AU10TIX

ANDROID SIMPLE SAMPLE STEP-BY-STEP GUIDE

SDK version 1.0.3

Document version: 1

AU10TIX

DISCLAIMER

This document contains trade secrets and confidential commercial or financial information exempt from disclosure under the Freedom of Information Act, 5 U.S.C. § 552(b)(4). It is submitted only within the framework of the NDA between the parties. All the information contained in this document is confidential and proprietary information belonging to Au10tix Ltd. Any person, firm, organization and/or entity in possession of this document, shall hold same in confidence, and shall be entitled to review and use the information in this document solely for the purpose of evaluating and/or pursuing business cooperation with Au10tix Ltd. and for no other purpose whatsoever, and may not use, publicize, permit, authorize or instigate disclosure of any information, terms, strategies, ideas, concepts, designs, methodologies, applications or other provisions contained in this document, to any person, firm, organization or entity of any type whatsoever.

AU10TIX

CONTENTS

AU10TIX

OVERVIEW

This guide describes the key steps required to produce the sample application provided along with the SDK.

The sample project provides a single activity implementation suggestion and detection samples contained in fragments.

PREPARATIONS

PREREQUISITES

Familiarity with Android 5.1 LOLLIPOP_MR1 (API level 22) or higher.

SAMPLE IMPLEMENTATION STEPS

Implementation consists of the following main stages:

- **Preliminary operations** - starting a new project including integrating the SDK, updating the Gradle build file and declaring permission use.
- **Creating a complete implementation example** on a single activity.

PRELIMINARY OPERATIONS

This stage describes all infrastructure related operations required for a successful implementation.

Perform the following:

- Integrate the SDK as follows:
  Note: Before you begin, make sure you are implementing the latest SDK version.
  1. Start a new project.
  2. Import the Au10tix SDK library into your project.
  3. Run a gradle sync to apply the changes. You can now use the SDK.
  4. Update the Gradle Build File. This example requires the following dependency:

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

- Declare permission use. Add the following permissions to the AndroidManifest:

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

- Load the library and verify its dependencies are available by calling Common.isReady().
  Note: The SDK can be loaded asynchronously, prior to other SDK related resource consuming operations.
- Keep the Boolean result to determine whether Liveness (False) or Liveness2 (True) should be loaded.

IMPLEMENTATION OF THE SIMPLE SAMPLE EXAMPLE PROJECT

The following implementation conducts all detection operations in separate fragments hosted on a single activity. This document explains the main points of the AU10TIX SDK integration implementation sample assuming that the user is familiar with Android SDK implementation.

The AU10TIX SDK integration implementation project application sample is implemented using base fragments to streamline the detection procedure flow and provide abstract methods to override by the inheriting detection fragments (see Fragment Inheritance Structure).

Note: The selected root class for this sample is: android.support.v4.app.FragmentActivity.

BASE CLASSES

A base class is the parent class of a derived class. In the current implementation example, create the following base classes for Activity and Fragments:

- Base Activity – handles library initialization
- Base Fragment – handles permission requests
- Base Detection Fragment – streamlines the detection procedure flow

CREATE THE BASE ACTIVITY

Class: BaseActivity.java
The Base Activity handles library initialization by loading the native library asynchronously and verifying that the required dependencies are available.

```
abstract class BaseActivity extends FragmentActivity {
    boolean dependenciesAreReady = false;
    static AsyncTask dependAsync;
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

- Calling Common.isReady() loads the SDK, and verifies that the device can perform Liveness2 detection operations. The SDK can be loaded asynchronously, prior to any other SDK related resource consuming operations.

```
        dependAsync = new AsyncTask<Void, Void, Void>() {
```

```
            @Override
            protected Void doInBackground(Void... voids) {
                dependenciesAreReady =  Common.isReady(BaseActivity.this);
                Log.d(TAG, "Vision dependencies ready: "+
dependenciesAreReady);
                return null;
            }
        }.execute();

    }

    protected boolean visionDependenciesPrepared(){

        return dependenciesAreReady;
    }

}
```

CREATE THE BASE FRAGMENT

Class: BaseFragment.java

The Base Fragment handles runtime permission logic and provides a protected abstract method `continueWithPermissions()` for the derived fragments to implement.

BASE DETECTION FRAGMENT

Class: BaseDetectionFragment.java

The Base Detection Fragment (the main parts shown below) holds the Au10tixCameraVision instance and its listeners. It also streamlines the detection procedure flow and provides abstract methods to override by the inheriting detection fragments.

Creating the Base Detection Fragment defines a single uniform behavior for the derived fragments and provides them with a common pool of variables and methods. This implementation makes sure all inheriting fragments will safely handle resources that require permission.

These are:

- Detection fragment (face, document, liveness, and barcode detections)
- Liveness2 fragment
- Custom Liveness2 Detection fragment
- Custom Controls Face Detection fragment
- Custom Document Quality Indication Detection fragment

- The following are the inherited Au10tixCameraVision listeners and the Au10tixCameraVision instance.

