Friends Camera SDK Developer Guide

Version 1.0 – 2016.04.01

LGD_C1TD_SDK_DG_V1.0EN

LG Developer

Mobile Communication Company
Mobile Handset R&D Center

Revision History

Document Version	Date	Comment
1.0	2016-04-01	Initial Release

Contents

1 Introduction 11

1.1 LG 360 CAM Overview 12

1.1.1 Connectivity 12

1.1.2 Commands Responder 13

1.2 Camera Control Application Overview 14

1.2.1 Discovery 14

1.2.2 Commands Requester 14

1.2.3 Supported Devices 15

1.3 Use Case 16

2 Setup 17

2.1 Android Development Environment 18

2.2 Installing Friends Camera SDK 19

2.3 Testing Environment 20

2.3.1 Device Settings 20

2.3.2 Running on a real device 20

3 Development 21

3.1 Creating an Android Project 22

3.2 Importing Library 23

3.3 Manifest Declarations 25

3.3.1 Adding Permissions 25

3.3.2 Adding Features 25

3.3.3 Declaring Application 25

3.3.4 Modifying the AndroidManifest.xml File 25

3.4 HTTP 26

3.4.1 Multiple Connections 26

3.4.2 HTTPS 26

3.4.3 XSS and XSRF Vulnerabilities 26

3.4.4 Content-Type Header 26

3.4.5 GET and POST Request 26

3.4.6 HTTP Status Code 26

3.5 Key Features 27

3.5.1 BLE and Wi-Fi Connection 27

3.5.2 Using Camera 35

3.5.3 Accessing Storage 36

3.5.4 Applying Settings 37

4 API References 38

4.1 OSC Protocol API 39
4.1.1 /osc/checkForUpdates 39
4.1.2 /osc/commands/execute 40
4.1.3 /osc/commands/status 43
4.1.4 /osc/info 44
4.1.5 /osc/state 46
4.2 OSC Commands API 48
4.2.1 camera.delete 48
4.2.2 camera.getFile 49
4.2.3 camera.getLivePreview 50
4.2.4 camera.getMetadata 51
4.2.5 camera.getOptions 56
4.2.6 camera.listFiles 57
4.2.7 camera.setOptions 60
4.2.8 camera.startCapture 61
4.2.9 camera.stopCapture 62
4.2.10 camera.takePicture 63
4.2.11 camera._getRecordingStatus 64
4.2.12 camera._liveSnapshot 65
4.2.13 camera._manualMetadata 66
4.2.14 camera._pauseRecording 67
4.2.15 camera._resumeRecording 68
4.2.16 camera._startPreview 69
4.2.17 camera._stopPreview 70
4.3 Device Options 72
4.3.1 Options 72
4.4 Settings 79
4.4.1 /settings/get 79
4.4.2 /settings/set 80
4.4.3 Setting Options 81
4.5 connectionManager API 82
4.5.1 BLE Connection APIs 82
4.5.2 Wi-Fi Connection APIs 82
5 Sample Application 83
5.1 Environment Setup 84
5.1.1 Development Environment 84
5.1.2 Running Environment 85
5.2 Application Structure 86

5.2.1 Project Structure 86
5.2.2 Layout Structure 87
5.3 OSC Library 89
5.3.1 OSC Protocol API 89
5.3.2 OSC Commands API 89
5.4 OSC Hello-world 91
5.5 Common Features 93
5.5.1 Connection Management 93
5.5.2 Camera Information and State 95
5.5.3 Camera Options 96
5.5.4 Permission 98
5.5.5 Result Dialog 98
5.6 Gallery Features 100
5.6.1 Working with Image Files in the Camera 100
5.6.2 Working with Image Files downloaded from the Camera 109
5.7 Video Features 112
5.7.1 Working with Video Files in the Camera 112
5.7.2 Working with Video Files in the Camera 118
6 Tips 119
6.1 Permission Management 120
6.1.1 Permissions at run time 120
7 Release Notes 121
8 Attribution 123

Tables

Table 1 Default Endpoint Address 13
Table 2 Required Content-Type Headers 26
Table 3 Intent Filters 29
Table 4 connectionManager API and Intent Mappings 29
Table 5 Wi-Fi State Table 30
Table 6 Keys for ScanResult 30
Table 7 Soft AP ON Result Code 31
Table 8 Wi-Fi Connection Result Table 31
Table 9 Wi-Fi Connection Result Table 33
Table 10 Main UI Components 87
Table 11 OSC API Implementation 89
Table 12 Common Features 93
Table 13 Gallery Features 100
Table 14 Video Features 112
Table 15 State Machine Elements 116
Table 16 State Transition Table 116

Figures

Figure 1 Camera Discovery Scenario 12
Figure 2 HTTP Communication Between App and Camera 15
Figure 3 Connection Sequence 32
Figure 4 Project Structure of the Sample Application 86
Figure 5 Main UI Layout of 'apidemo' Module 87
Figure 6 Main UI Layout of 'helloFriendsCamera' Module 91
Figure 7 Device Scanning Result 93
Figure 8 Connecting to a Device 94
Figure 9 Camera Information Dialog 95
Figure 10 Camera State Dialog 96
Figure 11 Device Options Layout 97
Figure 12 Setting Options Layout 98
Figure 13 Result Dialog Example 99
Figure 14 List Dialog for an Image Stored in the Camera 101
Figure 15 Image Information Dialog 102
Figure 16 Downloading an Image from the Camera 102
Figure 17 Deleting an Image from the Camera 103

Figure 18 TakePictureActivity layout 104
Figure 19 Manual Metadata 105
Figure 20 PreviewActivity UI 105
Figure 21 START PREVIEW and STOP PREVIEW Responses 106
Figure 22 Preview Stream from the Camera 107
Figure 23 CaptureIntervalActivity UI 108
Figure 24 UI Layout of Interval Capture 109
Figure 25 Viewer List Dialog 110
Figure 26 Viewer Mode Option Dialog 110
Figure 27 Cardboard View Mode 111
Figure 28 VR View Mode 111
Figure 29 List Dialog for a Video Stored in the Camera 113
Figure 30 Video Information Dialog 114
Figure 31 Downloading a Video from the Camera 114
Figure 32 Deleting a Video from the Camera 115
Figure 33 State Diagram for Video Recording Operation 116
Figure 34 RecordVideoActivity UI 117
Figure 35 Viewer List Dialog 118

About This Document

This document provides a guide to developing an Android™ application that can connect to and control LG 360 CAM.

Who This Guide is for

This guide is intended for:

- Those who wish to develop an Android application that connects to and controls LG 360 CAM

Conventions

Notes & Cautions

Note

Provides useful information for developers.

Caution

Provides important information for developers.

Related Documents

Google Open Spherical Camera API Specification (available on the following Google Developers site)
https://developers.google.com/streetview/open-spherical-camera/

Organization

This document consists of the following contents:

Chapter 1 Introduction briefly explains the interaction between LG 360 CAM and application developed with Friends Camera SDK.

Chapter 2 Setup describes how to set up an Android development environment with Friends Camera SDK.

Chapter 3 Development describes how to develop an application that controls any connected LG 360 CAM device.

Chapter 4 API References provides descriptions on Friends Camera API and connectionManager API.

Chapter 5 Sample Application describes how to develop an Android application with Friends Camera SDK by going through the details of the sample application.

Chapter 6 Tips discusses useful tips for application development.

Chapter 7 Release Notes lists the history of SDK releases.

Attribution Attribution lists license and use of brand attributions that are applicable to this document.

Terms

Terms used in Friends Camera SDK are defined in the following glossary table:

Term	Definition
LG 360 CAM	LG's spherical camera that offers an interface for any app developed with Friends Camera SDK to control the behavior of the camera
OSC API	Open Spherical Camera (OSC) API. Google's open standard for the control of spherical cameras.
Friends Camera API	LG's OSC API extension which includes Google OSC API and LG's vendor-specific implementations
connectionManager API	APIs for BLE and Wi-Fi connection with Friends Camera devices

1 Introduction

This chapter briefly explains the interaction between LG 360 CAM and application developed with Friends Camera SDK.

Contents

1.1 LG 360 CAM Overview
This section describes the behavior of an LG 360 CAM device as appeared to an application developed with Friends Camera SDK.

1.2 Application Overview
This section describes the behavior of the application developed with Friends Camera SDK as appeared to the LG 360 CAM device.

1.3 Use Case
This section introduces a typical use case scenario.

1.1 LG 360 CAM Overview

LG 360 CAM is a spherical camera that produces spherical images and video contents. LG 360 CAM can be connected to mobile devices. Images and videos can be taken or recorded remotely from the connected mobile device. Once connected, LG 360 CAM listens for commands from the connected mobile device and performs the requested actions which include shooting scenes, recording videos, downloading the files from the camera to the mobile device, etc.

1.1.1 Connectivity

BLE

LG 360 CAM supports Bluetooth® Low Energy (BLE) pairing for quick discovery and connection between the camera and the application. If the pairing is established, the connection transits to a Wi-Fi connection by turning on the Wi-Fi Soft AP mode (LG 360 CAM's Soft AP mode gets enabled).

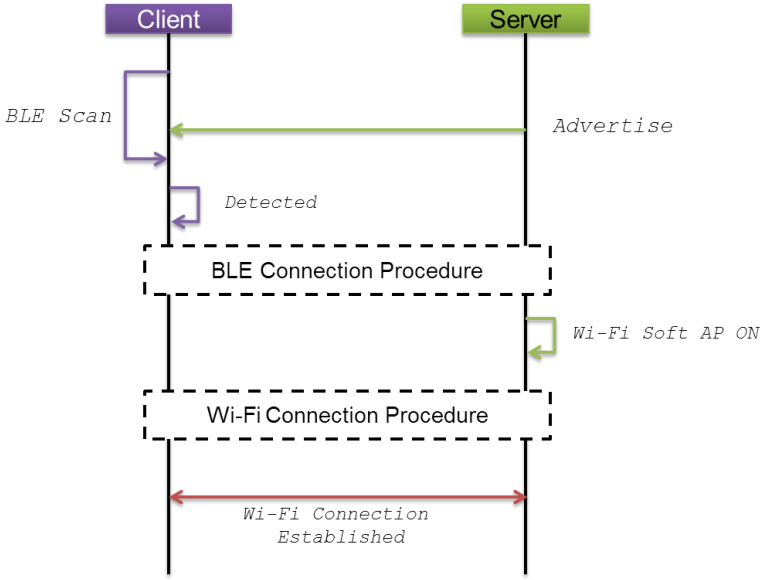


Figure 1 Camera Discovery Scenario

See BLE and Wi-Fi Connection for more information on how to pair an application with an LG 360 CAM device through the BLE pairing mechanism.

Wi-Fi

LG 360 CAM serves as a Wi-Fi® Soft AP to provide a Wi-Fi network. The Wi-Fi connection between the camera and the application is enabled through BLE pairing.

Once the connection is established, the LG 360 CAM device and the application communicate with each other over Wi-Fi channel only.

The default SSID name is of the form "LGR105_XXXXXX.OSC", where Xs are the last 6 digits of the device serial number.

The Wi-Fi connection is protected by WPA2-PSK. The WPA2-PSK password must be at least 8 characters long, containing letters, numbers and symbols.

1.1.2 Commands Responder

LG 360 CAM implements an HTTP/1.1 server to receive and respond to requests from the client that is the connected mobile device (the camera control application). The client sends its requests to the endpoint of the camera. The default IP address and port number are given in the following table:

Table 1 Default Endpoint Address

IP address	Port
192.168.43.1	6624

The HTTP/1.1 server of the camera is able to process GET and POST requests from clients.

1.2 Camera Control Application Overview

Developers can create an application that can connect to an LG 360 CAM device and control the camera with Friends Camera SDK. The two fundamental features that the application has to offer: device discovery and control of the camera. Device discovery can be achieved by using the Bluetooth and Wi-Fi features of the underlying Android System. Friends Camera SDK offers a set of APIs to provide the application with camera control features.

1.2.1 Discovery

The application discovers the camera by BLE pairing, as discussed in the Connectivity section of LG 360 CAM Overview. The application can use the Android System's Bluetooth and Wi-Fi features to establish connection with the camera. See BLE and Wi-Fi Connection for more information on how to pair an application with an LG 360 CAM device through the BLE pairing mechanism.

1.2.2 Commands Requester

The application sends requests, in the form of HTTP/1.1 GET or POST command, to the LG 360 CAM device over the established Wi-Fi network.

Friends Camera SDK offers a set of APIs and commands to provide camera control features. What can be accomplished by using the APIs and commands are as follows:

- managing connection
- controlling the camera (shooting / recording)
- manipulating images taken or videos recorded with LG 360 CAM
- configuring device settings

Friends Camera API implements the standard OSC API and adds vendor-specific commands and options as extension. For more information on Friends Camera API, see API References.

The following figure shows an example of HTTP request and response between the application and the camera.

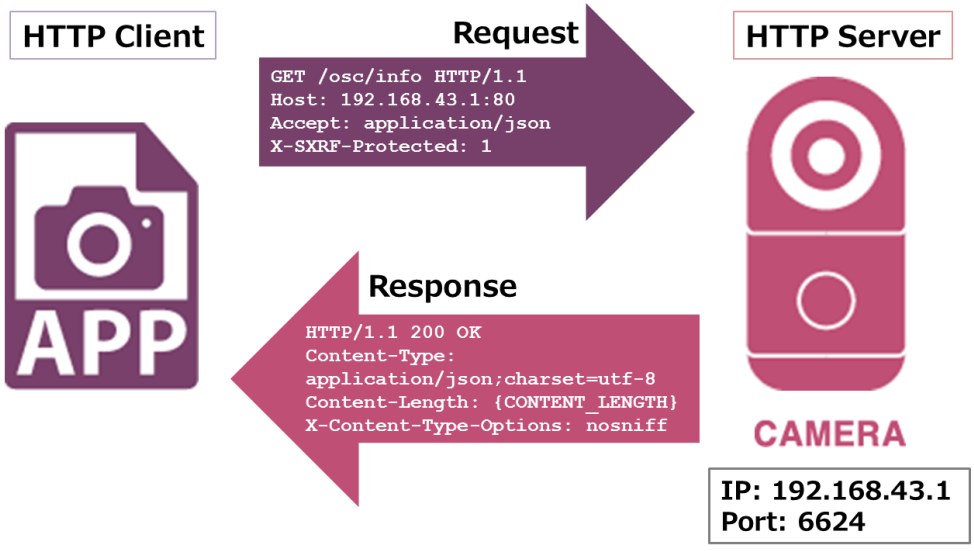


Figure 2 HTTP Communication Between App and Camera

1.2.3 Supported Devices

(As of Apr 01 2016)
Mobile devices running Android 5.0 (API level 21) or above
* BLE feature required

1.3 Use Case

A typical use case of interaction between an LG 360 CAM device and an application developed with Friends Camera SDK would be as follows:
1) Connecting over a Wi-Fi network through BLE pairing

2) Taking an image or record video

3) Manipulating the image taken or video recorded

4) Configuring the device settings

Details of how such actions are performed will be later discussed in the Development chapter and the Sample Application chapter.

2 Setup

This chapter describes how to set up an environment for Android development with Friends Camera SDK.

Contents

2.1 Android Development Envrionment
This section describes how to set up an environment for Android application development.

2.2 Installing Friends Camera SDK
This section describes how to install the Friends Camera SDK package.

2.3 Testing Environment
This section describes how to set up a testing environment.

2.1 Android Development Environment

Developing applications for Android with Friends Camera SDK requires a set of tools listed below:

- JDK (7.0 or higher)
- IDE (Android Studio 1.5 or higher)
- Android SDK (API level 21 or higher)

See the following Android Developer site for more information on environment setup: Developer Workflow

2.2 Installing Friends Camera SDK

Download the Friends Camera SDK package and unpack it to a working directory.

The Friends Camera SDK package includes:

- Developer Guide
- Sample Application
- FriendsLink Libraries

The sample application is built upon the following configuration:

- IDE: Android Studio 1.5
- Build: Gradle 2.8
- JDK: 7.0

2.3 Testing Environment

2.3.1 Device Settings

Make sure to configure the devices with the following settings:

- Turn your LG 360 CAM device on
- Turn your mobile device on and enable Bluetooth and Wi-Fi
- Launch your application developed with Friends Camera SDK

2.3.2 Running on a real device

Applications developed with Friends Camera SDK can be tested on real devices.

Connect a real mobile device to the computer via USB. Make sure “USB Debugging” is enabled on the mobile device. Refer to the following Android Developer site for information on how to enable USB debugging on Android-powered mobile devices: Enable USB debugging on your device.

Note

Get and install the LG Mobile USB driver to connect mobile devices to the computer. Download is available from the LG Support site. Select Region/Country and go to the product support page to get the software.

From the Android Studio IDE, select ‘Run’ from the menu to run the application on the connected mobile device.

3 Development

This chapter describes how to develop an application that controls any connected LG 360 CAM device.

Contents

3.1 Creating an Android Project
This section describes how to create a project with the Android Studio IDE.

3.2 Importing Library
This section describes how to import the BLE connectivity library to a project.

3.3 Manifest Declararions
This section discusses manifest declarations relevant to application development with Friends Camera SDK.

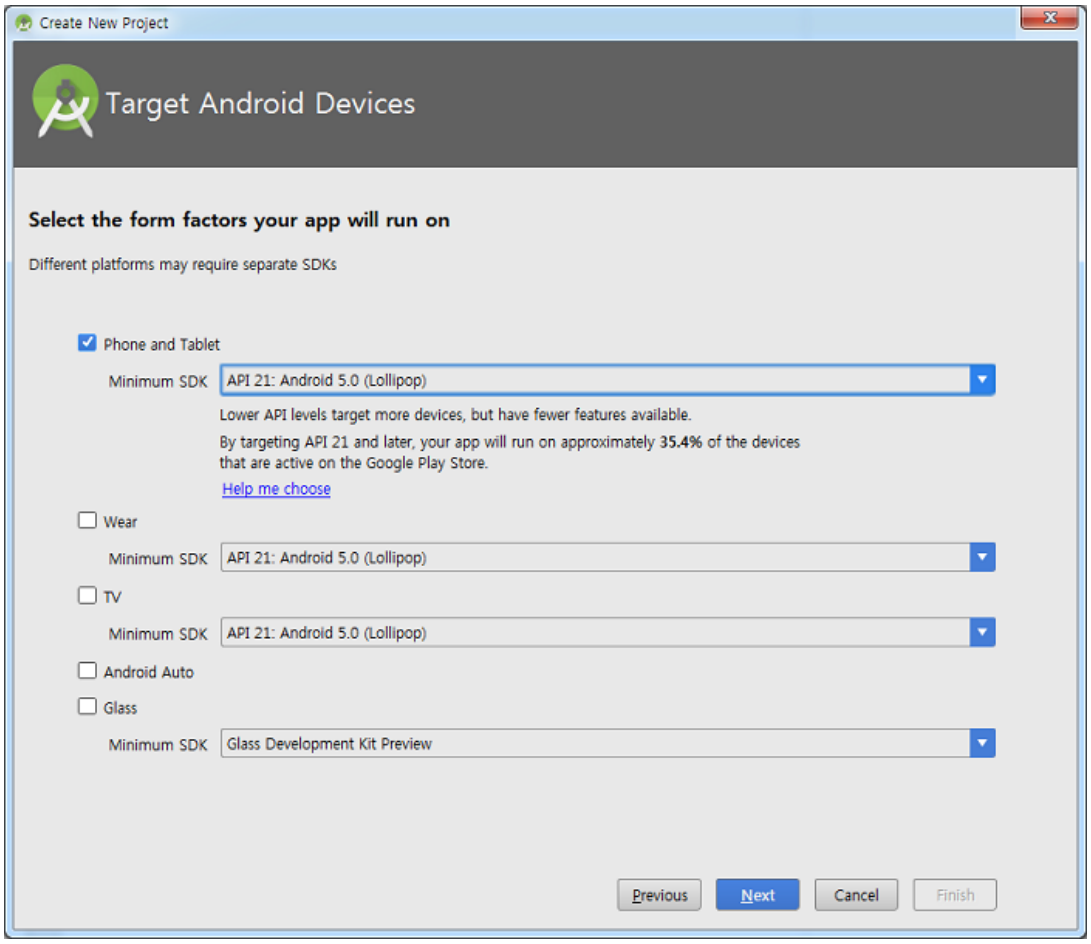
3.4 HTTP
This section discusses the requirements for HTTP communication.

3.5 Key Features
This section discusses key features that can be implemented with Friends Camera API.

3.1 Creating an Android Project

Create an Android project from Android Studio by the following steps:

1) Select 'New Project' from the 'File' menu or 'Start a new Android Studio project' from 'Quick Start'.
2) Fill in the required fields in the 'New Project' window.
3) Choose the 'Phone and Tablet' platform from the 'Target Android Devices' window.
4) Select 'API 21: Android 5.0 (Lollipop)' for 'Minimum SDK'.
5) Add an Activity to the project and complete creating the basics of the project.



For more information on how to create a project with Android Studio, please refer to the following Android Developer site: Create a Project with Android Studio.

