AU10TIX

ANDROID SDK IMPLEMENTATION GUIDE

SDK version 1.0

LEGAL NOTICE

CONTENTS

GETTING STARTED

This guide explains the core functionality of the SDK, and details the steps required for a successful implementation.

ABOUT THE SDK

The AU10TIX SDK is a single AAR file that offers image gathering and manipulation functionality.

SDK FEATURES

The following table describes the SDK features.

SDK Feature	Description
Scan document	Allows you to scan documents such as passports and driving licenses.
POA	Allows you to scan documents such as water or electric bills as Proof-of-Address.
Scan barcode	Allows you to scan documents to extract barcode data.
Selfie	Allows you to capture a selfie image.
Liveness	Allows you to detect whether the captured face image presents a real live person by way of a camera video session. This feature uses complex algorithms for detection and analysis.
Live face detection animation	The live face animation detects and tracks the found rectangle.
Live document detection animation	The live document animation detects and tracks the corners of a found rectangle.
Quality	Checks the quality of the captured images at the application level to control reflections, sharpness, brightness, etc.
Image manipulations	Allows you to crop and transform (image size and proportions) the captured image to maximize the authentication capabilities on the AU10TIX server with optimal image data.
EXIF data attachment to captured images	Allows you to attach Exchangeable Image File Format data to captured images such as location, platform, etc.
Branding	Allows you to apply your own customized User Interface in accordance to your branding.
Offline capabilities	Allows you to perform all image capturing and manipulations offline apart from the final authentication performed on the AU10TIX server.

SDK SPECIAL CAMERA CAPABILITIES

The following are the special camera capabilities of the SDK:

- Face detection
- Document detection
- Barcode detection
- Auto focus
- Auto exposure
- Quality detection.

PREREQUISITES

Applications targeting Android 5.1 LOLLIPOP_MR1 (API level 22) or higher.

INTEGRATING THE SDK TO YOUR ANDROID PROJECT

Note: Before you begin, make sure you are implementing the latest SDK version.

The following procedures allow you to integrate the SDK into your android project. First you need to import the SDK library into your project. Second, you need to set the SDK library module as a dependency.

To import the AU10TIX SDK library into your project:

1. Go to **File** -> **New** -> **New Module**.
2. Select **Import .JAR/.AAR Package**.
3. Locate and select the AU10TIX SDK AAR file.

To set the AU10TIX SDK library module as a dependency:

1. Go to **File** -> **Project Structure**.
2. Select the main app module.
3. Click the **Dependencies** tab.
4. Click **+** and add a new Module dependency.
5. Select the AU10TIX SDK module.
6. Run a gradle sync to apply the changes.

SDK SIZE AND ARCHITECTURE

The AU10TIX SDK supports all common Application Binary Interface (ABI) types, among which are the MIPS and MIPS64 ABIs, added for backward compatibility.

To reduce the downloaded application size, it is recommended to perform ABI splitting, and upload each ABI-specific APK separately to Google Play.

Once split, the approximate separated ABI size will range between ~4MB to ~6MB, except for x86/x86_64 ABIs which range between ~13MB and ~15MB.

The following is an ABI split example:

```
splits {
    abi {
        enable true
        reset()
        include "armeabi","armeabi-v7a",
        "arm64-v8a","mips","mips64","x86","x86_64"
        universalApk false
    }
}
```

SDK DEPENDENCIES

The AU10TIX SDK depends on Google Play Services.

To implement this dependency:

Add the following to your app module's build.gradle file:

```
dependencies {

    implementation 'com.google.android.gms:play-services-vision:17.0.2'}
```

Add the following to your app manifest file:

```
<Application

Android:...>

<meta-data android:name="com.google.android.gms.version"

    android:value="@integer/google_play_services_version"/>

<meta-data

    android:name="com.google.android.gms.vision.DEPENDENCIES"

    android:value="face" />
```

PERMISSIONS

The AU10TIX SDK uses the device camera and GPS sensor to produce photos containing location parameters as part of the EXIF metadata.

Both Camera and Location permissions must be declared, requested, and granted for the SDK to behave as expected.

The SDK requires the following permissions:

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

Note: Starting from Android Marshmallow (6.0), these permissions must also be requested at runtime.

PROGUARD

If necessary, add the following rules to your Proguard file:

```
-keep class com.senticore.au10tix.** {;}
-dontwarn com.senticore.au10tix.**
-keepattributes Exceptions,InnerClasses,Signature,*Annotation*
```

AU10TIX

INITIALIZATION

To use SDK library components independently, run the following Method - “isReady()” located under the “Common” class.

Notes:

- This procedure is mandatory for all implementations.
- This method loads the internal SDK libraries, and its Boolean result determines whether the device/session has the dependencies required for a Liveness2 Detection scenario.
- If false, the implementing application must fallback to load the Liveness Detection instead.
- This method can be called asynchronously and is independent of any other SDK operation.

```
        public static boolean isReady(this);
```

USER INTERFACE CUSTOMIZATION

The AU10TIX SDK resources can be overridden by the implementing application’s resources. To override a resource value, provide your own resource with the same resource id.

For a detailed description, see *Localization and Resource Customization*.

PURPOSE AND USAGE

The AU10TIX SDK is used to gather user data for server authentication.

DATA GATHERING

The AU10TIX SDK gathers the following types of data:

- **Barcode**: A byte[ ] representing the raw barcode data.
- **Face**: An image of the user, cropped to center on their face.
- **Document**: A quality image of an identification document, that is cropped, transformed to have a 90° angle, and includes geolocation EXIF data.
- **Liveness**: A set of numerical values used by the server to authenticate the gathered face image.
- **Liveness2**: A set of numerical values used by the server to authenticate the gathered face image, in addition to the Liveness values.

STANDARD IMPLEMENTATION PROCEDURE OVERVIEW

The following implementation steps are consistent for all detection types:

1. Verify camera and location permissions.
2. Build an Au10tixCameraVision object, by defining the type of detection to perform. For further details, see *AU10TIX CAMERA VISION* or *AU10TIX IQ CAMERA VISION*.
3. Register listeners to receive detection result data and events relevant to your implementation.
4. Add it as a view into a containing layout.
5. Start the detection procedure and stop it properly.

