity, license, activity, cardType,  
region, isBarcodeSide);
```

The width values are mandatory, they are set to indicate the width and height of the cropped cardimage.

A Drawable can be provided before calling `getInstanceAndShowCameraInterface` method in order to be displayed in the barcode scanning functionality. If not, no image will be shown. `currentOptionType` is one of the `AcuantCardType` possibilities: Driver License, Medical Insurance or Passport.

`stringMessageMessage` to show.

`currentOptionType` is one of the `AcuantCardType` possibilities: passport.

b Show the manual camera interface methods

```
AcuantAndroidMobileSDKControllerInstance.setWidth(myWidth);
```

```
acuantAndroidMobileSdkControllerInstance.showManualCameraInterface(mainActivity,  
CardType.DRIVERS_LICENSE, cardRegion, isBackSide);
```

The width values are mandatory, they are set to indicate the width and height of the cropped cardimage.

After the user taps the screen, the cropping process begins, there are two callback methods:

- **public void** `onCardCroppedStart(Activity activity);`

`activity`: the activity of the full screen Window, or the activity owner of the modal dialog (in case of Passport and Tablet for example)

- **public void** onCardCroppedFinish(Bitmap bitmap);
bitmap: the image card result
This function returns the cropped card image is returned.
- **public void** onCardCroppedFinish(final Bitmap bitmap, boolean scanBackSide);
bitmap: the image card result
This function returns the cropped card image is returned.
scanBackSide: A flag to alert the user to capture the back side of the card.
- **public void** onOriginalCapture(Bitmap bitmap);
bitmap: the image before the cropping process begins.
This function returns the card image without crop process.
- **public void** onCancelCapture();
Called when the user tap the back button.

c Show the barcode camera methods

```
AcuantAndroidMobileSDKControllerInstance.setWidth(myWidth);
```

```
AcuantAndroidMobileSDKControllerInstance.setPdf417BarcodeImageDrawable(getResources().getDrawable(R.drawable.barcode));
```

```
acuantAndroidMobileSdkControllerInstance.showCameraInterfacePDF417(mainActivity, CardType.DRIVERS_LICENSE, cardRegion);
```

The width values are mandatory, they are set to indicate the width and height of the cropped cardimage.

A Drawable can be provided before calling showCameraInterfacePDF417 method in order to be displayed in the barcode scanning functionality. If not, no image will be shown.

After the user opens the camera, the detection process begins, there are only one callback methods:

- **public void** onPDF417Finish (String result);
result: the barcode string result
- **public void** onBarcodeTimeOut();
This function will trigger to alert that the capture is pending without closing the camera view
- **getBarcodeCameraContext();**

return: The current barcode camera context.
This function return null if the barcode camera is close.

- **pauseScanningBarcodeCamera();**

This function pause the barcode camera detection

- **resumeScanningBarcodeCamera();**

return: The current barcode camera context.
This function resume the barcode camera detection

- **finishScanningBarcodeCamera();**

return: The current barcode camera context.
This function close the barcode camera.

- **public void onCancelCapture();**

Called when the user tap the back button.

B Optional, Add the following methods to customize.

setPdf417BarcodeImageDrawable: Customize the barcode interface with an image, default empty.

```
AcuantAndroidMobileSDKControllerInstance.setPdf417BarcodeImageDrawable(getResources().getDrawable(R.drawable.barcode));
```

setWatermarkText: method to see the watermark on your camera

```
AcuantAndroidMobileSDKController.setWatermarkText("Powered By Acuant",0,0,30,0);
```

setInitialMessageDescriptor: Customize the initial message, default implementation says "Align and Tap" or "Tap to Focus".

```
setInitialMessageDescriptor(R.layout.hold_steady);
```

```
setInitialMessageDescriptor(message, red, green, blue, alpha);
```

setFinalMessageDescriptor : Customize the capturing message, default implementation says "hold steady".

```
setFinalMessageDescriptor(R.layout.align_and_tap);
```

```
setFinalMessageDescriptor(message, red, green, blue, alpha);
```

setFlashlight: Enable or disable the flashlight, by default is false.