3.2 Importing Library

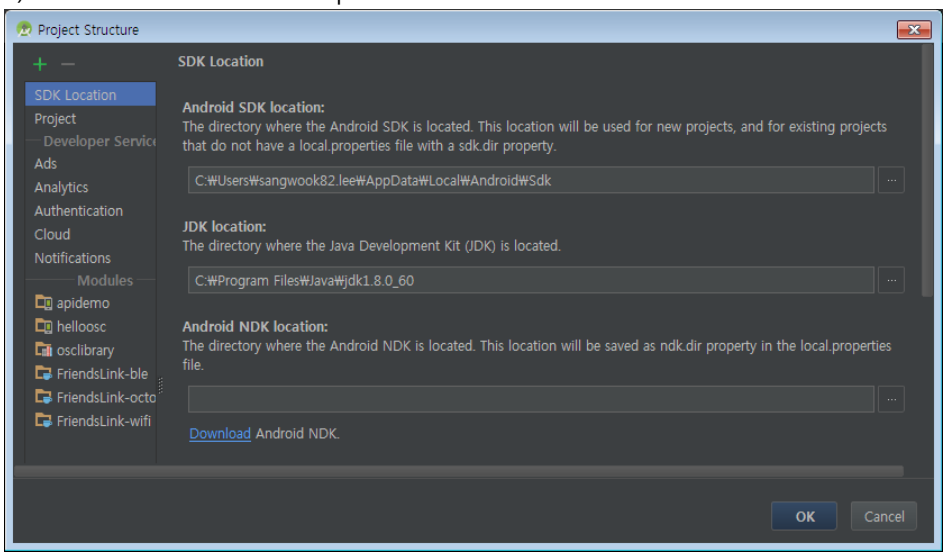
Import the BLE libraries in the Friends Camera SDK package to a working project. The SDK contains the following libraries:

- FriendsLink-ble: BLE library
- FriendsLink-octopus: connectionManager API library
- FriendsLink-wifi: Wi-Fi library

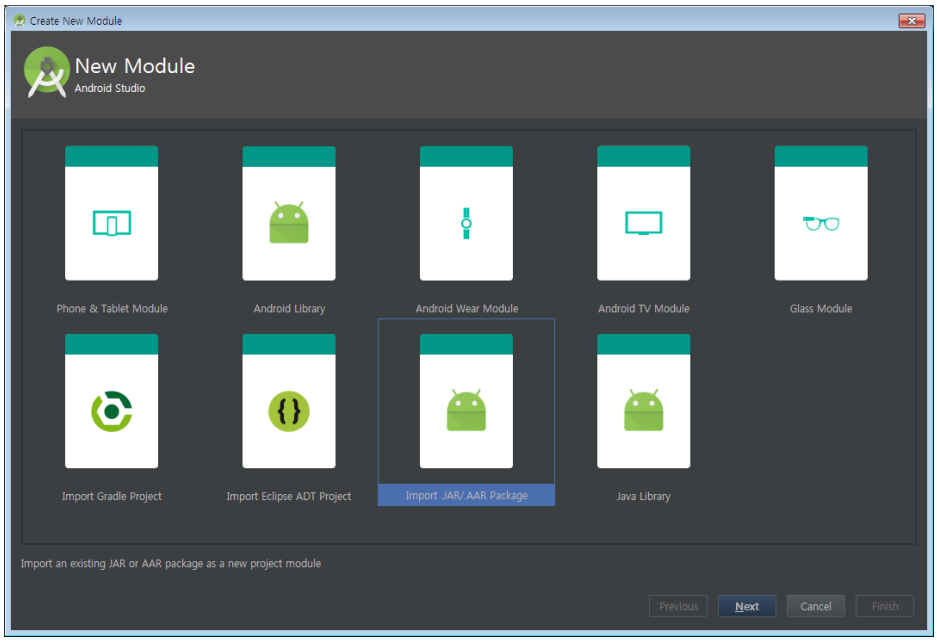
The aar library files are named according the following manner:
FriendsLink-"library name"-"main version"."sub version".aar
ex) FriendsLink-wifi-1.0.aar

Import the aar libraries to the project as described below:

1) Open up the project structure by right-clicking on your project and choosing "Open Module Settings" or choosing "File" > "Project Structure..."
2) Click the "+" button in the top left to add a new module.



3) Choose "Import .JAR or .AAR Package" and click the "Next" button.



4) Find your file using the ellipsis button ("…") beside the "File name" field. Studio will automatically create a subproject name. Just click "Finish".

5) Repeat these steps for all the libraries.

6) Add the dependency declaration to the 'build.gradle' build file under the application project.

```
dependencies {
  ...
  compile project(':FriendsLink-wifi')
  compile project(':FriendsLink-octopus')
  compile project(':FriendsLink-ble')
}
```

7) Click 'Sync Project with Gradle Files' button from the Toolbar of the Android Studio.

3.3 Manifest Declarations

This section discusses manifest declarations relevant to application development with Friends Camera SDK.

3.3.1 Adding Permissions

Applications developed upon Friends Camera SDK use the following permissions:

- android.permission.BLUETOOTH
- android.permission.BLUETOOTH_ADMIN
- android.permission.ACCESS_FINE_LOCATION
- android.permission.INTERNET
- android.permission.ACCESS_NETWORK_STATE
- android.permission.ACCESS_WIFI_STATE
- android.permission.CHANGE_WIFI_STATE

3.3.2 Adding Features

Applications developed upon Friends Camera SDK use the following feature:

- android.hardware.bluetooth_le

3.3.3 Declaring Application

Applications developed upon Friends Camera SDK include the following application declaration:

- <application android:name=".FriendsCameraApplication">

3.3.4 Modifying the AndroidManifest.xml File

Declare the permissions, features and applications required in the AndroidManifest.xml file:

<AndroidManifest.xml>

```
<manifest ...>
    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

    <uses-feature
        android:name="android.hardware.bluetooth_le"
        android:required="true" />

    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />

    <application android:name=".FriendsCameraApplication">
        ...
    </application>
</manifest>
```

3.4 HTTP

3.4.1 Multiple Connections

LG 360 CAM allows only one connection from clients; multiple simultaneous connections to the camera are not supported. The HTTP connection between the LG 360 CAM device and the application is established as one-to-one, ensuring the session over HTTP communication to be in exclusive use.

3.4.2 HTTPS

HTTPS communication is not supported.

3.4.3 XSS and XSRF Vulnerabilities

Addressing the XSS and XSRF prevention, the OSC API Specification requires the following header fields for all HTTP messages:

- For all HTTP request messages, include "X-XSRF-Protected: 1" in the HTTP header
- For all HTTP response messages, include "X-Content-Type-Options: nosniff" in the HTTP header

See the OSC API Specification for more details.

3.4.4 Content-Type Header

All HTTP response messages include "Content-Type: application/json; charset=utf-8 "in the HTTP header. One exception to the Content-Type header: commands that return images, videos or preview live video streams require the Content-Type header to be their respective media type instead. The following table summarizes the required Content-Type headers for each media type supported.

Table 2 Required Content-Type Headers

Media Type	HTTP Header
Image	"Content-Type: image/jpeg"
Video	"Content-Type: video/mp4"
Preview	"Content-Type: image/jpeg"

3.4.5 GET and POST Request

The **/osc/info** API is the only request that uses GET method. All other APIs and commands use POST method.

3.4.6 HTTP Status Code

If any API or command is processed with no errors, HTTP status '200 OK' will be returned. All client originated errors will return HTTP status '400 Bad Request'. All server originated errors will return HTTP status '500 Internal Server Error'.

3.5 Key Features

This section discusses key features that can be implemented with Friends Camera API.

Note

UI/UX design and concepts are up to developers to make the decision that best suits the needs of the application.

3.5.1 BLE and Wi-Fi Connection

BLE pairing and Wi-Fi connection are established by using connectionManager API. The connectionManager API provides a set of methods for BLE and Wi-Fi connection. See connectionManager API for more information.

Caution

The following libraries should be imported to a working project so that the application can use connectionManger APIs. See Importing Library for more information on how to import libraries:

FriendsLink-ble, FriendsLink-octopus, FriendsLink-wifi

Application Setup

Any application built with Friends Camera SDK should instantiate the `FriendsCameraApplication` class instead of using the base Application class of the Android System in order to use connection features. Declare the application name in the <application> tag of the AndroidManifest.xml file.

```
<application android:name=".FriendsCameraApplication">
```

The `FriendsCameraApplication` class implementation is given below:

<FriendsCameraApplication.java>

```
public class FriendsCameraApplication extends OctopusApplication {

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

Using connectionManager API

Create an instance of connectionManager class and call the public methods of the connectionManager class, as demonstrated below:

```
private ConnectionManager mConnectionManager;
mConnectionManager = OctopusManager.getInstance(this).getConnectionManager();

mConnectionManager.StartScanFriends();
```

```
mConnectionManager.StopScanFriends();
mConnectionManager.enableFriendWifiAp(address);
mConnectionManager.connect(ssid, passwd);
mConnectionManager.disconnect();
mConnectionManager.isConnected(ssid);
```

See connectionManager API for detailed descriptions on connectionManager API.

Implementing BroadcastReceiver

The result of a connectionManager API call is received from the following BroadcastReceiver:

```
private final BroadcastReceiver mUpdateReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {

        final String action = intent.getAction();
        if (action.startsWith(Central.PREFIX)) {
            // Ble Intent actions
                processBle();
        } else if (action.startsWith(WifiClient.PREFIX)) {
            // Wifi Intent actions
                processWifi();
        } else {
            Log.e(TAG, "Error !! out of action");i
        }
    }
};
```

The `onReceive()` method is designed to respond to known BLE and Wi-Fi Intent actions and ignore all other Intents. In order to avoid namespace collision, connectionManager defines prefix strings for BLE and Wi-Fi Intent action names in Central.PREFIX and WifiClient.PREFIX, respectively.

Registering BroadcastReceiver

Register the BroadcastReceiver with LocalBroadcastManager. Using LocalBroadcastManager ensures that only FriendsCameraApplication can receive BLE and Wi-Fi broadcasts of connectionManager.

```
LocalBroadcastManager.getInstance(getApplicationContext()).registerReceiver(mUpd
ateReceiver, getFilter());
```

Setting up Intent Filter

Add Intent actions to the Intent filter which is used for BLE and Wi-Fi connection. The following Intent actions are added to the filter:

- Central.ACTION_LE_SCANRESULT
- Central.ACTION_FRIENDS_LE_RESULT
- WifiClient.ACTION_FRIENDS_WIFI_RESULT
- WifiClient.ACTION_WIFI_STATE

```
private IntentFilter getFilter() {
    IntentFilter scanFilter = new IntentFilter();
    // BLE Intent filter
    scanFilter.addAction(Central.ACTION_LE_SCANRESULT);
    scanFilter.addAction(Central.ACTION_FRIENDS_LE_RESULT);
```

```
    //Wifi Intent filter
    scanFilter.addAction(WifiClient.ACTION_FRIENDS_WIFI_RESULT);
    scanFilter.addAction(WifiClient.ACTION_WIFI_STATE);
    return scanFilter;
}
```

See the following table for descriptions on the Intent filters:

Table 3 Intent Filters

BLE Intent Filter	
Central. ACTION_LE_SCANRESULT	The result of BLE scanning
Central.ACTION_FRIENDS_LE_RESULT	The result of GATT request
Wi-Fi Intent Filter	
WifiClient.ACTION_FRIENDS_WIFI_RESULT	The result of Wi-Fi connection request
WifiClient.ACTION_WIFI_STATE	The Wi-Fi connection state

The following table summarizes the relationship between connection API and the Intent generated by the API call:

Table 4 connectionManager API and Intent Mappings

connectionManager API	**Intent**	**Description**
StartScanFriends()	Central.ACTION_LE_SCANRESULT	Intent is broadcasted each time a new device is found after starting device scan
StopScanFriends()	-	No Intents are broadcasted
enableFriendWifiAp(String address)	Central.ACTION_FRIENDS_LE_RESULT	The result of GATT request to the server is returned in the Intent
connect(String ssid, String passwd)	WifiClient.ACTION_FRIENDS_WIFI_RESULT	The result of Wi-Fi connection request with the specified SSID is returned in the Intent
disconnect()	WifiClient.ACTION_WIFI_STATE	The Wi-Fi connection state (connected/disconnected) of the Friends device is returned in the Intent
isConnected(String ssid)	-	No Intents are broadcasted. The method itself returns a boolean value indicating the Wi-Fi connection state.

Constants for BLE Actions

<Wi-Fi State>

See the following code for the definitions of Wi-Fi states:

```
public static class STATE {
    public static final int WIFI_OFF = 0;
    public static final int WIFI_ON = 8;
    public static final int WIFI_BUSY = 16;
```

```
    public static final String[] toString = new String[]{"WIFI_OFF", "WIFI_ON",
"WIFI_BUSY"};

    private STATE() {
    }
}
```

Table 5 Wi-Fi State Table

Wi-Fi State	Description
Central.WIFI_OFF	The Friends device's Wi-Fi is turned off
Central.WIFI_ON	The Friends device's Wi-Fi is turned on
Central.WIFI_BUSY	The Friends device's Wi-Fi is already connected with other device.

<Keys Associated With Device Information>
See the following code for the definitions of key values to retrieve device information:

```
String LE_DEV_NAME = "com.lge.octopus.tentacles.ble.LE_DEV_NAME";
String LE_BLE_ADDRESS = "com.lge.octopus.tentacles.ble.LE_BLE_ADDRESS";
...
String LE_WIFI_STATE = "com.lge.octopus.tentacles.ble.LE_WIFI_STATE";
...
String LE_DEVICE_SERIAL = "com.lge.octopus.tentacles.ble.LE_DEVICE_SERIAL";
...
String LE_CAMERA_PROTOCOL_OSC =
"com.lge.octopus.tentacles.ble.LE_CAMERA_PROTOCOL_OSC";
String LE_CAMERA_FACTORY_MODE =
"com.lge.octopus.tentacles.ble.LE_CAMERA_FACTORY_MODE";
```

Table 6 Keys for ScanResult

Key	Description
Central.LE_DEV_NAME	Device name
Central.LE_DEVICE_SERIAL	Device's serial number
Central.LE_BLE_ADDRESS	Device's BLE address
Central.LE_CAMERA_PROTOCOL_OSC	Whether the device support OSC Protocol or not
Central.LE_WIFI_STATE	Wi-Fi state of the device (OFF / ON / BUSY)
Central.LE_CAMERA_FACTORY_MODE	The device has not connected with any other devices since the device was initialized. (The password is initialized to "00" followed by the last 6 digits of the serial number after factory reset.)

<Result of Soft AP ON Request>
A request to Wi-Fi Soft AP ON returns the result in the Intent filter as an extra attribute. See the following code for the definitions of the results of Wi-Fi Soft AP ON request:

```
public static class CBOPCODE {
    public static final byte WIFI_SOFT_AP_OFF = 1;
    public static final byte WIFI_SOFT_AP_ON = 2;
    ...
```

```
    public static final String[] toString = new String[]{"", "WIFI_SOFT_AP_OFF",
"WIFI_SOFT_AP_ON", ...};

    private CBOPCODE() {
    }
}
```

Table 7 Soft AP ON Result Code

Result Code	Description
Central.CBOPCODE.WIFI_SOFT_AP_OFF	Friends device's Wi-Fi has been turned off
Central.CBOPCODE.WIFI_SOFT_AP_ON	Friends device's Wi-Fi has been turned on

Constants for Wi-Fi Actions

<Wi-Fi Connection Result>

Table 8 Wi-Fi Connection Result Table

Result	Decription
WifiClient.RESULT.CONNECTED	Wi-Fi connected with the Friends device
WifiClient.RESULT.DISCONNECTED	Disconnected with the Friends device
WifiClient.RESULT.CONNECT_FAIL	Failed to connect with the Friends device
WifiClient.RESULT.TIME_EXPIRED	Wi-Fi connection request timed out (20 seconds)
WifiClient.RESULT.INCORRECT_PASSPHRASE	The Wi-Fi password is incorrect

Connection Sequence

The following diagram shows the connection procedure:

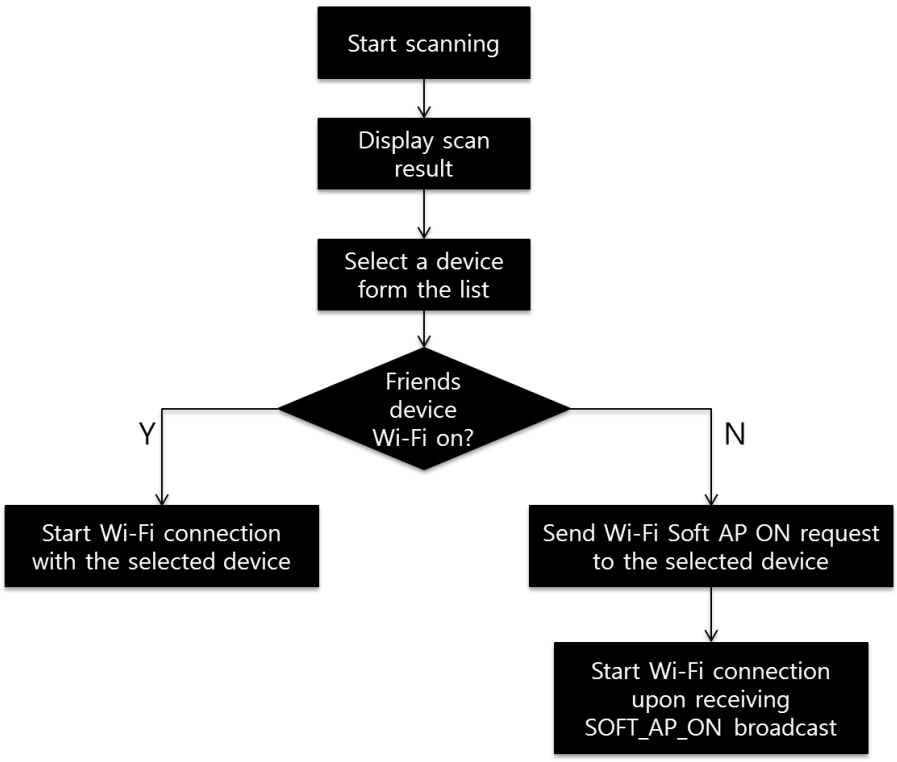


Figure 3 Connection Sequence

Scanning for devices gets started by calling the `connectionManager.StartScanFriends()` method. Select a device to connect with from the scanned device list. Check the Wi-Fi connection state of the selected device by using the `ScanResult.getWifistate()` method. Choose appropriate connection actions depending on the Wi-Fi state of the camera device. See Contants for BLE Actions for detailed information on Wi-Fi states. See the following code for the implementation of the connection sequence:

```
private void connect() {
    if (selectedDevice.getWifiState() == Central.STATE.WIFI_ON) {
        // Friend device's Wi-Fi is on
        // send Wi-Fi connection request to the device
    } else if (selectedDevice.getWifiState() == Central.STATE.WIFI_OFF) {
        // Friends device's Wi-Fi is off
        // send Wi-Fi Soft AP ON request to the device
    } else { //Central.STATE.WIFI_BUSY
        // Friends device's Wi-Fi is already in use
    }
}
```

BLE and Wi-Fi Intent Action Process

From the BroadcastReceiver, the Intent filter is applied to select the BLE and Wi-Fi connection broadcasts of connentionManager.

If BLE Intent actions are received, the BroadcastReceiver calls the processBle() method to process BLE actions.

```
private void processBle(Intent intent) {

    final String action = intent.getAction();

    if (Central.ACTION_LE_SCANRESULT.equals(action)) {
        // process BLE scan result
        // get the scan result contained in the Bundle
        Bundle scanResult = intent.getExtras();
        ScanResult foundDevice = new ScanResult(scanResult, true);
        boolean isNew = mLeDeviceListAdapter.addDevice(foundDevice);
        if(isNew) {
            mLeDeviceListAdapter.notifyDataSetChanged();
        }

    } else if (Central.ACTION_FRIENDS_LE_RESULT.equals(action)) {
        // check the result of BLE GATT request
        int result = intent.getIntExtra(Central.EXTRA_RESULT,
Central.RESULT_FAIL);
        int state = intent.getIntExtra(Central.EXTRA_STATE,
Central.LE_GATT_STATE.DISCONNECTED);
        String message = intent.getStringExtra(Central.EXTRA_DATA);

        if (result == Central.RESULT_FAIL) {
            // BLE connection failed
            // Constants.Result.FAIL_GATT_CONNECTION
            Utils.showTextDialog(mContext, "Fail:", "Fail to connect Gatt server");
```

```
        } else if (state == Central.LE_GATT_STATE.TURN_ON_AP && message != null)
{
            // BLE connection success & process the result of TURN_ON_AP request
            if
(Central.CBOPCODE.toString[WIFI_SOFT_AP_ON].equalsIgnoreCase(message)) {
                // Wi-Fi Soft AP ON success
                mConnectionManager.connect(getSsid(selectedDevice),
getDefaultPassphrase());
            }
        }
    }
}
```

Central.ACTION_LE_SCANRESULT filter is used to check the presence of any newly discovered devices. Intent.getExtras() returns a Bundle that contains device information. See the description on ScanResult for how to extract desired information from the Bundle. The keys to retrieve device information are defined in Constants. Newly discovered devices are added to the device list adapter so that they can be displayed by a ListView.

Central.ACTION_FRIENDS_LE_RESULT filter is used to get the result of GATT request to the server. The following keys are used to get data from the Intent actions:

Table 9 Wi-Fi Connection Result Table

Key	Description
Central.EXTRA_RESULT	Whether the GATT request has succeeded or failed
Central.EXTRA_STATE	Checks the type of the GATT request
Central.EXTRA_DATA	Gets the response to the GATT request

Check the received result from Central.LE_GATT_STATE.TURN_ON_AP request. If the result is WIFI_SOFT_AP_ON, the Wi-Fi of the camera device has successfully been enabled. Call `connect()` to start Wi-Fi connection with the camera device.

If Wi-Fi Intent actions are received, the BroadcastReceiver calls the processWifi() method to process Wi-Fi actions.

```
private void processWifi(Intent intent) {
    String action = intent.getAction();
    int result = intent.getIntExtra(WifiClient.EXTRA_RESULT,
WifiClient.RESULT.DISCONNECTED); // the result of Wi-Fi connection request to
the Friends device

    if (WifiClient.ACTION_WIFI_STATE.equals(action) && result ==
WifiClient.RESULT.CONNECTED) {
        // Wi-Fi connection is established with the Friends device
    } else if (WifiClient.ACTION_FRIENDS_WIFI_RESULT.equals(action)) {
        // process the result of Wi-Fi connection request to the Friends device
        processWifiResult(result);
    }
}
```

WifiClient.ACTION_WIFI_STATE filter is used to check the Wi-Fi connection state. The return value is either WifiClient.RESULT.CONNECTED or WifiClient.RESULT.DISCONNECTED.

WifiClient.ACTION_FRIENDS_WIFI_RESULT filter is used to inform the result of Wi-Fi connection request. The result is passed to processWifiResult() to take further actions.