After a successful integration and initialization of the SDK, your app module is dependent on the AU10TIX SDK module and is granted the permissions required to perform detections.

ACTIVITY LAYOUT

Have your Activity's content view include a FrameLayout and add the Au10tixCameraVision instance to it once it is built.

LAYOUT CUSTOMIZATION

Note: Customization is not mandatory. Skip this step to use the default Au10TixCameraVision user interface.

To initiate a detection procedure without using the default detection animation, see **showLiveFinder**, described under *AU10TIX CAMERA VISION*.

To customize the layout, Au10tixCameraVision requires the following ImageButtons as layout elements in the Activity's content view:

- **"@+id/toggle"**: Front or back camera feed selection.
- **"@+id/back"**: Back navigation.
- **"@+id/capture"**: Photo capture.

In addition, for liveness detection:

- **"@+id/timer"**: For remaining liveness test time indication. Relevant to Liveness implementations only, but should be implemented to allow a clean Liveness2 detection fallback to Liveness detection.

The common root layout containing these views must be given an id for the Au10tixCameraVision's controlView() builder method.

Note: As of version 1.0.2, the recommended practice for SDK implementation is to hide all unnecessary ImageButtons.

STARTING AND STOPPING DETECTIONS

Detections carried by Au10tixCameraVision must be safely started and stopped at the onResume() and onPause() activity lifecycle events.

```
@Override

    public void onResume() {

        super.onResume();

        if (checkLocationAndCameraPermissions())

            buildAu10tixCameraVision()*;

            mAu10tixCameraVision.start();

    }

    @Override

    public void onPause() {

        super.onPause();

        mCameraVision.stop();

        mCameraVision.release();

    }
```

*Examples of what the `buildAu10tixCameraVision()` method could include are described separately per each specific detection setup. See *Barcode Detection*, *Face Detection*, *Document Detection*, *Liveness Detection*, and *Liveness2 Detection* for further details.

BARCODE DETECTION

To gather raw barcode data:

1. Implement OnBackListener, and OnCaptureListener.
2. Declare Au10tixCameraVision as a private field at the class level:

```
public class MainActivity extends Activity implements
Au10tixCameraVision.OnCaptureListener,
Au10tixCameraVision.OnBackListener{

    private Au10tixCameraVision mAu10tixCameraVision;
```

3. At the onResume() lifecycle event, build Au10tixCameraVision using the Au10tixCameraVision builder.

```
@Override
```

```
    protected void onResume {

        super.onResume;

        mAu10tixCameraVision = (new Au10tixCameraVision.Builder(this))
```

4. Set the detection mode value to Au10tixCameraVision.BARCODE_DETECTION_MODE.
5. As barcode detection is performed using the back camera, set the detection mode value to Au10tixCameraVision.CAMERA_FACING_BACK.

```
        .detectionMode(Au10tixCameraVision. BARCODE_DETECTION_MODE)

        .facingMode(Au10tixCameraVision.CAMERA_FACING_BACK)
```

6. Register the Activity to listen to events received from Au10tixCameraVision.

```
        .onBack(this)

        .onCapture(this)
```

7. If using a custom layout, set the layout containing the back, toggle, and capture ImageButtons as the control view.

```
        .controlView(findViewById(R.id.root_container))
```

8. Build Au10tixCameraVision.

```
        .build();
```

9. Add the Au10tixCameraVision to the activity's FrameLayout container, described in the *Activity Layout* section.

```
((FrameLayout) findViewById(R.id.container)).addView(mCameraVision,

new ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,

                ViewGroup.LayoutParams.MATCH_PARENT));
```

10. Override onResume() and onPause() to safely start and stop detections, as described in the *Starting and Stopping Detections* section.
11. At the OnCapture listener implementation, cast the received detection result to BarcodeDetectionResult.

```
@Override

    public void onCapture(DetectionResult detectionResult) {

        if (detectionResult == null)

            return;

        BarcodeDetectionResult result =
        (BarcodeDetectionResult) detectionResult;

//Additional code
```

12. To get the raw barcode data received in the detection result, call the BarcodeDetectionResult getResults method to return a FindBarcode.Result array.

```
FindBarcode.Result[] results = result.getResults();
```

13. Finally, get the raw barcode data as a byte[], The gathered raw data can be converted to a string.

```
for (byte b : findBarcodeResult.getRawdata()) {

      rawData.put((int) b);

     }
```

FACE DETECTION

Note: To ensure a genuine face image, face detection should be performed as part of the liveness or liveness2 detection scenarios. The following procedure should be taken after proper consideration.

To gather a cropped face image:

1. Implement the OnBackListener, and OnCaptureListener.
2. Declare Au10tixCameraVision as a private field at the class level:

```
public class MainActivity extends Activity implements
Au10tixCameraVision.OnCaptureListener,
Au10tixCameraVision.OnBackListener{

    private Au10tixCameraVision mAu10tixCameraVision;
```

3. At the OnResume() lifecycle event, build Au10tixCameraVision using the Au10tixCameraVision builder.

```
@Override

    protected void onResume{

        super.onResume;

        mAu10tixCameraVision = (new Au10tixCameraVision.Builder(this))
```

4. Set the detection mode value to Au10tixCameraVision.FACE_DETECTION_MODE.
5. As face detection is performed using the front camera, set the detection mode value to Au10tixCameraVision.CAMERA_FACING_FRONT.

```
        .detectionMode(Au10tixCameraVision.FACE_DETECTION_MODE)

        .facingMode(Au10tixCameraVision.CAMERA_FACING_FRONT)
```

6. Register the Activity to listen to events received from Au10tixCameraVision.

```
        .onBack(this)

        .onCapture(this)
```

7. If using a custom layout, set the layout containing the back, toggle, and capture ImageButtons as the control view.

```
        .controlView(findViewById(R.id.root_container))
```

