

Comments (7)DependenciesDuplications (0)Blocking (0)Resources (3)

Duplicate

 of [160869539](#)


BugP4

+ Add Hotlist

 STATUS UPDATE

No update yet.

Edit

 DESCRIPTION

ul...@gmail.com created issue [#1](#)

Please note: This component is for the CameraX API used in Jetpack. Please DO NOT file Pixel Camera issues here.

Please describe your issue and include details such as the version of CameraX you are using and any relevant logs related to your issue.

// If at all possible, capture an Android logcat (<https://developer.android.com/studio/command-line/logcat>) when you're experiencing the issue, preferably while the camera is still active.

CAMERAX VERSION (beta06) beta06

CAMERA APPLICATION NAME AND VERSION: (Settings > Apps > (app name) > version) com.apptasticnv.AiOneMerchant

ANDROID OS BUILD NUMBER: (Settings > About > Build number) NJH467F relase-keys

DEVICE NAME: (Nexus 5X, Samsung S6, etc) OnePlus One

DESCRIPTION: After processing a second CameraX analyzer image the app crashes and seems to SEGFAULT

LIST ANY EXPERIMENTAL FEATURES: (As an example - @ExperimentalCamera2Interop) @SuppressLint("UnsafeExperimentalUsageError") Image medialmage = image.getImage();

STEPS TO REPRODUCE: 1.Return a image from the Analyzer component 2.Return a second image from the analyzer component App seems to crash at step 2

OBSERVED RESULTS: App crashes when returning a second image from the imageAnalyzer

EXPECTED RESULTS: App returns image from the imageAnalyzer and showcases it to the user.

REPRODUCIBILITY: (5 of 5, 1 of 100, etc) 5 of 5 on certain devices, it seems to not crash on other devices like a Samsung Galaxy A20

ADDITIONAL INFORMATION:

CODE FRAGMENTS (this will help us troubleshoot your issues): Activity Code

```
package com.apptasticnv.aionemerchant.ui.authentication.mrz;

import android.Manifest;
import android.app.Activity;
import android.app.Dialog;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.util.Size;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.ProgressBar;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.camera.core.CameraSelector;
import androidx.camera.core.ImageAnalysis;
import androidx.camera.core.Preview;
import androidx.camera.lifecycle.ProcessCameraProvider;
import androidx.camera.view.PreviewView;
import androidx.core.content.ContextCompat;

import com.apptasticnv.aionemerchant.R;
import com.apptasticnv.aionemerchant.interfaces.MRZInterface;
import com.apptasticnv.aionemerchant.mrz.MRZAnalyzer;
import com.google.android.material.floatingactionbutton.FloatingActionButton;
import com.google.common.util.concurrent.ListenableFuture;
import com.karumi.dexter.Dexter;
import com.karumi.dexter.PermissionToken;
import com.karumi.dexter.listener.PermissionDeniedResponse;
import com.karumi.dexter.listener.PermissionGrantedResponse;
import com.karumi.dexter.listener.PermissionRequest;
import com.karumi.dexter.listener.single.PermissionListener;

import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
```

```

import java.util.concurrent.Executors;

import static com.apptasticnv.aionemerchant.helpers.StorageHelper.saveImageToInternalStorage;
import static java.lang.Math.round;

public class MRZActivity extends AppCompatActivity {
    //

    Intent returnIntent = new Intent();

    MRZInterface anInterface = new MRZInterface() {
        @Override
        public void returnMRZ(String MRZ, Uri mrzBitmap) {
            Log.d(TAG, "MRZ returned");
        }

        @Override
        public void returnFront(Uri front, boolean valid) {
            Log.d(TAG, "Front returned");

            returnIntent.putExtra("Front", front.toString());
            Log.d(TAG, "Front returned as: " + front );

            runOnUiThread(() -> showDialog("Attach as the front of ID?", front));
        }

        @Override
        public void returnBack(Uri back, boolean valid, String MRZ) {
            Log.d(TAG, "Back returned");

            returnIntent.putExtra("Back", back.toString());
            returnIntent.putExtra("MRZ", MRZ);

            Log.d(TAG, "Back returned as: " + back );

            try {
                runOnUiThread(() -> showDialog("Attach as the Back of ID?", back));
            } catch (Exception e) {
                e.printStackTrace();
                setResult(Activity.RESULT_OK, returnIntent);
                finish();
            }
        }

        @Override
        public void setLoadingState(boolean loading) {
            if (loading) {
                progressBar.setVisibility(View.VISIBLE);
            } else {
                progressBar.setVisibility(View.GONE);
            }
        }
    };

    Context context = this;
    private ExecutorService executor = Executors.newSingleThreadExecutor();
    private String TAG = this.getClass().getSimpleName();

    // States range from FRONT }} BACK }} MRZ
    // initialize as front
    private String state;

    private ListenableFuture<ProcessCameraProvider> cameraProviderFuture;

    PreviewView previewView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_mrz_capture_guided);