```
private void processWifiResult(int result) {

    Log.e(TAG, "WIFI_RESULT: result = " + WifiClient.RESULT.toString[result]);

    switch (result) {
        case WifiClient.RESULT.CONNECTED:
            //successfully connected
            break;

        case WifiClient.RESULT.DISCONNECTED:
        case WifiClient.RESULT.CONNECT_FAIL:
        case WifiClient.RESULT.TIME_EXPIRED:
            // failed to connect
            break;

        case WifiClient.RESULT.INCORRECT_PASSPHRASE:
            // Wi-Fi password is not correct
            // request user input for password again
            break;
    }
}
```

See Constants for Wi-Fi Actions for more detailed information on Wi-Fi connection result.

ScanResult

The scan result contained the Bundle Object can be obtained by using Bundle.getString(), Bundle.getBoolean(), Bundle.getInt() methods according to value types. Device information contained in the Bundle can be retrieved by using the given key values in Constants for BLE Actions. The following code shows an example of ScanResult implementation. Developers can implement their own class that holds device information using the given keys.

```
public ScanResult(Bundle scanResult) {
      mScanResult = scanResult;
      mDeviceName = mScanResult.getString(Central.LE_DEV_NAME, "");
      mBleAddress = mScanResult.getString(Central.LE_BLE_ADDRESS, "");
      mSerial = mScanResult.getString(Central.LE_DEVICE_SERIAL, "");
      mIsOsc = mScanResult.getBoolean(Central.LE_CAMERA_PROTOCOL_OSC, false);
      mWifiState = mScanResult.getInt(Central.LE_WIFI_STATE,
Central.STATE.WIFI_OFF);
      mIsFactory = mScanResult.getBoolean(Central.LE_CAMERA_FACTORY_MODE, false);
   }
```

See Constants for BLE Actions for more information on key values.

Related APIs: ConnectionManager.StartScanFriends(), ConnectionManager.StopScanFriends(), ConnectionManager.enableFriendWifiAp(address), ConnectionManager.connect(ssid, passwd), ConnectionManager.disconnect(), ConnectionManager.isConnected(ssid)

3.5.2 Using Camera

Camera actions include taking a picture, previewing a scene, recording a video, etc. In addition to camera commands of OSC API, LG's vendor-specific commands add preview and video recording features as well.

Caution

Some commands handling camera actions do not return results immediately after the submission of the command. Continuously poll the previously submitted command with the **/osc/commands/status** API to determine whether the command has finished. See the API description of the commands listed below for more details.

Related Commands: **camera.takePicture**, **camera.startCapture**, **camera.stopCapture**, **camera._liveSnapshot**

Taking a Picture

To take a picture, send **camera.takePicture** command to the server (camera). The server responds with an URL which tells the location of the captured image.

Executing an Interval Capture

camera.startCapture command can be used for interval image capture if `captureMode` option is set to "`interval`". If `captureNumber` option is set 0, the camera executes an open-ended capture. Otherwise the camera executes a non-open-ended capture. Open-ended capture can only be stopped by sending **camera.stopCapture** command. Non-open-ended capture stops automatically when the camera has taken `captureNumber` number of images, or can terminate early by sending **camera.stopCapture** command. For open-ended capture and early termination of non-open-ended capture, **camera.stopCapture** command returns the result of captured image URLs. For automatic termination of non-open-ended capture, **camera.startCapture** command returns the result of captured image URLs.

Recording a Video

camera.startCapture command records video if `captureMode` option is set to "`video`". **camera.stopCapture** command stops recording and returns the result of recorded video URL. Supplementary commands are available to pause or resume recording by sending **camera._pauseRecording** or **camera._resumeRecording** command, respectively. The status of recording operation can be checked with **camera._getRecordingStatus** command.

Taking a Live Snapshot

camera._liveSnapshot command can be used to capture an image while the camera is recording a video. This command can be used only when the camera is in 180° modes.

Watching Preview

To watch preview, send **camera._startPreview** command to the server to start previewing. The server returns an Uri from which to receive a continuous video stream of preview. Send **camera._stopPreview** command to stop previewing.

camera.getLivePreview is another command to get preview. The response contains one shot of the current preview frame of the camera. Repeatedly send this command to build preview stream.

Note that the preview's field of view is determined in accordance with the direction of the camera lens. The camera generates preview images in fish-eye format.

The application should devise a video player on the playback side to show video streams delivered from the camera.

Getting and Setting Device Options

Use **camera.setOptions** to set any option to the camera before performing camera actions. **camera.getOptions** give information on the current camera option settings. Refer to the Device Options section for more information on device options.

Commands Related: **camera.takePicture**, **camera.startCapture**, **camera.stopCapture**, **camera._pauseRecoring**, **camera._resumeRecording**, **camera._getRecordingStatus**, **camera._startPreview**, **camera._stopPreview**, **camera.getLivePreview**, **camera.setOptions**, **camera.getOptions**

3.5.3 Accessing Storage

Downloading a File

Images and video files are stored in the storage of LG 360 CAM. The application can access the files in the storage of the camera and download the files to the application side. The camera returns binary data of image or video to the application as a response to **camera.getFile** command. Images and videos from the camera can be played on the application side if they are downloaded to the application.

Listing All Files

The whole list of image and/or video files in the camera can be obtained by using **camera.listFiles** command. Image and video files can be distinguished by examining the file extension type specified in the `name` field of the response message.

Getting Image or Video Header Information

The EXIF and Photo Sphere XMP header information of an image or the Video XMP information of a video can be obtained by using **camera.getMetadata** command.

Getting Thumbnails

Thumbnail images are appended to the response message if the option parameter `maxThumbSize` is set in **camera.listFiles** command. Note that a full size image or video will be returned if `maxThumbSize` is set to 0 or greater than the actual file size. If `maxThumbSize` is set to null, the response does not contain thumbnail images.

An alternative way to get thumbnails is using **camera.getFile** command. Use of `maxSize` parameter in **camera.getFile** command is exactly the same as that of `maxThumbSize` in **camera.listFiles** command.

Deleting a File

Files in the camera storage can be deleted by specifying their file URLs in **camera.delete** command.

Commands Related: **camera.getFile**, **camera.getMetadata**, **camera.listFiles**, **camera.delete**

3.5.4 Applying Settings

Introducing Settings Interface

In addition to the OSC API interface, LG 360 CAM provides the Settings interface between the application and the camera. Similar to OSC APIs whose Request-URL is identified by the '**/osc**' prefix, Settings APIs are identified by the '**/settings**' prefix in the Request-URL field.

Getting and Setting Setting Options

With **/settings/get** and **/settings/set**, one can get or set device configuration settings such as sleep time, sound on/off, etc. Supported Setting options are listed in the Setting Options section in the API References chapter.

Commands Related: **/settings/get**, **/settings/set**

4 API References

This chapter contains descriptions on Friends Camera API and connectionManager API. Friends Camera API consists of Google's OSC API and LG's vendor-specific implementations. The connectionManager API consists of methods for BLE and Wi-Fi connection with Friends devices.

Contents

4.1 OSC Protocol API
The Google Open Spherical Camera Protocol (OSC Protocol) APIs

4.2 OSC Commands API
The Google Open Spherical Camera API (OSC API) specification
With the extension of LG's vendor-specific commands (denoted by _ underscore prefix)

4.3 Device Options
Options available to set and get by **camera.setOptions** and **camera.getOptions** commands
With the extension of LG's vendor-specific options (denoted by _ underscore prefix)

4.4 Settings
LG's vendor-specific APIs to set or get setting options

4.5 connectionManager API
APIs for BLE and Wi-Fi connection with Friends devices

More information on Google's Open Spherical Camera API is available on the following Google Developers site: Open Spherical Camera API

Note

See Attribution chapter for attribution to Google regarding use of content of Open Spherical Camera API specification from Google Developers site.

Caution

LG 360 CAM supports OSC API version 2.0. For backward compatibility, OSC API version 1.0 has not been removed. However, avoid using deprecated OSC API version 1.0 which might cause unexpected errors; there's no guarantee that deprecated OSC API would function properly. For this reason, Friends Camera SDK does not provide any guide, API reference or technical support in regard to use of OSC API version 1.0.

4.1 OSC Protocol API

4.1.1 /osc/checkForUpdates

Description:

The /osc/checkForUpdates API updates the fingerprint to reflect the current camera state by comparing it with the fingerprint held by the client.
A timeout scheme can be used to periodically check the camera state at intervals of 'timeout' seconds. If the client's fingerprint and the camera's current fingerprint are equal, the camera waits for 'timeout' and returns its response. The camera returns the new fingerprint to the client immediately if the camera's fingerprint has been changed.

- The fingerprint algorithm

3 digit(batteryLevel)
1 digit(batteryStatus) - 0: discharging, 1: charging(USB), 2: charging(AC), 3: charged
1 char(fileChanged) - X: not changed, A: file added, D: file deleted
1 digit(SD card status) - 0: unmount, 1: mount
1 digit(Capture status) - 0 : idle, 1 : recording, 2 : shooting, -1 : error (camera open failed.)
1 digit(Camera ID) - 0 : half-front-camera, 1 : half-rear-camera, 2 : full-front-camera, 3 : full-rear-camera
4 digit(Current time) – current time suffix to make the fingerprint a unique one

Example 0940X1010182

Syntax:

```
POST /osc/checkForUpdates HTTP/1.1
```

Parameters:

Parameter	Data Type	Description	Required
stateFingerprint	String	The fingerprint held by the client (acquired from /osc/state API)	Required
waitTimeout	Integer	If the fingerprint has not been changed, the camera waits for this timeout seconds and then returns the response.	Optional

Command Output:

Parameter	Data Type	Description	Required
stateFingerprint	String	The current fingerprint of the camera device	Required
throttleTimeout	Integer	Timeout for the client to wait before making another update call. The server always returns 0 since it operates in a synchronous manner.	Optional

Error Code	Data Type	Description
missingParameter	String	stateFingerprint is not specified.
invalidParameterName	String	One or more input parameter names is unrecognized.
invalidParameterValue	String	The parameter names are recognized, but one or more values is invalid; for example, waitTimeout is out of range or its type is incorrect.
serverError	String	The server was unable to determine an appropriate throttleTimeout value for its response. The server problem will be identified by the 5XX value returned in the response. Camera manufacturers should supply a table of 5XX codes and corresponding server states that can throw this error.

Examples

```
Request
POST /osc/checkForUpdates HTTP/1.1
Host: [camera ip address]:[httpUpdatesPort]
Content-Type: application/json;charset=utf-8
Accept: application/jsonContent-Length: {CONTENT_LENGTH}
X-XSRF-Protected: 1

{
   "stateFingerprint": "0940X1010182",
   "waitTimeout": 300
}

Response
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Content-Length: {CONTENT_LENGTH}
X-Content-Type-Options: nosniff

{
   "stateFingerprint": "0940X1010182",
   "throttleTimeout": 0
}
```

4.1.2 /osc/commands/execute

Description:

The /osc/commands/execute API executes specified commands on the camera. The output is a command object.

Syntax:

```
POST /osc/commands/execute HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
```

```
Accept: application/json
Content-Length: {CONTENT_LENGTH}
X-XSRF-Protected: 1
```

Parameters

Parameter	Data Type	Description	Required
name	String	The command to be executed.	Required
parameters	Object	Command input parameters according to the command definitions specification. See OSC Commands API section for details.	Required

Command Output

Return Value	Data Type	Description	Required
name	String	The command to be executed.	Required
state	String	State of the command. Should be one of the following: done - Complete, results have been returned in this response. inProgress - Execution is still in progress. error - Failed, see error in the response.	Required
id	String	Command ID. This value is required for commands returning the status inProgress. For example, the camera.takePicture command takes a few seconds due to the need for stitching. See the "Status" section for more details.	Optional
results	Object	Command results. This value is required for commands returning state done if the command is expected to return results; for example, "results" : { "AAA": "BBB", ... } Please refer to OSC API Specification for examples.	Optional
error	Object	Command error description. This value is required for commands returning state error; for example, "error": { "code": "badSessionId" }	Optional
progress	Object	Command progress description. This value is required for commands returning state inProgress; for example, "progress": { "completion": 0.8 }	Optional

Error Code	Data Type	Description
unknownCommand	String	Requested command is unknown.
cameraInExclusiveUse	String	Camera is already in exclusive use, new session can't be started.
missingParameter	String	One or more required parameters were not specified.

invalidParameterName	String	One or more input parameter or option name was unrecognized or unsupported.
invalidParameterValue	String	The parameter or option names were recognized, but one or more values is invalid; for example, the sessionId doesn't exist, is inactive, its type is incorrect, the continuationToken is out of range, or an option value is incorrect.

Examples

```
Request
POST /osc/commands/execute  HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH} X-XSRF-Protected: 1

{
  "name": "camera.setOptions", "parameters":  {
    "sessionId": "12ABC3",
    "options":  {
      "iso": 200,
      "exposureCompensation": -2
    }
  }
}

Response
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Content-Length: {CONTENT_LENGTH} X-Content-Type-Options: nosniff

{
  "name": "camera.setOptions",
  "state": "done"
}

Request
POST /osc/commands/execute  HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH} X-XSRF-Protected: 1

{
  "name": "camera.takePicture",
  "parameters":  {
    "sessionId": "12ABC3"
  }
}

Response
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Content-Length: {CONTENT_LENGTH} X-Content-Type-Options: nosniff

{
  "name": "camera.takePicture",
  "state": "inProgress",
```

```
  "id": "90ABCD",
  "progress": {
    "completion":  0
  }
}
```

4.1.3 /osc/commands/status

Description:

The /osc/commands/status API returns the status for previous inProgress commands. The status API is useful for polling the progress of a previously issued command; for example, determining whether camera.takePicture has completed.

Syntax:

```
POST /osc/commands/status HTTP/1.1
```

Parameters

Parameter	Data Type	Description	Required
id	String	Command ID returned by a previous call to /osc/commands/execute	Required

Command Output

The output is also a command object (see Output of /osc/commands/execute)

Error Code	Data Type	Description
missingParameter	String	The id is not specified. This parameter requires a command ID returned by a previous call to /osc/commands/execute.
invalidParameterName	String	The input parameter is unrecognized.
invalidParameterValue	String	The input parameter is recognized, but its value is invalid; for example, id value doesn't exist or its type is incorrect.

Examples

```
Request
POST /osc/commands/status HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH}
X-XSRF-Protected: 1

{
   "id": "90ABCD"
```

```
}

Response
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Content-Length: {CONTENT_LENGTH}
X-Content-Type-Options: nosniff

{
    "name": "camera.takePicture",
    "state": "done",
    "results": {
        "fileUri": "ABC.JPG"
    }
}
```

4.1.4 /osc/info

Description:

The /osc/info API returns the basic information of the LG 360 CAM device.

Syntax:

```
GET /osc/info HTTP/1.1
```

Parameters

This command takes no input.

Command Output

Return Value	Data Type	Description	Required
manufacturer	String	The camera manufacturer.	Not defined
model	String	The camera model.	Not defined
serialNumber	String	Serial number.	Not defined
firmwareVersion	String	Current firmware version.	Not defined
supportUrl	String	URL for the camera's support webpage.	Not defined
gps	Boolean	True if the the camera has GPS.	Not defined
gyro	Boolean	True if the camera has Gyroscope.	Not defined
uptime	Integer	Number of seconds since the camera boot.	Not defined
api	String	List of supported APIs.	Not defined
endpoints*	Object	A JSON object containing information about the camera's endpoints. See the next table for endpoints Object.	Not defined
apiLevel	String	OSC API level. (Default: "v2")	Not defined

* endpoints Object

This JSON object provides information on the ports the camera uses for endpoints.

Return Value	Data Type	Description	Required
httpPort	Integer	Port for HTTP server. (Default: 6624)	Not defined
httpUpdatesPort	Integer	Port to receive updates over HTTP (Default: 6624)	Not defined

This command returns no errors.

Examples

```
Request
GET /osc/info HTTP/1.1
Host: [camera ip address]:[httpPort]
Accept: application/json
X-XSRF-Protected: 1

Response
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Content-Length: {CONTENT_LENGTH}
X-Content-Type-Options: nosniff

{
    "manufacturer": "LGE",
    "model": "LG-R105",
    "serialNumber": "LGR105cf99f269",
    "firmwareVersion": "R105160302",
    "supportUrl": "developer.lge.com/friends",
    "endpoints": {
        "httpPort": 6624,
        "httpUpdatesPort": 6624
    },
    "gps": true,
    "gyro": true,
    "uptime": 48,
    "apiLevel": "v2",
    "api": [
        "/osc/checkForUpdates",
        "/osc/commands/execute",
        "/osc/commands/status",
        "/osc/info",
        "/osc/state"
     ]
}
```

4.1.5 /osc/state

Description:

The /osc/state API returns the device information that change over time such as battery level, battery state, etc.

Syntax:

```
POST /osc/state HTTP/1.1
```

Parameters

This command takes no input.

Command Output

Return Value	Data Type	Description	Required
fingerprint	String	The current fingerprint of the camera device	Not defined
state*	Object	An Object that contains the camera's current information	Not defined

* state Object

Return Value	Data Type	Description	Required
storageChanged	Boolean	SD card detected: true SD card not detected: false	Not defined
_batteryLevel	Integer	Remaining battery level in percentage	Not defined
_chargeState	String	Battery charging state	Not defined
_sdCardState	Boolean	SD card not inserted: "false" SD card inserted: "true"	Not defined
_fileChanged	String	Most recent file activity Added: "A", deleted: "D", no change: "X"	Not defined
_captureStatus	String	Camera's recording state whether in standby, recording or capturing state	Not defined
_cameraId	String	The state of camera lens settings. full-front-camera,full-rear-camera,half-front-camera,half-rear-camera	Not defined

This command returns no errors.

Examples

```
Request
POST /osc/state HTTP/1.1
Host: [camera ip address]:[httpPort]
```

```
Accept: application/json
X-XSRF-Protected: 1

Response
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Content-Length: {CONTENT_LENGTH}
X-Content-Type-Options: nosniff

{
   "fingerprint": "0951X1039021",
   "state": {
      "batteryLevel": 0.95,
      "_batteryState": "charging(USB)",
      "_fileChanged": "X",
      "_sdCardState": true,
      "storageChanged": true,
      "_captureStatus": "idle",
      "_cameraId": "full-rear-camera"
   }
}
```

4.2 OSC Commands API

Commands input/output JSON and are part of HTTP requests/responses respectively.

4.2.1 camera.delete

Description:

Deletes an image given its URL.

Syntax:

```
{
  "name": "camera.delete",
  ...
}
```

Parameters

Parameter	Data Type	Description	Required
fileUrls	String Array	A list of URLs for files to be deleted from the camera. fileUrl is the absolute URL of the file, obtained previously, e.g. from listFiles or takePicture commands. There are three special cases: • The list is empty: delete all files in the camera. • The list only contains string “image”: delete all images in the camera. • The list only contains string “video”: delete all videos in the camera.	Required

Command Output

Error Code	Data Type	Description
missingParameter	String	The required fileUrl is not specified.
invalidParameterName	String	An input parameter name is unrecognized
invalidParameterValue	String	The input parameter name is recognized, but its value is invalid; for example, the fileUrl value doesn't exist, its data type is incorrect, or maxSize has the wrong data type.

Examples

```
POST /osc/commands/execute  HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH} X-XSRF-Protected: 1

{
  "name": "camera.delete",
```

```
  {
    "parameters":  {
      "fileUrl": "/media/i/temp.jpg"
    }
  }
}
```

4.2.2 camera.getFile

Description:

Returns either an image or a video given its URL. The client should know if the file it is requesting is an image or a video.
* If it is an image: the request should include "Accept: image/jpeg" and the response should include "Content-Type: image/jpeg".
* If it is a video: the request should include "Accept: video/mp4" and the response should include "Content-Type: video/mp4".
* If maxSize is specified, then the thumbnail image of the target file is returned instead.

Syntax:

```
{
  "name": "camera.getFile",
  ...
}
```

Parameters

Parameter	Data Type	Description	Required
fileUrl	String	URL of the target file	Required
maxSize	Integer	(Optional) Maximum size of the requested thumbnail image for the image or the video. If it is an image: if maxSize is omitted or larger than the full-size image, the full-size image is returned. If it is a video: if maxSize is omitted, the video is returned.	Optional

Command Output

Binary data(JPEG or MP4) or thumbnail image(JPEG)

Error Code	Data Type	Description
missingParameter	String	The required fileUrl is not specified.
invalidParameterName	String	An input parameter name is unrecognized
invalidParameterValue	String	The input parameter name is recognized, but its value is invalid; for example, the fileUrl value doesn't exist, its data type is incorrect, or maxSize has the wrong data type.

Examples

```
POST /osc/commands/execute HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: image/jpeg
Content-Length: {CONTENT_LENGTH}
X-XSRF-Protected: 1

{
    "name": "camera.getFile",
    {
        "parameters": {
            "fileUrl": "/media/i/temp.jpg"
        }
    }
}
```

4.2.3 camera.getLivePreview

Description:

Returns one shot of the current preview frame of the camera. Recursively call this command to build preview stream.

Syntax:

```
{
  "name": "camera.getLivePreview",
  ...
}
```

Parameters

This command takes no input.

Command Output

Binary data(JPEG)

Error Code	Data Type	Description
disabledCommand	String	Command is currently disabled.

Examples