8. Build Au10tixCameraVision.

```
        .build();
```

9. Add the Au10tixCameraVision to the activity's FrameLayout container, described in the *Activity Layout* section.

```
((FrameLayout) findViewById(R.id.container)).addView(mCameraVision,

new ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,

                ViewGroup.LayoutParams.MATCH_PARENT));
```

10. Override onResume() and onPause() to safely start and stop detections, as described in the *Starting and Stopping Detections* section.
11. At the OnCapture listener implementation, cast the received detection result to FaceDetectionResult.

```
@Override

    public void onCapture(DetectionResult detectionResult) {

        if (detectionResult == null)

            return;

        FaceDetectionResult result =

         (FaceDetectionResult) detectionResult;

//Additional code
```

12. To crop the face according to the quadrangle coordinates depicting its location, instantiate a new ImageContainer, by passing the image received in the result into its constructor.

```
ImageContainer imageContainer = new ImageContainer(result.getJpegImage());
```

13. Instantiate a new CropFace object, by passing the ImageContainer into its constructor.
14. Use this CropFace instance to generate a cropped image, by passing the result quadrangle coordinates to the crop method.

```
CropFace cropFace = new CropFace(imageContainer);
```

```
ImageContainer resultImageContainer = cropFace.crop(result.getQuadrangle());
```

15. Use the result ImageContainer to get the JpegImage object by passing it to the ImageContainer.getJpegPhoto() static method.

```
JpegImage resultImage = ImageContainer.getJpegPhoto(resultImageContainer);
```

DOCUMENT DETECTION

As document detection images are required to meet a quality standard, create a custom layout for insufficient image quality notifications.

To gather cropped document images:

1. Create a new xml layout with TextView as the root object.
2. Customize the layout to match your theme, and set the TextView id to "@+id/text".

```
<?xml version="1.0" encoding="utf-8"?>

<TextView

    xmlns:android="http://schemas.android.com/apk/res/android"

    android:id="@+id/text"

    android:layout_width="wrap_content"

    android:layout_height="wrap_content"

    android:textColor="@android:color/white"

    android:text="Bad image quality"

    android:textSize="14sp"
```

3. Implement the OnBackListener, and OnCaptureListener.
4. Declare Au10tixCameraVision as a private field at the class level:

```
public class MainActivity extends Activity implements
Au10tixCameraVision.OnCaptureListener,
Au10tixCameraVision.OnBackListener{

    private Au10tixCameraVision mAu10tixCameraVision;
```

5. At the OnResume() lifecycle event, build Au10tixCameraVision using the Au10tixIQCameraVision builder.

```
@Override

    protected void onResume{

        super.onResume;
        mAu10tixCameraVision = (new Au10tixIQCameraVision.Builder(this))
```

6. As the first builder method, set the value of customQualityToast builder method to the image quality layout id.
7. Set the detection mode value to Au10tixCameraVision.DOCUMENT_DETECTION_MODE.
8. As document detection is performed using the back camera, set the detection mode value to Au10tixCameraVision.CAMERA_FACING_BACK.

```
        .customQualityToast(R.layout.view_quality_dialog)

        .detectionMode(Au10tixCameraVision.DOCUMENT_DETECTION_MODE)

        .facingMode(Au10tixCameraVision.CAMERA_FACING_BACK)
```

9. Register the Activity to listen to events received from Au10tixCameraVision.

```
        .onBack(this)

        .onCapture(this)
```

10. If using a custom layout, set the layout containing the back, toggle, and capture ImageButtons as the control view.

```
        .controlView(findViewById(R.id.root_container))
```

11. Build Au10tixCameraVision.

```
        .build();
```

12. Add the Au10tixCameraVision to the Activity's FrameLayout container, described in the *Activity Layout* section.

```
((FrameLayout) findViewById(R.id.container)).addView(mCameraVision,

new ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,

                ViewGroup.LayoutParams.MATCH_PARENT));
```

13. Override onResume() and onPause() to safely start and stop detections, as described in the *Starting and Stopping Detections* section.
14. At the OnCapture listener implementation, cast the received detection result to DocumentDetectionResult.

```
@Override

    public void onCapture(DetectionResult detectionResult) {

        if (detectionResult == null)

            return;

        DocumentDetectionResult result =
          (DocumentDetectionResult) detectionResult;
```

Note: The captured document corners can be presented over the result image, to allow for user confirmation, that the captured corners fit the document's borders. See Quadrangle for further information.

15. To crop and transform the document image received in the detection result according to the quadrangle coordinates, instantiate a new ImageContainer by passing the image received in the result into its constructor.

```
ImageContainer imageContainer = new ImageContainer(result.getJpegImage());
```

16. Instantiate a new Transform object, by passing the ImageContainer into its constructor.

```
Transform transform = new Transform(imageContainer);
```

17. Use this transform instance to generate a Transform.Result, by passing the result quadrangle coordinates to the detection result's tranformByRectangle method.

```
Transform.Result transformResult =
                 transform.transformByRectangle(result.getQuadrangle());
```

18. Use the transform result to instantiate a new ImageContainer, and get the JpegImage object by passing it to the ImageContainer.getJpegPhoto() static method.

```
ImageContainer resultCont = transformResult.getResult();

JpegImage resultImage = ImageContainer.getJpegPhoto(resultCont);
```

LIVENESS DETECTION

AU10TIX SDK provides a liveness test to detect a real person on a live video stream.