```

```

        previewView = findViewById(R.id.previewView);

        checkPermissions();

        findElements();

    }

    ProgressBar progressBar;
    TextView textViewHelper, textViewInstruction;
    View bottomContainer;
    FloatingActionButton fab;

    private void findElements() {
        progressBar = findViewById(R.id.progressBar);

        fab = findViewById(R.id.fab);
        fab.setOnClickListener(this::takePictureHandler);

        textViewInstruction = findViewById(R.id.textViewInstruction);
    }

    private void takePictureHandler(View view) {
        switch (state) {
            case "FRONT":
                analyzer.setAllowFrontPictureCapture(true);
                break;
            case "BACK":
                analyzer.setAllowBackPictureCapture(true);
                break;

            default:
                Log.wtf(TAG, "Sorry not supported");
                break;
        }
    }

    private void checkPermissions() {
        // dexter code
        Dexter.withActivity(this)
            .withPermission(Manifest.permission.CAMERA)
            .withListener(new PermissionListener() {
                @Override
                public void onPermissionGranted(PermissionGrantedResponse response) {
                    initCamera();
                }

                @Override
                public void onPermissionDenied(PermissionDeniedResponse response) {
                    finish();
                }

                @Override
                public void onPermissionRationaleShouldBeShown(PermissionRequest permission, PermissionToken token) { /* ... */ }
            }).check();
    }

    private void initCamera() {
        cameraProviderFuture = ProcessCameraProvider.getInstance(context);

        cameraProviderFuture.addListener(() -> {
            try {
                ProcessCameraProvider cameraProvider = cameraProviderFuture.get();
                startMRZCamera(cameraProvider);
            } catch (ExecutionException | InterruptedException e) {
                // No errors need to be handled for this Future.
                // This should never be reached.
            }
        }, ContextCompat.getMainExecutor(context));
    }

    MRZAnalyzer analyzer;

```

```

void startMRZCamera(@NonNull ProcessCameraProvider cameraProvider) {
    Preview preview = new Preview.Builder()
        .build();

    CameraSelector cameraSelector = new CameraSelector.Builder()
        .requireLensFacing(CameraSelector.LENS_FACING_BACK)
        .build();

    preview.setSurfaceProvider(previewView.createSurfaceProvider());

    ImageAnalysis imageAnalysis =
        new ImageAnalysis.Builder()
            .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
            .setTargetRotation(previewView.getDisplay().getRotation())
            .setTargetResolution(new Size(1920, 1080))
            .build();

    analyzer = new MRZAnalyzer(context, anInterface, state);

    imageAnalysis.setAnalyzer(executor, analyzer);

    cameraProvider.bindToLifecycle(this, cameraSelector, imageAnalysis, preview);

    state = "FRONT";
    handleState();
}

private void showDialog(String titleMessage, Uri bitmap) {
    Dialog dialog = new Dialog(this);
    dialog.setCancelable(true);

    View view = getLayoutInflater().inflate(R.layout.fragment_image_result, null);
    dialog.setContentView(view);

    final TextView title = view.findViewById(R.id.textView);
    final ImageView imageView = view.findViewById(R.id.imageView);

    final Button save = view.findViewById(R.id.buttonSave);
    final Button retake = view.findViewById(R.id.buttonRetake);

    imageView.setImageURI(bitmap);
    title.setText(titleMessage);

    save.setOnClickListener(viewSave -> {
        switch (state) {
            case "FRONT":
                state = "BACK";
                handleState();
                break;
            case "BACK":

                setResult(Activity.RESULT_OK, returnIntent);
                runOnUiThread(this::finish);

                break;

        }

        dialog.dismiss();
    });
    retake.setOnClickListener(viewClose -> dialog.dismiss());

    dialog.show();
}

private void handleState() {
    switch (state) {
        case "FRONT":
            analyzer.setState("FRONT");
            textViewInstruction.setText("Take a picture of the front of your ID card");

            break;
        case "BACK":
            analyzer.setState("BACK");
    }
}

