FireflyApi Programming Interface

Overview

FireflyApi SDK, Containing IDCard (ID card), ICCard, QR code and other related functions, FireflyApi flexibly develops upper-layer software applications based on its related business needs and the relevant interfaces of the SDK.
Applicable scene:

- The device cannot be connected to the Internet of Things and is offline;
- The public security intranet, etc;
- The network speed is slow, to avoid taking up too much bandwidth and other environments;

Historical branch

Date	Version	Explanation	ModifiedBy
2020-12-21	1.2	Optimize IDCard function	XiaoDatao,SongJianfeng
2020-07-17	1.1	V2	XiaoDatao,SongJianfeng
2020-05-21	1.0	FirstEditionCompleted	SongJianfeng

Development environment

1. Android Studio >=3.6.2
2. Gradel Version: 3.6.2
3. Gradle Plugin Version: 3.6.2
4. SDK Tool >=25.2.3

SDK Integration instructions

1.Copy all the so packages in the fireflyApi/libs/armeabi-v7a directory of the Demo project to the libs/armeabi-v7a folder corresponding to the AS project;
2.Copy the fireflyapi.jar package in the fireflyApi/libs/ directory of the Demo project to the libs folder corresponding to the AS project;

For details, please refer to the technical case Demo project configuration and doc/FireflyApi_instructions.png ;

FireflyApi Interface Description

1.1 IDCard/ICCard

1. Overview

There are three main hardware configurations of current equipment:

Configuration	ICCard Swipe	IDCard Swipe
IDCard Module	YES	YES
NFC Module	YES	NO
Nothing	NO	NO

2. Connected devices

> Start the monitoring service and monitor the card swiping operation, it is recommended to execute in the on Resume() method;

```
/*
   Open the background monitoring service
*/
IDCardUtil.getInstance().bindIDCardService(Context context);

/*
   The NFC module is reported in the form of key value
*/
@Override
public boolean dispatchKeyEvent(KeyEvent event) {
     if (IDCardUtil.getInstance().handleEvent(event)) {return true;}

     return super.dispatchKeyEvent(event);
}
```

3. Check function support

> Since the monitoring service is started asynchronously, it is not necessarily accurate to perform function support detection immediately after connecting the device.

> It is recommended to check the function support again in the monitoring callback of 6 on Machine Connect

```
/*
  Check if the device supports ID card recognition
  boolean: true means support, false means not support
*/
boolean result =IDCardUtil.getInstance().isSupportIDCard();

/*
  Check if the device supports ICCard recognition
  boolean: true means support, false means not support
*/
boolean result = IDCardUtil.getInstance().isSupportICCard();
```

4. Set swipe mode

> Although the ID card module supports both ID card and IC Card, it does not support simultaneous reading of both, and the card reading mode needs to be set as required
>
> NFC module only supports ICCard

```
/*
 According to the specified reading method when swiping the card;
 There are two main types：READCARD_MODE_IDENTITY_CARD and READCARD_MODE_IC_CARD；

  IDCardConfig.READCARD_MODE_IDENTITY_CARD = 0;  //IDCard
  IDCardConfig.READCARD_MODE_IC_CARD = 1; //ICCard
  IDCardConfig.READCARD_MODE_IDENTITY_CARD_UUID = 2; //IDCard UUID
*/
IDCardUtil.getInstance().setModel(int readMode);
```

5. Set the endian mode for reading IC Card

> Set the endian mode when reading the IC Card, the default is big endian

```
/*
  boolean:true Bigendian, false Little endian
*/
IDCardUtil.getInstance().setICCardEndianMode(boolean useBig);
```