```
setFlashlight(showFlashlight);
```

```
setFlashlight(left, top, right, bottom);
```

setCropBarcode: Enable or disable the barcode image cropping. By default is false.
setCropBarcode(canCropBarcode);

setShowActionBar: Enable or disable the action bar. By default is false.
setShowActionBar (false);

setShowStatusBar: Enable or disable the status bar. By default is false.
setShowStatusBar (false);

setShowInitialMessage: Enable or disable the barcode camera message. By default is false.
setShowInitialMessage (false);

setCanShowBracketsOnTablet: Enable or disable the guiding brackets for tablets
setCanShowBracketsOnTablet(true);

C Add the following methods to set the size of the card.

If the proper card size is not set, MobileSDK will not be able to process the card.

For Driver's License Cards

`AcuantAndroidMobileSDKControllerInstance.setWidth(1250);`

For Medical Insurance Cards

`AcuantAndroidMobileSDKControllerInstance.setWidth(1012);`

For Passport Documents

`AcuantAndroidMobileSDKControllerInstance.setWidth(1478);`

7 Processing a card

After the capture and the crop process, you can retrieve information through processing of the cropped image.

A Add a callback for the web service.

`AcuantAndroidMobileSDKControllerInstance.setWebServiceListener(callback);`

B Call the web service to process the card image

a For Driver's License Cards

```
ProcessImageRequestOptions options =
ProcessImageRequestOptions.getInstance();
options.autoDetectState = true;
options.stateID = -1;
options.reformatImage = true;
options.reformatImageColor = 0;
options.DPI = 150;
options.cropImage = false;
options.faceDetec = true;
options.signDetec = true;
options.iRegion = region;
options.imageSource = 101;
options.acuantCardType = cardType;
```

```
AcuantAndroidMobileSDKControllerInstance.callProcessImageServices(frontSideCa
rdImage, backSideCardImage, barcodeString, callerActivity, options);
```

Explanation of the parameters:

region - Integer parameter for the Region ID. Parameter value -

United States - 0

Australia - 4

Asia - 5

Canada - 1

America - 2

Europe - 3

Africa - 7

General Documents - 6

autoDetectState- Boolean value. True - SDK will auto detect the state of the ID. False - SDK wont auto detect the state of the ID and will use the value of ProcState integer.

stateID - Integer value of the state to which ID belongs to. If AutoDetectState is true, SDK automatically detects the state of the ID and stateID value is ignored. If AutoDetectState is false, SDK uses stateID integer value for processing. For a complete list of the different countries supported by the SDK and their different State integer values, please see Appendix F of ScanW document -

<http://www.id-reader.com/ftp/applications/sdk/docs/ScanW.pdf>

faceDetec - Boolean value. True - Return face image. False - Won't return face image.

signDetec - Boolean value. True - Return signature image. False - Won't return signature image.

reformatImage - Boolean value. True – Return formatted processed image. False – Won't return formatted image. Values of ReformatImageColor and ReformatImageDpi will be ignored.

reformatImageColor - Integer value specifying the color value to reformat the image. Values –
 Image same color – 0
 Black and White – 1
 Grayscale 256 – 2
 Color 256 – 3
 True color – 4
 Enhanced Image – 5

DPI - Integer value up to 600. Reformats the image to the provided DPI value. Size of the image will depend on the DPI value. Lower value (150) is recommended to get a smaller image.

cropImage - Boolean value. When true, cloud will crop the RAW image. Boolean value. Since MobileSDK crops the image, leave this flag to false.

imageSource – To identify the source of the image. 101 is the value for MobileSDK.

b For Medical Insurance Cards

```
ProcessImageRequestOptions options
=ProcessImageRequestOptions.getInstance();
options.reformatImage = true;
options.reformatImageColor = 0;
options.DPI = 150;
options.cropImage = false;
options.acuantCardType = cardType;
```