```

```

        textViewInstruction.setText("Take a picture of the back of your ID card");

        break;
    }
}

}

```

## Analyzer Code

```

package com.apptasticnv.aionemerchant.mrz;

import android.annotation.SuppressLint;
import android.content.Context;
import android.graphics.Bitmap;
import android.media.Image;
import android.net.Uri;
import android.util.Log;

import androidx.annotation.NonNull;
import androidx.camera.core.ImageAnalysis;
import androidx.camera.core.ImageProxy;

import com.apptasticnv.aionemerchant.interfaces.MRZInterface;
import com.google.android.gms.tasks.OnFailureListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.firebase.ml.vision.FirebaseVision;
import com.google.firebase.ml.vision.common.FirebaseVisionImage;
import com.google.firebase.ml.vision.text.FirebaseVisionText;
import com.google.firebase.ml.vision.text.FirebaseVisionTextRecognizer;

import java.util.ArrayList;
import java.util.Objects;
import java.util.Timer;
import java.util.TimerTask;

import static com.apptasticnv.aionemerchant.helpers.ImageHelper.fauxBitmap;
import static com.apptasticnv.aionemerchant.helpers.StorageHelper.saveImageToInternalStorage;
import static com.apptasticnv.aionemerchant.mrz.BitmapToolkit.rotateBitmap;
import static com.apptasticnv.aionemerchant.mrz.BitmapToolkit.toBitmap;

public class MRZAnalyzer implements ImageAnalysis.Analyzer {

    private String TAG = this.getClass().getSimpleName();
    private Context context;
    private MRZInterface anInterface;
    private String MRZ = "Done";

    private BlockHandler blockHandler = new BlockHandler();

    public MRZAnalyzer(Context context, MRZInterface anInterface, String state) {
        this.context = context;
        this.anInterface = anInterface;
        this.state = state;
    }

    public void setState(String state) {
        this.state = state;
    }

    private Bitmap correctedImage;

    private boolean allowFrontPictureCapture = false;
    private boolean allowBackPictureCapture = false;

    public void setAllowFrontPictureCapture(boolean allowFrontPictureCapture) {
        this.allowFrontPictureCapture = allowFrontPictureCapture;
    }
}