```
Request
POST /osc/commands/execute HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: image/jpeg
Content-Length: {CONTENT_LENGTH}
X-XSRF-Protected: 1
{
```

```
    "name": "camera.getLivePreview"
}

Response
HTTP/1.1 200 OK
Content-Type: image/jpeg
Content-Length: {CONTENT_LENGTH}
X-Content-Type-Options: nosniff
Binary Data (JPEG)
```

4.2.4 camera.getMetadata

Description:

Returns file metadata given its URL. The image header lists the Exif and XMP fields.

Syntax:

```
{
  "name": "camera.getMetadata",
  ...
}
```

Parameters

Parameter	Data Type	Description	Required
fileUrl	String	Absolute URL of the target file of type String	Optional

Command Output

The image header:

Return Value	Data Type	Description	Required
exif *	Object	EXIF information in JSON format	Not defined
photosphere **	Object	Photo Sphere XMP information in JSON format	Not defined

The video header

Return Value	Data Type	Description	Required
videoXmp ***	Object	The following table lists all *required* fields for Spherical Video XMP, adapted from Allowed Global Metadata in this link. Please see also this link for encoding settings.	Not defined

* exif Obect

Return Value	Data Type	Description	Required
ExifVersion	String	The supported Exif version. Absence of this field is signifies nonconformance to the standard (see section 4.2). A 4-byte	Not defined

		ASCII code representing the version indicates conformance to the standard: `0220`. Since the type is UNDEFINED, there is no NULL for termination.	
`ImageDescription`	`String`	Image title/name.	Not defined
`DateTime`	`String`	Image creation/modification date and time (Exif uses a single field to represent either creation or last modification). The format is, `YYYY:MM:DD HH:MM:SS`. Time is 24-hour format. Date and time are separated by a blank character `[20.H]`. When the date and time are unknown, all the character spaces except colons (":") may be filled with blank characters, or else the Interoperability field may be filled with blank characters. The character string length is 20 bytes including NULL for termination. When the field is left blank, it is treated as unknown.	Not defined
`ImageWidth`	`Number`	The number of pixels in a row of image data.	Not defined
`ImageLength`	`Number`	The number of rows of image data.	Not defined
`ColorSpace`	`Number`	Defines the color space in which the image is intended to display. For further details see section 4.6.5.B.	Not defined
`Compression`	`Number`	The compression scheme used for image data.	Not defined
`Orientation`	`Number`	The image orientation viewed in terms of rows and columns.	Not defined
`Flash`	`Number`	Provides the status of flash used during image capture.	Not defined
`FocalLength`	`Number`	Focal length of the lens, in mm.	Not defined
`WhiteBalance`	`Number`	Provides white balance setting used during image capture.	Not defined
`ExposureTime`	`Number`	Exposure time (in seconds).	Not defined
`FNumber`	`Number`	F number used during image capture.	Not defined
`ExposureProgram`	`Number`	Class of the program used during image capture.	Not defined
`ISOSpeedRatings`	`Number`	ISO Speed and Latitude of the camera,	Not defined

		as specified in ISO 12232.	
ShutterSpeedValue	Number	Shutter speed. The unit is the APEX (Additive System of Photographic Exposure) setting (see Annex C).	Not defined
ApertureValue	Number	Lens aperture as an APEX value.	Not defined
BrightnessValue	Number	APEX value of image brightness. Ordinarily falls within the range of -99.99 to 99.99. Note that a value of FFFFFFFF.H indicates *unknown*.	Not defined
ExposureBiasValue	Number	APEX value of exposure bias. Ordinarily falls within the range of –99.99 to 99.99.	Not defined
GPSProcessingMethod	String	Names the geolocation method used. The first byte indicates the character code used (Table 6, Table 7), followed by the method name.	Not defined
GPSLatitudeRef	String	Indicates northern or southern latitude. N indicates northern, S is southern.	Not defined
GPSLatitude	Number	The latitude at which the image was captured.	Not defined
GPSLongitudeRef	String	Indicates eastern or western longitude. E indicates eastern, and W is western.	Not defined
GPSLongitude	Number	The longitude at which the image was captured.	Not defined
Make	String	The camera manufacturer.	Not defined
Model	String	The camera model name or number.	Not defined
Software	String	The name and version of the camera software or firmware used for image capture.	Not defined
Copyright	String	The copyright notice of the person or organization claiming rights to the image.	Not defined
MakerNote	String	A tag for manufacturers of Exif writers to record any desired information. The contents are up to the manufacturer, but this tag should not be used for any other than its intended purpose.	Not defined

** photosphere Object

Return Value	Data Type	Description	Required
`ProjectionType`	`Open Choice of Text`	Projection type. Google currently support `equirectangular`.	Not defined
`UsePanoramaViewer`	`Boolean`	Display the image in a panorama viewer (`True`), or as a flat image (`False`). Default to `True`. Please note both have capital initials.	Not defined
`PoseHeadingDegrees`	`Real`	Compass heading, in degrees, for the image center. Value range is >= 0 and < 360.	Not defined
`CroppedAreaImageWidthPixels`	`Integer`	Original width in pixels of the unedited image (equal to the actual image's width for unedited images).	Not defined
`CroppedAreaImageHeightPixels`	`Integer`	Original height in pixels of the image (equal to the actual image's height for unedited images).	Not defined
`FullPanoWidthPixels`	`Integer`	Original full panorama width from which the image was cropped. Or, if only a partial panorama was captured, this specifies the width of what the full panorama would have been.	Not defined
`FullPanoHeightPixels`	`Integer`	Original full panorama height from which the image was cropped. Or, if only a partial panorama was captured, this specifies the height of what the full panorama would have been.	Not defined
`CroppedAreaLeftPixels`	`Integer`	Column where the left edge of the image was cropped from the full sized panorama.	Not defined
`CroppedAreaTopPixels`	`Integer`	Row where the top edge of the image was cropped from the full sized panorama.	Not defined

*** videoXmp Object

Return Value	Data Type	Description	Required
`Spherical`	`Boolean`	Flag indicating if the video is a spherical	Not defined

		video	
`Stitched`	`Boolean`	Flag indicating if the video is stitched.	Not defined
`StitchingSoftware`	`String`	Software used to stitch the spherical video.	Not defined
`ProjectionType`	`Open Choice of Text`	Projection type used in the video frames. Google products currently support `equirectangular`.	Not defined
`SourceCount`	`Integer`	Number of cameras used to create the spherical video.	Not defined
`Timestamp`	`Integer`	Epoch timestamp of when the first frame in the video was recorded.	Not defined
`CroppedAreaImageWidthPixels`	`Integer`	Width of the video frame to display (e.g. cropping).	Not defined
`CroppedAreaImageHeightPixels`	`Integer`	Height of the video frame to display (e.g. cropping).	Not defined
`FullPanoWidthPixels`	`Integer`	Width of the encoded video frame in pixels.	Not defined
`FullPanoHeightPixels`	`Integer`	Height of the encoded video frame in pixels.	Not defined
`CroppedAreaLeftPixels`	`Integer`	Column where the left edge of the video frame was cropped from the full sized panorama.	Not defined
`CroppedAreaTopPixels`	`Integer`	Row where the top edge of the video frame was cropped from the full sized panorama.	Not defined

Error Code	Data Type	Description
missingParameter	String	fileUrl is not specified.
invalidParameterName	String	The input parameter name is unrecognized
invalidParameterValue	String	The input parameter is recognized, but its value is invalid; for example, the fileUrl doesn't exist, or its data type is incorrect.

Examples

```
Request
POST /osc/commands/execute HTTP/1.1
```

```
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH}
X-XSRF-Protected: 1

{
    "parameters": {
        "fileUrl": "file URI"
    }
}

Response
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Content-Length: {CONTENT_LENGTH}
X-Content-Type-Options: nosniff
{
    "name": "camera.getMetadata",
    "state": "done",
    "results": {
        "exif": {
            ...
            "ImageWidth": 2000,
            "ImageLength": 1000,
            ...
        },
        "photosphere": {
            "ProjectionType": "equirectangular",
            "UsePanoramaViewer": true,
            ...
        }
    }
}
```

4.2.5 camera.getOptions

Description:

Returns current settings for requested properties.

Syntax:

```
{
  "name": "camera.getOptions",
  ...
}
```

Parameters

Parameter	Data	Description	Required
optionNames	Array	A String Array of property names to return.	Required

Command Output

Return Value	Data Type	Description	Required
options	Object	JSON <key, value> pairs of the requested properties. The value can be any of the following types: String, String Array, Number, Number Array, Boolean, Object, Object Array. See Options.	Required

Error Code	Data Type	Description
missingParameter	String	One or more required parameters is missing; for example, options is not specified.
invalidParameterName	String	One or more input parameter option or option name is unrecognized or unsupported.
invalidParameterValue	String	One or more input parameter or option name is recognized, but its value is invalid; for example, a data type is incorrect.

Examples

```
Request
POST /osc/commands/execute HTTP/1.1
Host: [camera ip address]:[httpPort]
Accept: application/json
X-XSRF-Protected: 1
{
    "parameters": {
       "optionNames": [
           "iso",
           "isoSupport"
      ]
    }
}

Response
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Content-Length: {CONTENT_LENGTH}
X-Content-Type-Options: nosniff
{
  "results": {
      "options": {
          "iso": 200,
          "isoSupport": [100, 200, 400, 800, 1600]
      }
  }
}
```

4.2.6 camera.listFiles

Description:

Lists all images/all videos/all images and videos in the camera. It might take several requests to list all files.

Note: File list MUST be sorted by capture time, newest first.

It is highly recommended that maxThumbSize be set to null (omit this parameter); otherwise it takes too long to retrieve return messages due to a large amount of data transfer.
A token (continuationToken) used to fetch image files cannot be used as a token to fetch video files.

Syntax:

```
{
  "name": "camera.listImages",
  ...
}
```

Parameters

Parameter	Data Type	Description	Required
fileType	String	Type of the files to be listed, should be any of the three: "image", "video", "all".	Required
entryCount	Integer	Desired number of entries to return.	Required
maxThumbSize	Integer	Maximum size of thumbnail images (either for images or for videos). It is set to null when the client wants to omit thumbnail images from the result.	Optional
continuationToken	String	(Optional) An opaque continuation token of type String, returned by previous listFiles call, used to retrieve next files. Omit this parameter for the first listFiles call of the same type ("image", "video" or "all").	Optional

Command Output

Return Value	Data Type	Description	Required
entries *	Array of Objects	A list of image properties. Each entry should contain the following fields except for latitude and longitude, which are optional	Not defined
totalEntries	Integer	Total number of entries in storage.	Not defined
continuationToken	String	(Optional) Set only if the result is incomplete (incomplete means any listing that does not include the last image). To fetch remaining entries, the client should call listImages command again with the token. The token is formatted in the following manner: "image_11_20" – the returned list is an image list, the next index of the list is 11, and the total number of image files in the storage is 20.	Optional

* entries Object

Return Value	Data Type	Description	Required
Name	String	Name of the image/video.	Not defined

fileUrl	String	Absolute URL of the image/video, which can be used to download from the camera directly.	Not defined
Size	String	Size in bytes of the image/video.	Not defined
dateTimeZone	String	Date, time, and time zone for the image, in the format: YYYY:MM:DD HH:MM:SS+(-)HH:MM. Use 24-hour format for the time. Date and time are separated by one blank character. Time zone is offset from UTC time.	Not defined
lat	Double	Latitude of the image capture location.	Optional
lng	Double	Longitude of the image capture location.	Optional
width	Integer	Width of the image or each video frame.	Not defined
height	Integer	Height of the image or each video frame.	Not defined
thumbnail	String	Base64 encoded string for thumbnail image (when maxThumbSize != null).	Not defined
isProcessed	Boolean	A boolean value indicating if the file is processed (e.g. stitched) or it is only a preview. This should be true by default unless delayProcessing is set to true.	Not defined
_orientation	Integer	Orientation of the image or video, represented by rotated angle. [0, 90, 180, 270]	Not defined
_recordTime	Integer	The length of the recorded video file	Not defined

Error Code	Data Type	Description
missingParameter	String	Any required parameter is not specified; for example, entryCount is not specified.
invalidParameterName	String	The input parameter name is unrecognized.
invalidParameterValue	String	The input parameter name is recognized, but its value is invalid; for example, the continuationToken doesn't exist, is out of range, its data type is incorrect, maxThumbSize has the wrong data type.

Examples

```
POST /osc/commands/execute HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH}
X-XSRF-Protected: 1

{
    "name": "camera.listFiles",
    {
```

```
        "parameters": {
            "fileType": "all",
            "entryCount": 50,
            "maxThumbSize": 200,
            "continuationToken": "image_11_20"
        }
    }
}
```

4.2.7 camera.setOptions

Description:

Sets values for specified properties; for example, GPS on/off, date & time, ISO, white balance, shutter speed, sleep/power-off delay, and so on.

Syntax:

```
{
  "name": "camera.setOptions",
  ...
}
```

Parameters

Parameter	Data Type	Description	Required
options	Object	JSON <key, value> pairs of the properties to set. The value can be any of the following types: String, String Array, Number, Number Array, Boolean, Object, Object Array. See Options.	Required

Command Output

This command returns no result.

Error Code	Data Type	Description
missingParameter	String	One or more required parameters is missing; for example, options is not specified.
invalidParameterName	String	One or more input parameter option or option name is unrecognized or unsupported.
invalidParameterValue	String	One or more input parameter or option name is recognized, but its value is invalid; for example, a data type is incorrect.

Examples

```
POST /osc/commands/execute  HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH} X-XSRF-Protected: 1
```

```
{
  "name": "camera.setOptions",
  {
    "parameters": {
        "options": {
            "iso": 200
        }
    }
  }
}
```

4.2.8 camera.startCapture

Description:

Starts video capture or interval image capture depending on value of `captureMode`. "image", "video", "interval" can be set to `captureMode` (the command doesn't work if "image" is set). Use this command after setting `captureMode` with **camera.setOptions** command.

Please note this command doesn't return fileUrls immediately since the capture takes `captureNumber` to complete (in case of non-open-ended capture). The progress of the command can be obtained by polling the camera periodically. For example, poll after an estimated duration (time = captureInterval * captureNumber) using the **/osc/commands/status** API and fileUrls is the result returned when the command finishes.

Syntax:

```
{
    "name": "camera.startCapture"
}
```

Parameters

This command has no parameter in request body.

Command Output

Return Value	Data Type	Description	Required
_recordingRemainTime	Number	Returns the amount of time available for recording (in seconds)	Optional
_recordingTime	Number	Returns the amount of time the camera has been recording (in seconds)	Optional
_recordingMaxTime	Number	The maximum possible recording time (in seconds – applied only when recording in maximum resolution)	Optional
fileUrls	String Array	A list of URLs of captured image files (for interval capture only)	Optional

Error Code	Data Type	Description

disabledCommand	String	Command is currently disabled; for example, the camera is in video shooting mode

Examples

```
POST /osc/commands/execute HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH}
X-XSRF-Protected: 1

{
    "name": "camera.startCapture"
}
```

4.2.9 camera.stopCapture

Description:

Stops video capture or open-ended interval image capture, determined by capture mode in Options. If the corresponding startCapture is for interval image capture and the termination condition is set (e.g. it terminates after capturing 10 images), then stopCapture is not needed, however, it should still be treated as a valid command for early termination.

Poll the progress of the command by using the **/osc/commands/status** API to determine whether the command has completed so that the full list of captured files (populated in fileUrls) can be obtained.

Syntax:

```
{
    "name": "camera.stopCapture"
}
```

Parameters

This command has no parameter in request body.

Command Output

Return Value	Data Type	Description	Required
fileUrls	String Array	The file URL of image/video	Required

Error Code	Data Type	Description
disabledCommand	String	Command is currently disabled; for example, the camera is in video shooting mode

Examples

```
None
```

4.2.10 camera.takePicture

Description:

Captures an equirectangular image, saving lat/long coordinates to EXIF (if your camera features its own GPS or GPS is enabled on connected mobile phones). Call **camera.setOptions** prior to this command call if needed.

Please note this command doesn't return fileUrl immediately since the capture (including in-camera stitching) usually takes a while (varies from seconds to minutes) to complete. The progress of the command is obtained by polling the camera periodically using the **/osc/commands/status** API and fileUrl is the result returned when the command finishes.

Syntax:

```
{
  "name": "camera.takePicture"
}
```

Parameters

This command has no parameter.

Command Output

Return Value	Data Type	Description	Required
fileUrl	String	URI of the captured image. Manufacturers decide whether to use absolute or relative URIs; for example, "http://android.com/robots.txt" is an absolute URI while "robots.txt" is a relative URI. Clients may treat this as an opaque identifier.	Required
fileUri	String	URI of the captured image. Manufacturers decide whether to use absolute or relative URIs; for example, "http://android.com/robots.txt" is an absolute URI while "robots.txt" is a relative URI. Clients may treat this as an opaque identifier.	Required

Error Code	Data Type	Description
disabledCommand	String	Command is currently disabled; for example, the camera is in video shooting mode

Examples

```
POST /osc/commands/execute HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH}
X-XSRF-Protected: 1

{
   "name": "camera.takePicture"
}
```

4.2.11 camera._getRecordingStatus

Description:

Gets the recording status. This command is used while the camera is recording a video. If used while not recording, invalidParameterValue error is returned.

Syntax:

```
{
  "name": "camera._getRecordingStatus"
}
```

Parameters

This command has no parameter.

Command Output

Return Value	Data Type	Description	Required
_recordingPaused	Boolean	This key value indicates whether the camera is in the middle of recording or the recording is paused.	Required
_recordingRemainTime	Number	The time remaining the storage becomes full (in seconds)	Required
_recordingTime	Number	Returns the amount of time the camera has been recording (in seconds)	Required
_recordingMaxTime	Number	The maximum possible (in maximum resolution 20 minutes) recording time (in seconds)	Required
_cameraState	String	If invalidParameterValue error (command used while not recording) or a server oriented error is returned for camera._getRecordingStatus command, the camera state is assigned to this key. ex)idle, recording, shooting, interval, timer_counting, idle_by_temperature, error	Optional

Error Code	Data Type	Description
missingParameter	String	sessionId is missing.
invalidParameterName	String	The input parameter name is unrecognized.
invalidParameterValue	String	The input parameter name is recognized, but its value is invalid; for example, the sessionId doesn't exist, is no longer active, or its data type is incorrect.

Examples

```
POST /osc/commands/execute HTTP/1.1
Host: [camera ip address]:[httpPort]
```

```
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH}
X-XSRF-Protected: 1

{
   "name": "camera._getRecordingStatus"
}
```

4.2.12 camera._liveSnapshot

Description:

Captures an image during recording, saving lat/long coordinates to EXIF (if your camera features its own GPS or GPS is enabled on connected mobile phones). This command does not work and returns disabledCommand error if the command is used while the camera is not in recording.

Note that this command only works in 180° modes (corresponding _cameraId option is set to either “half-front-camera” or “half-rear-camera”).

Poll the progress of the command by using the **/osc/commands/status** API to determine whether the command has completed so that the captured file (populated in fileUrl) can be obtained.

Syntax:

```
{
  "name": "camera._liveSnapshot"
}
```

Parameters

This command has no parameter.

Command Output

Return Value	Data Type	Description	Required
fileUrl	String	URI of the captured image. Manufacturers decide whether to use absolute or relative URIs; for example, "http://android.com/robots.txt" is an absolute URI while "robots.txt" is a relative URI. Clients may treat this as an opaque identifier.	Required

Error Code	Data Type	Description
disabledCommand	String	Command is currently disabled; for example, the camera is in video shooting mode

Examples

```
POST /osc/commands/execute HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
```

```
Content-Length: {CONTENT_LENGTH}
X-XSRF-Protected: 1

{
    "name": "camera._liveSnapshot"
}
```

4.2.13 camera._manualMetadata

Description:

Requests to the camera current settings of White Balance, Exposure Value, ISO, Shutter speed information. Use this command after setting exposureProgram to manual (NOT_DEFINE = 0, MANUAL = 1, NORMAL_PROGRAM = 2) with **camera.setOptions**.

Syntax:

```
{
  "name": "camera._manualMetadata"
}
```

Parameters

This command has no parameter.

Command Output

Return Value	Data Type	Description	Required
_meta_wb	Number	White Balance Manual Meta data	Required
_meta_ev	Number	exposure compensation Manual Meta data	Required
_meta_iso	Number	iso manual meta data	Required
_meta_ss	Number	shutter speed manual meta data	Required

Error Code	Data Type	Description
disabledCommand	String	exposureProgram is not manual program

Examples

```
POST /osc/commands/execute HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH}
X-XSRF-Protected: 1

{
    "name": "camera._manualMetadata"
}
```

4.2.14 camera._pauseRecording

Description:

Pauses recording.

Syntax:

```
{
  "name": "camera._pauseRecording"
}
```

Parameters

This command has no parameter.

Command Output

Return Value	Data Type	Description	Required
_cameraState	String	Returns the camera state to indicate why PauseRecording failed at the time of command submission. ex)idle, recording, shooting, interval, timer_counting, idle_by_temperature, error	Optional

Error Code	Data Type	Description
missingParameter	String	sessionId is missing.
invalidParameterName	String	The input parameter name is unrecognized.
invalidParameterValue	String	The input parameter name is recognized, but its value is invalid; for example, the sessionId doesn't exist, is no longer active, or its data type is incorrect.

Examples

```
POST /osc/commands/execute HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH}
X-XSRF-Protected: 1

{
   "name": "camera._pauseRecording"
}
```

4.2.15 camera._resumeRecording

Description:

Resumes recording.

Syntax:

