Android Platform Guide – FLIR One/Cat S60

0. Prerequisites

Much of this guide assumes some familiarity with Android application development, the Java programming language, and common computer image storage and processing techniques. The following guides will prove helpful for those less familiar with these topics:

- Android Developers Portal
- Get Android Studio
- Android Bitmap Class

Introduction

Welcome to FLIR One Android app development. This document will introduce you to developing apps with the FLIR One accessory for Android (aka the FLIR One device). Note that this is not to be confused with the FLIR One Android app (aka the FLIR One app). New to this version of the SDK is support for the embedded FLIR thermal camera in the Cat® S60 smartphone.

This document is intended to serve as a quick start guide. It will introduce the developer to all of the important terms and methodologies related to the SDK. This document will help give an overall understanding of how the SDK works. Answers to commonly asked questions are also included.
For any support questions, contact FLIROneSDK@FLIR.com.

The FLIR One Android SDK allows you to receive, save, and load Frames from the camera in formats including MSX, Colorized thermal, and radiometric (kelvin) formats.

Release Notes

SDK 1.2.5

Changes:

- New VisibleUnalignedYUV888Image format replaces deprecated VisualJPEGImage
- Fixes issue of corrupt video would sometimes be seen after the FLIR One is disconnected and reconnected
- License violation check feature

Known issues: see version 1.2.2

SDK 1.2.4

Changes:

- Fixes crash when rendering frames on some devices.

Known issues: see version 1.2.2

SDK 1.2.3

Changes:

- Fixes crash that can occur when the FLIR One is disconnected and reconnected.
- Updates Android SDK and build tools versions used to generate the SDK module.

Known issues: see version 1.2.2

SDK 1.2.2

Changes:

- Resolves performance issues when rendering multiple image formats per frame.
- Updates libpng to fix security issues
- Fixes incorrect type reported for ThermalRGBA8888Image
- Adds new convenience method `RenderedImage#thermalPixelValues` for getting integer values for each thermal pixel
- Fixes signed short overflow issue with spot meter in example app

Known Issues:

- A race condition exists when saving and loading frames: do not immediately save and load frames in quick succession.

SDK 1.2

Changes:

- Supports the Cat® S60 Smartphone's embedded FLIR device.

Known Issues:

- Performance can be poor when rendering multiple image formats per frame such as VisibleAlignedRGBA8888Image and ThermalRGBA8888Image.

SDK 1.1.1

Changes:

- Fixes an issue that required an instance of the FrameProcessor class to be created to prevent a crash. **Note:** the constructor for the SimulatedDevice class has been changed as part of this fix.
  Known Issues:
- Performance can be poor when rendering multiple image formats per frame such as VisibleAlignedRGBA8888Image and ThermalRGBA8888Image.
- A race condition exists when saving and loading frames: do not immediately save and load frames in quick succession.

SDK 1.1

Changes:

- Deprecates the VisualJPEGImage and VisualYCbCr888Image formats, please use the new VisibleAlignedRGBA8888Image format.
- New convenience method `RenderedImage#thermalPixelData` for getting short pixels for thermal image types

Known Issues:

- Performance can be poor when rendering multiple image formats per frame such as VisibleAlignedRGBA8888Image and ThermalRGBA8888Image.
- A race condition exists when saving and loading frames: do not immediately save and load frames in quick succession.
- An instance of the FrameProcessor class is required to prevent a crash when connecting to a device

1. Initial Project Setup

Development environment requirements

The SDK and Example App have been developed with Android Studio 1.2 or later in mind.
Use of older versions of Android Studio or Eclipse is not supported.

Supported target hardware

The SDK can be run on any Android device with an ARMv7 CPU and Android 4.0 or greater.
In order to connect to a FLIR One device, the Android device must support USB host mode.
The SDK can be used on Android devices without USB host mode support, as well as Android Virtual Devices, but use is limited to the simulator and processing frames.

For a complete listing of supported devices, please refer to the FLIR One website.

Using the Android Library (AAR) file in a new or existing project

You can import the `flironesdk.aar` file in a new or existing Android Studio app project.

This file is required and must be downloaded from FLIR. The steps required to install these files along with any other project dependencies are included below.
All required external files are likely to be provided at the same place as this document. For any support questions, contact FLIROneSDK@FLIR.com.