```

```

public void setAllowBackPictureCapture(boolean allowBackPictureCapture) {
    this.allowBackPictureCapture = allowBackPictureCapture;
}

private String state;

@Override
public void analyze(@NonNull ImageProxy image) {

    switch (state) {
        case "FRONT":
            if (allowFrontPictureCapture) {
                allowFrontPictureCapture = false;

                @SuppressWarnings("UnsafeExperimentalUsageError") Image mediaImage = image.getImage();
                int rotationDegrees = image.getImageInfo().getRotationDegrees();
                Bitmap bitmapImage = toBitmap(Objects.requireNonNull(mediaImage));
                correctedImage = rotateBitmap(bitmapImage, rotationDegrees);

                correctedImage = fauxBitmap(correctedImage);
                Uri frontUri = saveImageToInternalStorage(context, correctedImage);

                anInterface.returnFront(frontUri, true);
            }

            new Timer().schedule(new TimerTask() {
                @Override
                public void run() {
                    image.close();
                }
            }, 250);
            break;

        case "BACK":
            if (allowBackPictureCapture) {
                allowBackPictureCapture = false;

                @SuppressWarnings("UnsafeExperimentalUsageError") Image mediaImage = image.getImage();
                int rotationDegrees = image.getImageInfo().getRotationDegrees();
                Bitmap bitmapImage = toBitmap(Objects.requireNonNull(mediaImage));
                correctedImage = rotateBitmap(bitmapImage, rotationDegrees);

                correctedImage = fauxBitmap(correctedImage);
                Uri backUri = saveImageToInternalStorage(context, correctedImage);
                anInterface.returnBack(backUri, true, this.MRZ);
            }

            Log.d(TAG, "Starting MRZ Processor");
            @SuppressWarnings("UnsafeExperimentalUsageError") Image mediaImage = image.getImage();
            int rotationDegrees = image.getImageInfo().getRotationDegrees();

            Bitmap bitmapImage = toBitmap(Objects.requireNonNull(mediaImage));
            correctedImage = rotateBitmap(bitmapImage, rotationDegrees);

            FirebaseVisionImage fImage = FirebaseVisionImage.fromBitmap(correctedImage);

            FirebaseVisionTextRecognizer detector = FirebaseVision.getInstance()
                .getOnDeviceTextRecognizer();

            detector.processImage(fImage)
                .addOnSuccessListener(successListener)
                .addOnFailureListener(failureListener);

            new Timer().schedule(new TimerTask() {
                @Override
                public void run() {
                    image.close();
                }
            }, 250);
            break;

        default:
            new Timer().schedule(new TimerTask() {
                @Override
                public void run() {

```

```
        image.close();
    }
    }, 250);

}

}

private OnSuccessListener<FirebaseVisionText> successListener = result -> {

    String resultText = result.getText();
    ArrayList<String> mrzBlocks = new ArrayList<>();
    for (FirebaseVisionText.TextBlock block : result.getTextBlocks()) {
        String blockText = block.getText();
        mrzBlocks.add(blockText);
    }

    if (mrzBlocks.size() != 0) {
        anInterface.setLoadingState(true);

        // call MRZ validator from SMRZ.
        String MRZ = blockHandler.process(mrzBlocks);
        if (MRZ != null && MRZ.length() > 59) {
            Log.wtf(TAG, "FOUND MRZ \n" + MRZ);
            this.MRZ = MRZ;
        }

        if (MRZ == null) {
            Log.i(TAG, "Starting Next Pass");
        } else {
        }

    }
};

private OnFailureListener failureListener = e -> Log.wtf("MRZAnalyzer", "Analysis failure.");

}
```



crash.txt

17 KB

[View](#)

[Download](#)

✓ Mentioned issues (1)    ✓ Links (2)

🔍 Mentioned issues (1)

P4   MLKit documentation missing critical information regarding how to close a image processing task.   ["https://issuetracker.google.com/162809057"](https://issuetracker.google.com/162809057)

↔ Links (2)

"// If at all possible, capture an Android logcat ( <https://developer.android.com/studio/command-line/logcat> ) when you're experiencing the issue, preferably while the camera is still active."

" ...d this error by enclosing the imageProxy.close() and mediaImage.close() into an OnCompleteListener attached to the process Task object. See details ↔ [here](#) . This does not appear to be a bug in

#### COMMENTS



**fu...@google.com** <fu...@google.com> [#2](#)

Hi, SIGSEGV may indicate a crash in a native layer of the application, similarly the jni-helper.cc logcat suggests this also.