```
{
  "name": "camera._resumeRecording",
  ...
}
```

Parameters

This command has no parameter.

Command Output

Return Value	Data Type	Description	Required
_recordingRemainTime	Number	Returns the amount of time available for recording (in seconds)	Required
_recordingTime	Number	Returns the amount of time the camera has been recording (in seconds)	Required
_recordingMaxTime	Number	The maximum possible recording time (in seconds – applied only when recording in maximum resolution)	Required
_cameraState	String	Returns the camera state to indicate why resumeRecording failed at the time of command submission. ex)idle, recording, shooting, interval, timer_counting, idle_by_temperature, error	Optional

Error Code	Data Type	Description
missingParameter	String	sessionId is missing.
invalidParameterName	String	The input parameter name is unrecognized.
invalidParameterValue	String	The input parameter name is recognized, but its value is invalid; for example, the sessionId doesn't exist, is no longer active, or its data type is incorrect.

Examples

```
POST /osc/commands/execute  HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH}
```

```
X-XSRF-Protected: 1

{
  "name": "camera._resumeRecording",
}
```

4.2.16 camera._startPreview

Description:

Obtains information on preview data from the camera.
protocol : UDP socket
format : mpegts
encoder : h.264 baseline profile
resolution
preview only : 768x768 (full sphere mode) / 1280x720 (half sphere mode) @14~15fps, VBR
preview on recording : 448x448 (full sphere mode) / 448x252 (half sphere mode) @8~9 fps, VBR

Syntax:

```
{
  "name": "camera._startPreview",
  ...
}
```

Remark:

This command can be used together with **camera.startCapture** command so that preview can be obtained while capturing.

Parameters

Parameter	Data Type	Description	Required
sessionId	String	Set this value to “LGFRIENDSCAMERA”	Required
_streamType	String	Stream Type: supports only "UDP"	Optional

Command Output

The server returns information on preview in the `_previewUri` field with which one can receive preview stream. The server starts transmitting to the port assigned to client’s IP address through udp socket. The client can receive preview stream either with its own udp socket or from udp://localhost:1234.

Error Code	Data Type	Description
invalidParameterName	String	The input parameter name is unrecognized.
invalidParameterValue	String	The input parameter name is recognized, but its value is invalid; for example, the sessionId doesn't exist, is no longer active, or its data type is incorrect.

Examples

```
POST /osc/commands/execute HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH}
X-XSRF-Protected: 1

{
    "name": "camera._startPreview",
    {
        "parameters": {
             "sessionId": "LGFRIENDSCAMERA",
             "_streamType": "UDP"
        }
    }
}

HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Content-Length: {CONTENT_LENGTH}
X-Content-Type-Options: nosniff
{
    "results": {
    "_previewUri" : "udp://:1234"},
    "name" : "camera._startPreview",
    "state":"done"
}
```

4.2.17 camera._stopPreview

Description:

Stops transferring preview video stream from the camera.

Syntax:

```
{
  "name": "camera._stopPreview",
  ...
}
```

Parameters

Parameter	Data Type	Description	Required
sessionId	String	Set this value to “LGFRIENDSCAMERA”	Required

Command Output

None.

Error Code	Data Type	Description
invalidParameterName	String	The input parameter name is unrecognized.
invalidParameterValue	String	The input parameter name is recognized, but its value is invalid; for example, the sessionId doesn't exist, is no longer active, or its data type is incorrect.

Examples

```
POST /osc/commands/execute  HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH}
X-XSRF-Protected: 1

{
  "name": "camera._stopPreview",
  {
    "parameters":  {
       "sessionId": " LGFRIENDSCAMERA"
    }
  }
}
```

4.3 Device Options

4.3.1 Options

The following table shows all options available to set and get by **camera.setOptions** and **camera.getOptions**. An option cannot be changed when its corresponding support is empty or contains only one option.

Parameter	Data Type	Description	read/write
captureMode	String	Current capture mode. Default to image.	rw
captureModeSupport	String Array	"captureModeSupport": ["image", "video", "interval"] List of capture modes currently available. Minimum requirement for API v1 is ["image"], and minimum requirement for API v2 is ["image", "interval"], where "interval" represents the mode of capturing a series of images spaced at a certain interval, please see also "captureInterval" and "captureIntervalSpport". Two additional modes ("video" and "walkaround")are supported by API v2, so the complete supported set by API v2 is ["image", "interval", "video", "walkaround"], where "video" represents the video capture mode and "walkaround" represents the mode of capturing two images in sequence, the first with the users standing anywhere relative to the camera and the second with the users standing on the opposite side (relative to the first standing point) of the camera. This allows the camera to remove the users from the final image by combining the two images. When "walkaround" is supported by the camera and set to be the current capture mode, the client should expect to send two takePicture commands to the camera, where the response of the first command (when it finishes) indicates the camera is ready to take the second image while the second command should return the final image when it finishes. To add more capture modes not supported yet, please prefix vendor-specific modes with an underscore (_);	r
exposureProgram	Number	Current exposure program.	rw
exposureProgramSupport	Number Array	"exposureProgramSupport": [0, 1, 2] List of exposure programs currently available; for example, [0, 1, 2, 3, 4]. Each integer represents a different exposure program: 0 = Not defined 1 = Manual 2 = Normal program 3 = Aperture priority 4 = Shutter priority Select the following link to download further details about ExposureProgram. Note: Please use number 9 for “ISO priority” if needed.	r
iso	Number	Current ISO speed setting. Default to 0 when ISO is in auto mode.	rw

isoSupport	Number Array	"isoSupport": [0, 100, 200, 400, 800, 1600, 3200] List of ISO settings currently available; for example, [0, 100, 200, 400, 800, 1600], where 0 represents auto mode.	r
shutterSpeed	Number	Current shutter speed setting. Default to 0 when it is in auto mode.	rw
shutterSpeedSupport	Number Array	"shutterSpeedSupport": [0, 0.5, 0.25, 0.125, 0.06666667, 0.033333335, 0.016666668, 0.008, 0.004, 0.002, 0.001, 5.0E-4, 2.5E-4, 1.6666666E-4] List of shutter speeds currently available; for example, [0, 0.067, 0.033, 0.017, 0.008], where 0 represents auto mode.	r
aperture	Number	Current aperture setting, in f-stops.	rw
apertureSupport	Number Array	"apertureSupport": 1.8 List of aperture settings currently available, expressed in f/number; for example, [1.4, 2, 2.8, 4, 5.6, 8, 11] or [] when it is auto mode.	r
whiteBalance	String	Current white balance setting; for example, daylight. Default to auto.	rw
whiteBalanceSupport	String Array	"whiteBalanceSupport": ["auto", "incandescent", "fluorescent", "warm-fluorescent", "daylight", "cloudy-daylight", "twilight", "shade", "manual"] List of white balance settings currently available, can be a subset of the predefined list:["auto", "incandescent", "fluorescent", "daylight", "cloudy-daylight", "shade", "twilight"]. Values for each: incandescent, around 3200K fluorescent, around 4000K datalight, around 5200K cloudy-daylight, around 6000K shade, around 7000K twilight, around 12000K Prefix vendor-specific setting names with an underscore; for example: _vendor-setting	r
exposureCompensation	Number	Current exposure compensation.	rw
exposureCompensationSupport	Number Array	"exposureCompensationSupport": [-2.000004, -1.833337, -1.66667, -1.500003, -1.333336, -1.166669, -1.000002, -0.833335, -0.666668, -0.500001, -0.333334, -0.166667, 0, 0.166667, 0.333334, 0.500001, 0.666668, 0.833335, 1.000002, 1.166669, 1.333336, 1.500003, 1.66667, 1.833337, 2.000004] List of of exposure compensations currently available, usually in step 0.33 or 0.5; for example: [-1, -0.67, -0.33, 0, 0.33, 0.67, 1]	r
fileFormat	Object	Current image type and resolution; for example: { "type": "jpeg", "width": 2000, "height": 1000 }	rw

fileFormatSupport	Object Array	List of the file formats currently available depending on captureMode; for example: [{ "type": "jpeg", "width": 2000, "height": 1000 }, ... { "type": "raw", "width": 200, "height": 100 }] or [{ "type": "mp4", "width": 1920, "height": 1080, "framerate": 24 }, ... { "type": "mp4", "width": 1280, "height": 720, "framerate": 30 }] Select the following link for more information about all types.	r
speedOptimized	Boolean	Default to false, which means image/video should be captured at the highest quality available. When it is set to true, cameras should capture/process at an optimized speed, e.g. by capturing at a lower but still reasonable resolution or doing other optimizations.	rw
previewFormat	Object	Not supported	rw
previewFormatSupport	Object Array	"previewFormatSupport": {"width": 768, "height": 768, "framerate": 1}	r
exposureDelay	Number	Current delay between the takePicture command and when the exposure is started, in seconds.	rw
exposureDelaySupport	Number Array	"exposureDelaySupport": 0 List of exposure delays currently available, in seconds; for example: [0, 1, 2, 5, 10, 30, 60]	r
captureInterval	Number	Current interval between capturing two consecutive images, in seconds.	rw
captureIntervalSupport	Object	"captureIntervalSupport": {"minInterval": 2, "maxInterval": 60} Minimum and maximum intervals allowed between capturing two consecutive images, in seconds, e.g. { "minInterval": 10, "maxInterval": 60 } It might vary depending on fileFormat.	r
captureNumber	Number	Number of images to be captured for one interval capture session. Default to 0, which means the capture needs to be terminated explicitly by stopCapture command; otherwise, the capture stops automatically after it captures images of the specified number. The capture can also be interrupted when the camera is out of battery or when it is turned off intentionally.	rw
captureNumberSupport	Object	"captureIntervalSupport": {"minInterval": 2, "maxInterval": 60} Minimum and maximum number of images that can be captured during an interval capture , e.g. { "minNumber": 2, "maxNumber": 50 } It might change depending on remaining storage.	r
sleepDelay	Number	Current delay before the camera goes to sleep, in seconds.	rw

sleepDelaySupport	Number Array	"sleepDelaySupport": [0, 60, 120, 300] List of the sleep delays currently available, in seconds; for example, [0, 60, 120, 300], where 0 disables sleep mode.	r
totalSpace	Number	Read-only: Number of bytes of total storage.	r
remainingSpace	Number	Read-only: Number of bytes of free storage.	r
remainingPictures	Number	Read-only: Estimated number of remaining pictures based on current capture setting.	r
remainingVideoMinutes	Number	Estimated number of remaining video minutes based on current capture setting.	r
gpsInfo	Object	Current GPS information. Set using setOptions using phone's GPS: { "lat": 23.532, "lng": -132.35 } The lat and lng properties are decimal degrees, with lat in the range, [-90, 90], and lng in the range, [-180, 180]. When provided by a phone, please note that each GPS location is valid only until the next update from the phone. The phone must determine when to update the GPS location; for example, right before each takePicture command. 65535 notifies the camera that the current GPS location is invalid and the camera should ignore it and use its own GPS if it exists; for example, setOptions({"gpsInfo": {"lat": 65535, "lng": 65535}}) means the GPS location is invalid.	rw
dateTimeZone	String	Current date and time information. Set by setOptions using phone's date, time, and time zone. The format is, YYYY:MM:DD HH:MM:SS+(-)HH:MM. Time is in 24-hour format, date and time are separated by a blank space, andtime zone is an offset from UTC time; for example, 2014:05:18 01:04:29+8:00 is China Time Zone (UTC+8:00)	rw
hdr	String	Current HDR mode.	rw
hdrSupport	String Array	"hdrSupport": "off" If you don't support HDR mode, then it is ["off"]; if you supports one HDR mode, it is ["off", "hdr"], if multiple HDR modes (different algorithms based) are supported, then it is ["off", "hdr", "hdr1", "hdr2", ...]	r
exposureBracket	Object	Current exposure bracket setting. Available only when hdrSupport contains at least one HDR mode. If the camera uses manual exposure bracketing, the object contains two entries: 1) shots, an integer containing the number of shots to be taken; 2) increment, a float number containing an f-stop increment between shots. For example: { "shots": 3, "increment": 1.33 } If the camera uses auto exposure bracketing, the object contains: { "autoMode": true } Default to empty {} when hdrSupport contains only "off". Otherwise, manufacturers decide default values (for example, auto exposure bracketing).	rw

exposureBracketSupport	Object	Exposure bracket settings currently available; for example: { "autoMode": true, "shotsSupport": [1, 3, 5, 7], "incrementSupport": [0.33, 0.67, 1, 1.33, 1.67, 2] } Default to empty {} if hdrSupport contains only "off". When hdrSupport contains any HDR mode, but auto exposure bracketing is not supported, then autoMode will be false.	r
gyro	Boolean	Set to true to enable the camera's gyroscope module, or false to disable this feature. Default to true if the camera supports it, otherwise false. This setting can be true only when gyroSupport == true.	rw
gyroSupport	Boolean	"gyroSupport": true If the camera has a gyroscope this value should be true, otherwise, false	r
gps	Boolean	Enables/disables the camera GPS module. Value must be true to enable, or false to disable. Default value is true if the camera supports it, otherwise false. Setting is true only when gpsSupport == true.	rw
gpsSupport	Boolean	"gpsSupport": false This value should be true if the camera has its own GPS module, otherwise false.	r
imageStabilization	String	Current image stabilization operation; for example, off.	rw
imageStabilizationSupport	String Array	"imageStabilizationSupport": "off" Image stabilization options currently available. The pre-defined list is, ["off", "on"].If the camera doesn't support image stabilization, return ["off"], otherwise return ["off", "on"]. Prefix vendor-specific strings with an underscore (_); for example, ["off", "_horizontal_stabilization", "_vibration_correction"].	r
wifiPassword	String	At least 8 characters long, containing letters, numbers, symbols. It can be changed only when the camera is connected to a client device. Once it is changed, the camera must disconnect and auto reconnect using the new wifiPassword. The camera must provide a reset mechanism in case the password is forgotten; for example, a reset button to restore the factory default password.	rw
delayProcessing	Boolean	True when processing (e.g. stitching) has a lower priority than capturing, or in other words, another capture is allowed before the processing of the previous captured image is finished. This is highly recommended, especially for cameras that require long processing time.	rw

delayProcessingSupport	Bolean Array	"delayProcessingSupport": false [true] means processing (e.g. stitching) has a lower priority than capturing, and is always delayed by default. [false] means processing happens right after capturing. [true, false] means there is a choice between these two modes.	r
_sceneMode	String	auto, party, night, sports, landscape	rw
_sceneModeSupport	String	"_sceneModeSupport": ["auto", "party", "night", "sports", "landscape"]	r
_timer	Integer	shutter timer	rw
_timerSupport	Integer	"_timerSupport": [0, 3, 5, 10]	r
_storage	Integer	Storage info	r
_batteryLevel	Integer	Bettery level	r
_angle	String	Change angle in half mode	rw
_angleSupport	String	"_angleSupport": ["wide", "normal", "narrow"]	r
_audioChannel	String	changes audio channel used when recording	rw
_audioChannelSupport	String	"_audioChannelSupport": ["5.1", "2"]	r
_cameraId	String	Camera ID	rw
_cameraIdSupport	String	"_cameraIdSupport": ["full-front-camera", "full-rear-camera", "half-front-camera", "half-rear-camera"]	r
_lgISO	Number	LG ISO auto = 0	rw
_lgISOSupport	Number	"_lgISOSupport": [50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 600, 700, 800, 1000, 1500, 2000, 2700] LG ISO List depends on Device	r
_lgShutterSpeed	String	LG shutter speed "0" is auto	rw
_lgShutterSpeedSupport	String	"_lgShutterSpeedSupport": ["1/2", "1/4", "1/8", "1/15", "1/30", "1/60", "1/125", "1/250", "1/500", "1/1000", "1/2000", "1/4000", "1/6000"] 1/2,1/4,1/8,1/15,1/30,1/60,1/125,1/250,1/500,1/1000,1/2000,1/4000,1/6000 depends on Device	r

_lgWhiteBalance	Number	LG white balance depends on device	rw
_lgWhiteBalanceSupport	Number	"_lgWhiteBalanceSupport": [2300, 2400, 2500, 2600, 2700, 2800, 2900, 3000, 3100, 3200, 3300, 3400, 3500, 3600, 3700, 3800, 3900, 4000, 4100, 4200, 4300, 4400, 4500, 4600, 4700, 4800, 4900, 5000, 5100, 5200, 5300, 5400, 5500, 5600, 5700, 5800, 5900, 6000, 6100, 6200, 6300, 6400, 6500, 6600, 6700, 6800, 6900, 7000, 7100, 7200, 7300, 7400, 7500] LG white balance list 0 = auto 2300,2400,2500,2600,2700,2800,2900,3000,3100,3200,3300,3400,3500,3600,3700,3800,3900,4000,4100,4200,4300,4400,4500,4600,4700,4800,4900,5000,5100,5200,5300,5400,5500,5600,5700,5800,5900,6000,6100,6200,6300,6400,6500,6600,6700,6800,6900,7000,7100,7200,7300,7400,7500 depends on device	r

4.4 Settings

4.4.1 /settings/get

Description:

Gets the values of the setting options. Can get multiple option values by listing the key names of the options in a String Array.

Syntax:

```
{
    "parameters" : ["key", "key2", ... ]
}
```

Parameters

Parameter	Data Type	Description	Required
key	String	A set of option key names	Required

Command Output

Parameter	Data Type	Description	Required
results	Object	A set of JSON <key, value> pairs of settings properties.	Not defined

Error Code	Data Type	Description
state	String	Processing result
code	String	Error code
message	String	Error reason

Examples

```
POST /settings/get HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH}
X-XSRF-Protected: 1

{
    "parameters" : ["key", "key2", ... ]
}
```

4.4.2 /settings/set

Description:

Sets the values to the setting options. Can set multiple option values by listing the JSON key-value pairs of the options.

Syntax:

```
{
   "parameters"  : [
     {"key": "value"},
      .....
   ]
 }
```

Parameters

Parameter	Data Type	Description	Required
key	String	Option name	Required
value	String	Value to set to the specified option key	Required

Command Output

Parameter	Data Type	Description	Required
results	String	Processing result	Not defined

Error Code	Data Type	Description
state	String	Processing result
code	String	Error code
message	String	Error reason

Examples

```
POST /settings/set  HTTP/1.1
Host: [camera ip address]:[httpPort]
Content-Type: application/json;charset=utf-8
Accept: application/json
Content-Length: {CONTENT_LENGTH} X-XSRF-Protected: 1