To gather liveness parameters and a face image:

1. Implement the OnBackListener, and OnCaptureListener.
2. Declare Au10tixCameraVision as a private field at the class level:

```
public class MainActivity extends Activity implements
Au10tixCameraVision.OnCaptureListener,
Au10tixCameraVision.OnBackListener{

    private Au10tixCameraVision mAu10tixCameraVision;
```

3. At the OnResume() lifecycle event, build Au10tixCameraVision using the Au10tixCameraVision builder.

```
@Override

    protected void onResume{

        super.onResume;

        mAu10tixCameraVision = (new Au10tixCameraVision.Builder(this))
```

4. Set the detection mode value to Au10tixCameraVision.LIVENESS_DETECTION_MODE.
5. As liveness detection is performed using the front camera, set the detection mode value to Au10tixCameraVision.CAMERA_FACING_FRONT.

```
        .detectionMode(Au10tixCameraVision.LIVENESS_DETECTION_MODE)

        .facingMode(Au10tixCameraVision.CAMERA_FACING_FRONT)
```

6. Register the Activity to listen to events received from Au10tixCameraVision.

```
        .onBack(this)

        .onCapture(this)
```

7. If using a custom layout, add a TextView to your layout with a "@+id/ timer " id, and set the layout containing the back, toggle, and capture ImageButtons and timer TextView as the control view.

```
        .controlView(findViewById(R.id.root_container))
```

8. Build Au10tixCameraVision.

```
        .build();
```

9. Add the Au10tixCameraVision to the Activity's FrameLayout container, described in the *ACTIVITY LAYOUT* section.

```
((FrameLayout) findViewById(R.id.container)).addView(mCameraVision,
new ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,

                ViewGroup.LayoutParams.MATCH_PARENT));
```

10. Override onResume() and onPause() to safely start and stop detections, as described in the *STARTING AND STOPPING DETECTIONS* section.
11. At the OnCapture listener implementation, cast the received detection result to LivenessDetectionResults.

```
@Override

    public void onCapture(DetectionResult detectionResult) {

        if (detectionResult == null)

            return;

        LivenessDetectionResults result =

         (LivenessDetectionResults) detectionResult;
```

12. Get the FaceDetectionResult from the LivenessDetectionResults object.

```
FaceDetectionResult faceDetectionResult = result.getFaceDetectionResult();
```

13. To crop the face according to the quadrangle coordinates depicting its location, instantiate a new ImageContainer, by passing the image received in the result into its constructor.

```
ImageContainer faceImageContainer = new
ImageContainer(faceDetectionResult.getJpegImage());
```

14. Instantiate a new CropFace object, by passing the ImageContainer into its constructor.
15. Use this CropFace instance to generate a cropped image, by passing the result quadrangle coordinates to the crop method.

```
CropFace cropFace = new CropFace(faceImageContainer);

ImageContainer resultImageContainer =
cropFace.crop(faceDetectionResult.getQuadrangle());
```

16. Use the result ImageContainer to get the JpegImage object by passing it to the ImageContainer.getJpegPhoto() static method.

```
JpegImage resultImage = ImageContainer.getJpegPhoto(resultImageContainer);
```

17. To get the Liveness parameters:

```
livenessDetectionResult.getLivenessDetectionFinalResult();
```

18. Calculate the average score for each parameter and forward the result.

Note: To avoid memory leaks, always call the release() method on ImageContainer instances once they are no longer required.

LIVENESS2 DETECTION

Note: If for any reason the Liveness2 Detection Mode cannot properly initialize, it will safely fail-over to Liveness Detection Mode. In addition, handle a Liveness detection result scenario to allow fail over.

To gather liveness and liveness2 parameters along with a face image:

1. Implement the OnBackListener, OnCaptureListener, and OnLiveness2SessionEvents, as described in *AU10TIXCAMERAVISION EVENT LISTENERS*.
2. Declare Au10tixCameraVision as a private field at the class level:

```
public class MainActivity extends Activity implements
Au10tixCameraVision.OnCaptureListener,
Au10tixCameraVision.OnBackListener,
Au10tixCameraVision.OnLiveness2SessionEvents{

    private Au10tixCameraVision mAu10tixCameraVision;
```

3. At the OnResume() lifecycle event, build Au10tixCameraVision using the Au10tixCameraVision builder.

```
@Override

    protected void onResume{

        super.onResume;

        mAu10tixCameraVision = (new Au10tixCameraVision.Builder(this))
```

4. Set the detection mode value to Au10tixCameraVision.LIVENESS2_DETECTION_MODE.
5. As gesture detection is performed using the front camera, set the detection mode value to Au10tixCameraVision.CAMERA_FACING_FRONT.

```
        .detectionMode(Au10tixCameraVision.LIVENESS2_DETECTION_MODE)

        .facingMode(Au10tixCameraVision.CAMERA_FACING_FRONT)
```

6. Register the Activity to listen to events received from Au10tixCameraVision.

```
        .onBack(this)

        .onCapture(this)

        .onLiveness2SessionEvents(this)
```

7. Build Au10tixCameraVision.

```
        .build();
```

8. Add the Au10tixCameraVision to the Activity's FrameLayout container, described in the *ACTIVITY LAYOUT* section.

```
((FrameLayout) findViewById(R.id.container)).addView(mCameraVision,

new ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,

                ViewGroup.LayoutParams.MATCH_PARENT));
```

9. Override onResume() and onPause() to safely start and stop detections, as described in the *STARTING AND STOPPING DETECTIONS* section.

10. At the OnCapture listener implementation, cast the received detection result to Liveness2DetectionResults.

    Note: Make sure to handle the Liveness detection result as well, in order to support a smooth Liveness2 failover. See *LIVENESS DETECTION* for further details.

```
@Override

public void onCapture(DetectionResult result) {

    Liveness2DetectionResults liveness2DetectionResults =
     ((Liveness2DetectionResults) result);

    if (liveness2DetectionResults.getSessionResultCode() == null) {

    liveness2DetectionResults
    .setSessionResultCode(Liveness2SessionResultCode
    .Liveness2SessionResultFAIL);

    }
```

The Liveness2 Detection Result is comprised of the following parameters:

- **resultCode**: Enumerated representation of a single gesture challenge result.
- **gestureType**: Enumerated representation of the tested gesture type.
- **detectionTime**: Gesture challenge duration.
- **faceTracked**: Face tracking indication.

