FLIR Atlas Android SDK migration guide

Migration from FLIR One Android SDK (versions 3.+)

Teledyne Confidential; Commercially Sensitive Business Data

Table of Contents

Changes history

Change	Version
Initial version	0.1
Added missing functionalities	0.2
Refactored hardly readable tables into sections	0.3
Text formatting unification and minor corrections	0.4
Add support for bookmarks navigation in PDF reader	0.5
Update code snippets to reflect API changes	0.6
Update name to FLIR Atlas Android SDK	1.0

Introduction

Welcome to a developers guide helping to migrate from FLIR One Android SDK (versions 3.+) to much more powerful and feature rich **FLIR Atlas Android SDK**.

This document will introduce you to developing apps with FLIR One accessory for Android (aka the FLIR One device) using **FLIR Atlas Android SDK** (a successor of FLIR One Android SDK).

The intention of this document is to serve as a quick reference for developers previously using FLIR One Android SDK. API mappings between SDKs will be served in form of a look up table with code examples for **FLIR Atlas Android SDK**.

Note that **FLIR Atlas Android SDK** features set is much wider comparing to the FLIR One Android SDK. This document describes only part of available APIs. Refer to **FLIR Atlas Android SDK** official documentation and samples for full features overview.

In the next sections you can read about features, which were supported by FLIR One Android SDK (version 3.+) and how to implement them using **FLIR Atlas Android SDK**. You can jump between sections using convenient bookmarks navigation in your PDF reader.

Teledyne Confidential; Commercially Sensitive Business Data

Discover USB devices

FLIR One Android SDK

- *Device.startDiscovery()*
- *Device.stopDiscovery()*

FLIR Atlas Android SDK

- *DiscoveryFactory.scan()*
- *DiscoveryFactory.stop()*

Code example (using FLIR Atlas Android SDK)

```
DiscoveryEventListener aDiscoveryEventListener = new DiscoveryEventListener() {

    @Override
    void onCameraFound(Identity identity) {
        // identity describes a device and is used to connect to device
    }
};
DiscoveryFactory.getInstance().scan(aDiscoveryEventListener,
CommunicationInterface.USB);
```

Additional comments

Old *Device.startDiscovery()* automatically performs a connection, while new *DiscoveryFactory.scan()* just finds a device.

Teledyne Confidential; Commercially Sensitive Business Data

Connect to device

FLIR One Android SDK

Connection is done automatically after successful discovery, but it is possible to use:

- *UsbCommunicator.connect()*
- *UsbCommunicator.pause()*
- *UsbCommunicator.close()*

FLIR Atlas Android SDK

- *Camera.connect()*
- *Camera.disconnect()*

Code example (using FLIR Atlas Android SDK)

```
// with given Identity from DiscoveryEventListener.onCameraFound() callback
Camera cameraInstance = new Camera();
ConnectionStatusListener listener = new ConnectionStatusListener() {
    @Override
    onConnectionStatusChanged(ErrorCode error) {
         // error are reported here
    }
};
// calling connect() is blocking
cameraInstance.connect(identity, listener);
```

Additional comments

Connecting to device does not start live streaming automatically.

Stream frames

FLIR One Android SDK

- *Device.startFrameStream()*

FLIR Atlas Android SDK

- *Camera.getStreams()*
- *Stream*
- *ThermalStreamer*
- *ThermalStreamer.withThermalImage()*

Code example (using FLIR Atlas Android SDK)

```
// after being connected: ConnectionStatusListener.onConnectionStatusChanged()
with ConnectionStatus.CONNECTED
final Stream stream = camera.getStreams().get(0);
if (stream.isThermal()) {
    final ThermalStreamer streamer = new ThermalStreamer(stream);
    stream.start(arg -> {
        // using Streamer.withThermalImage () function is thread-safe
        streamer.withThermalImage(thermalImage ->
        {
           // thread-safe access to thermalImage instance
           // new incoming frame will not interfere with data on which we operate
        });
    });
}
```

Additional comments

Image is available when *Stream.onReceived()* is called. Inside you use: Streamer.withThermalImage*(ThermalImage thermalImage)* for a thread-safe access to the *ThermalImage*.

