**Flow SDK**

**Camera SDK Function**

**Initialization**

First add the maven address to your build file ( file in the project root directory)build.gradle

|  |
| --- |
| Java allprojects {  repositories {  ...  maven {  url 'http://nexus.arashivision.com:9999/repository/maven-public/'  credentials {  // see sdk demo  username = '\*\*\*'  password = '\*\*\*'  }  }  } } |

Second import the dependent library in your file of app directorybuild.gradle

|  |
| --- |
| Java dependencies {  implementation 'com.arashivision.sdk:sdkcamera:1.6.6\_flow\_2' } |

Then initialize FlowSDK in Application

|  |
| --- |
| Java public class MyApp extends Application {   @Overridepublic void onCreate() {  super.onCreate();   // Init SDKInstaCameraSDK.*initFlow*(this);  } } |

**Scan / Connect / Disconnect / Obersve**

**scan**

First, you need to register an IScanBleListener to listen for the results of the Bluetooth scan.

|  |
| --- |
| Java FlowCameraManager.*getInstance*().setScanBleListener(new IScanBleListener() {  @Override  public void onScanStartSuccess() {  Log.*d*("fmg", "onScanStartSuccess");  }   @Override  public void onScanStartFail() {  Log.*d*("fmg", "onScanStartFail");  }   @Override  public void onScanning(BleDevice bleDevice) {  Log.*d*("fmg", "onScanning: " + bleDevice.getName());  }   @Override  public void onScanFinish(List<BleDevice> bleDeviceList) {  Log.*d*("fmg", "onScanFinish" + bleDeviceList.size());  } }); |

After that, you can start scanning.

|  |
| --- |
| Java FlowCameraManager.*getInstance*().startBleScan(30\_000); |

**stopScan**

|  |
| --- |
| Java FlowCameraManager.*getInstance*().stopBleScan(); |

**Connect**

|  |
| --- |
| Java InstaCameraManager.getInstance().connectBle(bleDevice); |

**Disconnect**

|  |
| --- |
| Java FlowCameraManager.*getInstance*().disconnectBle(); |

**Obersve**

You can on multiple pages to observe fmg status changedregister / unregister IFmgStatusListener

|  |
| --- |
| Java public class FlowActivity extends AppCompatActivity {  @Override  protected void onCreate(@Nullable Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  FlowCameraManager.*getInstance*().addFmgStatusListenerList(fmgStatusListener);  }   @Override  protected void onDestroy() {  super.onDestroy();  FlowCameraManager.*getInstance*().removeFmgStatusListenerList(fmgStatusListener);  }    private IFmgStatusListener fmgStatusListener = new IFmgStatusListener() {  @Override  public void onFmgConnectSuccess() {  Log.*d*("fmg", "onFmgConnectSuccess");  }    @Override  public void onFmgDisconnect() {  Log.*d*("fmg", "onFmgDisconnect");  }    @Override  public void onFmgConnectError(int errorCode) {  Log.*d*("fmg", "onFmgConnectError: " + errorCode);  }   @Override  public void onFmgBatteryUpdate(int battery) {  Log.*d*("fmg", "onCameraBatteryUpdate: " + battery);  }   @Override  public void onFmgModeChanged(FmgModel.PtzMode mode) {  Log.*d*("fmg", "onFmgModeChanged: " + mode.name());  }   @Override  public void onFmgLimitYawChanged(boolean isFmgLimitYaw) {  Log.*d*("fmg", "onFmgLimitYawChanged: " + isFmgLimitYaw);  }   @Override  public void onFmgLimitPitchChanged(boolean isFmgLimitPitch) {  Log.*d*("fmg", "onFmgLimitPitchChanged: " + isFmgLimitPitch);  }   @Override  public void onFmgStallChanged(boolean isFmgStalled) {  Log.*d*("fmg", "onFmgStallChanged: " + isFmgStalled);  }   @Override  public void onFmgPayloadChanged(boolean isFmgPayload) {  Log.*d*("fmg", "onFmgPayloadChanged: " + isFmgPayload);  }   @Override  public void onFmgImbalanceChanged(boolean isFmgImbalance) {  Log.*d*("fmg", "onFmgImbalanceChanged: " + isFmgImbalance);  }   @Override  public void onFmgLowTemperatureChanged(boolean isLowTemperature) {  Log.*d*("fmg", "onFmgLowTemperatureChanged: " + isLowTemperature);  }   @Override  public void onFmgHighTemperatureChanged(boolean isHighTemperature) {  Log.*d*("fmg", "onFmgHighTemperatureChanged: " + isHighTemperature);  }   @Override  public void onFmgSleepChanged(boolean isFmgSleep) {  Log.*d*("fmg", "onFmgSleepChanged: " + isFmgSleep);  }   @Override  public void onFmgHvModeChanged(FmgModel.PtzHvMode hvMode) {  Log.*d*("fmg", "onFmgHvModeChanged: " + hvMode.name());  } }; } |