6. Set up listening and callback functions

```
/*
  Binding credit card monitoring callback
  context
  callback Listen callback
*/
IDCardUtil.getInstance().setIDCardCallBack(IDCardUtil.IDCardCallBack callBack);
```

```
//Listen callback
IDCardUtil.IDCardCallBack callBack = new IDCardUtil.IDCardCallBack() {

    //The monitoring service is started asynchronously, if a card reading device is found after startup, it will call back on Machine Connect
    @Override
    public void onMachineConnect() {
      Log.i("firefly", "onMachineConnect ");
    }

    //Callback when ID card is swiped
    @Override
    public void onSwipeIDCard(final IDCardBean info) {
      Log.i("firefly",
        "picture:"+           + "\n" +
        "name：" + info.getName() + "\n" +
        "sex：" + info.getSex() + "\n" +
        "nation：" + info.getNation() + "\n" +
        "birthDate：" + info.getBirthDateStr() + "\n" +
        "address：" + info.getAddress() + "\n" +
        "number：" + info.getNum() + "\n" +
        "issue：" + info.getIssue() + "\n" +
        "expiration date：" + info.getCreateTimeStr() + "-" + info.getValidTimeStr() + "\n" +
         "picture:"+           + info.getPhoto() + "\n" );
    }

    // Execute callback when swiping IC card
    @Override
    public void onSwipeICCard(final ICCardBean info) {
      Log.i("firefly", "onSwipeICCard IC=" + info.getIcID());
    }

    // When setting the card reading mode to READCARD_MODE_IDENTITY_CARD_UUID Time, swipe ID card callback
    @Override
    public void onSwipeIDCardUUID(final String uuid) {
      Log.i("firefly", "onSwipeIDCardUUID uuid=" + uuid);
    }
  };
```

7. Disconnect

> Remove the monitoring callback and stop the monitoring service, it is recommended to execute in the on Stop () method;

```
IDCardUtil.getInstance().setIDCardCallBack(null);
IDCardUtil.getInstance().unBindIDCardService(Context context);
```

1.2 Fill light control

There are 4 types of fill light: infrared fill light, white fill light, red fill light and green fill light.

- Check whether the device supports infrared fill light;

```
/*
   Boolean: true means support, false means not support
*/
HardwareCtrl.isSupportInfraredFillLight();
```

- Operation of infrared fill light;

```
/*
   isChecked true means open, false close
*/
HardwareCtrl.setInfraredFillLight(isChecked);
```

- Check whether the device supports the brightness adjustment of the white fill light, and obtain the range of brightness adjustment;

```
/*
   Boolean:true Indicates that infrared fill light is supported, false indicates that it is not supported
*/
HardwareCtrl.isFillLightBrightnessSupport();

/*
   Support the maximum value of brightness adjustment;
*/
HardwareCtrl.getFillLightBrightnessMax();

/*
   Support the minimum value of brightness adjustment;
*/
HardwareCtrl.getFillLightBrightnessMin();
```

- White fill light operation;

```
/*
   On/Off;
   isChecked true means open and set the maximum value; false close
*/
HardwareCtrl.ctrlLedWhite(isChecked);

/*
   On/Off, if brightness adjustment is supported, the corresponding brightness value can be passed in when the light is turned on;
   isChecked  true means open and set the maximum value; false close
*/
HardwareCtrl.ctrlLedWhite(isChecked,brightness);
```

- Red fill light operation;

```
/*
   isChecked true means open and set the maximum value; false close
*/
HardwareCtrl.ctrlLedRed(isChecked);
```

- Green fill light operation;

```
/*
   isChecked true means open and set the maximum value; false close
*/
HardwareCtrl.ctrlLedGreen(isChecked);
```

1.3 Signal control

- "Rs485/232" signal operation, of which 485 serial port: /dev/ttyS4; 232 serial port: /dev/ttyS3, you can view the set value changes through cat /dev/ttyS4 or cat /dev/ttyS3;

```
/*

   Get serial number serialPort
   485 serial port: /dev/ttyS4; 232 serial port: /dev/ttyS3
*/
SerialPort serialPort = HardwareCtrl.openSerialPortSignal(new File("/dev/ttyS3"), 19200, new SerialPort.Callback() {

      //rs485/232After sending the signal, the return value received
       @Override
       public void onDataReceived(byte[] bytes, int i) {
```

```
        String result = StringUtils.bytesToHexString(bytes, size);
        Log.i("firefly", "result = "+result);
      }
    });

/*
  Send ‘48562311’ signal
*/
HardwareCtrl.sendSerialPortHexMsg(serialPort, "48562311")

/*
  Close the serial port (it is recommended to close the serial port when the page exits)
*/
HardwareCtrl.closeSerialPortSignal(serialPort);
```