Render received frame

FLIR One Android SDK

- *FrameProcessor.processFrame()*
- *Device.StreamDelegate.onFrameReceived()*

And get actual image using:

- *FrameProcessor.Delegate.onFrameProcessed()*

FLIR Atlas Android SDK

- *Camera.getStreams()*
- *Stream*
- *ThermalStreamer*
- *ThermalStreamer.withThermalImage()*

Code example (using FLIR Atlas Android SDK)

```
ImageView streamPreview = ...
final Stream stream = camera.getStreams().get(0);
if (stream.isThermal()) {
    final ThermalStreamer streamer = new ThermalStreamer(stream);
    stream.start(arg -> {
        streamer.withThermalImage(thermalImage ->
        {
            // do something with the ThermalImage, i.e.:
            JavaImageBuffer javaBuffer = thermalImage.getImage();
            android.graphics.Bitmap bmp =
BitmapAndroid.createBitmap(javaBuffer).getBitMap();
            // refresh UI with the new android.graphics.Bitmap
            streamPreview.setBitmap(bmp); // call this on UI thread!
        });
    });
}
```

Get and set parameters for incoming frame (i.e. palette, rotation, emissivity)

FLIR One Android SDK

Using FrameProcessor class, i.e.:

- *FrameProcessor.setImagePalette()*

FLIR Atlas Android SDK

Directly on the given ThermalImage, i.e.:

- *ThermalImage.setPalette()*

Code example (using FLIR Atlas Android SDK)

```
// get one of the available palettes
Palette palette = PaletteManager.getDefaultPalettes().get(0);
// under Stream.start(OnReceived.onReceived)
streamer.withThermalImage(thermalImage -> {
    thermalImage.setMsx(); // fusion mode MSX with default alpha
    thermalImage.setPalette(palette);
    ImageParameters params = thermalImage.getImageParameters();
    params.setEmissivity(0.91);
    thermalImage.setRotation(RotationAngle.ROTATION_ANGLE_90_DEGREES);
    // then render image, save image (take snapshot), etc.
}
```

Read image information (i.e. width)

FLIR One Android SDK

- *Device.StreamDelegate.onFrameReceived()*
- *FrameProcessor.Delegate.onFrameProcessed()*
- *RenderedImage class*

FLIR Atlas Android SDK

Directly from the given *ThermalImage*, i.e.:

- *ThermalImage.getImage().width*

Code example (using FLIR Atlas Android SDK)

```
// under Stream.start(OnReceived.onReceived)
streamer.withThermalImage(thermalImage -> {
    JavaImageBuffer javaBuffer = thermalImage.getImage();
    // read image width using: javaBuffer.width
}
```

Send data to device

FLIR One Android SDK

- *UsbCommunicator class*

FLIR Atlas Android SDK

- *N/A*

Additional comments

 Not implemented. Basically this is required for FW update, which is not covered by FLIR Atlas Android SDK (as far as now).

Validate device

FLIR One Android SDK

- *UsbCommunicator class*

FLIR Atlas Android SDK

- *N/A*

Additional comments

 Not exposed. Such validation is done internally by the SDK during discovery process, so only FLIR devices will show up.

Save image (take snapshot)

FLIR One Android SDK

- *RenderedImage.getFrame().save()*

FLIR Atlas Android SDK

- *ThermalImage.saveAs()*

Code example (using FLIR Atlas Android SDK)

```
// under Stream.start(OnReceived.onReceived)
streamer.withThermalImage(thermalImage -> {
    thermalImage.saveAs("destination_absolute_path");
}
```

Tuning/NUC

FLIR One Android SDK

- *Device.performTuning()*
- *Device.setAutomaticTuning()*
- *Device.Delagate.onTuningStateChanged()*
- *Device.Delegate.onAutomaticTuningChanged()*

FLIR Atlas Android SDK

- *Camera.getRemoteControl()*
- *RemoteControl.getCalibration()*
- *Calibration.nuc()*
- *Calibration.subscribeCalibrationState()*

We miss the following features:

1. *TuningState* - currently we only get active/inactive in the *NucStateListener* callback. While FLIR One Android SDK has *TuningStates:*
    a. *TuningRequired*
    b. *InProgress*
    c. *Tuned*
    d. *TuningSuggested*
    e. *ApproximatelyTuned*
    f. *Unknown*
2. Set automatic tuning on/off – currently it defaults to ON without possibility to turn it off