```
public abstract class BaseDetectionFragment extends BaseFragment {
    …
    protected Au10tixCameraVision.OnBackListener tixBackListener;
    protected Au10tixCameraVision.OnCaptureListener tixCaptureListener;
    protected FrameLayout cameraVisionContainer;
    protected Au10tixCameraVision mCameraVision;
    private OnDetectorFragmentInteractionListener mListener;
```

- BaseDetectionFragment holds the OnDetectorFragmentInteractionListener that provides the `onDetectionResult(DetectionResult result)` and `onBack()` events, that streamlines detection result and back navigation handling.

```
public interface OnDetectorFragmentInteractionListener {
    void onDetectionResult(DetectionResult result);

    void onBack();
}
```

- Upon detection capture, stops camera vision and passes the result to the MainActivity (implementing the OnDetectorFragmentInteractionListener).

```
 public BaseDetectionFragment() {

        tixBackListener = new Au10tixCameraVision.OnBackListener() {
            @Override
            public void OnBack() {
                onBackButtonPressed();
            }
        };

        tixCaptureListener = new Au10tixCameraVision.OnCaptureListener() {
            @Override
            public void onCapture(DetectionResult detectionResult) {
                mCameraVision.stop();
                onDetectionResult(detectionResult);
            }
        };

    }
```

- The following passes the back button pressed event to the MainActivity (implementing the OnDetectorFragmentInteractionListener).

```
    protected void onBackButtonPressed() {
        if (mListener != null) {
            mListener.onBack();
        }
```

```
}
```

- The following passes the detection result from the fragment derived from this class to the MainActivity.

```
protected void onDetectionResult(DetectionResult result) {

    if (mListener != null) {
        mListener.onDetectionResult(result);
    }
}
```

- *init()* Called upon onResume. Checks if the permissions are granted. If so, it calls the Abstract buildCameraVision implemented by the derived fragments and used to build the Au10tixCameraVision instance.

```
protected void init() {

    if (!requiredPermissionsGranted()) {

        if (!shouldStopRequesting) {
            requestPermissions(permissions, PERMISSIONS_CODE);
        }
    } else if (mCameraVision == null) {
        buildCameraVision();
    }

}

protected void continueWithPermissions() {
    if (mCameraVision == null) {
        buildCameraVision();
    }
}
```

- The Au10tixCameraVision instance is initialized upon onResume.

```
@Override
public void onResume() {
    super.onResume();
    init();
}
```

- The Au10tixCameraVision instance is removed from the cameraVisionContainer, stopped and released upon onPause.

```
@Override
public void onPause() {
    super.onPause();
    cameraVisionContainer.removeView(mCameraVision);

    if (mCameraVision != null) {
        mCameraVision.stop();
        mCameraVision.release();
        mCameraVision = null;
```

```
        }
    }
```

- The Abstract buildCameraVision is implemented by the derived fragments used to build the Au10tixCameraVision instance and adds it to the cameraVisionContainer FrameLayout.