11. Get the Liveness2 result data in accordance to your implementation requirements.
12. Get the LivenessDetectionResults from the GestureDetectionResults object.
13. Process the liveness detection results.

```
if (null != liveness2DetectionResults.getLivenessDetectionResults()) {

    jpegImage = liveness2DetectionResults

    .getLivenessDetectionResults()

    .getFaceDetectionResult()
    .getJpegImage();

    quadrangle = liveness2DetectionResults

    .getLivenessDetectionResults()

    .getFaceDetectionResult()

    .getQuadrangle();
```

IMAGE QUALITY EVALUATION

ImageQuality is used to evaluate the quality of a provided document image or of multiple images received from the camera video stream.

A new **ImageQuality** object is instantiated by passing an ImageContainer instance to its constructor.

To instantiate an imageQuality object:

```
ImageQuality iq = new ImageQuality(imageContainer);
```

ImageQuality provides the following method:

- **getQuality**: a method that accepts any of the separate quality test parameters or a combination of such. In addition, a Boolean parameter (isStill) differentiates between two different quality test types: One that runs on a video stream and one that runs on a still image and returns an ImageQuality.Result.

ImageQuality.Result

Image quality test results contain getters that return double values for each of the separate quality tests. These getter methods are:

- **getSharpness**
- **getBrightness**
- **getSaturation**
- **getReflection**
- **getNoise**
- **getContrast**

To produce an ImageQuality.Result for all quality tests on a still image:

```
ImageQuality.Result r = iq.getQuality(ImageQuality.IC_ALL, true);
```

For a combination of quality tests:

```
ImageQuality.Result r = iq.getQuality(ImageQuality.IC_REFLECTION |
ImageQuality.IC_SHARPNESS, true);
```

ImageQuality.Result provides the following method:

- **isUsed**: A method that accepts a predefined quality parameter integer constant and returns a boolean.

```
If(result.isUsed(ImageQuality.IC_SHARPNESS))

{…
```

Used to indicate whether the result was produced by an ImageQuality test that checked for the specified quality parameter. Returns `true` if used.

The following is a complete example producing a result from a DocumentDetectionResult object:

```
final DocumentDetectionResult r = (DocumentDetectionResult) result;

ImageContainer imageContainer =
ImageContainer.getContainerFromJpeg(((DocumentDetectionResult)
result).getJpegImage());

ImageQuality.Result qr = (new
ImageQuality(imageContainer)).getQuality(ImageQuality.IC_ALL );
```

API REFERENCE - MAIN OBJECTS

Following are the main objects required for a standard SDK implementation.

AU10TIX CAMERA VISION

Au10tixCameraVision is the central view used in any standard SDK implementation. It hosts the camera preview and controllers, handles detection sessions, and provides their results. The Au10tixCameraVision is instantiated using the builder pattern. The Au10tixCameraVision can be extended to permit custom implementations.

> **Note**: To avoid memory leaks, the Au10tixCameraVision must be released once it is no longer required.

Au10tixCameraVision offers the following event listeners:

- **OnSwitchCameraListener**: Enables adding additional behavior to the camera toggle action. Implementation is optional.
- **OnBackListener**: Enables setting a behavior for the back ImageButton. Implementation is optional, but since the back ImageButton must be present on the layout xml, view visibility must be set to GONE.
- **OnCaptureListener**: Provides DetectionResults. Implementation is mandatory.
- **Liveness2SessionListener**: Relevant for Liveness2 detection only.
- **OnProcessingListener:** Indicates that the detection data gathering process has initiated and is now being processed (`onCaptureInit()`). Implementation is optional.

For further details, see *AU10TIXCAMERAVISION EVENT LISTENERS*.
Once the listeners are implemented, Au10tixCameraVision can be instantiated via its Builder, according to the following example:

```
mCameraVision = (new Au10tixCameraVision.Builder(this))
         .facingMode(Au10tixCameraVision.CAMERA_FACING_BACK)
         .detectionMode(Au10tixCameraVision.BARCODE_DETECTION_MODE)
         .controlView(mLayout)
         .showLiveFinder(mLayout)
         .onLiveness2SessionEvents(this)
         .onCapture(this)
         .onBack(this)
         .onSwitchCamera(this)
         .onProcessing(this)
         .build();
```

- **facingMode**: Front or Back camera selection. Options are:

  - Au10tixCameraVision.CAMERA_FACING_BACK
  - Au10tixCameraVision. CAMERA_FACING_FRONT

- **detectionMode**: Detection mode selection. Options are:
  - Au10tixCameraVision.NO_DETECTION_MODE
  - Au10tixCameraVision.FACE_DETECTION_MODE
  - Au10tixCameraVision.DOCUMENT_DETECTION_MODE
  - Au10tixCameraVision.BARCODE_DETECTION_MODE
  - Au10tixCameraVision.LIVENESS_DETECTION_MODE
  - Au10tixCameraVision. LIVENESS2_DETECTION_MODE

- **controlView**: User Interface customization. Described in the *LAYOUT CUSTOMIZATION* section.
- **showLiveFinder**: Accepts a boolean to show or hide detection animations.
- **onBack**: Activity that implements the OnBackListener.
- **onSwitchCamera**: Activity that implements the OnSwitchCameraListener. Optional.
- **onLiveness2SessionEvents**: Activity that implements the onLiveness2SessionEvents. Optional.
- **onProcessing**: Activity that implements the OnProcessingListener (`onCaptureInit()`). Optional.
- **onCapture**: Activity that implements the OnCaptureListener.

Once the Au10tixCameraVision is properly instantiated, it is added to an already defined container view, as follows:

```
camera_vision_container.addView(mCameraVision,

new ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,

                    ViewGroup.LayoutParams.MATCH_PARENT));
```

To properly respond to the hosting Activity's lifecycle, override its OnResume and OnPause lifecycle events, and make sure the Au10tixCameraVision instance is started and stopped accordingly.