```
AcuantAndroidMobileSDKControllerInstance.callProcessImageServices(frontSideCardImage, backSideCardImage, null, callerActivity, options);
```

Explanation of the parameters:

reformatImage - Boolean value. True – Return formatted processed image. False – Won't return formatted image. Values of ReformatImageColor and ReformatImageDpi will be ignored.

reformatImageColor - Integer value specifying the color value to reformat the image. Values –
 Image same color – 0
 Black and White – 1
 Grayscale 256 – 2
 Color 256 – 3
 True color – 4
 Enhanced Image – 5

DPI - Integer value up to 600. Reformats the image to the provided DPI value. Size of the image will depend on the DPI value. Lower value (150) is recommended to get a smaller image.

cropImage – Boolean value. When true, cloud will crop the RAW image. Boolean value. Since MobileSDK crops the image, leave this flag to false.

c For Passport Cards

```
ProcessImageRequestOptions options =
ProcessImageRequestOptions.getInstance();
options.reformatImage = true;
options.reformatImageColor = 0;
options.DPI = 150;
options.cropImage = false;
options.faceDetec = true;
options.signDetec = true;
options.acuantCardType = cardType;
options.imageSource = 101;
```

```
AcuantAndroidMobileSDKControllerInstance.callProcessImageServices(frontSideCardImage, null, null, callerActivity, options);
```

Explanation of the parameters:

faceDetec - Boolean value. True - Return face image. False – Won't return face image.

signDetec- Boolean value. True – Return signature image. False – Won't return signature image.

reformatImage - Boolean value. True – Return formatted processed image. False – Won't return formatted image. Values of ReformatImageColor and ReformatImageDpi will be ignored.

reformatImageColor - Integer value specifying the color value to reformat the image. Values –
Image same color – 0
Black and White – 1
Grayscale 256 – 2
Color 256 – 3
True color – 4
Enhanced Image – 5

DPI - Integer value up to 600. Reformats the image to the provided DPI value. Size of the image will depend on the DPI value. Lower value (150) is recommended to get a smaller image.

cropImage – Boolean value. When true, cloud will crop the RAW image. Boolean value. Since MobileSDK crops the image, leave this flag to false.

C Finally, do your post-processing of the card information

The callback method:

`processImageServiceCompleted(AcuantCard card)`

`card`: a 'card' object with the scanned information

`status`: one of the constants of `AcuantErrorType`

`message`: error message from the server

is called when the web service completes. A 'card' with the card information is returned. It will be an instance of `DRIVERS_LICENSE`, `PASSPORT`, `MEDICAL_INSURANCE` according to the original card type you passed to the web service. You can retrieve state, signature, name, etc. from this class, for example for license driver's card, these are some properties:

```
String name;  
String licenceID;  
String address;  
String city;  
String zip;  
String state;  
String idCountry;  
String eyeColor;  
String hair;  
String height;  
String weight;  
String licenceClass;  
String restriction;  
String sex;  
String county;  
String dateOfBirth;  
String expirationDate;  
String nameLast;  
String nationality;  
String placeOfBirth;
```

```
Bitmap faceImage;  
Bitmap signImage;  
Bitmap reformatImage;
```

You can retrieve the name through:

```
card.getName()
```

also, you can check all the properties for all the card types in the API doc.

This is the implementation in the Sample project:

```

/**
 *
 */
@Override
public void processImageServiceCompleted(Card card) {
    if (Util.LOG_ENABLED) {
        Utils.appendLog(TAG, "public void processImageServiceCompleted(CSSNCard card, int
status, String errorMessage)");
    }
    isProcessing = false;

    Util.dismissDialog(progressDialog);

    String dialogMessage = null;

    try {
        DataContext.getInstance().setCardType(mainActivityModel.getCurrentOptionType());

        if (card == null || card.isEmpty()) {
            dialogMessage = "No data found for this license card!";
        } else {

            switch (mainActivityModel.getCurrentOptionType()) {
                case CardType.DRIVERS_LICENSE:
                    DataContext.getInstance().setProcessedLicenseCard((DriversLicenseCard) card);
                    break;

                case CardType.MEDICAL_INSURANCE:
                    DataContext.getInstance().setProcessedMedicalCard((MedicalCard) card);
                    break;

                case CardType.PASSPORT:
                    DataContext.getInstance().setProcessedPassportCard((PassportCard) card);
                    break;

                default:
                    throw new IllegalStateException("There is not implementation for processing the
card type "
                        + mainActivityModel.getCurrentOptionType() + "");
            }