```
protected abstract void buildCameraVision();
```

MAINACTIVITY IMPLEMENTATION

Class: MainActivity.java

The MainActivity (described below) exemplifies a single activity implementation of the SDK. The single activity loads all fragments. It is responsible for selection between different detection implementations and is responsible for the collection of all detection results. It also presents an example of the JSON request generator.

This includes the following implementation types:

- Face Detection
- Document Detection
- Barcode Detection
- Liveness Detection
- Liveness2 Detection

JSON REQUEST GENERATOR

The json Request Generator is a non-mandatory method responsible for creating a valid authorization request for the server. Once all required detection results are gathered and ready to send, pass them to the ServerRequestGenerator to generate the JSON string as shown below.

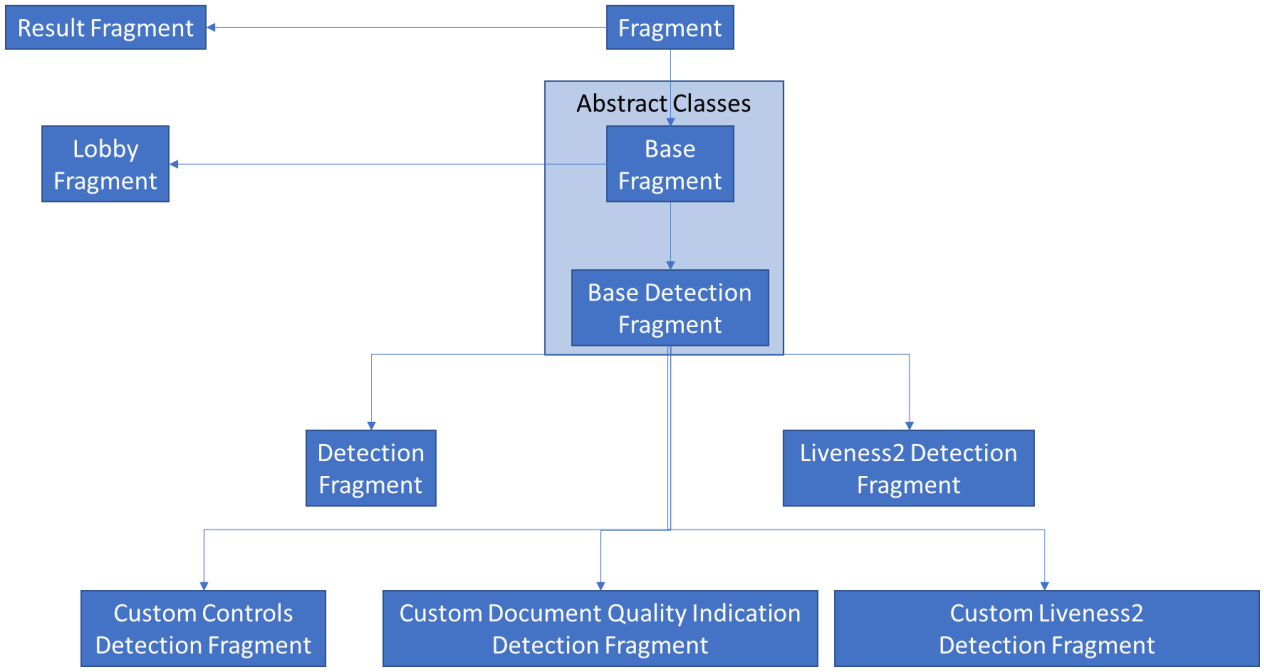
```
    public String generateCompoundProcessingRequestJson() {

        String returnedJsonString;
        ServerRequestGenerator compoundProcessingRequestBuilder =
                new ServerRequestGenerator(currentDocDetectionResult)
                        .withFrontSideQualityResult(currentDocQuality)
                        .withDocumentBackSide(currentDocDetectionResult)
                        .withBackSideQualityResult(currentDocQuality)
//                          .setRequestForDataExtractionOnly(true)
                        .withOptionalRequestTag("Optional Request Tag")
                        .withBarcode(currentBarcode)
                        .withPoaDocument(true)
                        .withOptionalPoaTag("Optional POA Tag")
                        .withLiveness2(currentLiveness2DetectionResult)
//                          .withLiveness(currentLiveness) //Provided for
backward compatibility purposes.
                ;
        returnedJsonString = compoundProcessingRequestBuilder.build();
```

```
        return returnedJsonString;
    }
}
```

FRAGMENT INHERITANCE STRUCTURE

The following diagram shows the Fragment Inheritance structure as implemented in the simple sample. Each block is explained in the following sections.



LOBBY FRAGMENT

Class: LobbyFragment.java – Extends Base Fragment

The lobby Fragment provides detection selection logic and all related permissions validation.

RESULT FRAGMENT

Class: ResultFragment.java – Extends Fragment

The ResultFragment presents detection results as received from the DetectionResultAsyncTask in a result object.

DETECTION FRAGMENT

Class: DetectionFragment.java – Extends Base Detection Fragment

Detection Fragment presents the most basic detection implementation. This example presents a default implementation that covers face, document, liveness, and barcode detections as shown below:

- Determines which detection type to load from the received arguments and defines the camera facing direction accordingly. Both are defined upon creation and used when the instance is built.

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {
            mDetectionType = getArguments().getInt(ARG_DETECTION_TYPE, 0);
            switch (mDetectionType) {
                case FACE_DETECTION_MODE:
                    mTitle =
getResources().getStringArray(R.array.detectionOptions)[FACE_DETECTION_MODE];

                case LIVENESS_DETECTION_MODE:
                    mFacing = CameraVision.CAMERA_FACING_FRONT;
                    break;

                case DOCUMENT_DETECTION_MODE:
                    mTitle =
getResources().getStringArray(R.array.detectionOptions)[DOCUMENT_DETECTION_MO
DE];
                    mFacing = CameraVision.CAMERA_FACING_BACK;
                    break;
                case BARCODE_DETECTION_MODE:
                    mTitle =
getResources().getStringArray(R.array.detectionOptions)[BARCODE_DETECTION_MOD
E];
                    mFacing = CameraVision.CAMERA_FACING_BACK;
                    break;
            }
        }
    }
```

