Top Image Systems

Digital Business Starts Here.

mobiFLOW

Android Library SDK Guide

Version 5

contact@topimagesystems.com

www.topimagesystems.com

11 January, 2018

Contents

About this guide 5
General notes 5
Project settings 5
Supported architectures 5
Libraries 6
Android Studio 6
Android SDK version 7
Permissions 7
Features 7
Activities 7
Library 8
Interaction with the Library 8
Session parameters 8
Use fragments 19
Android 6.0 permissions 20
License parameters 22
IQA parameters 22
For debug purposes only 26
Document types 27
Full Page capture session parameters 27
Custom capture parameters 27
Payment capture parameters 27
Check capture session parameters 28
Passport capture session parameters 28
Card capture session parameters 28
Video processing guidelines 28
Force still capture 28
Capture messages and errors 29
Session events 29
Error messages 30
UI events 32
Handle session continuation 33
Handle the session results 39
Library flow 46
Clear temporary files 46
Security recommendations 47

Set up a custom capture user interface 48
Change the look and feel 48
Change icons and captions 49
Change the text indicators 50
Change countdown image view 51
DebugRectView 52
Camera overlay color 52
Rectangle check frame 52
Custom view 52
Guidelines popup 53
Additional functionality in Custom view 53
Info screen configuration 53
Start a new activity that is not part of the Library 54
Leveler configuration 54
Configure levelerUI 54
OneUnitLeveler 55
Leveler parameters 55
TwoUnitsLeveler and Scale Leveler 55
Captions and messages 56
Reporting issues 59
Guidelines for successful capture 60
Contrast 60
Background homogeneity 60
Lighting 60
Shooting and rotation angles 60
Taking the picture 60
Checks only: Digital row (MICR) 61

About this guide

This guide describes the mobiFLOW image capture library, and explains how to use this library to integrate mobiFLOW into other Android apps, using Java.

Known issues and workarounds are also described.

General notes

- The minimal supported SDK version is 9.
- The easiest way to provide image detection in the application is to add a library project named *mobiFLOW*. The image capturing session can be called via Intent. The mobiFLOW library handles all aspects associated with the camera (integrating with the MICR algorithm for checks) and then providing the result in the Intent result.
- Image boundaries detection and contrast verification is performed on the image preview frames. The OCR algorithm used for this part is not very precise, but relatively fast.
- Before sending the image to the server, the image is cropped, binarized (with 1 channel) from a color image to a B&W image and set to TIFF with Group 4 Fax Encoding (CCITT T.6).

Project settings

This section describes the project settings for the Android SDK.

Supported architectures

The mobiFLOW package includes the following architectures by default: armeabi-v7a, arm64-v8a, x86.

If you wish to add additional ABI's (armeabi, mips, mips64, x86_64), contact TIS Support.

To reduce the APK size to a minimum, we recommend splitting the APK into multiple ABI's per architecture and uploading a number of APK files per architecture to Google Play.

If multiple APK files is not an option, and reducing the universal APK size is necessary, we recommend removing x86 Intel support devices: remove the folder that contains x86 so that files for Intel devices are not supported.

See the Showcase project settings for reference.

Libraries

Android Studio

The project can be integrated in manually or via jCenter. For static UI customization via XML, use manual integration.

Manual integration

Your project should contain 2 modules: *mobiFLOW* and *openCV*.

The *Build.gradle* file of the calling app should contain the mobiFLOW library. Add the mobiFLOW library to your dependencies, as shown here:

```
dependencies {
        compile fileTree(dir: 'libs', include: '*.jar')
        compile project(':mobiFLOW')
}
```

The *Build.gradle* file for mobiFLOW should include a dependency for OpenCV, as shown here:

```
dependencies {
        compile fileTree(dir: 'libs', include: '*.jar')
}
```

```
splits {
        abi {
  enable true
                reset()
                include 'armeabi-v7a', 'arm64-v8a', 'x86'
                universalApk true
  }
}
```

The *ndk* tag should be included under *defaultConfig* in the calling application, with supported ABI's:

```
defaultConfig{
…..
    ndk {
        abiFilters 'armeabi-v7a', 'arm64-v8a','x86'
   }
}
```

You can use the Showcase app as a reference and copy the configuration from there.

jCenter integration

Add compile *com.topimagesystems:MobiFlow:5.0* to your build gradle.

Note: The Android SDK version, permissions, and features must be added to the calling application Manifest.

Android SDK version

```
<uses-sdk android:minSdkVersion="9"/>
```

Permissions

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.FLASHLIGHT" />
```

Features

```
<uses-feature android:name="android.hardware.camera" android:required="true" />
<uses-feature android:name="android.hardware.camera.autofocus" />
<uses-feature android:name="android.hardware.camera.flash" android:required="true" />
```

Activities

Add full class paths to camera-related activities and instructions activities from the library project.

```
<activity
        android:name="com.topimagesystems.controllers.instructions.InstructionsController"
        android:screenOrientation="portrait" />
<!-- camera -->
<activity
        android:name="com.topimagesystems.controllers.imageanalyze.CameraManagerController"
        android:configChanges="keyboardHidden|orientation|screenSize" >
</activity>
<activity
        android:name="com.topimagesystems.controllers.imageanalyze.CameraController"
        android:configChanges="keyboardHidden|orientation|screenSize"
        android:hardwareAccelerated="false" >
</activity>
<activity
```