Au10tixCameraVision provides the following methods:

- **getFacingMode**: Get the camera facing direction. Returns an integer value equals to one of the above listed facing modes.
- **changeDetectionMode**: Switch between detection modes. Accepts one of the above listed detection modes.
- **setTitleText**: Set a title. Relevant only for implementations that do not require layout customization.
- **selectCameraSource**: toggle between front and back camera.
- **stop():** Stops the detection process.
- **release():** Releases the Au10tixCameraVision instance from memory

When extending the Au10tixCameraVision class, override the following methods:

- **createDetector**
- **protected void - init(android.content.Context context)**
- **protected void - initControls(android.view.View view, boolean isCustom)**
- **onBarcodeDetected(FindBarcode.Result[] results)**
- **onChallengeState(Liveness2DetectorResultCode challengeState)**
- **onCurrentFaceQuadrangle(Quadrangle quadrangle)**
- **onDocumentDetected(Detection detection)**
- **onFaceDetected(Detection detection)**
- **onFacePhotoConditionsMet(Liveness2Detector.PhotoListener photoTaken)**
- **onLiveness2Detected(java.util.List<Liveness2Detection> gestureDetections, java.util.List<LivenessDetection> livenessDetections)**
- **onLiveness2SessionResultCodeUpdate(Liveness2SessionResultCode liveness2SessionResultCode)**
- **onLivenessDetected(java.util.List<LivenessDetection> detections)**
- **onNewChallenge(GestureType nextChallenge)**
- **onProcessing()**
- **selectCameraSource(int facing)**
- **setTitleText(java.lang.String text)**
- **showMaskView(boolean toShow)**

AU10TIX IQ CAMERA VISION

Au10tixIQCameraVision is used for document quality assurance.

> **Note**: To avoid memory leaks, the Au10tixIQCameraVision must be released once it is no longer required.

Au10tixIQCameraVision receives ongoing quality detection results. These are used to determine whether it should show or hide the QualityToast error notification. When running an ongoing quality detection using Au10tixCameraVision, frames are tested for brightness, sharpness, and reflection in preset intervals.

Au10tixIQCameraVision extends Au10tixCameraVision, and provides additional builder methods to build Au10tixCameraVision instances with the following:

> **Note:** Setting a custom quality toast is mandatory for the image quality process to take place.

- **customQualityToast**: A custom layout file for presenting a message indicating bad image quality. This can be any layout file containing a TextView with a "@+id/text" id. Mandatory.
- **qualityPeriod**: Sets an other qualityCheck period interval, different from the default of 300L milliseconds (once every 0.3 seconds). Optional.

To build an Au10tixCameraVision using Au10tixIQCameraVision, set its builder methods first, as described in the following example:

```
mCameraVision = (new Au10tixIQCameraVision.Builder(this))

        .customQualityToast(R.layout.view_quality_dialog)

        .qualityPeriod(250L)

        .facingMode(Au10tixCameraVision.CAMERA_FACING_BACK)

        .detectionMode(Au10tixCameraVision.DOCUMENT_DETECTION_MODE)

        .onCapture(this)

        .onBack(this)

        .build();
```

AU10TIXCAMERAVISION EVENT LISTENERS

To handle events received from Au10tixCameraVision, have your Activity implement its listeners.

```
public class MainActivity extends Activity implements
Au10tixCameraVision.OnCaptureListener,
Au10tixCameraVision.OnSwitchCameraListener,
Au10tixCameraVision.OnBackListener,
Au10tixCameraVision.Liveness2SessionListener,
Au10tixCameraVision.OnProcessingListener{
```

These are:

- **OnCaptureListener**: Indicating that a detection process has finished by passing its detection results.
- **OnSwitchCameraListener**: Indicating the user had touched the Au10tixCameraVision toggle button (`onSwitchCamera()`).
- **OnBackListener**: Indicating that the user had touched the Au10tixCameraVision back button.
- **OnProcessingListener**: Indicating that the liveness session is over (`onCaptureInit()`).
- **Liveness2SessionListener**: Provides the following events to indicate different Liveness2 session states:
  - **onChallengeState**: provides *LIVENESS2 DETECTOR RESULT CODE* events.
  - **onLiveness2SessionResult**: provides *LIVENESS2 SESSION RESULT CODE* events.
  - **onGestureChallenge**: provides *GESTURE TYPE* events.

In most cases the OnBack listener would be implemented as follows, to perform the same functionality assigned to the device's back button:

```
@Override

    public void OnBack(){

        OnBackPressed();

    }
```

The OnCapture listener accepts detection results that are cast to the type of the received detectionResult. See the relevant detection section for further information.

```
@Override

    public void onCapture(DetectionResult detectionResult) {

        //Type specific handling

}
```

The Liveness2SessionListener is optional, and relevant for Liveness2Detection only.

Liveness2SessionListener.onChallengeState

The following custom implementation indicates challenge conclusion by responding to the received challenge result code. For a complete list of the provided challenge state events, see *LIVENESS2 DETECTOR RESULT CODE*.

```
    @Override

    public void onChallengeState(Liveness2DetectorResultCode resultCode) {

        switch (resultCode) {

            case Liveness2DetectorResultFAIL:

                challengeFailAudio.start();

                break;

            case Liveness2DetectorResultPASS:

                challengePassAudio.start();

                break;
```

Liveness2SessionListener.onLiveness2SessionResult

The following implementation indicates session conclusion by responding to the most basic received session result codes. For further details, see *LIVENESS2 SESSION RESULT CODE*.

```
@Override
```

```
public void onLiveness2SessionResult(Liveness2SessionResultCode result) {
        switch (result) {
            case Liveness2SessionResultFAIL:
            case Liveness2SessionResultPASS:
            case Liveness2SessionResultLiveness2RequirementsFAIL:
            case Liveness2SessionLivenessDetectionFAIL:
                mCameraVision.stop();
               break;
            case Liveness2SessionFaceDetectionFAIL:
                if (mCameraVision != null) {
                    //Face not found indication
                     CameraVision.stop();
                }
                break;
            case Liveness2SessionResultTimeoutFAIL:
                //Session timeout indication
                mCameraVision.stop();
                break;
        }
    }
```

Liveness2SessionListener.onGestureChallenge

The following implementation presents the next received challenge on a TextView label.
For further details, see *Gesture Type*.

```
@Override
public void onGestureChallenge(GestureType gestureType) {
    switch (gestureType) {
        case GestureTypeFaceForward:
```

```
        setLabelText(getResources()
        .getString(R.string.tix_face_forward));

            break;

        case GestureTypePanLeft:

        setLabelText(getResources()
        .getString(R.string.tix_turn_left));

            break;

        case GestureTypePanRight:

        setLabelText(getResources()
        .getString(R.string.tix_turn_right));

            break;



            case GestureTypeSmile:

            setLabelText(getResources()
            .getString(R.string.tix_smile));

            break;

            case GestureTypeEyesClosed:

            setLabelText(getResources()
            .getString(R.string.tix_close_eyes));

            break;

    }

}
```

DETECTORS

The following are the detectors used by Au10tixCameraVision for detection procedures. All detectors can be used to detect what they are supposed to detect on a single image.