            Util.unLockScreen(MainActivity.this);

            Intent showDataActivityIntent = new Intent(this, ShowDataActivity.class);
            this.startActivity(showDataActivityIntent);
        }
    }
}

```

```

    } catch (Exception e) {
        Utils.appendLog(TAG, e.getMessage());
        dialogMessage = "Sorry! Internal error has occurred, please contact us!";
    }

    if (dialogMessage != null) {
        Util.dismissDialog(alertDialog);
        alertDialog = Util.showDialog(this, dialogMessage, new DialogInterface.OnClickListener()
        {

            @Override
            public void onClick(DialogInterface dialog, int which) {
                isShowErrorAlertDialog = false;
            }
        });
        isShowErrorAlertDialog = true;
    }
}

```

8 Errors handling

In order to handle the errors or alert over SDK's action , you will receive the error on `didFailWithError(int code, String message)` method.

This is the implementation in the Sample project:

```

@Override
public void didFailWithError(int code, String message) {
    Util.dismissDialog(progressDialog);
    Util.unLockScreen(MainActivity.this);
    String msg = message;
    if (code == ErrorType.AcuantErrorCouldNotReachServer) {
        msg = getString(R.string.no_internet_message);
    } else if (code == ErrorType.AcuantErrorUnableToCrop) {
        updateModelAndUIFromCroppedCard(originalImage);
    }
    alertDialog = Util.showDialog(this, msg, new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            isShowErrorAlertDialog = false;
        }
    });
    isShowErrorAlertDialog = true;
}

```



```

if (Util.LOG_ENABLED) {
    Utils.appendLog(TAG, "didFailWithError:" + message);
}
// message dialogs
isValidating = false;
isProcessing = false;
isActivating = false;
}

```

9 Error Types

```

public final static int AcuantErrorCouldNotReachServer = 0; //check internet
connection
public final static int AcuantErrorUnableToAuthenticate = 1; //keyLicense are
incorrect
public final static int AcuantErrorUnableToProcess = 2; //image received by the server
was unreadable, take a new one
public final static int AcuantErrorInternalServerError = 3; //there was an error in
our server, try again later
public final static int AcuantErrorUnknown = 4; //there was an error but we were unable to
determine the reason, try again later
public final static int AcuantErrorTimedOut = 5; //request timed out, may be because
internet connection is too slow
public final static int AcuantErrorAutoDetectState = 6; //Error when try to detect the
state
public final static int AcuantErrorWebResponse = 7; //the json was received by the server
contain error
public final static int AcuantErrorUnableToCrop = 8; //the received image can't be
cropped.
public final static int AcuantErrorInvalidLicenseKey = 9; //Is an invalid license key.
public final static int AcuantErrorInactiveLicenseKey = 10; //Is an inactive license
key.
public final static int AcuantErrorAccountDisabled = 11; //Is an account disabled.
public final static int AcuantErrorOnActiveLicenseKey = 12; //there was an error on
activation key.
public final static int AcuantErrorValidatingLicensekey = 13; //The validation is
still in process.
public final static int AcuantErrorCameraUnauthorized = 14; //The privacy settings
are preventing us from accessing your camera.
public final static int AcuantNoneError = 200; //The privacy settings are preventing us from
accessing your camera.

```

10 Change Log

Acuant Android MobileSDK version 4.0.0.

Changes:

- Changed internal dependence to external dependence.
- Added callback when the user taps the back button.
- Made several improvements to barcode scanning interface.
- Replaced HTTPClient with OkHTTP.
- Fixed camera focus issues on Galaxy S6 and Nexus 9.
- Added a new field DocumentDetectNameShort for IDs/DLs.

Added:

- Added support for Camera2 API.