```
        android:name="com.topimagesystems.ui.InfoScreenActivity"
        android:theme="@android:style/Theme.Translucent.NoTitleBar" >
</activity>
```

Library

Interaction with the Library

Full sample code and integration can be found in the sample SDK application.

Session parameters

Parameter	Description
documentType	Document type set to one of the enums: ■ TISDocumentType.CHECK ■ TISDocumentType.PAYMENT ■ TISDocumentType.FULLPAGE ■ TISDocumentType.PASSPORT ■ TISDocumentType.CARD ■ TISDocumentType.CUSTOM **Note:** There no default. This must be set!
debugMode	In debug mode, images are stored on the device and logs are written to the console. Default value: FALSE

Parameter	Description
uxType	Static capture sets predefined boundaries on the screen according to the aspect ratio, while the document must be placed relatively within the shown boundaries. Live capture looks for a quadrilateral of a document in any size, with optional additional settings according to document type, and validates that the document is in the correct aspect ratio. Setting the aspect ratio to 0.0 both for *Minimum* and *Maximum* will skip validation in live mode and let you capture any document. *TISFlowUXType* can be set to one of the following values : ■ STATIC ■ LIVE Default value: LIVE
minHeightWidthAspectRatio	Minimum allowed ratio between the captured image's height and width. Default values: 0.35 (check, bill) 1.17 (full page) 0.65 (passport) 0.582 (card)
maxHeightWidthAspectRatio	Maximum allowed ratio between the captured image's height and width. Default values: 0.50 (check, bill) 1.56 (full page) 0.8 (passport) 0.7117 (card)
enableIQA	Enable or disable the IQA validations. Default value: FALSE

Parameter	Description
IQASettings	A class of type *IqaSettings* to set all the threshold parameters for the IQA validations. Can leave defaults if not in use.
showInfoScreen	Show the information screen if the user has difficulty capturing the document after a specific set time. Default value: TRUE
InfoScreenInterval	The number of milliseconds until the information screen appears on the camera overlay. Default Value: 10000
showGuidelinesIndicators	When set to FALSE, only two static indicators will be presented. ■ *TISFlowIndicatorAlign* – static indicator for alignment (the device should be aligned with the document) ■ *TISFlowIndicatorHold* – indicator for hold (the device should be held over the document) When set to TRUE, dynamic indicators will be presented. Default value: TRUE
outputGrayscaleImage	Enable the output of a grayscale JPG. Default value: TRUE
outputOriginalImage	Enable the output of the captured original image. Default value: TRUE
outputColorImage	Enable the output of the captured cropped color image. Default value: TRUE
colorImageCompression	A value of the factor by which the JPG cropped color image will be compressed. Values ranges from 0.0 for highest compression (lowest quality) to 1.0 (highest quality). Default value: 1.0
outputBinarizedImage	Enable the output of the captured black & white image. Default value: TRUE

Parameter	Description
grayScaleSize	Set the width and height of the grayscale output image. The parameter is of type *int*. Default value: [0,0] 0 value will not resize the image.
enableBlurDetection	When set to TRUE, mobiFLOW will check the sharpness of an image and will notify when the image is blurred. **Note:** Currently blur detection does not apply on the back side of a document. Set to FALSE for Checks. Default value: TRUE for Payment and Full Page, FALSE for Check.
videoFeedProcessing	Applicable only for devices with at least 1920*1080 video resolution. When set to TRUE, the picture will be taken directly from the video feed when the document is aligned properly with the frame. In this case, the device will not switch to still mode and will not present the countdown sequence. When set to FALSE, the device will switch to still mode to take the picture. For devices with video resolution lower than 1920*1080, the image will be taken in Stills mode, even if video feed processing was set to TRUE. See Captions and messages for more details. Default value: TRUE for Check, FALSE for other types
maxVideoFramesToCapture	When video feed processing is enabled and the device video resolution is higher than the minimum (), the library will try to process the image that was captured. In case of failure, this parameter is set to the maximum attempts to capture via video mode before switching back to still mode and countdown. This parameter should be set between 5 and 10 for better performance. This parameter is relevant only when *videoFeedProcessing* is set to TRUE. Default value: 7

Parameter	Description
showCountDown	Applicable for still mode only. When set to TRUE, once the user is in a position to take a picture, the frame turns green and a countdown is shown until the picture is taken automatically. When set to FALSE, no countdown is shown. The picture is taken when the frame is green and the user sees the *hold still* message on the screen. Default value: FALSE
enableCountdownSound	Enable a sound along with the image capture countdown. Default value: FALSE
dynamicStrings	Option to change *strings.xml* values at runtime. The key is the String ID as shown in the *Strings.xml*. The value is the string to replace. If a key does not have a value or all the object is null, the default value will be taken from the *Strings.xml* (set to null - no change from the original behavior). The parameter is of the type *HashMap<String, String>*. Default value: Null
showDefaultProcessingView	When set to FALSE, a new layout from *mbck_camera_layout* will be presented on the screen when the processing stage has started (*@+id/processingOverlay*). When set to TRUE, the progress bar will be presented over the camera overlay. Default value: TRUE
multiPageCapture	When set to TRUE, this parameter enables capture of multiple documents. After each capture, a prompt screen is displayed asking the user if they would like to capture another image. If the user selects **Finish**, the Library calls *OnActivityResult*. After every captured image, the *onMessageReceive* listener is called with the *TISFlowActionCallback* action MULTI_CAPTURE (see section 3.2.1), but the camera session stays open until the user finishes the multipage session. Default value: FALSE

Parameter	Description
binarizeBackSameAsFront	When set to TRUE, the same binarization algorithm that runs on the front side will run on the back side of the check. Default value: FALSE
binarizationThreshold	Threshold for the strength of the binarization algorithm. Values can be between 0.0 and 1.0. This should be set only when capturing a single size document. If the size varies, like Bills, then it should be set to 0.0 for optimization. 1.0 – darkest 0.0 – The SDK calculates the optimal threshold according to the image size. Default value: 0.0
scanFrontOnly	When set to TRUE, only the front side will be captured. When set to FALSE and *scanBackOnly* is set to FALSE, both front and back will be captured. Default value: FALSE
scanBackOnly	When set to TRUE, only the back side is captured. If *scanFrontOnly* is also TRUE, it will fail to initialize the Library. Default value: FALSE
softCapture	Provides the ability to capture the document while the device is held at an angle and not necessarily flat over the document. In this case, the document image will be straightened and aligned from the angled position to a flat position. Using this method may impact the quality of the final image. Default value: NO

Parameter	Description
scanBarcodeLocation	Of type *TISScanBarcodeLocation* enum value. When set to NONE, no barcode will be detected. Otherwise, it directs on which side of the document the barcode will be detected. Possible values are: BARCODE_FRONT BARCODE_BACK BARCODE_FRONT_AND_BACK BARCODE_NONE Default value: BARCODE_NONE

Parameter	Description
barcodeTypes	ArrayList<TISBarcodeType> Relevant only when *scanBarcodeLocation* is not NONE. Contains the barcode types that will be recognized during the barcode scan session. Once a barcode is detected, if there is a match with one of the barcode types, the barcode is parsed and the SDK continues to capture the document. If one of the barcode types includes QR_CODE, AZTEC_CODE or DATA_MATRIX_CODE, a square will appear instead of a rectangle for the barcode detection. Supported barcode types in the array: ■ UPCE_CODE ■ CODE_39_CODE ■ CODE_39_MOD_43_CODE ■ EAN_13_CODE ■ EAN_8_CODE ■ CODE_93_CODE ■ CODE_128_CODE ■ PDF_417_CODE ■ QR_CODE ■ AZTEC_CODE ■ INTERLEAVED_2_OF_5_CODE ■ ITF_14_CODE ■ DATA_MATRIX_CODE Default value: all barcode types
customView	When set to TRUE, uses the custom view that was set up. Default value: FALSE

Parameter	Description
ocrType	■ UNKNOWN (For Check only) ■ E13B (For Check only) ■ CMC7 (For Check only) ■ OCRA ■ MRZ (For Passport and Card only) ■ PAN (For Pancard subtype only) ■ OFF ■ CREDIT (For Credit Card only) Default value: OFF
useMaxResolution	This setting is valid for stills capture mode only. The SDK uses the optimal resolution to process an image automatically per document type and flow. You can set this to TRUE if the use of the maximum camera resolution of the device is required. **Note:** Setting this parameter to true can lead to a longer processing time. Default value: FALSE
minMICRLength (Check capture only)	Minimum MICR length (number of characters). Default value: 15
maxMICRLength (Check capture only)	Maximum MICR length (number of characters). Default value: 50
frontImageSize (Check capture only)	Size of the front black & white and grayscale images output. Should be passed as a parameter to the back scan according to the size output of the front scan when the back scan is done separately. The first value in the array is the image width and the second is the image height. Parameter type is *Int[]*.
portraitCapture (Custom document type only)	When set to TRUE, the camera opens in portrait mode. Default value: FALSE

Parameter	Description
binarizationType (All document types except Check)	Available only for the Custom document type. TIS_GENERAL_BINARIZATION: Default value for all document types except Check. TIS_CHECK_BINARIZATION: Default value for Check.
license	Of the type *TISLicenseParameters* class, which includes 3 members that must be initialized on the constructor. A valid license must be coded in order for the camera session to start, otherwise a license error message is displayed. See License parameters for more information about these parameters.
animateTransitionInLivePreview	For *TISFlowUXType.LIVE*. When set to TRUE, the green and red rectangles will switch with smooth transition animation. Default value: TRUE
softCaptureThreshold	When enabled, the calling app will have the option to control the strictness/softness of the capture and can allow wider angles and higher capture distance from the frame. Possible values are 0-1 (the default value is 0, the same threshold as previous versions). A higher value will make the capture experience less strict. **Note:** At the maximum threshold, capture at a wide angle may effect image quality. Default value: 0
tapToFocus	When set to TRUE, the phone will explicitly order the device to focus on the image after the user taps on the camera overlay. Default value: TRUE
enableBlurDetectionOnBackSide (Beta feature)	When enabled, the SDK performs blur detection on the back side of the document to avoid blurry back side images. Valid for all document types. Default value = FALSE

Parameter	Description
enableManualCapture (All document types except Passport and Credit Card)	When set to TRUE, a button will be added to the screen, allowing to take a still image immediately that will be sent to processing or to the Crop Controller. Default value: FALSE
enableCropController (All document types except Passport and Credit Card)	When set to TRUE, the image that was taken by manual capture, or automatically by the SDK, is sent to a Crop Controller to confirm the quality and cropping of the image, or to correct the cropping, before it is sent to processing. This feature is only available from Android API 11. Default value: FALSE
showErrorSignatureOverCMC7 (Check document type only)	When set to TRUE, if a signature is detected over an CMC7 MICR, an error is sent to the calling app. Default value: TRUE
showGridInLivePreview	If the *uxType* parameter is LIVE, there is an option to display the grid over the camera overlay. Default value = TRUE

To initiate image capturing, invoke an intent that will trigger the library code. Consider using a convenience class *CaptureIntent* that wraps intent invoking and result receiving.

Create a default parameter class and change any parameter if needed. This example is for bill payment. See Document types for information on the different types of document.

For Payments

```
CaptureIntent captureIntent = new CaptureIntent(activity);
// create CaptureIntent to interact with the library
paymentCaptureParams input = (paymentCaptureParams)
        CaptureIntent.getCaptureParams(TISDocumentType.PAYMENT);
// Create parameters class that matches the document type capture using casting
CaptureIntent.captureDocument(input);
```

For Checks

```
CaptureIntent captureIntent = new CaptureIntent(this); // this is your Activity context
IQASettingsIntent iqaSettings = IQASettingsIntent iqaSettings = new IQASettingsIntent().getIQASettingsDefault();//
init default IQA settings class
```

```
// check 51 init
// IQASettingsIntent iqaSettings = new IQASettingsIntent();
//iqaSettings = iqaSettings.getIQAStandart51Defaults();
checkCaptureParams input = (checkCaptureParams) captureIntent.getCaptureParams(TISDocumentType.CHECK);
// init default check configuration.
input.outputColorImage = false; // cropped color image, seems you don't need this from the settings.
input.IQASettings = iqaSettings;
input.uxType = TISFlowUXType.STATIC;
input.license = new TISLicenseParameters(… , … , ) // input the license here
CameraController.registerListener(this); // this == class context, register listener to get messages from the SDK,
does not have to be here, can be anywhere before .
captureIntent.captureDocument(input); // open camera screen.
CaptureIntent.getCaptureParams(TISDocumentType.CHECK);
```

First, initialize *CaptureIntent* to interact with the library.

After that, get the default class parameters (using casting by adding the correct enum that describes the document type), and then initialize the session with the parameters class.

Use fragments

CaptureIntent can be used from a *Fragment* class as well, and all the results will be returned to this fragment.

In this case, instead of sending the activity as a parameter to the *CaptureIntent* constructor, the fragment should be sent as a parameter. It can be a native fragment or a fragment from the support library.

This parameter type is *Object*, so an exception will be thrown if it is not a valid fragment object.