This might indicate some issue in the processing of the frame instead of CameraX. It might also be some interaction - perhaps an unexpected image size or an image being returned/dealloc

To debug further, it might be worth trying to first remove the processing code, and simply access a few pixels in Java to test the images are being received/returned as expected, then add m



**fu...@google.com** <fu...@google.com> [#3](#)

*Reassigned to bu...@google.com.*

Bugjuggler: wait 2w



**bu...@google.com** <bu...@google.com> [#4](#)

Hi. I've received your bug and will wait until 2020-08-03 13:41 -0700 PDT and then assign the bug to [camerax-bugs@google.com](mailto:camerax-bugs@google.com).

to...@gmail.com <to...@gmail.com> #5

I'm getting a similar error with a bit simpler code and built almost entirely from example code blocks from the docs. Not sure if this is a CameraX or MLKIT issue though.

CAMERAX VERSION 1.0.0-beta07

CAMERA APPLICATION NAME AND VERSION: (Settings > Apps > (app name) > version) I guess this is referring to the default camera app and not the one I'm developing? If so, I can't find the

ANDROID OS BUILD NUMBER: (Settings > About > Build number) M4B30Z

DEVICE NAME: (Nexus 5X, Samsung S6, etc) Nexus 5

DESCRIPTION: Code crashes after a call to my BarcodeScanner object's process() method (i.e. scanner.process(image)) and then seems to SEGFAULT

LIST ANY EXPERIMENTAL FEATURES: (As an example - @ExperimentalCamera2Interop) @SuppressWarnings("UnsafeExperimentalUsageError") Image mediaImage = image.getImage();

STEPS TO REPRODUCE: Run app from Android Studio and will crash within a few seconds of opening on the phone.

OBSERVED RESULTS: If I set breakpoints within the analyze function, I can step through via "Resume Program (F9)" as many times as I want it seems, but if I step through with F8, it will crash

EXPECTED RESULTS: Does not crash, and returns a simple Log "BarCode Recognized" when presented a barcode.

REPRODUCIBILITY: (5 of 5, 1 of 100, etc) 5 of 5.

ADDITIONAL INFORMATION:

CODE FRAGMENTS (this will help us troubleshoot your issues): Activity Code

## Code

```
package jp.oist.cameraxapp;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.camera.core.Camera;
import androidx.camera.core.CameraSelector;
import androidx.camera.core.ImageAnalysis;
import androidx.camera.core.ImageCapture;
import androidx.camera.core.ImageProxy;
import androidx.camera.core.Preview;
import androidx.camera.lifecycle.ProcessCameraProvider;
import androidx.camera.view.PreviewView;
import androidx.core.content.ContextCompat;
import androidx.lifecycle.LifecycleOwner;

import android.annotation.SuppressLint;
import android.graphics.Point;
import android.graphics.Rect;
import android.media.Image;
import android.os.Bundle;
import android.util.Log;
import android.util.Size;
import android.widget.Toast;

import com.google.android.gms.tasks.OnFailureListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.android.gms.tasks.Task;
import com.google.common.util.concurrent.ListenableFuture;
import com.google.mlkit.vision.barcode.Barcode;
import com.google.mlkit.vision.barcode.BarcodeScanner;
import com.google.mlkit.vision.barcode.BarcodeScannerOptions;
import com.google.mlkit.vision.barcode.BarcodeScanning;
import com.google.mlkit.vision.common.InputImage;

import java.util.List;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executor;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class MainActivity extends AppCompatActivity {

    private ListenableFuture<ProcessCameraProvider> cameraProviderFuture;
    private ExecutorService executor;

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```



```

executor = Executors.newSingleThreadExecutor();

PreviewView previewView = findViewById(R.id.previewView);

cameraProviderFuture = ProcessCameraProvider.getInstance(this);

cameraProviderFuture.addListener(() -> {
    try {
        // Camera provider is now guaranteed to be available
        ProcessCameraProvider cameraProvider = cameraProviderFuture.get();

        // Set up the view finder use case to display camera preview
        Preview preview = new Preview.Builder().build();

        // Choose the camera by requiring a lens facing
        CameraSelector cameraSelector = new CameraSelector.Builder()
            .requireLensFacing(CameraSelector.LENS_FACING_FRONT)
            .build();

        // Connect the preview use case to the previewView
        preview.setSurfaceProvider(
            previewView.createSurfaceProvider());