You must import the `flironesdk.aar` library file into your Android Studio project. This is most easily done in a 2 step process:
1. Use "Import .JAR or .AAR Package" wizard (via `File→New→New Module`) and select the `flironesdk.aar`
2. Add the flironesdk module as a dependency of your app module: select `File→Project Structure` , select you app module, go to the Dependancies tab, click the plus (+) icon and select "Module Dependency" to select the flironesdk module.

Note: The above method of importing the flironesdk.aar module requires Android Studio 1.2 or greater.

Using the bundled Example App as a starting point

It is highly recommended that developers examine the source code of the example app as a first point of reference. The example app can also be used as the base for a new app, since all the basic requirements of a FLIR One app are present.

2. Getting Up and Running: Streaming Frames

Getting a Device Object

The first step to receiving frames from the device is to implement the Device.Delegate interface, and use `Device.startDiscovery` to tell the device discovery service to check for a supported device being connected. This includes the FLIR One, the embedded device in a Cat® S60, and any future devices supported by this SDK.
The Device.Delegate interface defines methods to be called for device-specific events.

Device.startDiscovery

In your main activity's onResume method, call Device.startDiscovery and in onPause call Device.stopDiscovery()

Your device delegate that you pass to startDiscovery will have its onDeviceConnected method called when a FLIR One is attached to your phone and the user has given your app permission to use it.

See example app as a guide.

Application Manifest Device Filter

If you want to prompt the user to open your app when the device is connected, add the following to your AndroidManifest.xml

```
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <meta-data android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
        android:resource="@xml/device_filter" />
```

And add a file `res/xml/device_filter.xml` with

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <usb-device vendor-id="2507" product-id="6550" />
</resources>
```

Differences with Cat® S60

When running on the Cat® S60, the SDK will not connect to a FLIR One when using Device.startDiscovery. The SDK also opens the visible camera for you, asking the user for permission if needed. Your app may need to stop and start the frame stream in your activity's onRequestPermissionsResult so that the visible camera gets started. Alternatively, you can check for and handle the camera permission before calling startFrameStream. Because the SDK has an image capture session open for the camera, if you wish to turn on the flashlight, use the `Device#setTorchMode` method if `Device#hasTorch` returns true. `Device#hasTorch` returns false, when the device is a FLIR One, and you may use the Android Camera API to turn on the flashlight in the torch mode.

Implementing Callbacks and Rendering

The SDK utilizes a delegation pattern for passing data from the device to your code, as well as for passing processed frames to your code.

Following the same pattern as `Device.startDiscovery` , the `startFrameStream` method requires a `Device.StreamDelegate` instance to be passed. This delegate will handle the receiving of Frame objects which represent the raw data received from the device on each frame. Raw frames are rendered to usable image formats by an instance of the `FrameProcessor` class.

Note: Multiple frame processors can be created, and each can be configured with multiple frame formats, but keep in mind that processing multiple frame types and palettes requires high performance hardware.

Implement a `Device.StreamDelegate` instance with the `onFrameReceived` method to receive Frame objects in real time from the device. Then create an instance of the `FrameProcessor` class with your implementation of a `FrameProcessor.Delegate` and a set of rendered frame types and a palette. For a simple live stream application, your `onFrameReceived` method should pass the Frame object to your configured `FrameProcessor` instance.

Using the SimulatedDevice

The `SimulatedDevice` class can be instantiated to test your code. To use the SimulatedDevice class, copy the sampleframes.zip raw resource from the example app, and create an instance of SimulatedDevice like so:

```
new SimulatedDevice(deviceDelegate, getContext(), getResources().openRawResource(R.raw.sampleframes), 100)
;
```

The simulated device will call your device delegate callback methods just like a physical device.

There is one caveat: the simulated device does not have calibration data, so frames are not aligned and `ThermalRadiometricKelvinImage` will not have realistic temperature values.

Omitting sampleframes.zip from Release Builds

If you don't use the SimulatedDevice class in your release build, you can reduce your apk size by only including the sampleframes.zip raw resource in your debug builds. One way to do this is by placing the sampleframes.zip in the debug resoueces directory:

```
app/src/debug/res/raw/sampleframes.zip
```

and creating a zero-byte file in your main resources directory with the same resource name:

```
app/src/main/res/raw/sampleframes.fake
```

Remember that raw resource names ignore the file extension, so you can name the file with an extension that helps clarify its purpose.

Rendering and Displaying Frames

`FrameProcessor.Delegate.onFrameProcessed` will be called with `RenderedImage` objects, each indicating its type with the `imageType` method. The type names indicate the pixel format, and all can be easily made into Android `android.graphics.Bitmap` objects using the `getBitmap` method, or by using `Bitmap.createBitmap` with the `pixelData()` method.

In order to receive different formats, use the `setFrameTypes` method with a Set of as many formats as you want.
Warning: you may not use the `BlendedMSXRGBA8888Image` and `ThermalRGBA8888Image` types concurrently.

Note: while not critical, it is recommended to use an EnumSet when calling `setFrameTypes`

Descriptions of Rendered Frame Types

`ThermalLinearFlux14BitImage`

Linear 14 bit image data, padded to 16 bits per pixel. This is the raw image from the thermal image sensor.

`ThermalRGBA8888Image`

Thermal RGBA image data, with a palette applied.

`BlendedMSXRGBA8888Image`

MSX (thermal + visual) RGBA image data, with a palette applied. This shows an outline of objects using the visible light camera, overlaid on the thermal image.

`ThermalRadiometricKelvinImage`

Radiometric centi-kelvin (cK) temperature data. Note that is is in centi-kelvin, so a reading of 31015 is equal to 310.15K (98.6ºF or 37ºC). Since each pixel is a 16 bit short type, keep in mind that Java shorts are signed, but the source data was unsigned. Use the `thermalPixelValues` method to get the pixels as int values to prevent sign overflow.

`VisibleAlignedRGBA8888Image`

Visible-light RGBA image data, **aligned** with the thermal image. This image has been cropped and adjusted to line up with the thermal image.

`VisibleUnalignedYUV888Image`

Visible-light YUV packed 4:4:4 image data, **unaligned** with the thermal image. This image has not been cropped or adjusted to line up with the thermal image.

~~ `VisualJPEGImage` ~~

DEPRECATED
Visual JPEG image data, **unaligned** with the thermal.

~~ `VisualYCbCr888Image` ~~

DEPRECATED
Visual YCbCr image data, **aligned** with the thermal. This image has been cropped and adjusted to line up with the thermal image.

Selecting a Palette

When streaming, you can specify a palette to use by calling your FrameProcessor instance's `setFramePalette` method.

3. Using the FLIROneSDK, delegation and using delegates

Delegate interfaces allow for asynchronous events to be passed to your handler methods. Much like how an Android Activity has onPause and onResume methods, the Device.Delegate, Device.StreamDelegate, and FrameProcessor.Delegate interfaces
define callback methods that are called when events such as a device being connected, a device sending a frame, and a frame being finished processing, and allow for non-blocking method calls if needed.

4. Tuning the FLIR One device

What is Tuning?

Tuning is a quick recalibration process of the IR camera in the FLIR One device.
Tuning is required by the IR camera on a regular basis because the properties of the IR camera change in time based upon its internal temperature and other factors.
For this reason tuning is required whenever the FLIR One device determines that the IR properties have drifted by too much from the previous tuning.
During the tuning process, the FLIR One device will perform a calibration called a flat field calibration (FFC).

How to present tuning in the application

During tuning, the shutter is closed and frames should not be displayed. Frames are not guaranteed to be delivered by the SDK during tuning.
The user will see this as a gap, stutter, or otherwise useless frames unless the application displays a notification.
In the example application we use a progress bar in indeterminate mode, along with a shading of the last frame received, when the tuning state is reported.

5. Streaming frames from the FLIR ONE device

Controlling the frame stream

Use the Device mthods `startFrameStream` and `stopFrameStream` to control when you recieve frames from the device. Once started, frames will be delivered to your `StreamDelegate` until stopped or the device disconnects.

What is the `FrameProcessor` class?

The FrameProcessor class is like a photo lab where you get your pictures developed and printed. First you instantiate the class, tell it what formats you want the images rendered, and then pass it frames to render. Note, when processing frames from a device, there is a pipeline of frames created for efficient rendering and automatic gain adjustment.

How to save an image

Use the Frame method `save` to save the frame in FLIR's radiometric JPEG format. This allows the file to be opened by the SDK or other FLIR applications and re-rendered with another palette or additional temperature analysis.

6. Managing and editing media collected by the application

Using Android Best Practices for saved media

Follow standard Android SDK methods for selecting the path to save a frame, and running the media scanner so the saved image is shown in the gallery or photos app.

A good starting point is the Android developer guide for photo apps: https://developer.android.com/training/camera/photobasics.html#TaskPath

7. Saving and Accessing images on disk

Saving a Frame instance to disk with options