```
CaptureIntent captureIntent;
try {
        captureIntent = new CaptureIntent(fragment);
} catch (Exception e) {
        e.printStackTrace();
        return;
}
checkCaptureParams input = (checkCaptureParams) captureIntent.getCaptureParams(TISDocumentType.CHECK);
```

Implementation example for checks:

The best way to add a fragment is demonstrated in the *ShowCaseContainerActivity* (fragment used in activity) and ShowCaseContainerFragment (nested fragments: fragment used inside a fragment).

It is important when using *onActivityResult* in the container to call the super.*onActivityResult* so that it will handle the result of any child fragments used in the container.

If the fragment's container is an activity (use protected):

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
……..
// Handling Activity results
super.onActivityResult(requestCode, resultCode, data);
}
```

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
……..
// Handling Fragment results
super.onActivityResult(requestCode, resultCode, data);
}
```

If the fragment's container is a fragment (nested fragments – use public):

```
Fragment fragment = new ShowCaseFragment();
FragmentTransaction transaction = getFragmentManager().beginTransaction();
transaction.add(R.id.container, fragment);
transaction.commit();
```

To avoid the *illegalStateException* that occurs often when using fragments, it is best to inflate the fragment (no need to use the fragment tag in the layout XML).

Android 6.0 permissions

For Android M target applications, the SDK must have camera permission in order to start a session. You must choose whether to ask for permission before initiating mobiFLOW, or let mobiFLOW ask for permission.

Note: If you would like the alert to show on the calling app screen, you must ask for the camera permission before calling mobiFLOW.

The code below shows how to ask for camera permission from the calling application:

```
@TargetApi(23)
private boolean askPermission(){
Context c = getApplicationContext();

String permission = "android.permission.CAMERA";
```

```
int res = this.checkCallingOrSelfPermission(permission);
if (res == PackageManager.PERMISSION_GRANTED){
        // has permission start mobiFLOW
}
Else{
        // don't have permission, ask for permission – result will return to
        // onRequestPermissionResult method
        requestPermissions(
                new String[]{Manifest.permission.CAMERA},
                MY_PERMISSIONS_REQUEST_CAMERA);
        }
        return true;
}
// permission result callback
@Override
public void onRequestPermissionsResult(int requestCode,
                String permissions[], int[] grantResults) {
        switch (requestCode) {
        case Constants.CAMERA_PERMISSIONS_REQUEST: {
        // If request is cancelled, the result arrays are empty.
        if (grantResults.length > 0
        && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
        // permission was granted,
        // start mobiFLOW
        } else {
        // permission denied !
        }
        return;
    }
  }
}
```

If you use debug mode where images are saved on the device, you must grant storage access permission as well in the same way. This must be done from the calling app.

If mobiFLOW handles the camera permission, the alert will be shown before the camera opens. If the user approves the permission, the session will start as usual.

If the user denies permission, the session will be closed with the result code:

RESULT_CAMERA_PERMISSION_ACSSES_DENIED

License parameters

Each version of the SDK requires a license. If a license is not configured, mobiFLOW displays an error on the device's screen and does not start the camera session.

The license is individual per implementation and is made up of the licensee name, the license key, and the license itself. The license is either limited by expiration date or is unlimited.

The license is valid per SDK version and can only be used on that version, so upgrading to a newer version requires a new license that matches the version of the SDK used.

The following 3 values (which are provided by TIS) must be initialized:

Parameter	Description
licensee	The name of the licensee that the license is associated with. Usually, this will be the customer name or the project name.
licenseKey	A unique key that is given to each license or customer.
activeLicense	An encrypted string that contains the license information.

Sample code:

```
input.license = new TISLicenseParameters("ABCD","a70e52b0-e499-3562-afb1-17f04038356b","TqeRDhExXuGCLNdIcvb4OR9+QJYiTnWQ3ooFtcWx39OkkNeUYf4Ph0U+P5x6DaRldA84HwlWUzF5YMLA5k==");
```

If the license information was validated successfully, the camera session starts.

If the license validation failed, a notification is displayed on the screen to the user and the session ends with a result code: RESULT_LICENSE_INVALID.

To get the message you can call:

```
String licenseErrorMessage = data.getStringExtra(CaptureIntent.MOBIFLOW_ERROR_DETAILS);
```

See the Handle the session results section for more details.

IQA parameters

Create the *IQASettingsIntent* instance, and set its IQA properties.

IQA is used to define validation for image quality.

The parameters for *iQAParameters* should be set according to this table.

Parameter	Description	Default value
maxRotationSkew	Maximum skewing angle allowed.	7.5f
minDarknessFront	Minimum ratio of black pixels to total pixels for the front side.	0.009f
maxDarknessFront	Maximum ratio of black pixels to total pixels for the front side.	0.9f
minDarknessBack	Minimum ratio of black pixels to total pixels for back side.	0.0038f
maxDarknessBack	Maximum ratio of black pixels to total pixels for back side.	0.98f
numberOfSpotsFront	Maximum number of spots that are considered as spots allowed per square inch on average for the front side. Black areas count as spots if the size of the area is greater than 3 pixels and less than 20 pixels, and the black area is surrounded by white pixels.	5852
numberOfSpotsBack	Maximum number of spots that are considered as spots allowed per square inch on average for the back side. Black areas count as spots if the size of the area is greater than 3 pixels and less than 20 pixels, and the black area is surrounded by white pixels.	5852

Parameter	Description	Default value
cornerDataFrontTLH cornerDataFrontTLW cornerDataFrontTLA cornerDataFrontTRH cornerDataFrontTRW cornerDataFrontTRA cornerDataFrontBLH cornerDataFrontBLW cornerDataFrontBLA cornerDataFrontBRH cornerDataFrontBRW cornerDataFrontBRA	Thresholds for height, width and area (in inches) for every corner of the check on the front side. Use the function *setCornerFrontSameToAllCorners* to set the same height, width and area for all corners, or use *SetCornerFrontAll* to set a different threshold for each corner.	1.0f 1.0f 0.4f 1.0f 1.0f 0.4f 1.0f 1.0f 0.4f 0.8f 0.8f 0.3f
cornerDataBackTLH cornerDataBackTLW cornerDataBackTLA cornerDataBackTRH cornerDataBackTRW cornerDataBackTRA cornerDataBackBLH cornerDataBackBLW cornerDataBackBLA cornerDataBackBRH cornerDataBackBRW cornerDataBackBRA	Thresholds for height, width and area (in inches) for every corner of the check on the back side. Use the function *setCornerBackSameToAllCorners* to set the same height, width and area for all corners, or use *SetCornerBackAll* to set a different threshold for each corner.	0.3f 0.3f 0.1f 0.3f 0.3f 0.1f 0.8f 0.8f 0.3f 0.3f 0.3f 0.1f
edgeDataTH edgeDataTW edgeDataTA edgeDataRH edgeDataRW edgeDataRA edgeDataBH edgeDataBW edgeDataBA edgeDataLH edgeDataLW edgeDataLA	Thresholds for height, width and area (in inches) for every side of the check (top/bottom/left/right). Use the function *setEdgeSameToAllSides* to set the same height, width and area for all corners, or use *SetEdgeAll* to set a different threshold for each corner.	0.8f 0.8f 0.3f 0.8f 0.8f 0.3f 0.8f 0.8f 0.3f 0.8f 0.8f 0.3f

Parameter	Description	Default value
imagesizeFrontMin	The minimum file size for the TIFF image for the front side.	0.5f
imagesizeFrontMax	The maximum file size for the TIFF image for the front side.	200f
imagesizeBackMin	The minimum file size for the TIFF image for the back side.	0.5f
imagesizeBackMax	The maximum file size for the TIFF image for the back side.	200f
horizontalStreakSumOfBlackPixels	The minimum number of black pixels required to determine if the line is black (check front).	30
horizontalStreakLineWidth	The minimum width of the black line to detect (check front).	18
horizontalStreakNumLines	The minimum number of black lines for the horizontal streaks alert (check front).	4
carbonStripSumOfBlackPixels	The minimum number of black pixels required to determine if the line is black (check back).	25
carbonStripLineWidth	The minimum width of the black line to detect (check back).	12
carbonStripNumLines	The minimum number of black lines for the horizontal streaks alert (check back).	1
piggyBackMaxWidth	Maximum width threshold between two checks that overlap each other.	10
piggyBackMaxHeight	Maximum height threshold between two checks that overlap each other.	
piggyBackMaxAR	Top and bottom Location threshold between two checks that overlap each other.	20
maxImageDimensionsHeight		

Parameter	Description	Default value
maxImageDimensionsWidth		

Here is an example of how to initialize *IQASettingsIntent* and add it to the capture parameters:

```
checkCaptureParams input = (checkCaptureParams)
        CaptureIntent.getCaptureParams(TISDocumentType.CHECK);
IQASettingsIntent iqaSettings = new IQASettingsIntent();
iqaSettings.cornerDataFrontTLH = 1.0f;
iqaSettings.cornerDataFrontTLW = 1.0f;
input.IQASettings = iqaSettings; (can be found in sample SDK app).
```

To initialize default IQA parameters, use:

```
check 21
iqaSettings = new IQASettingsIntent();
iqaSettings = iqaSettings.getIQASettingsDefault();
check 51:
IQASettingsIntent iqaSettings = new IQASettingsIntent();
iqaSettings = iqaSettings.getIQAStandart51Defaults();
```

For debug purposes only

There are four parameters that can configure the camera with values different from the mobiFLOW defaults. This option is for debugging only and must be set to zero when not debugging.

The SDK will choose the video and still camera resolution that best fits the devices that were tested. For problematic devices, you can choose the resolution manually by changing these values. It is not recommended to change this without first consulting TIS.

These values are in the *integer.xml* file:

<integer name="videoWidthResolution">0</integer> - change video width resolution

<integer name="videoHeightResolution">0</integer> - change video height resolution

<integer name="stillWidthResolution">0</integer> - change still width resolution

<integer name="stillHeightResolution">0</integer> - change still height resolution

Note: Change these values only for debugging purposes.

Add Device Exceptions

To work with Camera API 2

To add a specific device to work with CameraAPI2, you must add an entry to the *arrays.xml* file under *exception_devices_use_cameraAPI2*.

The SDK will calculate the video resolution that best matches the screen aspect ratio. To suppress this calculation, you can disable *calculateVideoToScreenAspectRatio* by deleting the value *All* and upplay the AR calculation per device.

To add a device to exception list

Follow the structure below:

The format has to be Build.MANUFACTURER + <space> + Build.MODEL device brand, space, device model.

For example: <item>Samsung SM-G900V</item>

> **Important:** If you want to make this change other than for debug purposes, consult with the TIS Mobile Team first.

Document types

The available document types are: CHECK, PAYMENT, FULL_PAGE CUSTOM, CARD and PASSPORT.

All the document types below will be initialized with the default parameters that fit the relevant document type.

Full Page capture session parameters