{
  "parameters"  : [
    {"SSID": "myCamera"}
  ]
}
```

4.4.3 Setting Options

The following table shows all options available to set and get by **/settings/set** and **settings/get**. An option cannot be changed when its corresponding support is empty or contains only one option.

Parameter	Data Type	Description	Required
_wifiPW	String	password of wifi hotspot	Not defined
_sound	Integer	Device sound support on: 1 / off: 0	Not defined
_batteryLevel	Integer	Remaining battery level in percentage (read only)	Not defined
_batteryState	String	"charging"/"charged"/"disconnect" (read only)	Not defined
_plugType	String	"Unplugged"/"AC"/"USB" (read only)	Not defined
_sleepTime	Integer	Sleep time in seconds	Not defined
_totalCapacity	Long	Total storage capacity (read only)	Not defined
_freeCapacity	Long	Free storage capacity (read only)	Not defined

4.5 connectionManager API

Friends Camera SDK provides the connectionManager API, a set of methods for BLE and Wi-Fi connection. With these methods, developers can discover LG 360 CAM devices and connect the application (client) to the camera device (server).

4.5.1 BLE Connection APIs

Modifier and Type	Method
`void`	`StartScanFriends()`
`void`	`StopScanFriends()`
`void`	`enableFriendWifiAp(String address)`

public void StartScanFriends()

This method scans for Friends devices by BLE scanning.

public void StopScanFriends()

This method stops previously started BLE scanning for Friends devices.

public void enableFriendWifiAp(String address)

This method sends a request to turn on Wi-Fi Soft AP mode of the target Friends device specified by the `address` parameter.

Parameters	
address	String: The BLE address of the target Friends device

4.5.2 Wi-Fi Connection APIs

Modifier and Type	Method
`void`	`connect(String ssid, String passwd)`
`void`	`disconnect()`
`boolean`	`isConnected(String ssid)`

public void connect(String ssid, String passwd)

Thid method sends a Wi-Fi connection request with the SSID and Wi-Fi password provided in `ssid` and `passwd` paramters, respectively.

Parameters	
ssid	String: The SSID name of target Friends device
passwd	String: The Wi-Fi password. If null is assigned, search from the saved `wifiConfiguration` for the specified ssid.

public void disconnect()

This method disconnects the current Wi-Fi connection.

public boolean isConnected(String ssid)

This method examines the Wi-Fi connection state with the specified SSID name specified in `ssid` parameter.

Parameters	
ssid	String: The SSID name of target Friends device

5 Sample Application

This chapter describes how to develop an Android application with Friends Camera SDK by going through the details of the sample application.

Contents

5.1 Environment Setup
This section describes how to set up an environment to build and run the sample application.

5.2 Application Structure
This section shows the structure of the sample application.

5.3 OSC Library
This section explains how the 'osclibrary' module implements OSC APIs.

5.4 OSC Hello-world
This section demonstrates how to use OSC APIs with the 'helloFriendsCamera' tutorial application.

5.5 Common Features
This section illustrates how to implement common features that the sample application provides.

5.6 Gallery Features
This section illustrates how to implement gallery features that the sample application provides.

5.7 Video Features
This section illustrates how to implement video features that the sample application provides.

5.1 Environment Setup

5.1.1 Development Environment

The sample application is developed under the following configuration:

- IDE: Android Studio 1.5
- Build: Gradle 2.8
- JDK: 8.0

'apidemo' Module

Manifest configurations are as follows:
<SDK Version>

- minSdkVersion: 21
- targetSdkVersion: 23

<Permission>

- android.permission.BLUETOOTH
- android.permission.BLUETOOTH_ADMIN
- android.permission.ACCESS_FINE_LOCATION
- android.permission.INTERNET
- android.permission.ACCESS_NETWORK_STATE
- android.permission.ACCESS_WIFI_STATE
- android.permission.CHANGE_WIFI_STATE
- android.permission.READ_EXTERNAL_STORAGE
- android.permission.WRITE_EXTERNAL_STORAGE

<Feature>

- android.hardware.bluetooth_le

'helloFriendsCamera' Module

Manifest configurations are as follows:
<SDK Version>

- minSdkVersion: 21
- targetSdkVersion: 23

<Permission>

- android.permission.INTERNET
- android.permission.ACCESS_NETWORK_STATE

Viewer

The 'apidemo' application uses an image viewer module based on the Google Cardboard VR viewer. Download and import the following client libraries for Cardboard SDK for Android from the Google Developers site:

- cardboard.jar
- libprotobuf-java-2.6-nano.jar

Note

From Cardboard SDK for Android v0.6.0, the libprotobuf-java-2.6-nano.jar library is incorporated into the cardboard.jar library. For more details, see Release Notes of Cardboard SDK for Android.

Import the project file of the sample application to Android Studio to get started.

5.1.2 Running Environment

Enable Bluetooth and Wi-Fi from the Settings menu on the application side.

Note

The mobile device running the application must support BLE (Bluetooth Low Energy) to connect with LG 360 CAM.

5.2 Application Structure

5.2.1 Project Structure

The sample application consists of the following modules:

- 'osclibrary': an OSC API handler module
- 'helloFriendsCamera': a hello-world application that demonstrates how to use OSC APIs
- 'apidemo': a sample application that implements the fundamental camera control features

The following figure shows the project structure of the 'apidemo' application:

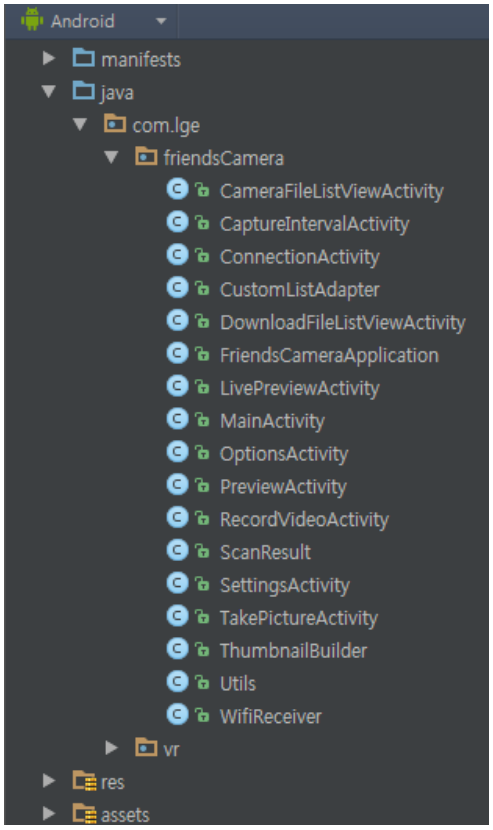


Figure 4 Project Structure of the Sample Application

5.2.2 Layout Structure

The following figure shows the layout of the main UI of the 'apidemo' module.

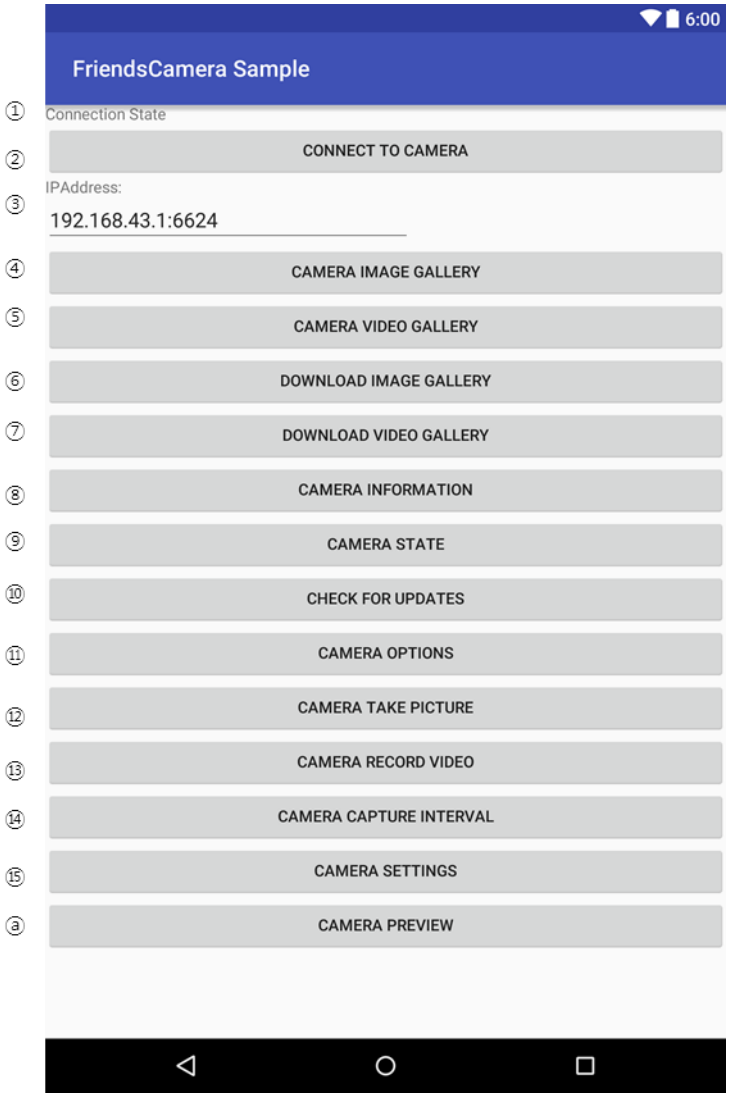


Figure 5 Main UI Layout of 'apidemo' Module

The main UI consists of the following components:

Table 10 Main UI Components

No.	Name	View Type	Description
①	Connection State	Text	Indicates the Wi-Fi connection state
②	CONNECT	Button	Connects to LG 360 CAM over Wi-Fi channel
③	IP Address	EditText	Configures the IP address of the camera
④	CAMERA IMAGE GALLERY	Button	Lists thumbnails of the images stored in the camera
⑤	CAMERA VIDEO GALLERY	Button	Lists thumbnails of the videos stored in the camera
⑥	DOWNLOAD IMAGE GALLERY	Button	Lists images downloaded from the camera

⑦	DOWNLOAD VIDEO GALLERY	Button	Lists videos downloaded from the camera
⑧	GET CAMERA INFORMAION	Button	Retrieves information of the camera that is connected to the application
⑨	GET CAMERA STATE	Button	Retrieves camera state
⑩	CHECK FOR UPDATES	Button	Checks whether the camera state has changed
⑪	CAMERA OPTIONS	Button	Gets/sets device options from/to the camera
⑫	CAMERA TAKE PICTURE	Button	Captures an image
⑬	CAMERA RECORD VIDEO	Button	Records a video
⑭	CAMERA CAPTURE INTERVAL	Button	Executes an interval capture
⑮	CAMERA SETTINGS	Button	Gets/sets settings options from/to the camera
ⓐ	CAMERA PREVIEW	Button	Watches live preview

At startup, only the CONNECT button, the DOWNLOAD IMAGE GALLERY button and the DOWNLOAD VIDEO GALLERY button are enabled if the application is not connected with any LG 360 CAM device. Other buttons get enabled when connection is established.

5.3 OSC Library

Communication between the application and the camera uses HTTP protocol. The sample application uses the `HttpAsynTask` class to perform network operations. The `HttpAsynTask` class extends `AsynTask` to send HTTP requests and handle HTTP responses in the background.

The 'osclibrary' module in the package implements classes that are responsible for each OSC API request and response handling.

5.3.1 OSC Protocol API

The sample application uses the `HttpAsynTask` class to send OSC APIs over HTTP. See the following table for OSC Protocol API implementations in the 'osclibrary' module.

Table 11 OSC API Implementation

OSC Protocol API	Java Class
/osc/info	OSCInfo.java
/osc/state	OSCState.java
/osc/commands/execute	OSCCommandsExecute.java
/osc/commands/status	OSCCommandsStatus.java
/osc/checkForUpdate	OSCCheckForUpdates.java

For the Settings interface (/settings APIs), the `FriendsCameraSettings` class is responsible for API implementation.

Key methods for HTTP communication:

- `doInBackground()` handling a request
- `onPostExecute()` handling a response

Classes whose respective API has input parameters override the `doInBackground()` method of the `HttpAsynTask` class to append input parameters. Other classes that do not have input parameters use the `super()` method of the super class. Response messages are handled by the `onPostExecute()` method.

5.3.2 OSC Commands API

All OSC commands are executed by the **/osc/commands/execute** API.

Command Execution

The `OSCCommandsExecute` class is responsible for sending commands to the camera and handling response messages. The **/osc/commands/execute** API has input parameters. This class overrides `doInBackground()` to append input parameters. For **/osc/commands/execute** API, the command to be executed is specified in the `name` parameter.

<OSCCommandsExecute.java>

```
@Override
protected Object doInBackground(Void... voids) {
```

```
    JSONObject data = new JSONObject();
    try {
        data.put("name", mCommand);
        if(mParameters != null) {
            data.put("parameters", mParameters);
        }
    } catch (JSONException e) {
        Log.v(TAG, "Error: Json error for put data in function");
        e.printStackTrace();
    }
    setHttpRequestData(data.toString());
    return super.doInBackground(voids);
}
```

Command Status

The status of command execution can be obtained by the **/osc/commands/status** API from the `OSCCommandsStatus` class. The **/osc/commands/status** API has an input parameter. This class overrides `doInBackground()` to append the input parameter. For **/osc/commands/status** API, the Command ID returned from the **/osc/commands/execute** is specified in the `id` parameter.

<OSCCommandsStatus.java>