- To build the au10tixcameravision instance, implement the abstract buildCameraVision method from the BaseDetection Class.
  qualityPeriod(250L) - Is relevant for document detection only, can be set generally for all detection types. Will not get used when not detecting documents.
  customQualityToast(R.layout.view_quality_dialog) - Is relevant for document detection only, can be set generally for all detection types. Will be ignored when not detecting documents.

```
    protected void buildCameraVision() {

        mCameraVision = (new Au10tixIQCameraVision.Builder(getActivity()))
                .qualityPeriod(250L)
                .customQualityToast(R.layout.view_quality_dialog)
                .showLiveFinder(true)
                .onBack(tixBackListener)
                .onCapture(tixCaptureListener)
```

```
                .facingMode(mFacing)
                .detectionMode(mDetectionType)
                .build();
```

- *cameraVisionContainer.addView(mCameraVision)* - Adds the Au10tixCameraVision instance to its FrameLayout container.

```
cameraVisionContainer.addView(mCameraVision);

        mCameraVision.setTitleText(mTitle);
        mCameraVision.start();
    }
```

- Defines the cameraVisionContainer view.

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View root = inflater.inflate(R.layout.fragment_detector, container,
false);
        cameraVisionContainer =
root.findViewById(R.id.cameraVisionContainer);

        return root;
    }

}
```

LIVENESS2 DETECTION FRAGMENT

Class: Liveness2DetectionFragment.java – Extends Base Detection Fragment

The Liveness2DetectionFragment presents a basic liveness2 detection implementation. It includes Liveness2-specific event listeners implementation example as shown below:

For basic customization the session listener should be implemented. This example holds the vibrator and MediaPlayer audio files.

```
public class Liveness2DetectionFragment extends BaseDetectionFragment
implements Au10tixCameraVision.Liveness2SessionListener {
    private static final String TAG = "Liveness2 Detection";

    Vibrator vib;
    MediaPlayer mpF, mpS;
```

- Initializing vibrator service.

```
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        vib = (Vibrator)
getActivity().getApplicationContext().getSystemService(Context.VIBRATOR_SERVI
CE);
    }
```

- Inflates the layout.

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View root = inflater.inflate(R.layout.fragment_detector, container,
false);
        cameraVisionContainer =
root.findViewById(R.id.cameraVisionContainer);

        return root;
    }
```

- Defines the MediaPlayer instances for audio challenge result indication.

```
@Override
    public void onResume() {
        super.onResume();
        mpS = MediaPlayer.create(getActivity().getApplicationContext(),
R.raw.success);
        mpF = MediaPlayer.create(getActivity().getApplicationContext(),
R.raw.fail);
    }
```

- Releases the MediaPlayer instances.

```
@Override
    public void onStop() {
        super.onStop();

        try {
            if (mpF != null) {
                mpF.release();
            }
            if (mpS != null) {
                mpS.release();
            }
        } catch (Exception e) {
            Log.e(TAG, e.getLocalizedMessage());
        }
```

- To build the au10tixcameravision instance implement the Abstract buildCameraVision method from the BaseDetection Class.

```
    protected void buildCameraVision() {

        Au10tixCameraVision cameraVision = (new
Au10tixCameraVision.Builder(getActivity().getApplicationContext()))
                .facingMode(Au10tixCameraVision.CAMERA_FACING_FRONT)
                .detectionMode(Au10tixCameraVision.LIVENESS2_DETECTION_MODE)
                .onBack(tixBackListener)
                .onCapture(tixCaptureListener)
                .onLiveness2SessionEvents(this)
```

```
                .build();

        mCameraVision = cameraVision;
        cameraVisionContainer.addView(mCameraVision);
    }
```

- Indicates single challenge resolution during the session.

```
@Override
    public void onChallengeState(Liveness2DetectorResultCode
liveness2DetectorResultCode) {

        switch (liveness2DetectorResultCode) {

            case Liveness2DetectorResultFAIL:
                challengeCounter++;

                if (vib.hasVibrator()) {

                    long[] pattern = {0, 20, 20, 10};

                    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
                        vib.vibrate(VibrationEffect.createWaveform(pattern, -
1));
                    } else {
                        vib.vibrate(pattern, -1);
                    }
                }

                mpF.start();
                break;
            case Liveness2DetectorResultPASS:

                if (vib.hasVibrator()) {

                    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
                        vib.vibrate(VibrationEffect.createOneShot(20,
VibrationEffect.DEFAULT_AMPLITUDE));
                    } else {
                        vib.vibrate(20);
                    }
                }
                mpS.start();
                break;
        }
    }
```