```
FullPageCaptureParams input = (FullPageCaptureParams)
        CaptureIntent.getCaptureParams(TISDocumentType.FULL_PAGE);
```

Custom capture parameters

```
customCaptureParams input = (customCaptureParams)
        CaptureIntent.getCaptureParams(TISDocumentType.CUSTOM);
```

Payment capture parameters

```
paymentCaptureParams input = (paymentCaptureParams)
        CaptureIntent.getCaptureParams(TISDocumentType.PAYMENT);
```

Check capture session parameters

```
checkCaptureParams input = (checkCaptureParams)
        CaptureIntent.getCaptureParams(TISDocumentType.CHECK);
```

Passport capture session parameters

```
passportParams input = (passportParams)
        CaptureIntent.getCaptureParams(TISDocumentType.PASSPORT);
```

Card capture session parameters

```
CardParamsinput = (CardParams)
        CaptureIntent.getCaptureParams(TISDocumentType.CARD);
```

All the default parameters for each class can be modified in the calling application by accessing each parameter in the *params* variable (in this example, *input*).

Once all are initialized, you call *CaptureDocument* to start the camera session:

```
CaptureIntent.captureDocument(input);
```

Video processing guidelines

The check capture and MICR recognition process is done directly from the video feed on supported devices, to achieve a better user experience and maximize performance. The video feed is supported on devices with at least 1980*1080. On other devices, the library starts in stills mode, even when video mode option is enabled. The MICR recognition on these devices is done only after the still picture was taken.

To avoid a long session on problematic checks, the maximum taken frame failure in video mode can also be configured with *maxVideoFramesToCapture* parameters. After X failures on a video frame, the Library will switch to still mode.

The resources should be also imported to the library *res* folder.

Force still capture

There is an option to add exceptional devices that will capture in still even if they support video capture (some devices capture better in still and have issues in video capture). This will enforce the devices on the list to capture in still when *videoFeedProcessing* is set to TRUE.

To add exceptional devices, add an entry to *exception_devices_name_stills_only* in the *arrays.xml*.

To add a device to the exception list, follow the structure below:

The format must be Build.MANUFACTURER + <space> + Build.MODEL device brand, space, device model.

For example: <item>Samsung SM-G900V</item>

Capture messages and errors

To capture messages from the library, the calling app activity should register with *CameraController.registerListener(this)* and implement the *TISmobiFLOWMessages* interface.

You must import *com.topimagesystems.controllers.imageanalyze.CameraController.TISmobiFLOWMessages*.

Three public methods will be generated automatically after implementing the listener.

Session events

Session events like *OCRResult*, *CaptureBack*, and *MultiCapture* will be received here:

```
public void onmobiFLOWGeneralMessageReceived (TISFlowOutputMessages message, Object[] extraData, Context context)
```

TISFlowGeneralMessages possible values are:

Value	Description
CAPTURE_BACK	Fires when the front capture ended before continuing to capture the back side, when both front and back are captured in the same session.
MULTI_CAPTURE	In video mode only, fires after every image capture in a multi-capture session.
CHECK_OCR_RESULT	In video mode only, fires when the document type is Check and the OCRType is E13B or CMC7 after recognition is done.
PAN_CARD_OCR_RESULT	In video mode only, fires when document type is Card and the OCRType is PAN after recognition is done.
PASSPORT_OCR_RESULT	In video mode only, fires when document type is Passport and OCRType is MRZ after recognition is done.
ID_CARD_OCR_RESULT	In video mode only, fires when the document type is Card and the OCRType is MRZ after recognition is done.
BACK_PRESSED	Fires when the Back button was pressed. This event must return a value when fired. It can use either CONTINUE_CURRENT_SESSION to cancel the back default behavior, or CONTINUE_MOBI_FLOW to apply the default back press action.
CREDIT_CARD_OCR_RESULT	In video mode only, fires when the document type is Card and the OCRType is CREDIT after recognition is done.

extraData – This object contains the OCR results that are returned from mobiFLOW and described above.

Validate the OCR Result

When one of the OCR results messages is fired, you can validate the OCR results here. See Handle session continuation for information on how to confirm or reject the OCR result.

Error messages

Error messages from the Library will be received here:

```
public void onmobiFLOWErrorMessageReceived (TISFlowErrorMessage error, Object[] extraData, Context context)
```

TISFlowErrorMessage available errors codes are:

enum TISFlowErrorCode {

ERROR_GENERAL_FAIL (R.string.TISErrorImageGeneral),

ERROR_OCR_READING (R.strings.TISFlowErrorReadingOCRMessage),

ERROR_NO_VALID_BOUNDING_BOX (R.string.TISFlowErrorNoValidBoundingBox),

ERROR_IQA_CORNER_DATA (R.string.TISFlowErrorIQACornerData),

ERROR_IQA_EDGE_DATA (R.string.TISFlowErrorIQAEdgeData),

ERROR_IQA_SKEW (R.string. TISFlowErrorIQASkew),

ERROR_IQA_DARKNESS (R.string.TISFlowErrorIQADarkness),

ERROR_IQA_NUM_SPOTS (R.string.TISFlowErrorIQANumSpots),

ERROR_IQA_HORIZONTAL_STREAK(R.string.TISFlowErrorHorizontalStreaks),

ERROR_IQA_CARBON_STRIP(R.string.TISFlowErrorCarbonStrip),

ERROR_IQA_PIGGY_BACK(R.string. TISFlowErrorPiggyBack),

ERROR_IQA_IMAGE_DIMENTIONS(R.string. TISFlowErrorMinImageDimensions),

ERROR_BLUR_DETECTED (R.string.TISErrorBlurFail),

ERROR_MICR_LENGHT (R.strings.TISFlowErrorReadingMessage),

ERROR_MICR_INTERUPPTED (R.strings.TISFlowMicrInterrupted),

ERROR_MICR_ON_BACK (R.strings.TISFlowWarningMICRDetectedOnCheckBack),

UNSUPPORTED_CAMERA,

UNSUPPORTED_AUTO_FOCUS,

UNSUPPORTED_CPU

}

The order in which the validations run is different when using stills mode and video mode, and so are the messages that are used. The following table shows the order of the validations and their application per document type and capture mode:

Validation description	Validation error code (enum)	Error message name	Message on video feed processing	Message on stills
Blur Detection**	ERROR_ BLUR_ DETECTED	TISErrorBlurFail	NO*	YES
Look For Document Rectangle	ERROR_NO_ VALID_ BOUNDING_ BOX	TISFlowErrorNoValid BoundingBox	NO*	YES
User is capturing the front side instead the of back side of the check***	ERROR_ MICR_ON_ BACK	TISFlowWarningMICR DetectedOnCheckBack	YES	YES
OCR Validation	ERROR_OCR_ READING	TISFlowErrorReading OCRMessage	NO	YES
MICR Length Validation***	ERROR_ MICR_ LENGHT	TISFlowErrorReadingMessage	YES	YES
MICR Line Interruption By Signature.CMC7 Only***	ERROR_ MICR_ INTERUPPTED	TISFlowMicrInterrupted	YES	YES
IQA Folded Corner***	ERROR_IQA_ CORNER_ DATA	TISFlowErrorIQA CornerData	YES	YES
IQA Folded Edge***	ERROR_IQA_ EDGE_DATA	TISFlowErrorIQA EdgeData	YES	YES
IQA Skew***	ERROR_IQA_ SKEW	TISFlowErrorIQASkew	YES	YES

Validation description	Validation error code (enum)	Error message name	Message on video feed processing	Message on stills
IQA Darkness***	ERROR_IQA_DARKNESS	TISFlowErrorIQADarkness	YES	YES
IQA Number of Spots***	ERROR_IQA_NUM_SPOTS	TISFlowErrorIQANumSpots	YES	YES
IQA Horizontal Streaks***	ERROR_IQA_HORIZONTAL_STREAKS	TISFlowErrorHorizontalStreaks	YES	YES
IQA Carbon Strip***	ERROR_IQA_CARBON_STRIP	ERROR_IQA_CARBON_STRIP	YES	YES
IQA Piggy Back***	ERROR_IQA_PIGGY_BACK		YES	YES

* When no message is thrown, mobiFLOW will proceed to process the next frame.

** Enabled on documents without OCR.

*** Checks only.

Note: IQA validations are performed only for Checks and on B&W images.

UI events

This function is fired when any of the UI changes listed below happen:

```
public void on MobiFlowUIEventMessageReceived(TISFlowUIMessages message,ViewGroup cameraOverlayView)
```

Then, simply using *cameraOverlayView.findViewById*, you can inflate your additional UI element and add functionality to it.

The *message* parameter indicates which UI event is currently occurring in the camera controller.

TISFlowUIMessages optional values are:

Value	Description
BACK_PRESSED	Fires when the back button is pressed

Value	Description
BEFORE_PROCESSING	Fires before the processing view animation starts
AFTER_PROCESSING	Fires before the processing view animation finishes (library finished processing)
INIT_LAYOUT	Fires when the screen is refreshed
HINT_CHANGED	Fires when the capture hint is changed
INSTRUCTION_CHANGED	Fires when the capture instruction is changed

Accessibility

Using HINT_CHANGED and INSTRUCTION_CHANGED messages, you can control the accessibility of the controls and change them at runtime. See the code sample in the Handle session continuation section.

Handle session continuation

This section is relevant only for general and error message handling. It explains how to instruct mobiFLOW to continue the session after a message was handled.

The calling app can decide at every step how mobiFLOW will proceed with the enum *TISFlowInputMessages*.

> **Note:** You must return a message to mobiFLOW, so if you do not have any specific request, use the default: CONTINUE_MOBI_FLOW.

First, get the listener with:

```
returnMessage = CameraController.getManagerListener();
```

TISFlowInputMessages possible outgoing values are:

Value	Description
CONTINUE_MOBI_FLOW	Indicates to mobiFLOW to continue the normal flow. mobiFLOW alerts will be displayed. `returnMessage.onMessageReturn (TISFlowActionCallback.CONTINUE_MOBI_FLOW);`

Value	Description
CONTINUE_MOBI_FLOW_CUSTOM_UI	Indicates to mobiFLOW to continue the normal flow, but without mobiFLOW alerts. The calling app will get the screen context and add UI elements to the camera overlay to be displayed, instead of showing the mobiFLOW default alerts.
OCR_RESULT_FAILED	Indicates to mobiFLOW that OCR validation has failed. mobiFLOW will try to run OCR on the next frame.
OCR_RESULT_OK	Indicates to mobiFLOW that OCR is OK and it can proceed with the flow to process the image.
CONTINUE_CURRENT_SESSION	Relevant for BACK_PRESSED event only. It will not close the camera activity on BACK_PRESSED. The calling application can perform some custom actions here.
CANCEL_SESSION	Indicates to mobiFLOW to stop the current session. To fetch the image if it was successfully processed, you will be able to do so on *onActivityResult* in the case RESULT_CLOSE_SESSION. In code: `returnMessage.onMessageReturn (TISFlowActionCallback.CANCEL_SESSION);`

Code sample

```
@Override
public void onmobiFLOWGeneralMessageReceived (TISFlowGeneralMessages message, Object[] extraData,
Context
context) {
        // get messages from the library.
```

```
returnMessage = CameraController.getManagerListener();
switch (message) {
        case CAPTURE_BACK:
        if (errorMessageReceived) { // if got error on front image don't
        // proceed to capture back, close
        // session with image result.
        returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
        } else {
        returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
        }
        break;

case BACK_PRESSED:
        returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
        break;

case PAN_CARD_OCR_RESULT:
        String[] ocrData = (String[]) extraData;
        boolean panValidation = OcrValidationUtils.validationPanCard((String) extraData[1]);
        if (panValidation) {
        returnMessage.onMessageReturn(TISFlowInputMessages.OCR_RESULT_OK);
        } else {
        returnMessage.onMessageReturn(TISFlowInputMessages.OCR_RESULT_FAILED);
        }
        break;

case ID_CARD_OCR_RESULT:
        returnMessage.onMessageReturn(TISFlowInputMessages.OCR_RESULT_OK);
        break;
        case CHECK_OCR_RESULT:
        String[] ocrCheckData = (String[]) extraData;
        returnMessage.onMessageReturn(TISFlowInputMessages.OCR_RESULT_OK);
        break;

case PASSPORT_OCR_RESULT:
        boolean passportValidation = OcrValidationUtils.validatePassport((String) extraData[0],
        Integer.valueOf((String) (extraData[1])));
        // get passport result as key value hash map
        HashMap<String, String> passportResultParsed = (HashMap<String, String>) extraData[4];
        if (passportValidation) {
```

```
                returnMessage.onMessageReturn(TISFlowInputMessages.OCR_RESULT_OK);
                } else {
                returnMessage.onMessageReturn(TISFlowInputMessages.OCR_RESULT_FAILED);
                }
                break;

        case MULTI_CAPTURE:
                // get the image here!!! image data can be taken
                if (extraData != null) {
                // save multi capture images
                if (isScanFrontOnly) {
                saveMultiCaptureFrontSideImages((String) extraData[0]);
                } else if (isScanBackOnly) {
                saveMultiCapturBackSideImages((String) extraData[0]);
                } else {
                saveMultiCaptureFrontSideImages((String) extraData[0]);
                saveMultiCapturBackSideImages((String) extraData[1]);
                }
                }
                saveDataForMulticapture((String[]) extraData);
                returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
                break;

        default: // must use default value here!
                returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
                break;
        }
}

@Override
public void onmobiFLOWErrorMessageReceived (TISFlowErrorMessage error, Object[] extraData, Context context) {
        // get Error messages from the library.
        returnMessage = CameraController.getManagerListener();
        switch (error) {
        case ERROR_OCR_READING:
                returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
                errorMessageReceived = true;
                break;
        case ERROR_IMAGE_CONTRAST:
                returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
```

```
            errorMessageReceived = true;
            break;
        case ERROR_MICR_LENGHT:
            returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
            errorMessageReceived = true;
            break;
        case ERROR_NO_VALID_BOUNDING_BOX:
            returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
            break;
        case ERROR_BLUR_DETECTED:
            returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
            break;
        case ERROR_IQA_DARKNESS:
            returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
            break;
        case ERROR_IQA_CORNER_DATA:
            returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
            break;
        case ERROR_IQA_EDGE_DATA:
            returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
            break;
        case ERROR_IQA_SKEW:
            returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
            break;
        case ERROR_IQA_NUM_SPOTS:
            returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
            break;
        case ERROR_MICR_INTERUPPTED:
            returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
            break;
        case ERROR_MICR_ON_BACK:
            returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
            break;
        case UNSUPPORTED_CAMERA:
            returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
            break;
        case UNSPORTTED_CPU:
            returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
            break;
        default:
```

```
                returnMessage.onMessageReturn(TISFlowInputMessages.CONTINUE_MOBI_FLOW);
                break;
        }
        return;
}

// For accessibility support
private TextView hintTextView;
private TextView instructionTextView;

@Override
public void onmobiFLOWUIEventMessageReceived (TISFlowUIMessages message, ViewGroup cameraOverlayView)
{
        // get UI messages from the library.
        returnMessage = CameraController.getManagerListener();
        switch (message) {
        case INIT_LAYOUT:
                if (AccessibilityUtils.isAccessibilityEnabled(this)) {
                if (isDynamicCapture)
                hintTextView = (TextView) cameraOverlayView.findViewById(R.id.DynamicTxtIndicator);
                else if (isCustomView) {
                hintTextView = (TextView) cameraOverlayView.findViewById(R.id.customTxtIndicator);
                instructionTextView = (TextView)
                cameraOverlayView.findViewById(R.id.customTxtCapture);
                } else {
                hintTextView = (TextView) cameraOverlayView.findViewById(R.id.txtIndicator);
                instructionTextView = (TextView) cameraOverlayView.findViewById(R.id.txtCapture);
                }
        }
        break;
        case AFTER_PROCESSING:
                break;
        case BEFORE_PROCESSING:
                break;
        case HINT_CHANGED:
                if (AccessibilityUtils.isAccessibilityEnabled(this)) {
                if (hintTextView != null) {
                String hintText = hintTextView.getText().toString();
                if (hintText.equalsIgnoreCase(getResources().getString(R.string.TISFlowIndicatorAlignFlat))
```

```
            hintTextView.setContentDescription(getResources().getString
            (R.string.TISFlowIndicatorAlignFlatDescription));
            }
        }
        break;
        case `:
        if (AccessibilityUtils.isAccessibilityEnabled(this)) {
        if (instructionTextView != null)
            instructionTextView.setContentDescription("instruction: " + instructionTextView.getText());
        }
        break;
        default:
        break;
        }
}
```

See the more detailed sample code in the Showcase app in the *ShowCaseActivity*.

Handle the session results

After *CaptureIntent* has finished and *onActivityResult()* function is called. The object that is transferred back to the original calling function is *SessionResultParams* (see example later on this section).

Request code CaptureIntent.MOBI_FLOW_REQUEST_CODE - contains the following: RESULT_OK and RESULT_CANCELLED.

Get the images from the library

By default, the images are not saved to the device (to do so, call *saveImagesToDevice()*).

All the images are stored in the device memory and can be retrieved from:

- SessionResultParams.tiffFront;
- SessionResultParams.jpegBWFront;
- SessionResultParams.grayscaleFront;
- SessionResultParams.colorFront;
- SessionResultParams.originalFront;
- SessionResultParams.tiffBack;
- SessionResultParams.jpegBWBack;
- SessionResultParams.grayscaleBack;
- SessionResultParams.colorBack;
- SessionResultParams.originalBack;

The sample (later in the section) demonstrates how to retrieve the result from the library. It uses the optional *saveImageToDevice* function which is given as well in the showcase.

For Checks, when using split capture for front and back, or for other document types, if you want to scale the back side the same as the front side, use this function to retrieve it and to set the *frontImageSize* for the back side capture.

Parameter	Description
getFrontImageRectArray()	The array contains the height and width of the image the front image that was captured.

For document type Card only with OCR PAN

To get Card OCR results as an array, call the method *OcrValidationUtils.parsePanCardResult (currentSessionResult.getOcrParams()[1]);*.

Method	Description
parsePanCardResult	An array with the card's results. The array size changes dynamically according to the number of results found, while the order of the results are by the x,y location of each field, from top to bottom and left to right.

For document type Card only with OCR MRZ

To get the Card OCR result as a HashMap, call the method *OcrValidationUtils.parseIDCardResult (ocrResult.ocrResultWithDelimiter)*:

```
String[] ocrData = currentSessionResult.getOcrParams();
OCRResult ocrResult = new OCRResult(ocrData);
HashMap<String,String> parsedData =
OcrValidationUtils.parseIDCardResult(ocrResult.ocrResultWithDelimiter);
```

Method	Description
parseIDCardResult	Returns a HashMap with the card's results. With the following keys: ■ kTISCard_Type ■ kTISCard_IssuingCountry ■ kTISCard_DocumentNumber ■ kTISCard_DateOfBirth ■ kTISCard_Sex ■ kTISCard_ExpirationDate ■ kTISCard_Nationality ■ kTISCard_Surname ■ kTISCard_FirstName ■ kTISCard_MiddleName

For document type Passport

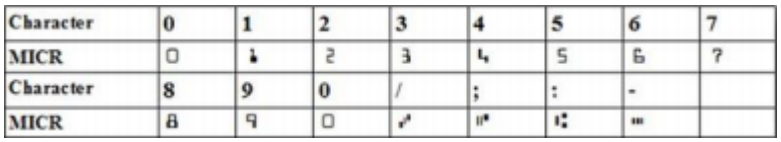
To parse the result of the *onActivityResult* method to a readable HashMap, you can use the method *OcrValidationUtils.parsePassportResult(currentSessionResult.getOcrParams()[1]);*.

The following properties with the results:

Method	Description
parsePassportResult	Dictionary with passport's results. The dictionary contains the following keys: ■ kTISPassport_Type; ■ kTISPassport_IssuingCountry; ■ kTISPassport_Surname; ■ kTISPassport_FirstName; ■ kTISPassport_PassportNumber; ■ kTISPassport_Nationality; ■ kTISPassport_DateOfBirth; ■ kTISPassport_Sex; ■ kTISPassport_ExpirationDate; ■ kTISPassport_PersonalNumber;

For document type Check only

The following parameters are available:

Parameter	Description
getOcrParams(String[])	Session OCR result.
getOcrParams (SessionResultParams.DIGITAL_ ROW_LENGTH)	The MICR length.
getOcrParams (SessionResultParams.OCR_ RESULT_WITH_DELIMETER)	The MICR result formatted in mobiFLOW format (special characters represented by a dash).
getOcrParams (SessionResultParams.OCR_ RAW_RESULT)	The result of every character in the MICR is represented by a number, separated by commas. 0,1,2,3,4,5,6,7,8,9,10,12,11,13 The numbers represent the MICR in the order given in the following table: 
getOcrParams (SessionResultParams.SCORE_ RESULT)	The score for each one of the recognized characters separated by commas, respective to the *rawResult*.

For document type Check only, with CMC7 MICR

Parameter	Description
signatureOverMicrDetected	Will be available if *showErrorSignatureOverCMC7* Was set to FALSE. If it returns TRUE, the SDK found a signature over the MICR line.

Barcode configuration and barcode result

The SDK searches for the barcode for “x” frames (default 13). Only frames with a valid rectangle are calculated. If a barcode was not found for "x" frames, the SDK skips the barcode recognition and only captures the image.

The number of frames can be modified in the variable name *max_barcode_tries* in the *integers.xml* file.

The barcode result can be retrieved from the *SessionResultParams* object using the *getBarcodeResult()* function.

Method	Return
getBarcodeResult()	*BarcodeResult* object that holds the data of the barcode scanning.

The *BarcodeResult* holds the barcode scanning data, and has the following get methods:

Method	Return
isEmpty()	Boolean. TRUE if no barcode data was detected, otherwise FALSE.
getBarcodeTypeFront()	Int representing the barcode type on the front side of the document. If no barcode is detected on the front side, this method returns -1.
getBarcodeDataFront()	String. The parsed data from the barcode located on the front side of the document. If no barcode is detected on the front side, this method will return null.
getBarcodeTypeBack()	Int representing the barcode type on the back side of the document. If no barcode is detected on the back side, this method will return -1.
getBarcodeDataBack()	String. The parsed data from the barcode located on the back side of the document. If no barcode is detected on the back side, this method will return null.

Parse barcode data for Driver's License for US/Canada

To parse the results from the Driver's License barcode when barcode type PDF_417_CODE was detected, use the following method, which will return a dictionary:

OcrValidationUtils.DLBarcodeParser.parseDLBarcode (String barcodeData)

The dictionary contains the following keys:

First Name	Date Of Birth	Audit Information
Middle Name	Sex	Ethnicity
Last Name	Issue Date	Compliance Type
Name Suffix	Restriction Code	First Name Truncation
Address	Endorsement Code	Middle Name Truncation
City	Limited Duration Document Indicator	Last Name Truncation
State	Document Number	Federal Commercial Vehicle Code
Postal Code	Country ID	Customer Specific Control Number

ID Number	Inventor Control Number	WA Specific Endorsements
Class	Card Revision Date	Transaction Types
Height	Temp Visitor	Under 18 Until
Weight	Address	Under 21 Until
Eye Color	Address Additional info	Revision Date
Hair Color	Duplicates	Social Security Number
Expiration Date	Organ Donor	

Exception handling

The Library will catch an exception that is thrown by the camera or any other runtime error.

The Exception details will be sent to the calling app as a string to the *onActivityResult* method.

The result type name is *CameraManagerController.RESULT_LIBRARY_ERROR*.

To get the exception details as a string, use *String errorMessage = data.getStringExtra (CaptureIntent.mobiFLOW_ERROR_DETAILS);*. Then the calling app can decide how to proceed with the error handling and beyond.

The example below shows how to use the error handling:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (CameraManagerController.sessionType == SessionType.TEST) {
        return;
        }

        if (requestCode == CaptureIntent.MOBI_FLOW_REQUEST_CODE) {
        switch (resultCode) {
        case RESULT_OK:
                // parse sesion image result.
                currentSessionResult = CaptureIntent.parseActivityResult(requestCode, resultCode, data);
                if (currentSessionResult.getOcrParams() != null) {
                String ocrResult = currentSessionResult.getOcrParams()[1];

                // for passport you can get results hashMap with this helper method
                //HashMap<String, String> passportResult =
                OcrValidationUtils.parsePassportResult(ocrResult);

        }
        // get the front image size if needed for split capture.
        if (currentSessionResult.getFrontImageRectArray() != null) {
        inputFrontImageArray = currentSessionResult.getFrontImageRectArray();
        }
```

```
// use this to get barcode results.
//currentSessionResult.getBarcodeResult();
// save multiple images on MultiCapture mode.
if (!isMultiCapture) {
// the images are received as byte[] on device memory,
// this is helper method to save the images to device file system. if needed
saveImagesToDevice();
}

break;
// user pressed on cancel button
case RESULT_CANCELED:
        break;
// user pressed cancel from Alert
case CameraManagerController.RESULT_CANCELED_FROM_ALERT
        break;
// get result after session has been closed.
case CameraManagerController.RESULT_CLOSE_SESSION:
// will reach here after
currentSessionResult = CaptureIntent.parseActivityResult(requestCode, resultCode, data);
// decode images, Tiff and jpg

if (currentSessionResult.getFrontImageRectArray() != null) {
inputFrontImageArray = currentSessionResult.getFrontImageRectArray();
}
saveImagesToDevice();
        break;
// handles camera permissions access denied
case CameraManagerController.RESULT_CAMERA_PERMISSION_ACSSES_DENIED:
// on api 23 target apps, mobiFLOW checking for camera permission
// if not approved will reach here.
String permissionErrorMessage =
        data.getStringExtra(CaptureIntent.mobiFLOW_ERROR_DETAILS);
if (permissionErrorMessage != null)
Log.e("error", permissionErrorMessage);
break;

// will get here if error or exception was thrown from the library.
case CameraManagerController.RESULT_LIBRARY_ERROR:
//the exception or error will be received here.
```

```
        String errorMessage = data.getStringExtra(CaptureIntent.mobiFLOW_ERROR_DETAILS);
        if (errorMessage != null)
        Log.e("error", errorMessage);
        // do something with the error
        // got unexpected Error from the Library or exception
        break;

        // invalid License result code
        case CameraManagerController.RESULT_LICENSE_INVALID:
        String licenseErrorMessage =
        data.getStringExtra(CaptureIntent.mobiFLOW_ERROR_DETAILS);
        if (licenseErrorMessage != null)
        Log.e("error", licenseErrorMessage);
        break;
        }
        // helper method if needed to remove the images byte array
        FileUtils.clearMemory();
        }
        // must use this to unregister the listeners
        CameraController.unregisterListener();
}
```

Library flow

Most users will not need to change the camera session capture flow section, and using the Library will be sufficient.

> **Note:** If you need to change a drawable or string resource, you can add a different resource with an identical name to your project and it will override the one used by the Library. Do this with caution.

To change the camera overlay, refer to the *CameraOverlayLayout* class and *mbck_camera_layout.xml*. All the visual overlay elements are held here.

Clear temporary files

The Library does not save the images on the device by default. If you wish to do so, you can use, for example, the code supplied above in the function*saveImagesToDevice();*

When using the *saveImagesToDevice()* function as described, you can use the method *FileUtils.clearHighResImages(this);* to delete the images. If you did not use *saveImagesToDevice()*, but saved the images to a different path, the method *FileUtils.clearHighResImages(this);* will not erase your image files.

All the above is implemented in the sample SDK app as an example.