For further details, see Per-Frame Detection Functionality.

> **Note**: To avoid memory leaks, all detectors must be released once no longer required.

- **DocumentDetector** – Returns quadrangle signifying the document borders.
- **FaceDetector** - Returns quadrangle signifying the face area.
- **LivenessDetector** – Returns face area and liveness results. For further details, see Detection Result.

- **Liveness2Detector** - Returns face area, liveness and liveness2 results. For further details, see Detection Result.
- **QualityDetector**, for further details see Image Quality Evaluation

IMAGE CONTAINER

ImageContainer is a class representing an image, used for image manipulation and quality indication.

Note: To avoid memory leaks, the ImageContainer must be released once it is no longer required.

The ImageContainer provides the following methods:

- **release**: Release the ImageContainer instance.

Release example:

```
container.release();
```

- **rotate**: Rotate the image by degrees.

Rotate example:

```
container = ImageContainer.rotate(container, detection.rotation);
```

- **mirror:** Flip the image horizontally.

Mirror example:

```
container = ImageContainer.mirror(container);
```

- **getContainerFromJpeg:** Create a new ImageContainer for a JpegImage.

getContainerFromJpeg example:

```
ImageContainer imageContainer =
ImageContainer.getContainerFromJpeg(result.getJpegImage());
```

- **getJpegPhoto:** Create a new JpegImage from an ImageContainer.

getJpegPhoto example**:**

```
JpegImage jpegImage = ImageContainer.getJpegPhoto(container);
```

- **writeGPS:** Add location data to the EXIF metadata.

writeGPS example**:**

```
imageContainer.writeGPS(mLocation.getLatitude(), mLocation.getLongitude(),

                  mLocation.getAltitude, mLocation.getTime());
```

See ***Error! Reference source not found.****Per-Frame Detection Functionality* for additional examples of u se.

JPEG IMAGE

JpegImage is an object that contains a byte-array representation of an image, its width and height, orientation data (rotation degree) as gathered from the device sensors, and camera facing direction (front or back).

QUADRANGLE

The **Quadrangle** is a four-sided polygon. It contains four Dot objects, each representing a vertex.

Used to hold the coordinates of a document or face detection result in an image.

Required for image cropping and transformation.

IQ SUFFICIENCY CHECK

The **IQSufficiencyCheck** is a helper class containing predefined double typed constants, that determine minimum and maximum values for each image quality test. These are used to verify that the values received are within the legal range expected by our servers, and that the ID document photo meets quality requirements.

IQSufficiencyCheck provides the following methods:

- **docID**: A static method that accepts an ImageQuality.Result instance, and returns a boolean as a pass or fail quality test result.

DETECTION RESULT

All detection results extend the DetectionResult abstract class. This class contains information regarding the detection result type.

Detection results are accepted by the Au10tixCameraVision.OnCaptureListener OnCapture method implementation.

Once accepted, a DetectionResult is cast into its respective type class, for further processing.

The available result types are:

FaceDetectionResult: Contains a JpegImage and a Quadrangle. The image is the entire frame as captured, and the quadrangle represents the face's location.
See CropFace under *IMAGE MANIPULATION FUNCTIONALITY* for further instructions.

DocumentDetectionResult: Contains a JpegImage and a Quadrangle. The image is the entire frame as captured, and the quadrangle represents the document's outline corners' location.
See Transform under *IMAGE MANIPULATION FUNCTIONALITY* for further instructions.

BarcodeDetectionResult: Contains a FindBarcode.Result array with a single FindBarcode.Result object. See Per-Frame Detection Functionality for further details.

LivenessDetectionResult: Contains parameters indicating liveness probability.

LivenessDetectionResults: Contains a FaceDetectionResult, and a LivenessDetectionResult array list.

Liveness2DetectionResults: Contains a LivenessDetectionResults, and gesture test result parameters.

LIVENESS2 ENUMERATORS

The following enumerators are used to indicate Liveness2 detection session states:

GESTURE TYPE

- **GestureTypeFaceForward**: Head angle looking forward.
- **GestureTypePanLeft**: Pan head movement to the left.
- **GestureTypePanRight**: Pan head movement to the right.
- **GestureTypeSmile**: Smile facial gesture.
- **GestureTypeEyesClosed**: Eyes closed facial gesture.

LIVENESS2 DETECTOR RESULT CODE

- **Liveness2DetectorResultFAIL**: Liveness2 detection session failed.
- **Liveness2DetectorResultPASS**: Liveness2 detection session passed.
- **Liveness2DetectorResultERROR**: Liveness2 detection session received an error.
- **Liveness2DetectorResultINPROGRESS**:Liveness2 detection session started.
- **Liveness2DetectorResultANALYSE**: Liveness2 detection result is being analyzed.
- **Liveness2DetectorResultENDED**: Liveness2 detection session ended.