- Indicates preliminary state and final session resolution.

```
@Override
    public void onLiveness2SessionResult(Liveness2SessionResultCode
liveness2SessionResultCode) {
        if (isVisible()) {
            switch (liveness2SessionResultCode) {
                case Liveness2SessionResultFAIL:
```

```
                case Liveness2SessionResultPASS:
                case Liveness2SessionResultLiveness2RequirementsFAIL:
                case Liveness2SessionLivenessDetectionFAIL:
                    mCameraVision.stop();
                    mCameraVision.release();
                    break;
                case Liveness2SessionFaceDetectionFAIL:
                    if (mCameraVision != null) {

showToast(getResources().getString(R.string.tix_face_missing));


mCameraVision.setTitleText(getResources().getString(R.string.tix_face_missing
));
                        mCameraVision.stop();
                        mCameraVision.release();

getActivity().getSupportFragmentManager().popBackStack();

                    }
                    break;
                case Liveness2SessionResultTimeoutFAIL:

showToast(getResources().getString(R.string.tix_timeout));

                    mCameraVision.stop();
                    break;
                case Liveness2SessionINPROGRESS:
                case Liveness2SessionFaceTrackingFAIL:
                case Liveness2SessionDeviceOrientationAngleFAIL:
                case Liveness2SessionFaceTooFarFAIL:
                case Liveness2SessionFaceTooNearFAIL:
                    break;
                case Liveness2SessionResultERROR:
                    showToast(getString(R.string.tix_error));
                    mCameraVision.stop();
                    getActivity().getSupportFragmentManager().popBackStack();
                    break;
            }
        }
    }
```

- Indicates the next gesture type.

```
 @Override
    public void onGestureChallenge(GestureType gestureType) {
    }
}
```

CUSTOM CONTROLS DETECTION FRAGMENT

Class: CustomControlsFaceDetectionFragment.java – Extends Base Detection Fragment

AU10TIX

Custom Controls enables UI buttons customization. The following example demonstrates custom control implementation running a face detection session.

- *fragment_custom_controls_detector.xml* is the custom layout held as a view and passed to the Au10tixCameraVision control view builder method.

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {

        root = inflater.inflate(R.layout.fragment_custom_controls_detector,
container, false);
        cameraVisionContainer =
root.findViewById(R.id.cameraVisionContainer);

        return root;
    }
```

- *.controlView(root)* - this is the custom controls view defined below.

```
 @Override
    protected void buildCameraVision() {

        mCameraVision = (new Au10tixCameraVision.Builder(getActivity()))
                .showLiveFinder(true)
                .onBack(tixBackListener)
                .onCapture(tixCaptureListener)
                .detectionMode(FACE_DETECTION_MODE)
                .facingMode(Au10tixCameraVision.CAMERA_FACING_FRONT)
                .controlView(root).build();

        cameraVisionContainer.addView(mCameraVision);

        mCameraVision.start();
    }

    @Override
    public void onResume() {
        super.onResume();

        if (mCameraVision != null) {

mCameraVision.setTitleText(getString(R.string.tix_custom_controls));
        }
    }
}
```

CUSTOM DOCUMENT QUALITY INDICATION DETECTION FRAGMENT

Class: CustomDocumentQualityIndicationDetectionFragment.java – Extends Base Detection Fragment

The CustomDocumentQualityIndicationDetectionFragment is responsible for customizing the UI for Face, Document and Document Quality. It presents the Au10tixIQCameraVisionQualityEvents class,

which is an example of inheritance from Au10tixCameraVision. The following example implements a custom document quality session.

- The UI holds three TextViews to hide or display according to reflection, sharpness and brightness values.

```
TextView
        reflectionTextView,
        sharpnessTextView,
        brightnessTextView;
boolean
        reflectTvVisible = true,
        sharpTvVisible = true,
        brightTvVisible = true;


@Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View root =
inflater.inflate(R.layout.fragment_custom_quality_detector, container,
false);
        reflectionTextView = root.findViewById(R.id.textViewReflection);
        sharpnessTextView = root.findViewById(R.id.textViewSharpness);
        brightnessTextView = root.findViewById(R.id.textViewBrightness);
        cameraVisionContainer =
root.findViewById(R.id.cameraVisionContainer);

        return root;
    }
```

- Evoked by the quality listener, displays or hides the quality indicating TextViews.

```
private void showNotification(int indicator, final boolean shouldShow) {
        switch (indicator) {
            case ImageQuality.IC_SHARPNESS:
                if (sharpTvVisible != shouldShow) {
                    sharpTvVisible = shouldShow;

                    sharpnessTextView.setVisibility(shouldShow ? View.VISIBLE
: View.INVISIBLE);
                }
                break;
            case ImageQuality.IC_BRIGHTNESS:
                if (brightTvVisible != shouldShow) {
                    brightTvVisible = shouldShow;

                    brightnessTextView.setVisibility(shouldShow ?
View.VISIBLE : View.INVISIBLE);
                }
                break;
            case ImageQuality.IC_REFLECTION:
                if (reflectTvVisible != shouldShow) {
                    reflectTvVisible = shouldShow;
```

```
                    reflectionTextView.setVisibility(shouldShow ?
View.VISIBLE : View.INVISIBLE);
                }
                break;
            }
        }
```

- This implementation permits separate quality parameter indications. Sets a custom behavior for image quality indication. In this example the show notification method is used directly under the setQualityListener builder method.