Code example (using FLIR Atlas Android SDK)

```
Camera cameraInstance = new Camera();
ConnectionStatusListener listener = ...
cameraInstance.connect(identity, listener);
// after connection is established
RemoteControl remoteControl = cameraInstance.getRemoteControl();
if(remoteControl != null) { // check if device supports RemoteControl
    Calibration calibration = remoteControl.getCalibration();
    if(calibration != null) { // check if device supports Calibration
        // register listener
        Calibration.NucStateListener nucListener = ...
        calibration.subscribeCalibrationState(nucListener);
        // perform NUC - this will trigger the NucStateListener event
        calibration.nuc();
    }
}
```

Additional comments

 Missing *TuningState* enum. Missing *setAutomaticTuning()* method.

Accuracy bound (upper/lower)

FLIR One Android SDK

Accuracy bound is not really useful, because it is not related to the sensor/camera used. It is hardcoded to 273.15K (lower) and 393.15K (upper).

- *Device.getUpperAccuracyBound()*
- *Device.getLowerAccuracyBound()*

FLIR Atlas Android SDK

We introduced more useful feature, which describes the current image temperature range.

- *Scale.getRangeMin()*
- *Scale.getRangeMax()*

Code example (using FLIR Atlas Android SDK)

```
// ThermalImage from obtained streaming or an opened thermal file
ThermalImage image = ...
double lowerTemp = image.getScale().getRangeMin();
double upperTemp = image.getScale().getRangeMax();
```

Additional comments


Accuracy bound as in FLIR One Android SDK is not implemented, but it was not useful for the users. Instead we introduced temperature range for the given image/frame.

Camera name and serial number

FLIR One Android SDK

- *Device.getModelName()*
- *Device.getSerialNumber()*

FLIR Atlas Android SDK

- *Camera.getRemoteControl()*
- *RemoteControl.getCameraInformation()*
- *CameraInformation.name*
- *CameraInformation.serialNumber*

Code example (using FLIR Atlas Android SDK)

```
// assuming cameraInstance is a Camera object in connected state
RemoteControl remoteControl = cameraInstance.getRemoteControl();
if(remoteControl != null) { // check if device supports RemoteControl
    CameraInformation cameraInfo = remoteControl.getCameraInformation();
    if(cameraInfo != null) { // check if device supports CameraInformation
        String modelName = cameraInfo.name;
        String serialNumber = cameraInfo.serialNumber;
    }
}
```

Torch

FLIR One Android SDK

- *Device.hasTorch()*
- *Device.setTorchMode()*

FLIR Atlas Android SDK

- *N/A*

Additional comments

 Valid only for CAT S60/S61 phones, which are currently not supported by FLIR Atlas Android SDK.

Battery

FLIR One Android SDK

- *Device.hasBattery()*
- *Device.PowerUpdateDelegate. onBatteryChargingStateReceived()*
- *Device.PowerUpdateDelegate. onBatteryPercentageReceived()*

FLIR Atlas Android SDK

- *Camera.getRemoteControl()*
- *RemoteControl.getBattery()*
- *Battery.getChargingState()*
- *Battery.subscribeChargingState()*
- *Battery.getPercentage()*
- *Battery.subscribePercentage()*

Code example (using FLIR Atlas Android SDK)

```
// assuming cameraInstance is a Camera object in connected state
RemoteControl remoteControl = cameraInstance.getRemoteControl();
if(remoteControl != null) { // check if device supports RemoteControl
    Battery battery = remoteControl.getBattery();
    if(battery != null) { // check if device supports Battery
        // read out battery status and percentage once
        Battery.ChargingState state = battery.getChargingState();
        int percentageLeft = battery.getPercentage();
        // subscribe for updates related with battery state and percentage left
        // assuming Battery.BatteryStateListener instance is stateListener
        // assuming Battery.BatteryPercentageListener instance is
percentageListener
        battery.subscribeChargingState(stateListener);
        battery.subscribePercentage(percentageListener);
    }
}
```

CAT S60/S61

FLIR One Android SDK

CAT phones are supported and user can access their IR camera in the similar way like FLIR One external accessory.

FLIR Atlas Android SDK

- *N/A*

Additional comments

 CAT phones are not supported. Using FLIR Atlas Android SDK on them may result in undefined behavior.