Security recommendations

The mobile calling application has the responsibility to protect the data returned by the SDK in the downstream flow until the mobile application is closed. The mobile calling application implementing the TIS SDK should adhere to security best practices in order to protect any sensitive data and customer information.

Some of the considerations while implementing and configuring the SDK:

- The mobile calling application is responsible for ensuring that any sensitive data received from the SDK process follows existing processes for safeguarding the data; it is assumed that whatever processes are used for manually entered data would be applied to data extracted from the SDK process.
- Upon closing the SDK, images and/or data are erased from memory. It is the responsibility of the mobile calling application to ensure that the SDK is closed and objects are released upon completion of the SDK process.
- When *debugMode* is set to TRUE, images captured by the SDK are stored on the device (as well as the logs). It is strongly recommended that *IsDebug* always be set to FALSE in the release mode of the application build (Production code), as the images and application data should not be physically stored outside the context of the mobile application. The images and data should only exist in the temporary memory of the mobile application and should not be accessible outside the application context.
- For on-device OCR of Checks (for account funding use cases), it is not recommended to return the check image to the user. Only the extracted data should be returned. To do this, the output settings should be set to **FALSE**:
  - outputGrayscaleImage = False
  - outputOriginalImage = False
  - outputColorImage = False
  - outputBinarizedImage = False
- The Android SDK documentation provides additional code samples (*saveImagesToDevice()*) to save the images on the device after retrieving them from the SDK. Similar code may also be implemented for iOS as well. It is not recommended to save any images/data available from the SDK to the device of the application build, especially in the release mode of the application (Production code). This code should only be used for testing and troubleshooting issues in the development cycle.

Set up a custom capture user interface

This topic explains how to customize the capture user interface.

Change the look and feel

User interface customization is performed by overriding corresponding resources in the library. Any resource residing in the *res* directory structure can be overridden by the user application, such as:

- Values (residing in *values*): string values, styles, dimensions, colors.
- Layouts (residing in *layout*). Primary layout files that are shown to the user by the library intents are:

Layout file name	Description
mbck_camera_layout.xml	Layout of the preview screen as shown to the end user. This layout also includes a processing screen named **processingOverlay**.

- Visual resources (residing in *drawable-...*): PNG files containing images displayed to the user.

Note: The components referred to programmatically in the layout XMLs should exist in your customized version of the XML file, otherwise run-time errors will be generated.

The procedure is essentially the same for all types of resources.

1. Select files you want to customize from the *res* folder of the library and paste them into the respective *res* directories in your project. For value resources, only corresponding lines should be copied. The name of the resource should remain the same.
2. Modify the resource.

After compilation, resources from your project will override the respective resources from the library.

Using this method, you can modify all visual aspects of the application. It is not possible to generate additional screen functionality; your modified resource may use only the component callbacks available on the original screen.

There is an alternative view with all elements, where each custom element starts with a custom prefix. See Additional functionality in custom view for more information.

Change icons and captions

You can keep some controls from the preview screen and change the image files and captions of what is shown on the capture screen. To do this, change the following files in the *res* directory.

File name	Description
logo_watermark.png	The logo of the company
btn_torch.png	The flash icon when not selected
btn_torch_selected.png	The flash icon when selected

You can also change the icons of the indicators and the frame:

File name	Description
ic_boundary_bottom.png	The bottom boundary of the frame when the check is not found
ic_boundary_top.png	The top boundary of the frame when the check is not found
ic_boundary_bottom_v.png	The bottom boundary of the frame when the check is found
ic_boundary_top_v.png	The top boundary of the frame when the check is found
ic_boundary_bottom_rl.png	The bottom-left boundary of the frame when the check is not found
ic_boundary_top_rt.png	The top-right boundary of the frame when the check is not found
ic_boundary_bottom_v_rl.png	The bottom-left boundary of the frame when the check is found
ic_boundary_top_v_rt.png	The top-right boundary of the frame when the check is found
btn_general.9.png	The **Cancel** button background when not selected
btn_general_selected.9.png	The **Cancel** button background when selected

Important: Do not change any other images or files in this folder.

The default messages of the label on top of the camera overlay are *TISFlowFrontCaption* for front and *TISFlowBackCaption* for back.

To change these captions at runtime, you must change the messages dynamically in the *dynamicStrings* hash map keys mentioned above and set new values.

The relevant messages to change are:

String name	Description
TISFlowPleaseCaptureCheckFront, TISFlowPleaseCaptureImage	The label caption at the top of the capture screen when capturing the front side of the check or other document.
TISFlowPleaseCaptureCheck TISFlowPleaseCaptureImageBack	The label caption at the top of the capture screen when capturing the back side of the check or other document. With combined front and back capture, this message is displayed after successful capture of the front .
TISSuccessfulReadingTitle	With combined front and back capture, the title of the message that is displayed after successful capture of the front.
TISFlowPleaseCaptureTheBarcode	The label caption at the top of the capture screen when capturing a barcode.

Change the text indicators

You can also change the text style and background in the XML - *mbck_camera_layout.xml*.

- *@+id/txtIndicator*: Indicators dynamic and static. TextView when not in capture mode.
- *@+id/txtHoldIndicator*: Capture mode TextView.

In the localization files, change the relevant string:

String name	Description
TISFlowIndicatorAlign	Indicator to hold the device flat over the check
TISFlowIndicatorDown	Indicator to move the device towards the bottom of the check
TISFlowIndicatorLeft	Indicator to move the device left
TISFlowIndicatorRight	Indicator to move the device right
TISFlowIndicatorTop	Indicator to move the device towards the top of the check
TISFlowIndicatorRotateLeft	Indicator to rotate the device left (check is at an angle)
TISFlowIndicatorRotateRight	Indicator to rotate the device right (check is at an angle)
TISFlowIndicatorZoomIn	Indicator to move closer to the check (check is too far from the frame)

String name	Description
TISFlowIndicatorZoomOut	Indicator to move away from the check (check is exceeding the frame)
TISFlowIndicatorLight	Indicator to turn on the flash (there is not enough light)
TISFlowIndicatorHold	Indicator to hold the camera when the check is found, before the picture is taken
TISFlowScanBarcode	Indicator to move the device towards the barcode
TISFlowIndicatorScanCreditCard	Indicator to align with the credit card boundaries
TISFlowInvalidRotation	Indicator that phone and document do not have the same orientation

Change countdown image view

In *mbck_camera_layout.xml*, change the following:

The custom view that inherits from *ImageView*. Can be modify android:id="@+id/counter".

You can change the colors and style in the *colors.xml* file:

String name	Description
counter_background	Change the circle background color.
camera_counter_color	Change the text color.
counter_border_color	Change the circle border color. Can get color as a resource (not with *#some color*).
countDownStartValue	When the counter for taking a still image is displayed, it will start from the number set in this parameter. Default value: 2.
countDownStopValue	When the counter for taking a still image is displayed, it will count down to the number set in this parameter. This must be a number lower than *countDownStartValue*. Default value 0.
counterTextSize	Change the counter size.

String name	Description
counterFont	Change the counter font with one of the following values: ■ BOLD_ITALIC ■ BOLD, ITALIC ■ NORMAL

DebugRectView

Optionally, you can allow the Library to draw the rectangle over the check on video processing. The rectangle is the frame that the algorithm finds over the check.

Use *app:showCurrentRectangleFound = true* in *com.topimagesystems.ui.DebugRectView* from the XML file.

Camera overlay color

You can change the color of the outside capture frame with the value *camera_overlay_color* located in *color.xml*.

You can also change dynamic capture colors in the *color.xml* file:

String name	Description
validRectFillColor	Fill color for a valid rectangle frame
validRectStrokeColor	Stroke color for a valid rectangle frame
invalidRectFillColor	Fill color for an invalid rectangle frame
invalidRectStrokeColor	Stroke color for an invalid rectangle frame
grid_line_color	Change the grid color

Rectangle check frame

When the Library goes into capture mode, there is an option to draw a green rectangle over the check frame. You can set this in *com.topimagesystems.ui.CheckBounderiesView* with the value *drawGreenRectangle="true"*.

Custom view

The custom view allows you to switch between the TIS classic view and any other UI you wish.

The view contains 2 XML files: *custom_mbck_camera_layout.xml* and *custom_mbck_camera_manager_layout.xml*.

There are another elements that do not exist in the classic view.

- *bannerTop – LinearLayout*: Allows you to change the color of the top banner.
- *bannerBottom - LinearLayout*: Allows you to change the color of the bottom of the banner.

Guidelines popup

customStaticTxtIndicator: Static text indicator if further details required guideline, the text will disappear after the capture has started.

Additional functionality in Custom view

You can add UI elements with additional functionality to the custom camera layout (the layout XML file *custom_mbck_camera_layout.xml*; see Custom view). An element can be added directly to the XML with a unique ID.

For example, to add a button:

```
<Button
        android:id="@+id/customExtraButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:text="extra button" />
```

For information on adding functionality to your new element, or any other UI elements, see UI events.

Info screen configuration

The default guidelines popup is displayed when the user has difficulties capturing the document after a certain time (customizable).

The popup animates from the top of the screen to the center, in landscape mode.

You can change the text in the *integer.xml* file:

String name	Description
info_screen_text	The text in the instructions screen

You can also change the animation speed, which is set to 600 milliseconds by default:

String name	Description
customLongAnimation	600

The whole view can be modified in the XML layout-land: *info_screen.xml*.