```
        mCameraVision = (new
Au10tixIQCameraVisionQualityEvents.Builder(getActivity()))
                .qualityPeriod(250L)
                .setQualityListener(result -> (new
Handler(Looper.getMainLooper())).post(() -> {
                    showNotification(ImageQuality.IC_SHARPNESS |
ImageQuality.IC_REFLECTION | ImageQuality.IC_BRIGHTNESS,
!IQSufficiencyCheck.docID(result));
```

- ImageQuality - Sharpness

```
                    if (result.isUsed(ImageQuality.IC_SHARPNESS)) {
                        Log.w("quality test", "Sharpness value: " +
result.getSharpness());
                        showNotification(ImageQuality.IC_SHARPNESS,
!IQSufficiencyCheck.docID(ImageQuality.IC_SHARPNESS, result));
                    }
```

- ImageQuality - Reflection

```
                      if (result.isUsed(ImageQuality.IC_REFLECTION)) {
                        Log.w("quality test", "Reflection value: " +
result.getReflection());

                        showNotification(ImageQuality.IC_REFLECTION,
!IQSufficiencyCheck.docID(ImageQuality.IC_REFLECTION, result));
                    }
```

- ImageQuality - Brightness

```
                    if (result.isUsed(ImageQuality.IC_BRIGHTNESS)) {
                        Log.w("quality test", "Brightness value: " +
result.getBrightness());

                        showNotification(ImageQuality.IC_BRIGHTNESS,
!IQSufficiencyCheck.docID(ImageQuality.IC_BRIGHTNESS, result));
                    }

                }))
                .facingMode(Au10tixCameraVision.CAMERA_FACING_BACK)
                .detectionMode(Au10tixCameraVision.DOCUMENT_DETECTION_MODE)
```

```
                .showLiveFinder(true)
                .onBack(tixBackListener)
                .onCapture(tixCaptureListener)
                .onProcessing(tixProcessingListener)
                .build();
```

CUSTOM LIVENESS2 DETECTION FRAGMENT

Class: CustomLiveness2DetectionFragment.java – Extends Base Detection Fragment

The CustomLiveness2DetectionFragment presents a basic liveness2 detection implementation with custom UI as shown below:

- *TextView userInteraction* is the label on which commands and states are displayed.

```
TextView userInteraction;
```

- Custom root view (fragment_custom_liveness2_detector.xml)

```
@Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View root =
inflater.inflate(R.layout.fragment_custom_liveness2_detector, container,
false);

        cameraVisionContainer =
root.findViewById(R.id.cameraVisionContainer);
        userInteraction = root.findViewById(R.id.userInteractionTextView);

        return root;
    }
```

- *.showLiveFinder(false)* - Permits the user to override the default Liveness2 UI. When building the cameravision set the showliveFinder builder method to false in order to override the default liveness UI.

```
    @Override
    protected void buildCameraVision() {

        mCameraVision = (new Au10tixCameraVision.Builder(getActivity()))
                .facingMode(CameraVision.CAMERA_FACING_FRONT)
                .detectionMode(CameraVision.LIVENESS2_DETECTION_MODE)
                .onBack(tixBackListener)
                .onCapture(tixCaptureListener)
                .onLiveness2SessionEvents(this)
                .showLiveFinder(false)
                .build();
```

- Customize session result event handling.

```
@Override
    public void onLiveness2SessionResult(Liveness2SessionResultCode
liveness2SessionResultCode) {
        if (isVisible()) {
            switch (liveness2SessionResultCode) {
                case Liveness2SessionResultFAIL:
                case Liveness2SessionResultPASS:
                case Liveness2SessionResultLiveness2RequirementsFAIL:
                case Liveness2SessionLivenessDetectionFAIL:

                    if (mCameraVision != null) {
                        mCameraVision.stop();
                    }
                    break;
                case Liveness2SessionFaceDetectionFAIL:
                    if (mCameraVision != null) {
                        mCameraVision.stop();
                        Toast.makeText(getActivity().getApplicationContext(),
getResources().getString(R.string.tix_face_missing),
Toast.LENGTH_SHORT).show();
                    }
                    getActivity().getSupportFragmentManager().popBackStack();
                    break;
                case Liveness2SessionResultTimeoutFAIL:

                    if (mCameraVision != null) {
                        mCameraVision.stop();
                    }
                    Toast.makeText(getActivity().getApplicationContext(),
getResources().getString(R.string.tix_timeout), Toast.LENGTH_SHORT).show();
                    break;
                case Liveness2SessionINPROGRESS:
                    break;
                case Liveness2SessionFaceTrackingFAIL:
                    break;
```

- Pre-session result event handling.