```
@Override
protected Object doInBackground(Void... voids) {
    JSONObject data = new JSONObject();
    try {
        data.put("id",commandId);
    } catch (JSONException e) {
        e.printStackTrace();
    }
    setHttpRequestData(data.toString());
    return super.doInBackground(voids);
}
```

5.4 OSC Hello-world

The 'helloFriendsCamera' module is a hello-world tutorial application that demonstrates a typical OSC API call sequence. This application shows an example of taking a picture using OSC APIs.

The following figure shows the layout of the UI of the 'helloFriendsCamera' module:

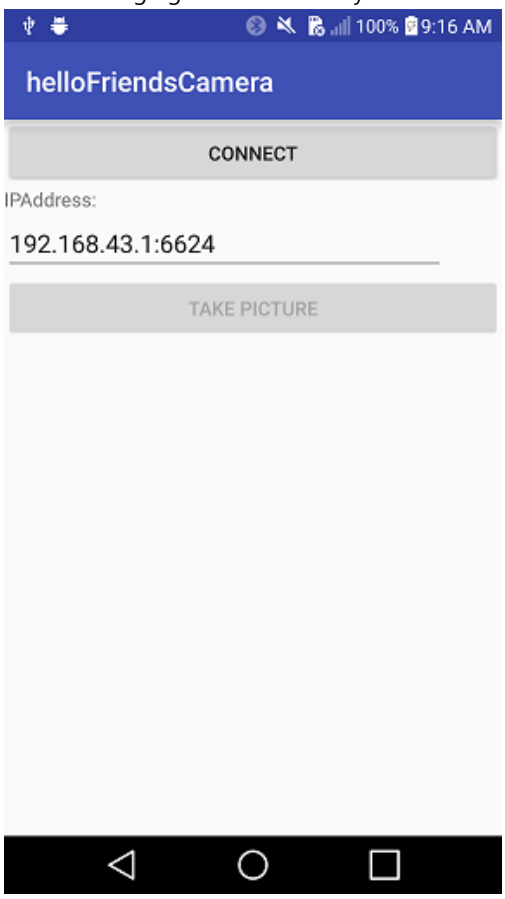


Figure 6 Main UI Layout of 'helloFriendsCamera' Module

The 'CONNECT" button starts the `ConnectionActivity` which is responsible for connection management. See Connection Management for more information. Scan for Friends devices and select a device from the scan result list. Then the application sends BLE and Wi-Fi connection requests to the selected camera device. See BLE and Wi-Fi Connection for more information.

If the connection is successfully established, the application can use 'TAKE PICTURE' to capture an image. Check the calling sequence of methods provided below:

1) `setIPPort()`
The `setIPPort()` method sets the IP address and port number for HTTP connection.

2) `takePicture()`
If 'TAKE PICTURE' button is clicked, then the `takePicture()` method is called to capture an image by using the **camera.takePicture** command.

3) `checkTakePictureState()`

If `state` of the response to **camera.takePicture** is “inProgress”, then the application calls the `checkTakePictureState()` method recursively until the state becomes “done” or it gets any error response. The `checkTakePictureState()` method sends the **/osc/commands/status** API to check the status.

4) `handleFinishTakePicture()`
If `state` becomes “done”, then the application calls the `handleFinishTakePicture()` method to save `fileUrl` of the captured image.

5.5 Common Features

The 'apidemo' sample application provides the following common features:

Table 12 Common Features

Feature	Description
Connection	Connects the application with the camera
Info and State	Gets the camera information and operating state
Options	Gets and sets the camera options
Permission	Checks whether the application has the permission required
Result Dialog	Shows the result of API / command submitted in a dialog

5.5.1 Connection Management

Connect

By clicking the 'CONNECT' button, the sample application starts the `ConnectionActivity` class and checks whether the application is connected. The sample application starts device scanning by clicking the 'START SCAN' button. From the `checkBleScanPermissionAndStartScan()` method, the sample application checks required permissions, enables Wi-Fi and Bluetooth, requests the permissions if not granted and then starts device scanning by calling the `startScanDevice()` method which calls the connectionManager's `StartScanFriends()` method to scan for Friends devices. Scanned devices are listed below the 'START SCAN' button. The following figure shows an example of device scan result:

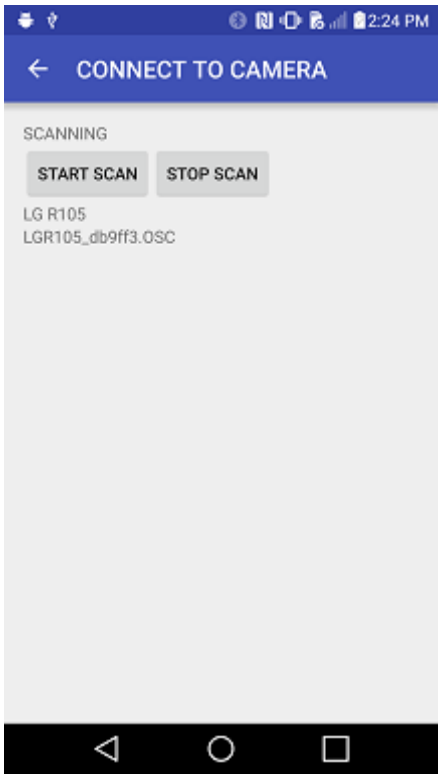


Figure 7 Device Scanning Result

By clicking the 'STOP SCAN' button, the sample application stops scanning devices by calling the `stopScanDevice()` method which calls the connectionManager's `StopScanFriends()` method.

If any device name in the scanned list is selected, the sample application starts Wi-Fi connection with the chosen device by calling the ConnectionActivity's `connect()` method which calls the connectionManager's `connect()` method. If it's the first time connecting to the selected camera device, the following pop-up is presented to ask for Wi-Fi password. The sample application saves the password information and uses this information in later connection attempts without challenging for Wi-Fi password.

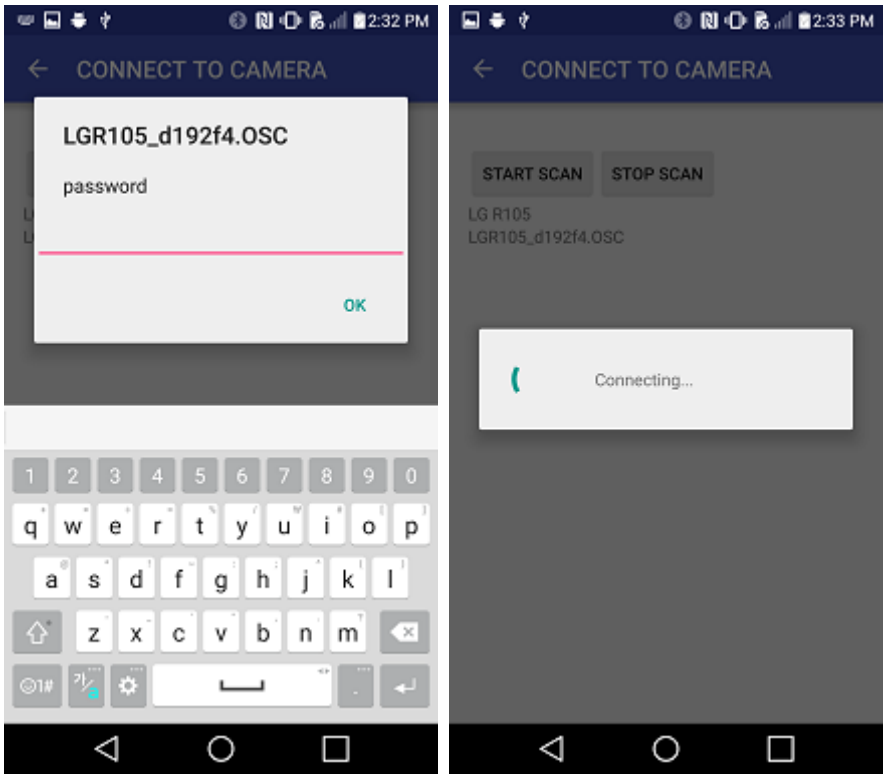


Figure 8 Connecting to a Device

If the connection is successfully established, the application returns to the main UI and all the buttons get enabled.

Disconnect

The `WifiReceiver` class is responsible for managing Wi-Fi connection between the application and the camera. If the sample application is connected with any camera, the text of 'CONNECT' button is set to "DISCONNECT". By clicking the button in this state, the sample application disconnects the current Wi-Fi connection with the camera. The sample application calls the Android System's API `disconnect()` to disconnect with the camera. The sample application returns to the Main screen when disconnected.

If any download job is in progress when the connection is disconnected, the download job needs to be cancelled. The `deleteUnfinishedDownloadFile()` method of `CameraFileListViewActivity` deletes the file being downloaded.

Connection Status Info

The `updateStateBasedOnWifiConnection()` method updates the texts of 'Connection State' and 'CONNECT' button based on the Wi-Fi connection state.

5.5.2 Camera Information and State

Camera Information

Basic properties of the camera can be obtained by the **/osc/info** API. The `OSCInfo` class is responsible for sending HTTP GET request of **/osc/info** and handling the response message from the camera. Refer to the OSC Library section for how the OSC Library manages OSC API request and response messages.

The sample application presents a dialog with the information obtained from the response message.

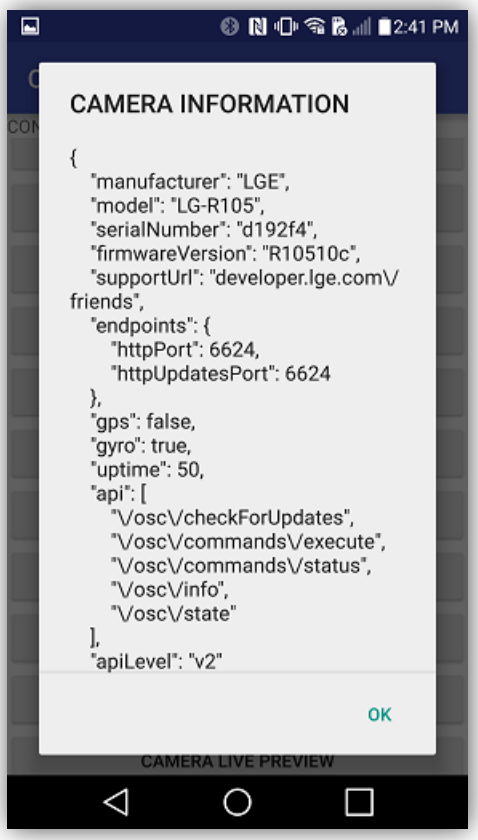


Figure 9 Camera Information Dialog

Camera State

Non-static properties of the camera can be obtained by the **/osc/state** API. The `OSCState` class is responsible for sending HTTP POST request of **/osc/state** and handling the response message from the camera. Refer to the OSC Library section for how the OSC Library manages OSC API request and response messages.

The sample application presents a dialog with the information obtained from the response message.

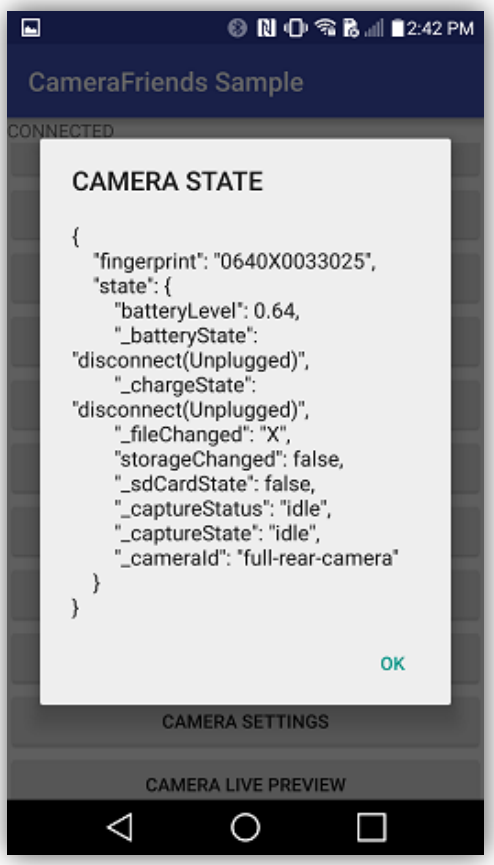


Figure 10 Camera State Dialog

'CHECK FOR UPDATE' checks whether the camera state has been changed or not. By clicking the button, the sample application sends the **/osc/checkForUpdate** API and compares the returned `fingerprint` with old value to notify the user whether the state has been changed or not. The `fingerprint` value represents the current state of the camera. See the API description on **/osc/checkForUpdate** for information on fingerprint algorithm.

5.5.3 Camera Options

Device Options

Device options can be get or set from the `OptionsActivity` class. The following figure shows the layout of the 'CAMERA OPTIONS'.

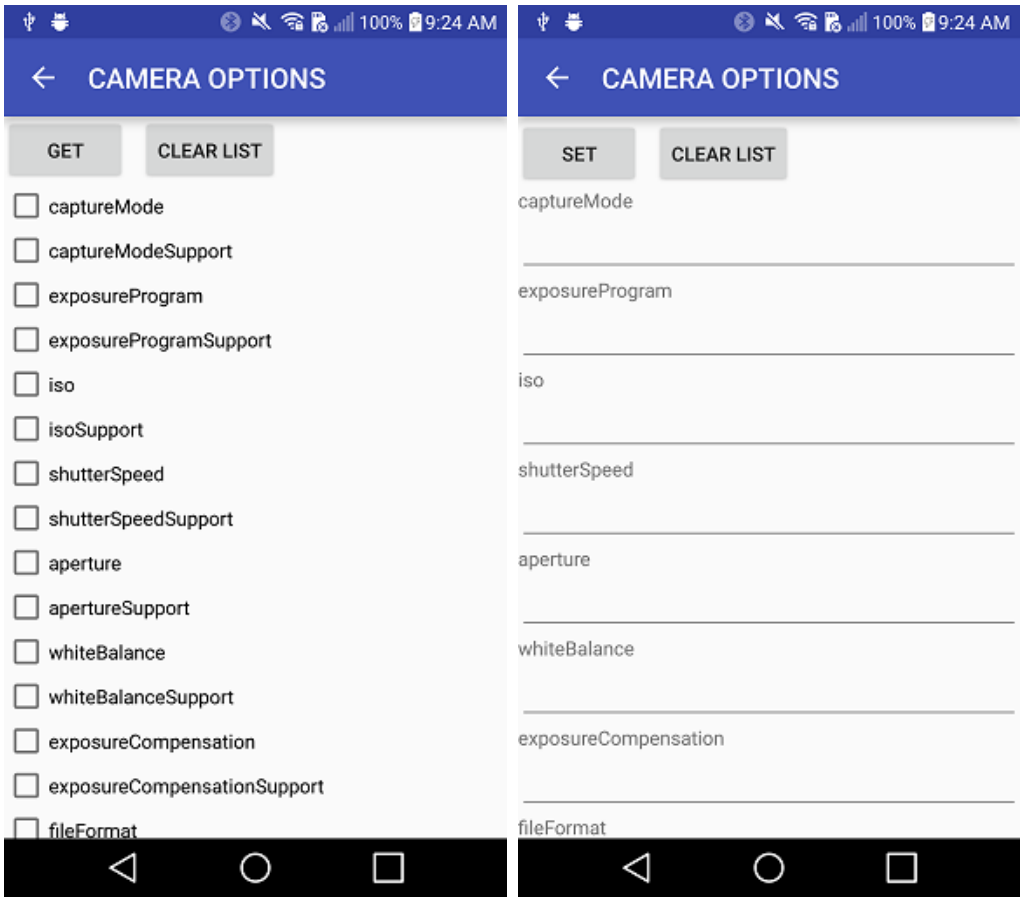


Figure 11 Device Options Layout

Select an option by checking the box and click the 'GET' button. Then the result will be shown in a Dialog window. To set options, select an option name and set the value for the selected option. Click the 'SET' button and the option value is set to the camera. Users can request to get or set multiple options. The sample application lists the selected options in the `options` input parameter and sends **command.getOptions** or **command.setOptions** command to get or set the options.

Setting options

Setting options can be get or set from the `SettingsActivity` class. The following figure shows the layout of the 'CAMERA SETTINGS'.

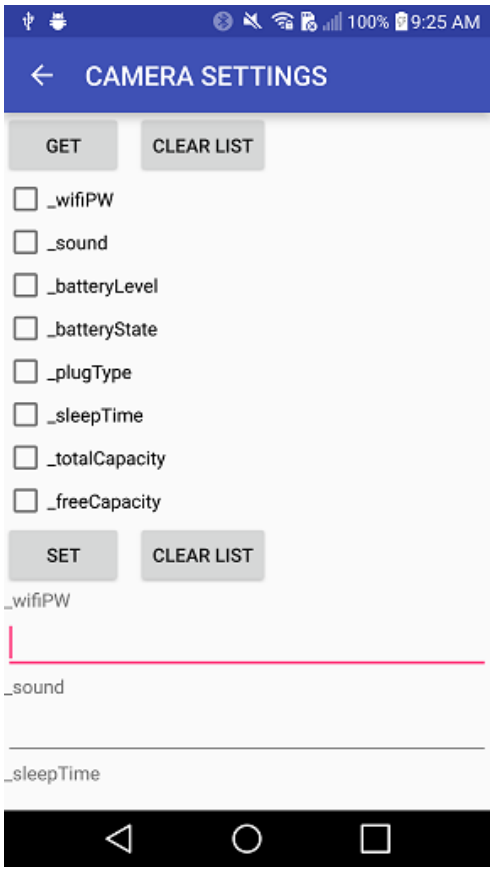


Figure 12 Setting Options Layout

Select an option by checking the box and click the 'GET' button. Then the result will be shown in a Dialog window. To set options, select an option name and set the value for the selected option. Click the 'SET' button and the option value is set to the camera. Users can request to get or set multiple options. The sample application lists the selected options in the `parameters` String Array and sends **/settings/get** or **/settings/set** API to get or set the options.

5.5.4 Permission

At the startup of the `MainActivity`, the sample application checks whether it has the permission of WRITE_EXTERNAL_STORAGE. The `checkFileWritePermission()` method handles the permission check. If the application does not have the permission, then it requests the permission to be granted. Refer to Permission Management for more information.

5.5.5 Result Dialog

Upon receiving a response message from the camera after submitting an API or command, the sample application shows a dialog that presents the output of the command. The text in the dialog contains the raw JSON data of output so that developers can have better idea of how the camera has responded. The following figure shows an example of result dialog.

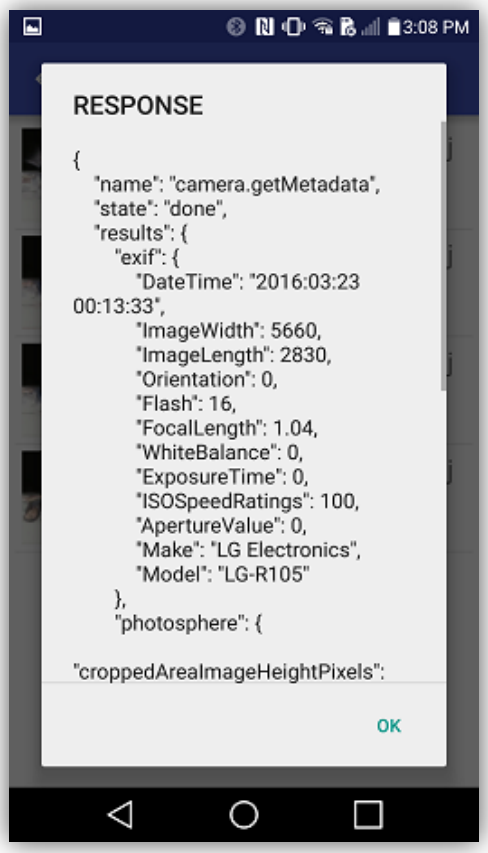


Figure 13 Result Dialog Example

5.6 Gallery Features

Images taken with the camera are stored in the storage of the camera. Using OSC commands APIs, those files can be downloaded to and viewed on the application side.

The sample application provides the following gallery features:

Table 13 Gallery Features

Feature	Description
Image List	Gets the list of image files in the camera
Thumbnail List	Gets the list of thumbnails of image files in the camera
Image Info	Gets the information of the target image file
Download	Downloads the target image file from the camera to the application
Delete	Deletes the target image file in the camera
View	Views images with the VR viewer
Shoot	Takes an image capture remotely from the application
Preview	Shows preview as a video stream

5.6.1 Working with Image Files in the Camera

Image List

The image files in the storage of the camera can be browsed from the application. The `CameraFileListViewActivity` class uses the `CustomListAdapter` class to save information of the images.

The list of images in the camera is obtained by executing **camera.listFiles** command. The `CameraFileListViewActivity` class implements `getListFiles()` to execute this action. Note that the method is called with `mediaType` set to "`image`".

If the camera cannot return the complete list, the `continuationToken` field is set in the response. Then the client should send **camera.listFiles** command again to obtain the remaining list of images. Repeat this command until the last item is obtained.

Thumbnail List

The sample application shows the thumbnail images as a list on the screen. The `getView()` method gets a thumbnail view for each image item in the list by using **camera.getFile** command. It ensures to display multiple thumbnail images in a seamless manner. The sample application assigns 60 to `maxSize`.

Image Information

By clicking on one of the thumbnail images listed on the screen, the sample application shows the following list dialog:

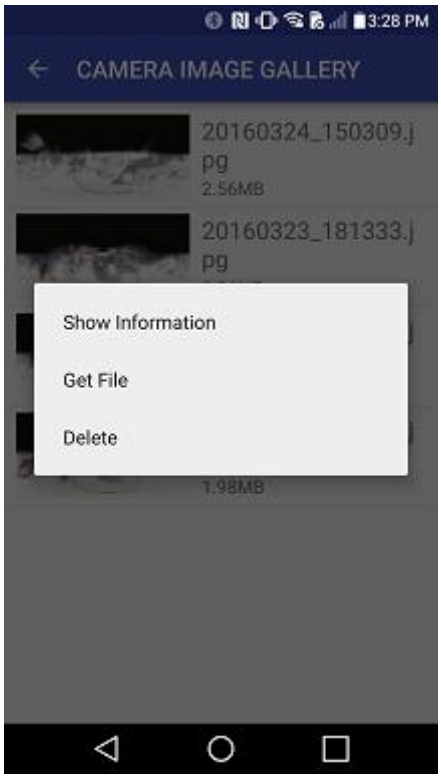


Figure 14 List Dialog for an Image Stored in the Camera

'Show Information' shows a dialog that displays information of the image selected. This information is obtained by **camera.getMetadata** command. The `CameraFileListViewActivity` class implements the `getFileMetadata()` method to execute this action.

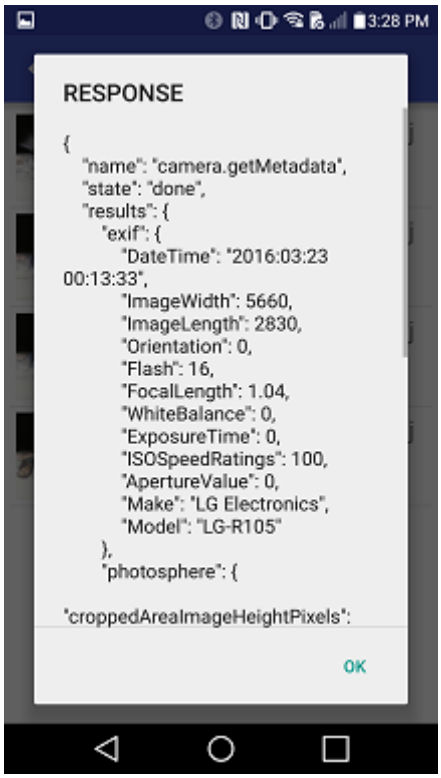


Figure 15 Image Information Dialog

Download Image

'Get File' downloads the selected image from the camera to the smartphone that runs the sample application. The `CameraFileListViewActivity` class implements the `getFullFile()` method to download the file by using **camera.getFile** command. Make sure to update the HTTP header to include "`Accept: image/jpeg`".

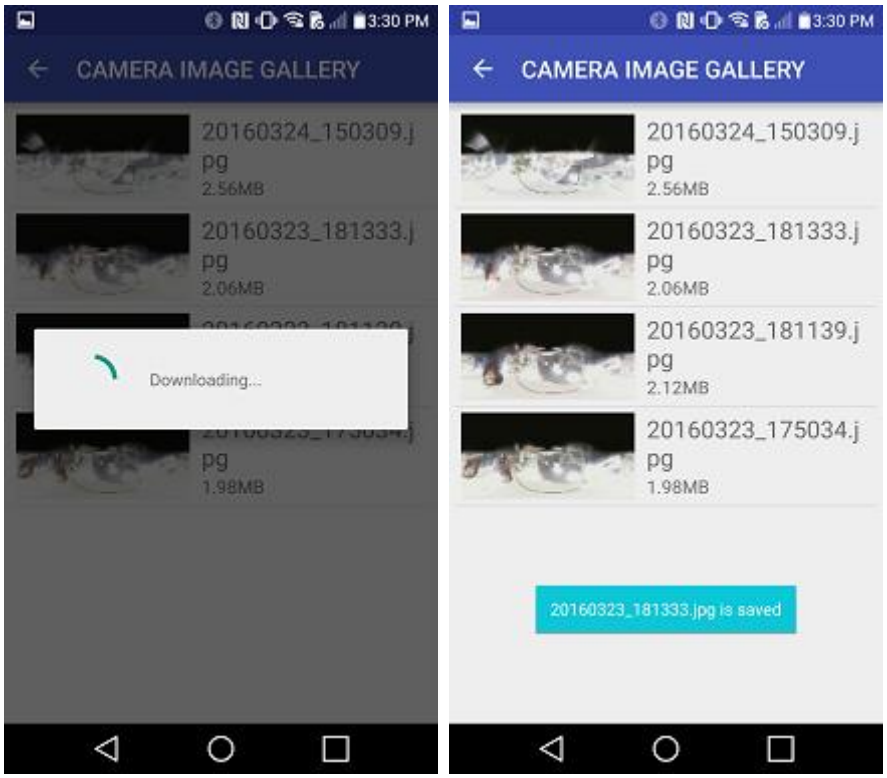


Figure 16 Downloading an Image from the Camera

Delete Image

'Delete' deletes the image from the camera. The `CameraFileListViewActivity` class implements the `deleteFilesFromCamera()` method to delete the file by using **camera.delete** command. The `deleteFilesFromCamera()` method also updates the list object to remove the item from the list.

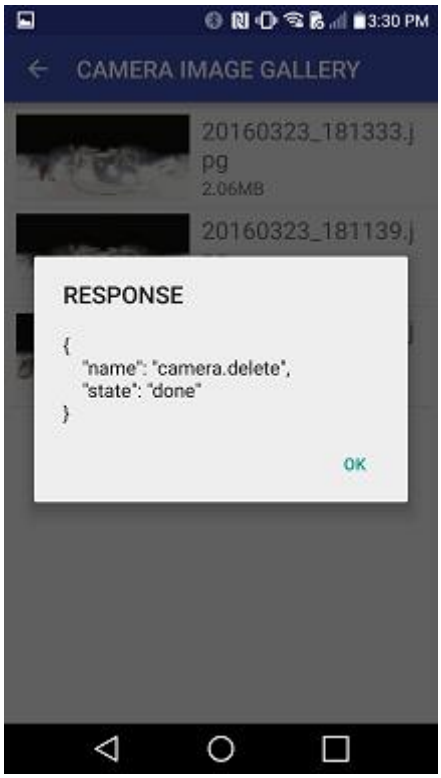


Figure 17 Deleting an Image from the Camera

Capture Image

By clicking the 'CAMERA TAKE PICTURE' button, the sample application starts `TakePictureActivity`. If the 'TAKE PICTURE' button is clicked, the sample application sends **camera.takePicture** command. The captured image is saved in the camera.

Figure 18 TakePictureActivity layout

By clicking the 'MANUAL METADATA' button, the sample application sends **camera._manualMetadata** command and shows its result as shown below. Note that `exposureProgram` option must be set to 1(manual) prior to command submission; otherwise the camera returns an error indicating that `exposureProgram` is not set to manual.

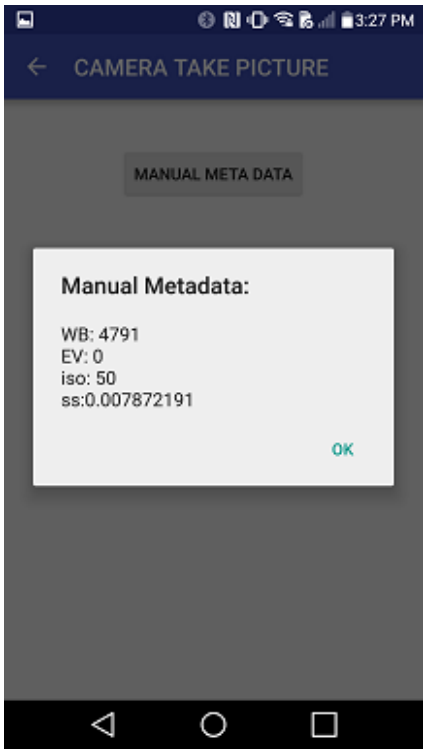


Figure 19 Manual Metadata

Preview

The previewAcitivity class is started by clicking the 'START PREVIEW' button, as show below:

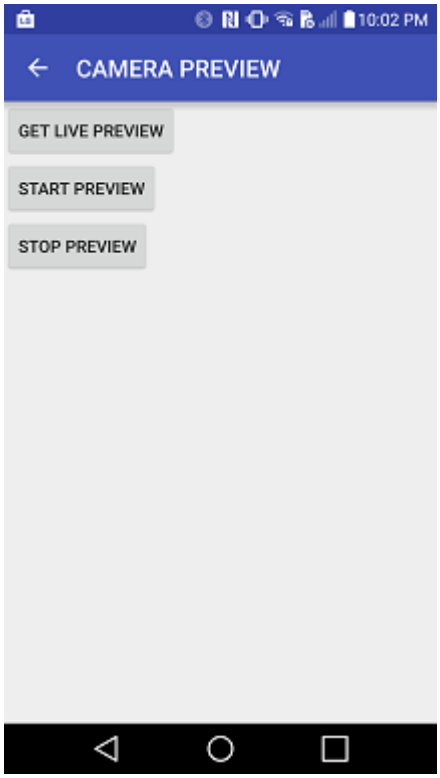


Figure 20 PreviewActivity UI

The 'START PREVIEW' button starts previewing by sending the **camera._startPreview** command to the camera. The 'STOP PREVIEW' button stops the previewing by sending the **camera._stopPreview** command to the camera. Note that the response to **camera._startPreview** contains URI information for preview in _previewUri. Use this URI to get the preview stream (UDP) from the camera. UDP player is required to play this preview stream.

Figure 21 START PREVIEW and STOP PREVIEW Responses

The `LivePreviewActivity` class is responsible for building preview stream. By using the `getLivePreview()` method, **camera.getLivePreview** command is repeatedly submitted as long as the previously submitted command returns no errors. This class uses `keepSendRequest` as a control variable whether to continue or stop recursive command submission. When the activity resumes (from `onResume()` method), the activity finishes if the application is not connected with any camera device.

The following figure shows an example of preview stream watched from the mobile device.

Figure 22 Preview Stream from the Camera

Interval Capture

The application can control the camera to execute interval capture action. The `CaptureIntervalActivity` class is responsible for managing the interval capture feature.

By clicking the 'CAMERA CAPTURE INTERVAL' button, the sample application starts `CaptureIntervalActivity`. The following figure shows the UI layout of `CaptureIntervalActivity`. When entering this Activity, the sample application presents a dialog where the user can easily change `captureMode` option to "interval" so that the user can readily use the interval capture features.

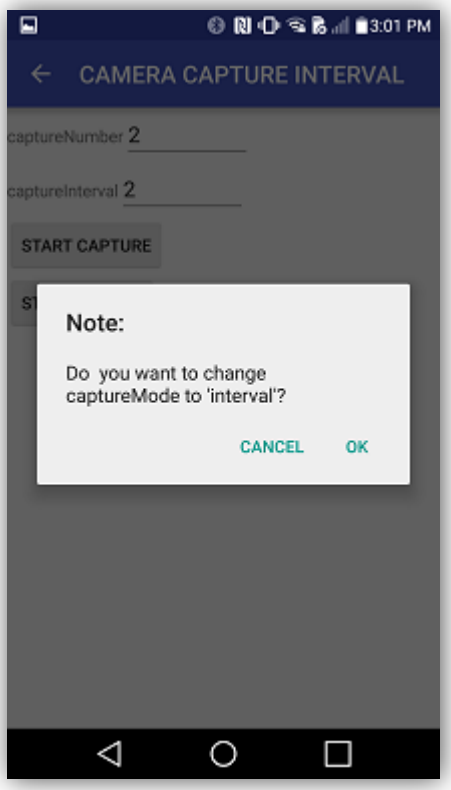


Figure 23 CaptureIntervalActivity UI