Start a new activity that is not part of the Library

Declare the activity name you want to launch in the *strings.xml* file. The class name should include the full path: *package name* + *class name*, as in the example below.

```
<string name=" TISCallingAppActivityName">com.topimagesystems.ui.InfoScreenActivity</string>
```

The Library will load the activity from the *string.xml* entry at runtime. The trigger to open the activity is the *onClick* method from *mbck_camera_layout.xml* (*android:onClick="startCallingAppActivity"*) with the method *startCallingAppActivity*.

Note: The activity class must be under the mobiFLOW project.

Open a new package name *src* folder and add the activity class to run.

Leveler configuration

To draw the leveler on every sensor movement, you must disable *android:hardwareAccelerated* in the *cameraController* activity.

Copy the following manifest to your calling application.

```
<activity
        android:name="com.topimagesystems.controllers.imageanalyze.CameraController"
        android:configChanges="keyboardHidden|orientation|screenSize"
        android:hardwareAccelerated="true" >
</activity>
```

Configure levelerUI

You can change the leveler parameters and location in the *mbck_camera_layout.xml* file.

You can configure the following three custom images in the XML file:

Image	Description
com.topimagesystems.ui.OneUnitLeveler	One unit leveler inherited from ImageView
com.topimagesystems.ui.TwoUnitsLeveler	Horizontal image and vertical image that inherited from ImageView
com.topimagesystems.ui.ScaleLeveler	Horizontal image and vertical image that inherited from ImageView and contains scale units

For the *TwoUnitsLeveler* and the *ScaleLeveler*, you must specify for both images their location and docking of in the XML file: left/right for portrait leveler and top/bottom for horizontal leveler.

To disable the leveler, add *android:visibility*="*gone*"; the leveler with then not be displayed on the screen.

OneUnitLeveler

The parameters declaration can be found in the *attr.xml* file. You can change the height and the width, as for any Android ImageView, with *layout_width* and *layout_height*.

Leveler parameters

Parameter	Description	Applies to
isFadeOutEnable	True/false fading enable/disable	All leveler types
isDraggingEnable	True/false drugging enable/disable	All leveler types
levelerThickness	Leveler border width (stroke width)	OneUnitLeveler
userColorsInScale	Boolean Customize scale leveler and set its colors – can be set to multi colors if set to true or single color if set to false.	ScaleLeveler
scaleUnitGap	The distance between the leveler's units. The number of units is dynamically calculated accordingly.	ScaleLeveler

TwoUnitsLeveler and Scale Leveler

The leveler height and width are set relative to the capturing frame size (inside red/green boundaries) and determined at run time with the calculation of the screen resolution and the document captured. For this reason, the *layout_width* and *layout_height* will only set the container size, not the image size:

```
levelerLocation: set the leveler location can receive:top,bottom,left,right
levelerLocation="left"
        android:layout_width="100dp"
        android:layout_height="wrap_content"
levelerLocation="top"
        android:layout_width="wrap_content"
        android:layout_height="100dp"
levelerLocation="right"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
levelerLocation="bottom"
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

To change the size of the leveler, change the following:

- *paddingLeftAndRight*: For a vertical image, start "x" dp right/left to the frame size (make the leveler smaller).
- *paddingTopAndBottom*: For a horizontal image start "x" dp from the bottom/top of the screen (makes the leveler smaller).

Captions and messages

Localization files (strings.xml) are available in the resources. You can change the captions and messages used in the Library during the process. Only English is provided by default, but you can support different languages in your application if required.

The relevant messages are:

Message Name	Description
TISFlowPleaseCaptureCheckFront	Caption displayed when capturing the front face of the image/check
TISFlowPleaseCaptureCheckBack	Caption displayed when capturing the back side of the image/check
TISFlowCancel	The **Cancel** button shows in the error messages
TISFlowOK	The OK button shows in the error messages
TISFlowPleaseCaptureBarcode	Instruction for the user to capture the barcode in Static capture, when barcode capture is enabled
TISFlowDigitalRowNotInScope (Checks only)	Message when the digital row is not within the length in the settings
TISFlowCancelCaptureBotton	The **Cancel** button caption shows on the capture screen
TISFlowPreparingForServerLocatingBoundaries	Instruction message displayed for notification *OCRNotificationStatusLocatingBounderies*

Message Name	Description
TISFlowPreparingForServerBinarizing	Instruction message displayed for notification *OCRNotificationStatusImageBinarazing*
TISFlowPreparingForServerCropping	Instruction message displayed for notification *OCRNotificationStatusImageCropping*
TISFlowErrorReading	Title for all error messages
TISFlowErrorReadingCheck	Message when mobiFLOW failed to read the digital row; recapture of the front is necessary
TISFlowErrorReadingImageContrast	Message when there are contrast issues in detecting colors for the digital row reading
TISFlowErrorReadingImageGeneral TISFlowErrorReadingCheckGeneral	General message about failure to validate the image, issued if a more specific message does not apply
TISFlowErrorNoValidBoundingBox	Message when case the bounding box of the image was not detected by the Library
TISFlowErrorIQACornerData	Message when one of the corners of the check is missing and over the accepted threshold
TISFlowErrorIQAEdgeData	Message when one of the edges of the check is missing and over the accepted threshold
TISFlowErrorIQASkew	Message when the check is skewed over the accepted threshold
TISFlowErrorIQADarkness	Message when the image is too dark over the accepted threshold
TISFlowErrorIQANumSpots	Message when the image has too much noise and the number of spots per square inch exceeds the accepted threshold
TISFlowErrorFileTooSmall	Message when the file generated by the Library is too small, below the minimum accepted threshold

Message Name	Description
TISFlowErrorMinImageDimensions	Message when the image is not within the dimensions or aspect ratio that is expected
TISFlowErrorUnknown	Message about IQA validation failure, issued if a more specific message does not apply General message about failure to validate the image, issued if a more specific message does not apply
TISErrorBlurFail	Message when the image is detected as blurry
TISFlowWarningMICRDetectedOnCheckBack (Checks only)	Message when the MICR was detected while the user tried to capture the back of the check (meaning they were capturing the front of the check instead of the back)
TISFlowWarningMicrInterrupted (Checks only)	Message when the recognition of the MICR detects that there is something interrupting the MICR recognition, such as stains or the signature
TISFlowMultiCaptureTitle	Title of the message for multi-capture
TISMultiCaptureShouldContinueCapture	Message to check whether the user wants to capture another document
TISFlowFinish	The caption on the button to finish multi-capture
TISFlowCapture	The caption on the button to continue and capture another document
TISFlowCancel	The caption of the **Cancel** button on alerts

Reporting issues

To report issues to TIS, you must reproduce the issue on the mobiFLOW Showcase app, setting debug mode to ON.

When debug mode is ON, images and logs will be saved on the device for debugging purposes. These images and logs can be sent to the TIS Support Team to enable them to investigate any issues or bugs that you may encounter. In debug mode, every image that was captured is saved, even if you received an error message after the capture.

To be able to access these images and logs, you need an app on the device that is able to explore the file system. An example of such an app is ES File Explorer, which can be downloaded from the Google Play Store.

The images will be saved on the device under the root folder in a folder named *.mobiflow* and the log file will be saved in a folder named *.debugmobiflow*. These two folders are hidden, so you will need to go on the app you are using to browse the file system and enable viewing of hidden folders.

Images - <root>/.mobiflow

Log - <root>/.debugmobiflow/log.txt

There is only one log file, and it will grow with every capture, so it is important to delete it before logging something that you want to report. Make sure the log contains only the data from the relevant capture you had issues with.

For every capture, all five images for the front and five for the back will be saved, depending on which images you decided to output (see Handle the session results).

When reporting an issue, please send the following:

- The log file containing only the issue you are reporting.
- All relevant images regarding the issue.
- A detailed description of the issue and step-by-step instructions on how to reproduce.
- Information about the device or devices and the operating system of the device relevant to the issue.
- Information about the Showcase or SDK version relevant to the issue.
- The configuration of all the parameters in the Showcase or SDK where the issue occurs.

If the issue is related to difficulties in capture, and you were not able to capture the document, you can take a picture of the document with your native camera app on the device and send it to the Support Team instead. It would also be very helpful if you can scan the document on a proper scanner and send a copy that the Support Team can print and test themselves.

Guidelines for successful capture

To ensure successful, optimal capture from the mobiFLOW library, you should include the following guidelines in your application's instructions, which should be followed before the user starts the capture process. These guidelines are not mandatory, and a document can still be captured even if the guidelines are not followed, but following them will ensure optimal capture and the best result.

Contrast

The document should be positioned on a background with a different color. Strong visual contrast near the document's boundaries is particularly advised. For documents with multiple colors around the boundaries, the document background should be a different color from any color on the document's boundaries.

Background homogeneity

The background should be clean and homogenous. In particular, strong lines on the background that do not belong to the document should be avoided. It is best to have the surface around the document clear of any objects about 6" (15cm) from each side of the document.

Lighting

Strong direct sunlight or artificial lighting on the document is not recommended. In particular, having strong light on one part of the document and shade over another part should be avoided at all times. Such a situation can result in a very poor B&W image of the part that was not in the shade.

Shooting and rotation angles

The phone's camera should be lying as flat as possible relative to the document's surface. Moreover, the in-plane rotation of the camera should be similar to that of the document, that is, the picture should be taken in landscape. The document should be positioned in the center of the screen within the frame that is displayed, as close as possible to the frame sides.

Taking the picture

When the *hold still* wording appears, stand still with the device over the document until the countdown is over and the still picture is taken. Moving or shaking during this process can result in a blurry image, which will lead to a failure or an unclear B&W image.

Checks only: Digital row (MICR)

Make sure that the digital row is clean and the signature is not stretching over it. Ensure that all the digits and special characters are readable.