```
                case Liveness2SessionDeviceOrientationAngleFAIL:

setLabelText(getResources().getString(R.string.tix_device_orientation));
                    break;
                case Liveness2SessionFaceTooFarFAIL:

setLabelText(getResources().getString(R.string.tix_face_too_small));
                    break;
                case Liveness2SessionFaceTooNearFAIL:

setLabelText(getResources().getString(R.string.tix_face_too_large));
                    break;
                case Liveness2SessionResultERROR:
                    showToast(getString(R.string.tix_error));

                    if (mCameraVision != null) {
                        mCameraVision.stop();
```

```
                    }
                    getActivity().getSupportFragmentManager().popBackStack();
                    break;
                }
            }
        }
```

- When onDetectionResult gets called the setLabelText sets the test ended string for the event resource.

```
    @Override
    protected void onDetectionResult(DetectionResult result) {
        setLabelText(getResources().getString(R.string.tix_test_ended));
        mCameraVision.stop();
        super.onDetectionResult(result);
    }
```

- When onGestureChallenge gets called the setLabelText sets the test ended string for the event resource.

```
    @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;
        }
    }
}
```

DETECTION RESULT ASYNC TASK

Class: DetectionResultAsyncTask.java

The DetectionResultAsyncTask presents the procedure required to gather the detection result products captured by the Au10tixCameraVision according to detection result type as shown below:

- Pass the detection result provided by the Au10tixCameraVision onCapture to the DetectionResultAsyncTask, to generate the ResultObject used in this sample.
  Note: Set the Gson library to be used to convert java objects to json.
  Note: Remember to release ImageContainer objects once they are no longer used.

```
    @Override
    protected ResultObject doInBackground(DetectionResult...
detectionResults) {

        DetectionResult currentResult = detectionResults[0];
        ResultObject resultObject = null;
        switch (currentResult.getType()) {
```

RESULT PRODUCT EXAMPLES

The following are examples of result products used with DetectionResultAsyncTask to produce final result objects used in the example.

- The following gathers the required Face Detection products and packages them into a ResultObject (described at the end of this section).

```
case FACE_DETECTION_RESULT: {
                FaceDetectionResult faceDetectionResult =
(FaceDetectionResult) currentResult;

                ImageContainer ic =
ImageUtil.jpegImageToImageContainer(faceDetectionResult.getJpegImage());
                CropFace cropFace = new CropFace(ic);
                ImageContainer icCropped =
cropFace.crop(faceDetectionResult.getQuadrangle());
                ic.release();
                JpegImage croppedFacedJpeg =
ImageContainer.getJpegPhoto(icCropped);
                icCropped.release();
                resultObject = new ResultObject(croppedFacedJpeg,
FACE_DETECTION_RESULT);
                resultObject.setExtra("");

                break;
            }
```

- The following gathers the required Document Detection products and packages them into a ResultObject (described at the end of this section).

```
            case DOCUMENT_DETECTION_RESULT: {
                DocumentDetectionResult documentDetectionResult =
```

```
(DocumentDetectionResult) currentResult;
                ImageContainer ic =
ImageUtil.jpegImageToImageContainer(documentDetectionResult.getJpegImage());
                ImageQuality.Result qr = (new
ImageQuality(ic)).getQuality(ImageQuality.IC_ALL, true);

                ImageQualityModel imageQualityModel = new
ImageQualityModel(qr);

                ImageContainer icTransformed = new
Transform(ic).transformByRectangle(documentDetectionResult.getQuadrangle()).g
etResult();
                ic.release();
                JpegImage croppedRectJpeg =
ImageContainer.getJpegPhoto(icTransformed);
                icTransformed.release();

                resultObject = new ResultObject(croppedRectJpeg,
DOCUMENT_DETECTION_RESULT);
                resultObject.setExtra(new Gson().toJson(imageQualityModel));
                break;
            }
```

- The following gathers the required Barcode Detection products and packages them into a ResultObject (described at the end of this section).

```
            case BARCODE_DETECTION_RESULT: {
                BarcodeDetectionResult barcodeDetectionResult =
(BarcodeDetectionResult) currentResult;
                StringBuilder barcodeBuilder = new StringBuilder();
                FindBarcode.Result[] results =
barcodeDetectionResult.getResults();
                for (FindBarcode.Result barcodeResult : results) {
                    barcodeBuilder.append(new
String(barcodeResult.getRawdata()));
                }
                resultObject = new ResultObject(null,
BARCODE_DETECTION_RESULT);
                resultObject.setExtra(barcodeBuilder.toString());
                break;
            }
```

- The following gathers the required Liveness Detection products and packages them into a ResultObject (described at the end of this section).