Set `captureNumber` and `captureInterval` options and click 'START CAPTURE' (**camera.startCapture**) to execute a non-open-ended interval capture. If `captureNumber` is set to 0, the sample application executes an open-ended capture.

Click 'STOP CAPTURE' (**camera.stopCapture**) to finish interval capture. Non-open-ended capture stops automatically when the camera completes the specified capture action. It can still be terminated earlier by sending **camera.stopCapture** command.

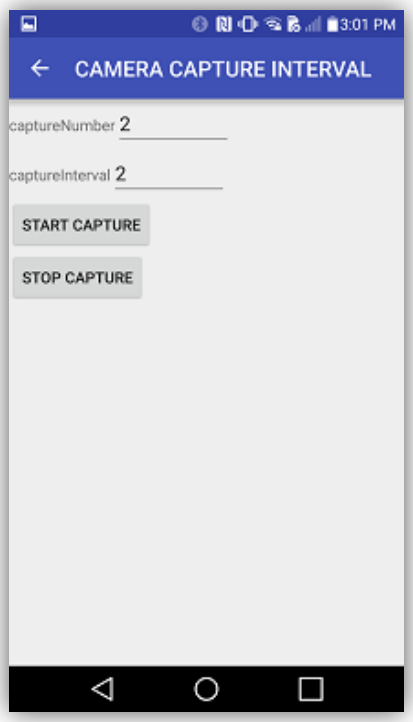


Figure 24 UI Layout of Interval Capture

Note that the camera does not return `fileUrls` response immediately; make sure to check the command status with the **/osc/commands/status** API. Although the command status returns "done", the camera might have not finished saving captured images and listing them in the fileUrls field. The sample application waits for `captureNumber * captureInterval` seconds after submitting **camera.startCapture** command to get the full list of fileUrls. For this reason, startCapture() method of `CaptureIntervalActivity` uses `postDelayed()` with a handler when executing interval capture action.

Note

It is a good design practice to make the application wait for a certain amount of time (`captureNumber * captureInterval`) after submitting **camera.startCapture** command so that the camera can respond with the full list of captured images.

5.6.2 Working with Image Files downloaded from the Camera

View

Images downloaded from the camera are stored in a subfolder under the mobile device's camera storage directory. By clicking on the 'DOWNLOAD IMAGE GALLERY' button, downloaded images are listed on the screen. The `DownloadImageListViewActivity` class is responsible for managing those downloaded image files.

By clicking on one of the images in the list, the sample application shows a dialog to select a viewer to show the image with. Currently, the Viewer (ViewerActivity in com.lge.vr) is the only available option in the sample application. Developers can implement or adopt their own viewers onto their applications.

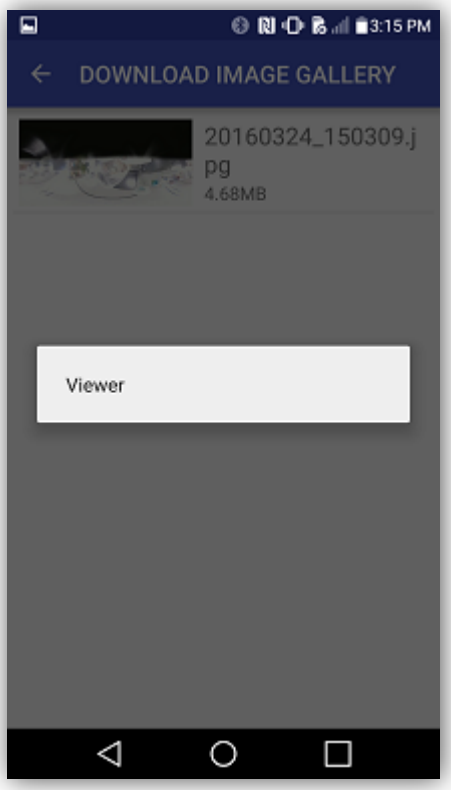


Figure 25 Viewer List Dialog

The sample application uses the VR viewer of the Google Cardboard SDK for Android. The client libraries of the viewer are imported in the 'libs' directory of the project. With the VR viewer, users have an option to see the image in Cardboard View mode (rendered as binocular image) or in VR View mode. Note that the image is drawn on a full-sphere canvas within the viewer.

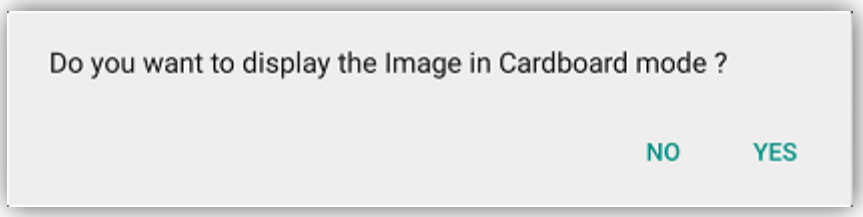


Figure 26 Viewer Mode Option Dialog

By clicking "YES" (Cardboard View mode), the following binocular image view is presented.

Figure 27 Cardboard View Mode

By clicking “NO” (VR View mode), the following VR image view is presented. Note that if the image is viewed with VR View mode, the scene displayed on the screen (field of view) is determined according the direction of the user’s view.

Figure 28 VR View Mode

5.7 Video Features

The sample application provides the following video features:

Table 14 Video Features

Feature	Description
Video List	Gets the list of video files in the camera
Thumbnail List	Gets the list of thumbnails of video files in the camera
Video Info	Gets the information of the target video file
Download	Downloads the target video file from the camera to the application
Delete	Deletes the target video file in the camera
View	Plays a video back with the VR viewer
Record	Records a video remotely from the application
Live Snapshot	Takes a snapshot image while the camera is recording a video

5.7.1 Working with Video Files in the Camera

Working with video files is similar to that of image files, since video and image use the same commands to work with the files.

Video List

The video files in the storage of the camera can be browsed from the application. The `CameraFileListViewActivity` class uses the `CustomListAdapter` class to save information of the videos.

The list of videos in the camera is obtained by executing **camera.listFiles** command. The `CameraFileListViewActivity` class implements `getListFiles()` to execute this action. Note that the method is called with `mediaType` set to "`video`".

If the camera cannot return the complete list, the `continuationToken` field is set in the response. Then the client should send **camera.listFiles** command again to obtain the remaining list of videos. Repeat this command until the last item is obtained.

Thumbnail List

The sample application shows the thumbnail images as a list on the screen. The `getView()` method gets a thumbnail view for each video item in the list by using **camera.getFile** command. It ensures to display multiple thumbnail images in a seamless manner. The sample application assigns `maxSize` to be 60.

Video Information

By clicking on the thumbnail image of a video file, the following list dialog is presented. Select 'Show Information' to call the `getFileMetadata()` method and show the video information.

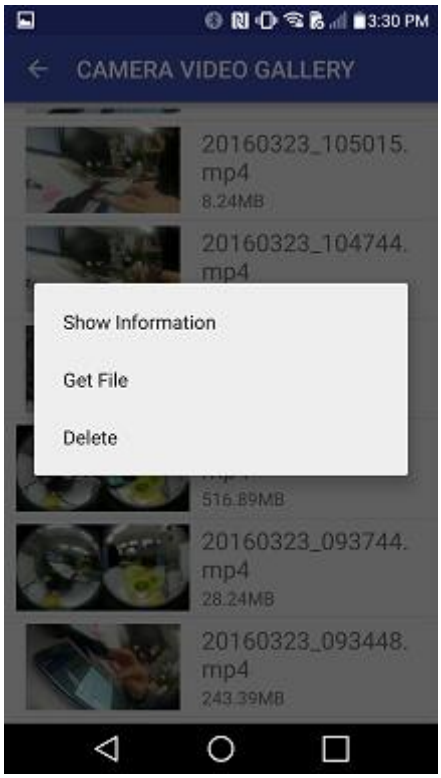


Figure 29 List Dialog for a Video Stored in the Camera

'Show Information' shows a dialog that displays information of the video selected. This information is obtained by **camera.getMetadata** command. The `CameraFileListViewActivity` class implements the `getFileMetadata()` method to execute this action.

Figure 30 Video Information Dialog

Download Video

Select 'Get Video' from the list dialog to download the video. **camera.getFile** command from the `getFullFile()` method obtains the specified video file from the camera. Make sure to update the HTTP header to include "`Accept: video/mp4`".

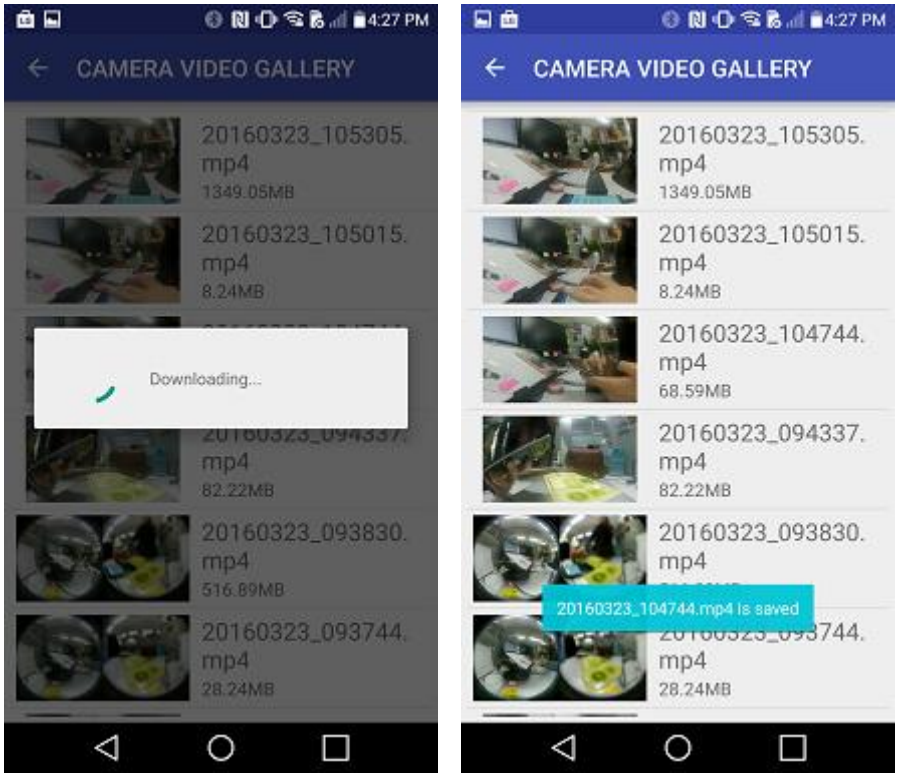


Figure 31 Downloading a Video from the Camera

Delete Video

Select 'Delete' from the list dialog to delete the video. **camera.delete** command from the `deleteFilesFromCamera()` method deletes the specified video file from the camera. The `deleteFilesFromCamera()` method also updates the list object to remove the item from the list.

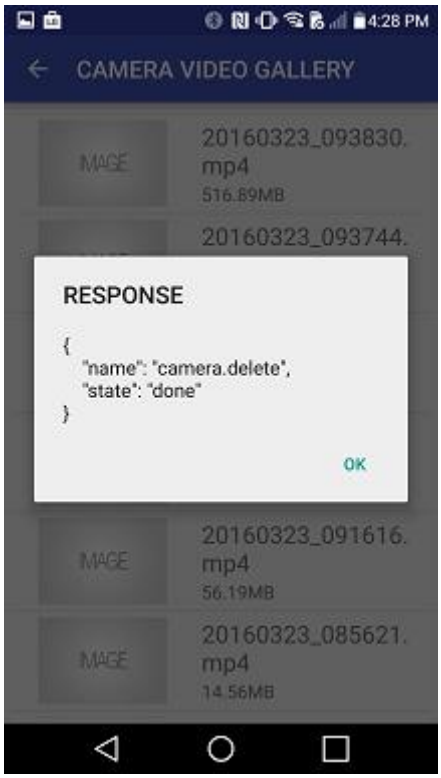


Figure 32 Deleting a Video from the Camera

Record Video

The sample application can control the camera remotely to start and stop recording a video.

If the 'CAMERA RECORD VIDEO' button is clicked, the sample application starts the `RecordVideoActivity` which has 'START RECORDING' and 'STOP RECORDING' buttons. By using these buttons the sample application manages video recording behavior based on the recording status.

Video Recording behavior can be modeled by the following state diagram.

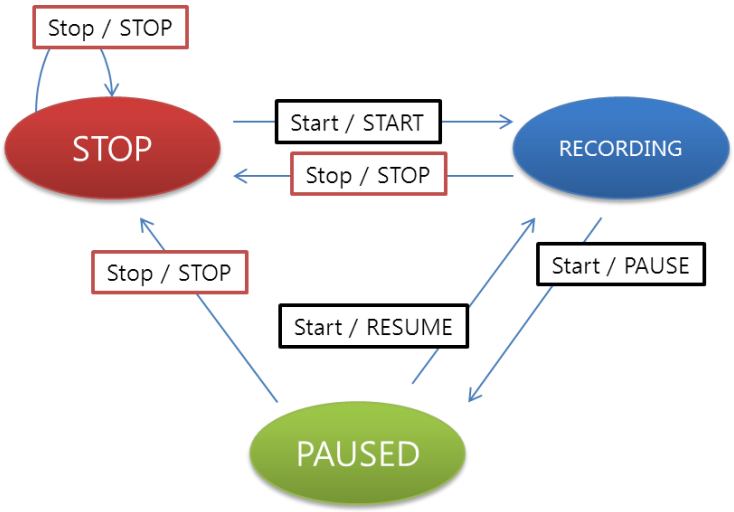


Figure 33 State Diagram for Video Recording Operation

Table 15 State Machine Elements

Element	Name	Description
State	STOP	Recording is stopped (initial state)
	RECORDING	Recording is in progress
	PAUSED	Recoding is paused
Event(Input)	Start	'**Start Recording**' button is clicked
	Stop	'**Stop Recording**' button is clicked
Action(Output)	START	Send **camera.startCapture** command. Toggle '**Start Recording**' button text to "Pause Recording".
	PAUSE	Send **camera._pauseRecording** command. Toggle '**Start Recording**' button text to "Resume Recording".
	RESUME	Send **camera._resumeRecording** command. Toggle '**Start Recording**' button text to "Pause Recording".
	STOP	Send **camera.stopCapture** command. Toggle '**Start Recording**' button text to "Start Recording".

Table 16 State Transition Table

Current State	Event(Input)	Next State	Action(Output)
STOP	Start	RECORDING	START
	Stop	STOP	STOP
RECORDING	Start	PAUSED	PAUSE
	Stop	STOP	STOP
PAUSED	Start	RECORDING	RESUME
	Stop	STOP	STOP

The following figure shows the UI layout of `RecordVideoActivity`. When entering this Activity, the sample application presents a dialog where the user is able to easily change `captureMode` option to "video" so that the user can readily use the video recording features.

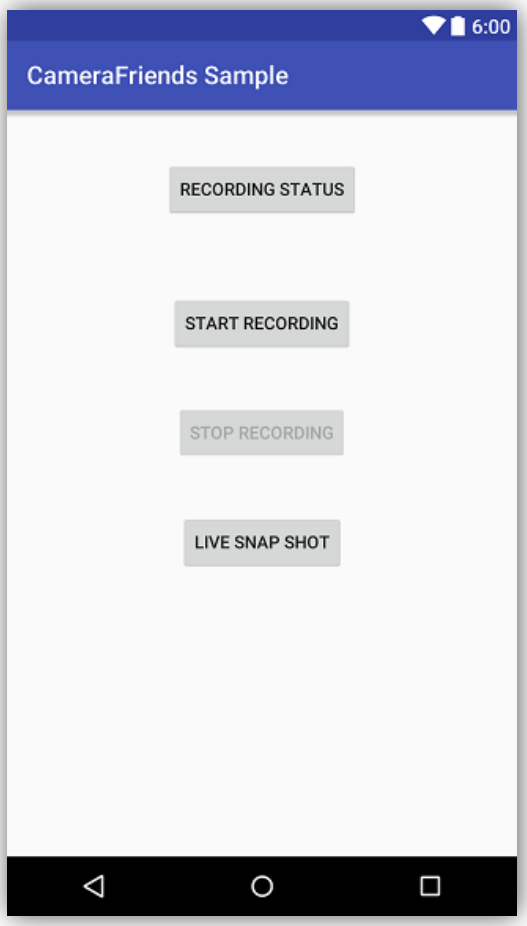


Figure 34 RecordVideoActivity UI

Recording Status

By clicking the 'RECORDING STATUS' button, the sample application checks the recording status of the camera by using **camera._getRecordingStatus** command (from `getRecordingStatus()` method). If the camera is not in recording state, **camera._getRecordingStatus** command returns an error message that indicates the current state of the camera.

Live Snapshot

By clicking the 'LIVE SNAP SHOT' button, the sample application takes a snapshot image while the camera is in recording status by using **camera._liveSnapshot** command (from `liveSnapShot()` method). Note that **camera._liveSnapshot** command can be used only when the camera is in 180° modes; _cameraId option needs to be set to either "half-front-camera" or "half-rear-camera".

5.7.2 Working with Video Files in the Camera

View

Download a video from the camera and play it back with a player.
The sample application registers the `setOnItemClickListener` callback to present the following dialog:

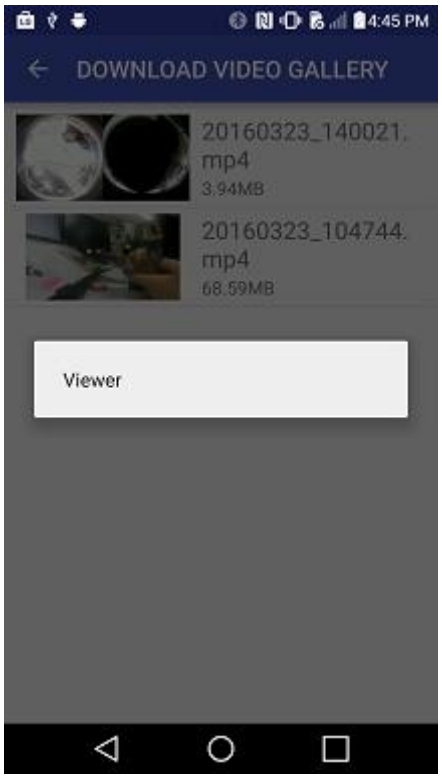


Figure 35 Viewer List Dialog

Select 'Viewer' to start the VR viewer. The `startViewer()` method is called and `ACTION_VIEW` intent is used to launch the `ViewerActivity` of the VR viewer. The video is played back with the VR viewer. Viewing videos are similar to viewing images with the VR Viewer; videos can be played in Cardboard View mode or in VR View mode. See Working with Image Files downloaded from the Camera for more information.

6 Tips

This chapter discusses useful tips for application development.

Contents

6.1 Permission Management
This section discusses how to manage permissions.

6.1 Permission Management

6.1.1 Permissions at run time

One of the major changes in Android 6.0 (API Level 23) is permission management. Previously (Android 5.1 and lower) permissions are granted at install time by declaring permissions in the manifest file. Beginning in Android 6.0, users can grant permissions at run time and choose to grant or revoke permissions individually.

To request a permission,

1) Check whether the application has the permission.
   Call `ContextCompat.checkSelfPermission()` method to determine whether the permission has been granted or not. The method returns PERMISSION_GRANTED if granted or PERMISSION_DENIED otherwise.

2) Request the permission if necessary.
   Call `requestPermissions()` to request the permission if the permission has not been granted (PERMISSION_DENIED returned from step 1).

3) Handle the request result callback.
   The `onRequestPermissionsResult()` callback is invoked when the user responds to the permission dialog box. Override this callback and perform appropriate actions according to the result passed to the callback.

Consider presenting the user with an explanation message before calling `requestPermissions()`. By calling this method, the Android System presents a permission dialog box to the user and does not allow this dialog to be modified. Thus the explanation has to be presented beforehand.
In other case, the user might keep revoking permission request that has to be granted in order to use necessary functionality. Then showing an explanation would be a reasonable approach to help the user understand why such permission is required.

7 Release Notes

Release Notes

Friends Camera SDK for Android Revision 1.0

Release date: 2016-04-01

Initial Release

Main Features

- Friends Camera SDK

Developers can create an application that can connect to an LG 360 CAM device and control the camera with the Friends Camera SDK. The Friends Camera SDK package includes:

- FriendsLink Libraries
- Developer Guide (API Reference included)
- Sample Application

- Friends Camera API

Friends Camera API consists of the following Google's OSC API and LG's vendor-specific implementations:

- OSC Protocol APIs: The Google Open Spherical Camera Protocol (OSC Protocol) APIs
- OSC Commands APIs: The Google Open Spherical Camera API (OSC API) specification with the extension of LG's vendor-specific commands (denoted by _ underscore prefix)
- Setting APIs: LG's vendor-specific APIs to set or get setting options

- connectionManager API

APIs for BLE and Wi-Fi connection with Friends devices

- Sample App

The sample application consists of the following modules:

- 'osclibrary': an OSC API handler module
- 'helloFriendsCamera': a hello-world application that demonstrates how to use OSC APIs
- 'apidemo': a sample application that implements the fundamental camera control features

Known Issues

The followings are the known issues of this release:

- Remove the Wi-Fi configuration with the target Friends device previously established using any other app, otherwise connection request through FriendsLink libraries would fail.
- The use of sessionId parameter of camera._startPreview and camera._stopPreview commands will be deprecated in the future release.
- The /settings/set API returns no errors even if an invalid value is set to _sound or _sleepTime.
- The camera._getRecordingStatus command responses an error, but with HTTP Status code 200 OK, when the command is used while the camera is not in recording state.

- The gyro option value does not change to “false”, even though setting gyro to “false” with camera.setOption returns {“state”:”done”}.
- The camera.startCapture command does not work if the captureMode option is set to “image”.
- The camera._stopPreview command returns {“state”:”done”} even if camera._startPreview is not called prior to camera._stopPreview.

8 Attribution

- Use of Google Developers Site Content

Friends Camera API is in compliance with the implementation guidance of the Open Spherical Camera API of Google. API descriptions and code examples, excluding vendor-specific extensions, are reproduced from the content of the Google Developers site of the Open Spherical Camera API. API descriptions and code examples on the Google Developers site of the Open Spherical Camera API are licensed under the Creative Commons Attribution 3.0 License and the Apache 2.0 License, respectively. See the below for License Attribution.

- License Attribution

"Open Spherical Camera API" (https://developers.google.com/streetview/open-spherical-camera) by Google Developers (https://developers.google.com) is licensed under CC BY 3.0 (http://creativecommons.org/licenses/by/3.0/)

- Use of Brands

Android is a trademark of Google Inc.
All Android based and Google based marks are trademarks or registered trademarks of Google Inc.

Wi-Fi® is a registered trademark of Wi-Fi Alliance.

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. - See more at: https://www.bluetooth.com/marketing-branding/brand-best-practices-guidelines#sthash.9lwPIahG.dpuf