LIVENESS2 SESSION RESULT CODE

Session conclusion:

- **Liveness2SessionResultFAIL**: Liveness2 session failed.
- **Liveness2SessionResultPASS**: Liveness2 session passed.
- **Liveness2SessionResultERROR**: Liveness2 detection session failed with generic error.
- **Liveness2SessionINPROGRESS**: Liveness2 detection session started.
- **Liveness2SessionResultLiveness2RequirementsFAIL**: User failed to comply with Liveness2 commands minimum requirements.
- **Liveness2SessionResultTimeoutFAIL**: Liveness2 session timed out before user complied with minimum requirements.
- **Liveness2SessionFaceDetectionFAIL**: Face detection failed during session.
- **Liveness2SessionLivenessDetectionFAIL**: Failed to detect live person.
- **Liveness2SessionFaceTrackingFAIL**: Failed to track detected face continuously during session.

Session initialization:

- **Liveness2SessionDeviceOrientationAngleFAIL**: Wrong device orientation angle was detected before session start.
- **Liveness2SessionFaceTooFarFAIL**: Face of user is too far from the camera.
- **Liveness2SessionFaceTooNearFAIL**: Face of user is too close to the camera.

INTERNAL SDK OBJECTS

Common: Provides a GetLibVersionName() method, that returns the library version.

ServerRequestGenerator: Generates the JSON request expected by the AU10TIX server.

SERVER REQUEST GENERATOR

The ServerRequestGenerator class generates the JSON request required by the AU10TIX server.

The ServerRequestGenerator follows the builder pattern and can be customized to cover all possible request combinations.

```
ServerRequestGenerator compoundProcessingRequestBuilder =

    new ServerRequestGenerator(currentDoc)

        .withFrontSideQualityResult(currentDocQuality)

        .withDocumentBackSide(currentDoc)

        .withBackSideQualityResult(currentDocQuality)

        .setRequestForDataExtractionOnly(true)

        .withOptionalRequestTag("Optional Request Tag")

        .withBarcode(currentBarcode)

        .withPoaDocument(true)

        .withOptionalPoaTag("Optional POA Tag")

        .withLiveness2(currentLiveness2)

//      .withLiveness(currentLiveness);

    String Json = compoundProcessingRequestBuilder.build();
```

Note: The withLiveness() method is available for backward compatibility purposes. New implementations can disregard it.

IMAGE MANIPULATION FUNCTIONALITY

To maintain the secured integrity of an image captured with AU10TIX mobile SDK, follow these steps:

1. Create an imageContainer object from the JpegImage received in capture session result object.
2. Crop/transform the imageContainer object as described later in this section.
3. Produce a JpegImage from the cropped imageContainer, using ImageContainer.getJpegPhoto().

Note: You may create a bitmap from the jpegImage to display a preview. Make sure that this converted image is used for this purpose only. To maintain security integrity, only send JpegImage instances that were produced and manipulated as the following examples show.

Caution: Once the final jpg image is created it should not be tempered with until it reaches the AU10Tix authentication server.

Transform: crops and transforms document images by a quadrangle.
Modifies image proportions by quadrangle, to a rectangular shape.

Transform example:

```
ImageContainer imageContainer = new
ImageContainer(detectionResult.getJpegImage());

Transform transform = new Transform(imageContainer);

Transform.Result transformResult = transform
Transform.transformByRectangle(detectionResult.getQuadrangle());

ImageContainer resCont = transformResult.getResult();

JpegImage result =ImageContainer.getJpegPhoto(resCont);
```

CropFace: crops face images by quadrangle.

Crop example:

```
ImageContainer faceImageContainer = new
ImageContainer(detectionResult.getJpegImage());

CropFace cropFace = new CropFace(faceImageContainer);

ImageContainer croppedFaceContainer =
cropFace.crop(detectionResult.getQuadrangle());

JpegImage faceCropResultImage = ImageContainer
.getJpegPhoto(croppedFaceContainer);
```

PER-FRAME DETECTION FUNCTIONALITY

FindBarcode: Accepts an image container and returns a set of detection results.

FindBarcode.Result Structure

- rawdata: The barcode data represented as a byte[].
- points: Barcode vertex coordinates on the original image.
  format: An integer representation of barcode type. The supported types are CODE128 and PDF417.
- orientation: An integer representation of the scanned barcode orientation.
  Options are horizontal or vertical.
- Status: An integer representation of the barcode recognition status.
  Options are "ERROR", "OK", and " POSIBLE_PRESENT".

FindCorners: Accepts an image container and returns a document detection result that includes the quadrangle required for further manipulations.

FindFace: Accepts an image container and returns a face detection result that includes the quadrangle required for further manipulations.

ImageQuality: Produces a quality rating for a specific ImageContainer. See *IMAGE QUALITY EVALUATION* for a complete description.

Liveness: Accepts image containers and returns a set of liveness detection results.

LOCALIZATION AND RESOURCE CUSTOMIZATION

The AU10TIX SDK resources can be overridden by the implementing application's resources. To override a resource value, provide your own resource with the same resource id.

STRING RESOURCES

The following string resources can be replaced and localized:

LIVENESS2 TEST INSTRUCTIONS

The Liveness2 test instructs the user to perform simple gestures. These are:

```
    <string name="tix_face_forward">FACE FORWARD</string>

    <string name="tix_turn_right">TURN RIGHT</string>

    <string name="tix_turn_left">TURN LEFT</string>

    <string name="tix_close_eyes">CLOSE EYES</string>

    <string name="tix_smile">SMILE</string>

    <string name="tix_test_ended">TEST ENDED</string>
```

FACE DETECTION ALERT

Whenever the AU10TIX SDK performs the Liveness detection and fails to find a face, the following alert appears:

```
    <string name="tix_face_missing">Face not found!</string>
```

DIMENSION RESOURCES

The following dimension may be changed:

LIVENESS2 TEST LABEL TEXT SIZE

The Liveness test label has the following default text size value. A different value may be specifically set for a larger screen, if necessary.

```
    <dimen name="tix_challenge_label">20sp</dimen>
```

COLOR RESOURCES

The following colors may be replaced:

LIVENESS2 TEST FACE BORDER COLORS

The face border color is used to indicate a single challenge result.

```
    <color name="tix_base_color">#CC000000</color>

    <color name="tix_fail_color">#CC780000</color>

    <color name="tix_success_color">#CC007800</color>
```