If you need to directly obtain the value rather than listen, you can do so in the following way:

|  |
| --- |
| Java FlowCameraManager.*getInstance*().getBatteryLevel(); FlowCameraManager.*getInstance*().getFmgMode(); FlowCameraManager.*getInstance*().isFmgLimitYaw(); FlowCameraManager.*getInstance*().isFmgStalled(); FlowCameraManager.*getInstance*().isCharging(); FlowCameraManager.*getInstance*().isFmgPayload(); FlowCameraManager.*getInstance*().isHighTemperature(); FlowCameraManager.*getInstance*().isFmgImbalance(); FlowCameraManager.*getInstance*().isFmgSportMode(); FlowCameraManager.*getInstance*().isFmgSleep(); FlowCameraManager.*getInstance*().isLowTemperature(); FlowCameraManager.*getInstance*().isFmgLimitPitch(); FlowCameraManager.*getInstance*().getFmgHvMode(); |

PtzMode is an enumeration class with a total of 5 states. The first four correspond to the AUTO, F, PF, and FPV on the Flow dial. LOCK is the state triggered by long pressing the trigger button.

|  |
| --- |
| Java public static enum PtzMode {  *AUTO*,  *FOLLOW*,  *PITCH\_LOCK*,  *FPV*,  *LOCK*;  } |

**Button Event**

You can register/unregister ICameraController.IFmgButtonClickListener on multiple pages to observe FMG button events.

This operation must be performed after a successful connection.

|  |
| --- |
| Java private ICameraController.IFmgButtonClickListener buttonClickListener = new ICameraController.IFmgButtonClickListener() {  @Override  public void onModeButtonEvent(FmgModel.PtzButtonEvent buttonEvent) {  Log.*d*("fmg", "onModeButtonEvent: " + buttonEvent.name());  }   @Override  public void onShutterButtonEvent(FmgModel.PtzButtonEvent buttonEvent) {  Log.*d*("fmg", "onShutterButtonEvent: " + buttonEvent.name());  }   @Override  public void onHoldButtonEvent(FmgModel.PtzButtonEvent buttonEvent) {  Log.*d*("fmg", "onHoldButtonEvent: " + buttonEvent.name());  }   @Override  public void onMiddleButtonEvent(FmgModel.PtzButtonEvent buttonEvent) {  Log.*d*("fmg", "onMiddleButtonEvent: " + buttonEvent.name());  }   @Override  public void onTouchPanelEvent(FmgModel.PtzButtonEvent buttonEvent) {  Log.*d*("fmg", "onTouchPanelEvent: " + buttonEvent.name());  }   @Override  public void onDialWheelEvent(FmgModel.PtzButtonEvent buttonEvent) {  Log.*d*("fmg", "onDialWheelEvent: " + buttonEvent.name());  }   @Override  public void onDialWheelSrcValueEvent(int value) {  Log.*d*("fmg", "onDialWheelSrcValueEvent: " + value);  }   @Override  public void onPowerBtnEvent(FmgModel.PtzButtonEvent buttonEvent) {  Log.*d*("fmg", "onPowerBtnEvent: " + buttonEvent.name());  } };  private void addFmgButtonClickListener(ICameraController.IFmgButtonClickListener listener) {  FlowCameraManager.*getInstance*().addFmgButtonClickListener(listener); }  private void removeFmgButtonClickListener(ICameraController.IFmgButtonClickListener listener) {  FlowCameraManager.*getInstance*().removeFmgButtonClickListener(listener); } |

FmgModel.PtzButtonEvent is an enumeration class, implemented as follows：

|  |
| --- |
| Java public static enum PtzButtonEvent {  //\*\*\*\*\* ButtonEvent \*\*\*\*\*//  *SINGLE\_CLICK*(0),  *DOUBLE\_CLICK*(1),  *TRIPLE\_CLICK*(2),  *LONG*(3),  *LONG\_RELEASE*(4),  //\*\*\*\*\* MiddleButton \*\*\*\*\*//  *RC\_UP*(16),  *RC\_DOWN*(17),  *RC\_LEFT*(18),  *RC\_RIGHT*(19),  *RC\_IDLE*(20),  //\*\*\*\*\* TouchPanel \*\*\*\*\*//  *TOUCH\_CW*(32),  *TOUCH\_CCW*(33),  *TOUCH\_START*(34),  *TOUCH\_END*(35),  *TOUCH\_LEFT\_DOUBLE\_CLICK*(36),  *TOUCH\_RIGHT\_DOUBLE\_CLICK*(37),  //\*\*\*\*\* DialWheel\*\*\*\*\*//  *DW\_CW\_SINGLE*(48),  *DW\_CW\_START*(49),  *DW\_CW\_STOP*(50),  *DW\_CCW\_SINGLE*(51),  *DW\_CCW\_START*(52),  *DW\_CCW\_STOP*(53),  //\*\*\*\*\* DialWheelSrcValue \*\*\*\*\*//  *DW\_SRC\_VALUE*(54);   private int nativeValue;   private PtzButtonEvent(int nativeValue) {  this.nativeValue = nativeValue;  }   public int getNativeValue() {  return this.nativeValue;  } } |