Use the Frame method `save` to save the frame in FLIR's radiometric JPEG format. Arguments passed to the save method affect the preview image saved and are accessible when loading the image.

Rendering a saved Frame instance

Use the `LoadedFrame` class to open a file previously saved by `Frame.save` or another FLIR app. For rendering purposes, a `LoadedFrame` instance should be treated like a `Frame` object received from a Device.

8. Common pitfalls and questions

How are images saved?

Radiometric images are images saved as JPEG files with embedded thermal data. These images are saved with all of the meta-data required to reconstruct the radiometric data. This radiometric data includes the ability to read the temperature of a pixel at a given point and the ability to change the color palette and set the rotation. Keeping this data intact is desired for use with FLIR Tools: For iPhone, For OS X. The FLIR Tools application allows users extract advanced temperature readings from radiometric jpegs. Currently radiometric jpegs can be imported to the FLIR Tools application from the iPhone or shared from within an SDK using application via email.
See documentation on saving data or check out the `Frame` class documentation.

How to edit or change the color palette of an image?

To change how a saved Frame is displayed, simply pass it to a FrameProcessor instance with new frame type or palette options. To save the image with a new preview image format, call Frame.save with updated options.

Why am I not getting live streamed images

If your StreamDelegate's onFrameReceived method is not being called, make sure you passed
For further questions contact FLIROneSDK@FLIR.com for support.

Why am I not able to `import com.flir.flirone.*` ?

Make sure you have imported the `flironesdk.aar` file correctly in Android Studio. Please see the Example Application's configuration for an example.

How do I get the temperature out of the pixels?

In order to get the temperature of a pixel, you'll need to add `ThermalRadiometricKelvinImage` FrameType to the image processor. Once you receive a RenderedFrame in this format, you can use the width and height supplied to find the value of a particular pixel in this array. The values represent degrees Kelvin * 100. For example, the value 273.15ºK is represented by 27,315.

How do I get information about the FLIR One device battery level?

Although it is completely optional it is a good idea to inform your user of the power level of the FLIR One device.
To receive power updates, implement a `Device.PowerUpdateDelegate` and pass an instance to a connected device's setPowerUpdateDelegate method.

Note: the Cat S60 does not report battery level through this SDK.

What is FLIR Tools?

FLIR Tools is an application developed by FLIR for use with Mac OS X, Android and iPhone. The FLIR Tools application allows users extract advanced temperature readings from radiometric jpegs. Currently radiometric jpegs can be imported to the FLIR Tools application from the iPhone or shared from within an SDK using application via email. See documentation on sharing and uploading data or check out the FLIROneSDKShareActivity documentation. FLIR Tools can be found here: For iPhone, For OS X, or via a web search.

9. Open Source Licenses

JSMN

Copyright (c) 2010 Serge A. Zaitsev

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

libjpeg-turbo

This software is copyright (C) 1991-2012, Thomas G. Lane, Guido Vollbeding.
All Rights Reserved except as specified below.

Permission is hereby granted to use, copy, modify, and distribute this software (or portions thereof) for any purpose, without fee, subject to these conditions:
(1) If any part of the source code for this software is distributed, then this README file must be included, with this copyright and no-warranty notice unaltered; and any additions, deletions, or changes to the original files must be clearly indicated in accompanying documentation.
(2) If only executable code is distributed, then the accompanying documentation must state that "this software is based in part on the work of the Independent JPEG Group".
(3) Permission for use of this software is granted only if the user accepts full responsibility for any undesirable consequences; the authors accept NO LIABILITY for damages of any kind.

These conditions apply to any software derived from or based on the IJG code, not just to the unmodified library. If you use our work, you ought to acknowledge us.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the libjpeg-turbo Project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS", AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

OpenCV

Copyright (C) 2000-2015, Intel Corporation, all rights reserved.
Copyright (C) 2009-2011, Willow Garage Inc., all rights reserved.
Copyright (C) 2009-2015, NVIDIA Corporation, all rights reserved.
Copyright (C) 2010-2013, Advanced Micro Devices, Inc., all rights reserved.
Copyright (C) 2015, OpenCV Foundation, all rights reserved.
Copyright (C) 2015, Itseez Inc., all rights reserved.
Third party copyrights are property of their respective owners.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the names of the copyright holders nor the names of the contributors may be used to endorse or promote products derived from this software without specific prior written permission.

This software is provided by the copyright holders and contributors "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall copyright holders or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.