        // Set up the capture use case to allow users to take photos
        ImageCapture imageCapture = new ImageCapture.Builder()
            .setCaptureMode(ImageCapture.CAPTURE_MODE_MINIMIZE_LATENCY)
            .build();

        ImageAnalysis imageAnalysis =
            new ImageAnalysis.Builder()
                .build();

        imageAnalysis.setAnalyzer(ContextCompat.getMainExecutor(this), new ImageAnalysis.Analyzer() {
            private BarcodeScanner scanner = buildBarCodeScanner();

            @Override
            public void analyze(ImageProxy imageProxy) {
                @SuppressWarnings("UnsafeExperimentalUsageError") Image mediaImage = imageProxy.getImage();
                if (mediaImage != null) {
                    InputImage image =
                        InputImage.fromMediaImage(mediaImage, imageProxy.getImageInfo().getRotationDegrees());
                    // Pass image to an ML Kit Vision API
                    Task<List<Barcode>> result = scanner.process(image)
                        .addOnSuccessListener(new OnSuccessListener<List<Barcode>>() {
                            @Override
                            public void onSuccess(List<Barcode> barcodes) {
                                // Task completed successfully
                                Log.i("CameraXApp3", "Barcode Recognized");
                                processBarCodes(barcodes);
                            }
                        })
                        .addOnFailureListener(new OnFailureListener() {
                            @Override
                            public void onFailure(@NonNull Exception e) {
                                // Task failed with an exception
                            }
                        });
                    mediaImage.close();
                    imageProxy.close();
                }
            }

            private BarcodeScanner buildBarCodeScanner() {
                BarcodeScannerOptions options =
                    new BarcodeScannerOptions.Builder()
                        .setBarcodeFormats(
                            Barcode.FORMAT_QR_CODE)
                        .build();
                return BarcodeScanning.getClient(options);
            }

            //
            private void processBarCodes(List<Barcode> barcodes) {
                for (Barcode barcode : barcodes) {
                    String rawValue = barcode.getRawValue();
                    int valueType = barcode.getValueType();
                    // See API reference for complete list of supported types
                    if (valueType == Barcode.TYPE_TEXT) {
                        Toast.makeText(MainActivity.this, "Received Message:" + rawValue, Toast.LENGTH_SHORT).show();
                    }
                }
            }
        });
    }
}

```

```

    });
}

// Attach use cases to the camera with the same lifecycle owner
Camera camera = cameraProvider.bindToLifecycle(
    ((LifecycleOwner) this),
    cameraSelector,
    preview,
    imageCapture,
    imageAnalysis);

} catch (InterruptedException | ExecutionException e) {
    // Currently no exceptions thrown. cameraProviderFuture.get() should
    // not block since the listener is being called, so no need to
    // handle InterruptedException.
}

}, ContextCompat.getMainExecutor(this));
}

}

```

Logcat output below:

```

08-03 17:50:22.073 20115-20136/jp.oist.cameraxapp A/libc: Fatal signal 11 (SIGSEGV), code 1, fault addr 0x48 in tid 20136 (pool-2-thread-1)
08-03 17:50:22.084 215-20164/? E/mmm-camera: module_faceproc_port_event_func:886] MCT_EVENT_MODULE_BUF_DIVERT 131074, Cannot start FD, active 20003, framei
08-03 17:50:22.117 215-20164/? E/mmm-camera: module_faceproc_port_event_func:886] MCT_EVENT_MODULE_BUF_DIVERT 131074, Cannot start FD, active 20003, framei
08-03 17:50:22.150 215-20164/? E/mmm-camera: module_faceproc_port_event_func:886] MCT_EVENT_MODULE_BUF_DIVERT 131074, Cannot start FD, active 20003, framei
08-03 17:50:22.175 195-195/? A/DEBUG: *** **
08-03 17:50:22.175 195-195/? A/DEBUG: Build fingerprint: 'google/hammerhead/hammerhead:6.0.1/M4B30Z/3437181:user/release-keys'
08-03 17:50:22.175 195-195/? A/DEBUG: Revision: '11'
08-03 17:50:22.175 195-195/? A/DEBUG: ABI: 'arm'
08-03 17:50:22.175 195-195/? A/DEBUG: pid: 20115, tid: 20136, name: pool-2-thread-1 >>> jp.oist.cameraxapp <<<
08-03 17:50:22.175 195-195/? A/DEBUG: signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 0x48
08-03 17:50:22.184 215-20164/? E/mmm-camera: module_faceproc_port_event_func:886] MCT_EVENT_MODULE_BUF_DIVERT 131074, Cannot start FD, active 20003, framei
08-03 17:50:22.188 195-195/? A/DEBUG: r0 00000000 r1 b6d39ec0 r2 0000010f r3 00000001
08-03 17:50:22.188 195-195/? A/DEBUG: r4 00000023 r5 ab456200 r6 9f7592dc r7 00000000
08-03 17:50:22.188 195-195/? A/DEBUG: r8 00000000 r9 aeefe2800 s1 12ef88e0 fp 20000000
08-03 17:50:22.188 195-195/? A/DEBUG: ip 00000005 sp 9f7592b0 lr b496deb9 pc ab27e4d8 cpsr 200b0030
08-03 17:50:22.194 195-195/? A/DEBUG: backtrace:
08-03 17:50:22.194 195-195/? A/DEBUG: #00 pc 0001b4d8 /system/lib/libmedia_jni.so
08-03 17:50:22.194 195-195/? A/DEBUG: #01 pc 741830cd /data/dalvik-cache/arm/system@framework@boot.oat (offset 0xled6000)
08-03 17:50:22.217 215-20164/? E/mmm-camera: module_faceproc_port_event_func:886] MCT_EVENT_MODULE_BUF_DIVERT 131074, Cannot start FD, active 20003, framei
08-03 17:50:22.251 215-20164/? E/mmm-camera: module_faceproc_port_event_func:886] MCT_EVENT_MODULE_BUF_DIVERT 131074, Cannot start FD, active 20003, framei
08-03 17:50:22.284 215-20164/? E/mmm-camera: module_faceproc_port_event_func:886] MCT_EVENT_MODULE_BUF_DIVERT 131074, Cannot start FD, active 20003, framei
08-03 17:50:22.317 215-20164/? E/mmm-camera: module_faceproc_port_event_func:886] MCT_EVENT_MODULE_BUF_DIVERT 131074, Cannot start FD, active 20003, framei
08-03 17:50:22.666 195-195/? A/DEBUG: Tombstone written to: /data/tombstones/tombstone_05
08-03 17:50:22.666 195-195/? E/DEBUG: AM write failed: Broken pipe
08-03 17:50:22.687 198-20190/? E/Surface: queueBuffer: error queuing buffer to SurfaceTexture, -32
08-03 17:50:22.687 198-20190/? E/Camera3-OutputStream: returnBufferCheckedLocked: Stream 1: Error queueing buffer to native window: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Camera3-Device: Can't return buffer to its stream: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Surface: queueBuffer: error queuing buffer to SurfaceTexture, -32
08-03 17:50:22.687 198-20190/? E/Camera3-OutputStream: returnBufferCheckedLocked: Stream 1: Error queueing buffer to native window: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Camera3-Device: Can't return buffer to its stream: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Surface: queueBuffer: error queuing buffer to SurfaceTexture, -32
08-03 17:50:22.687 198-20190/? E/Camera3-OutputStream: returnBufferCheckedLocked: Stream 1: Error queueing buffer to native window: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Camera3-Device: Can't return buffer to its stream: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Surface: queueBuffer: error queuing buffer to SurfaceTexture, -32
08-03 17:50:22.687 198-20190/? E/Camera3-OutputStream: returnBufferCheckedLocked: Stream 1: Error queueing buffer to native window: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Camera3-Device: Can't return buffer to its stream: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Surface: queueBuffer: error queuing buffer to SurfaceTexture, -32
08-03 17:50:22.687 198-20190/? E/Camera3-OutputStream: returnBufferCheckedLocked: Stream 1: Error queueing buffer to native window: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Camera3-Device: Can't return buffer to its stream: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Surface: queueBuffer: error queuing buffer to SurfaceTexture, -32