```
            case LIVENESS_DETECTION_RESULT: {
                LivenessDetectionResults livenessDetectionResults =
(LivenessDetectionResults) currentResult;
                LivenessResultModel livenessResultModel = new
LivenessResultModel(livenessDetectionResults.getLivenessDetectionFinalResult(
));

                ImageContainer ic =
ImageUtil.jpegImageToImageContainer(livenessDetectionResults.getFaceDetection
Result().getJpegImage());
```

```
                CropFace cropFace = new CropFace(ic);
                ImageContainer icCropped =
cropFace.crop(livenessDetectionResults.getFaceDetectionResult().getQuadrangle
());
                ic.release();

                JpegImage croppedFacedJpeg =
ImageContainer.getJpegPhoto(icCropped);
                icCropped.release();

                resultObject = new ResultObject(croppedFacedJpeg,
LIVENESS_DETECTION_RESULT);
                resultObject.setExtra(new
Gson().toJson(livenessResultModel));

                break;
            }
```

- The following gathers the required Liveness2 Detection products and packages them into a ResultObject (described at the end of this section).

```
            case LIVENESS2_DETECTION_RESULT: {

                Liveness2DetectionResults liveness2DetectionResults =
((Liveness2DetectionResults) currentResult);

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

liveness2DetectionResults.setSessionResultCode(Liveness2SessionResultCode.Liv
eness2SessionResultFAIL);
                }
                if (null !=
liveness2DetectionResults.getLivenessDetectionResults()) {

                    ImageContainer ic =
ImageUtil.jpegImageToImageContainer(liveness2DetectionResults.getLivenessDete
ctionResults().getFaceDetectionResult().getJpegImage());

                    CropFace cropFace = new CropFace(ic);
                    ImageContainer icCropped =
cropFace.crop(liveness2DetectionResults.getLivenessDetectionResults().getFace
DetectionResult().getQuadrangle());
                    ic.release();
                    JpegImage croppedFacedJpeg =
ImageContainer.getJpegPhoto(icCropped);
                    icCropped.release();
                    resultObject = new ResultObject(croppedFacedJpeg,
LIVENESS2_DETECTION_RESULT);
                    Liveness2ResultModel liveness2ResultModel = new
Liveness2ResultModel(liveness2DetectionResults);

                    resultObject.setExtra(new
Gson().toJson(liveness2ResultModel));
                }
```

```
                    break;
                }

            }

            return resultObject;
        }
```

- The following is the ResultObject that was produced above. This is a comfortable data structure for reference purposes.

```
public class ResultObject {
    JpegImage jpegImage;
    int type = -1;
    String extra = "";

    public ResultObject(JpegImage jpegImage, int type) {
        this.jpegImage = jpegImage;
        this.type = type;
    }

    public int getType() {
        return type;
    }

    public String getExtra() {
        return extra;
    }

    public void setExtra(String extra) {
        this.extra = extra;
    }

    public Bitmap getPhotoBitmap(){
        Bitmap photo = null;
        if (jpegImage != null){
         photo = ImageUtil.jpegImageToBitmap(jpegImage);}
        jpegImage = null;
        return photo;
    }

}
```

AU10TIXIQCAMERAVISIONQUALITYEVENTS

Class: Au10tixIQCameraVisionQualityEvents.java

The Au10tixIQCameraVisionQualityEvents presents a custom extension of Au10tixCameraVision and its Builder. It lets the implementer receive ongoing quality detections and provides their own custom handling logic.

- This implementation extends Au10tixCameraVision and provides a quality listener for custom quality result indication.

```
public class Au10tixIQCameraVisionQualityEvents extends Au10tixCameraVision {
    static QualityDetector mQualityDetector;
    private boolean mUseQualityDetection;

    private Au10tixIQCameraVisionQualityEvents(Context context) {
        super(context);
        this.mUseQualityDetection = true;
        this.mQualityDetector = new QualityDetector();

    }

    @NonNull
    protected List<Detector> createDetector() {
        List<Detector> detectors = super.createDetector();
        if (this.mDetectionMode != 1) {
            return detectors;
        } else {
            if (this.mUseQualityDetection) {
                detectors.add(0, this.mQualityDetector);
            }
            return detectors;
        }
    }

    public static class Builder extends
com.senticore.au10tix.sdk.cameraVision.Au10tixCameraVision.Builder {
        public Builder(Context context) {
            super(context);
            this.mCameraVision = new
Au10tixIQCameraVisionQualityEvents(context);
        }


        @NonNull
        public Au10tixIQCameraVisionQualityEvents.Builder qualityPeriod(long
period) {
            ((Au10tixIQCameraVisionQualityEvents)
this.mCameraVision).mQualityDetector.setPeriod(period);
            return this;
        }
```

- Allows the implementer to receive ongoing quality detections and provide their own custom handling logic.

```
        @NonNull
        public Au10tixIQCameraVisionQualityEvents.Builder
setQualityListener(@NonNull QualityDetector.OnQualityDetectedListener
qListener) {
            mQualityDetector.setOnQualityDetectedListener(qListener);
            return this;
```

```
            }
        }
}
```