08-03 17:50:22.687 198-20190/? E/Camera3-OutputStream: returnBufferCheckedLocked: Stream 1: Error queueing buffer to native window: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Camera3-Device: Can't return buffer to its stream: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Surface: queueBuffer: error queuing buffer to SurfaceTexture, -32
08-03 17:50:22.687 198-20190/? E/Camera3-OutputStream: returnBufferCheckedLocked: Stream 1: Error queueing buffer to native window: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Camera3-Device: Can't return buffer to its stream: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Surface: queueBuffer: error queuing buffer to SurfaceTexture, -32
08-03 17:50:22.687 198-20190/? E/Camera3-OutputStream: returnBufferCheckedLocked: Stream 1: Error queueing buffer to native window: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Camera3-Device: Can't return buffer to its stream: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Surface: queueBuffer: error queuing buffer to SurfaceTexture, -32
08-03 17:50:22.687 198-20190/? E/Camera3-OutputStream: returnBufferCheckedLocked: Stream 1: Error queueing buffer to native window: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Camera3-Device: Can't return buffer to its stream: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Surface: queueBuffer: error queuing buffer to SurfaceTexture, -32
08-03 17:50:22.687 198-20190/? E/Camera3-OutputStream: returnBufferCheckedLocked: Stream 1: Error queueing buffer to native window: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Camera3-Device: Can't return buffer to its stream: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Surface: queueBuffer: error queuing buffer to SurfaceTexture, -32
08-03 17:50:22.687 198-20190/? E/Camera3-OutputStream: returnBufferCheckedLocked: Stream 1: Error queueing buffer to native window: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Camera3-Device: Can't return buffer to its stream: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Surface: queueBuffer: error queuing buffer to SurfaceTexture, -32
08-03 17:50:22.687 198-20190/? E/Camera3-OutputStream: returnBufferCheckedLocked: Stream 1: Error queueing buffer to native window: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Camera3-Device: Can't return buffer to its stream: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Surface: queueBuffer: error queuing buffer to SurfaceTexture, -32
08-03 17:50:22.687 198-20190/? E/Camera3-OutputStream: returnBufferCheckedLocked: Stream 1: Error queueing buffer to native window: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Camera3-Device: Can't return buffer to its stream: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Surface: queueBuffer: error queuing buffer to SurfaceTexture, -32
08-03 17:50:22.687 198-20190/? E/Camera3-OutputStream: returnBufferCheckedLocked: Stream 1: Error queueing buffer to native window: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Camera3-Device: Can't return buffer to its stream: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Surface: queueBuffer: error queuing buffer to SurfaceTexture, -32
08-03 17:50:22.687 198-20190/? E/Camera3-OutputStream: returnBufferCheckedLocked: Stream 1: Error queueing buffer to native window: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Camera3-Device: Can't return buffer to its stream: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Surface: queueBuffer: error queuing buffer to SurfaceTexture, -32
08-03 17:50:22.687 198-20190/? E/Camera3-OutputStream: returnBufferCheckedLocked: Stream 1: Error queueing buffer to native window: Broken pipe (-32)
08-03 17:50:22.687 198-20190/? E/Camera3-Device: Can't return buffer to its stream: Broken pipe (-32)
08-03 17:50:22.687
```

[illegible]

[illegible]







to...@gmail.com <to...@gmail.com> [#6](#)

I have resolved this error by enclosing the imageProxy.close() and mediaImage.close() into an OnCompleteListener attached to the process Task object. See details [↗here](#). This does not ap  
[↗here](#).



fu...@google.com <fu...@google.com> [#7](#)

Status: Duplicate of [160869539](#)

Glad the issue was resolved, thanks. We're also discussing ideas on making the interaction between MLKit/CameraX a bit easier. Thanks for filing the documentation bug.