- Wiegand signal operation;

```
/*
  Wiegand Input
*/
HardwareCtrl.openRecvMiegandSignal("/dev/wiegand");

/*
  Add Wiegand Input listening callback
*/
HardwareCtrl.recvWiegandSignal(new RecvWiegandCallBack() {
  @Override
  public void recvWiegandMsg(int i) {
      Log.i("firefly", "result = "+i);
  }
});

/*
  When the page exits, close Wiegand
*/
HardwareCtrl.closeRecvMiegandSignal();

/*
  Wiegand 26 Output
  Check the set value changes through cat /sys/devices/platform/wiegand-gpio/wiegand26.
*/
HardwareCtrl.sendWiegandSignal("123456789");

/*
  Wiegand 34 Output
  Check the set value changes through cat /sys/devices/platform/wiegand-gpio/wiegand34.
*/
HardwareCtrl.sendWiegand34Signal("123456789");
```

- Level signal/relay signal;

```
/*
   D0 level signal isChecked true means open, false close
*/
LevelSignalUtil.sendSignalD0(isChecked);

/*
   D1 level signal isChecked true means open, false close
*/
LevelSignalUtil.sendSignalD1(isChecked);

/*
   Relay signal isChecked true means open, false closed
*/
RelayUtil.sendRelaySignal(isChecked);
```

1.4 QRCode

- QRCode Operation

```
/*
   Check whether the device supports the QR code function;
   Boolean: true means support, false means not support
*/
QrCodeUtil.getInstance().isQrCodeSupport();

/*
   Turn on QR code scanning
*/
QrCodeUtil.getInstance().init();

/*
   Whether to turn on the QR code scanning fill light
   QrCodeUtil.LED_STATE_AUTO // Automatic
   QrCodeUtil.LED_STATE_ON // ON
   QrCodeUtil.LED_STATE_OFF // OFF
*/
QrCodeUtil.getInstance().setLedState(state);

/*
   Add QR code monitoring callback
*/
QrCodeUtil.getInstance().setQRCodeCallback(new QrCodeUtil.QRCodeCallback() {
    // Callback method when QR code is connected
   @Override
   public void onConnect() {
      Log.i("firefly","QRCode onConnect:");
   }

    // Content callback method for QR code recognition
   @Override
   public void onData(final String s) {
```

```
        Log.i("firefly","QRCode onData:"+s);
    }
});

/*
   When the page exits, turn off QR code scanning
*/
QrCodeUtil.getInstance().release();
```

1.5 body temperature

- Body temperature operation;

```
/*
   Check whether the device supports body temperature;
   Boolean: true means support, false means not support
*/
TempatureUtil.getInstance().isSupport();

/*
   Turn on body temperature detection
*/
TempatureUtil.getInstance().openDevice();

/*
  Add temperature detection monitoring callback
*/
TempatureUtil.getInstance().setTempatureCallback(new TempatureUtil.TempatureCallback() {
    // Callback method when body temperature detection is connected
   @Override
   public void onConnect() {
       Log.i("firefly","TempatureCallback onConnect:");
   }

    // Content callback method for body temperature detection
   @Override
   public void update(float ambientTempature, float objectTempature) {
       Log.i("firefly","TempatureCallback update:ambientTempature="+ambientTempature +
"objectTempature="+objectTempature);
   }
});

/*
   When the page exits, turn off the body temperature function
*/
TempatureUtil.getInstance().closeDevice();
```

1.6 Radar

- Radar operation;

```
/*
   Handling radar events, KeyEvent event
   In the Android Activity page, public boolean dispatchKeyEvent(KeyEvent event), handle KeyEvent
event
*/
RadarUtil.handleEvent(event);

/*
   Dispatch Key Event on the Android Activity page
*/
@Override
   public boolean dispatchKeyEvent(KeyEvent event) {
      int ret = RadarUtil.handleEvent(event);
      if (ret == RadarUtil.EVENT_HANDLE_RADAR_IN) {// Object entry
         Log.i("firefly","EVENT_HANDLE_RADAR_IN");
         return true;
      } else if (ret == RadarUtil.EVENT_HANDLE_RADAR_OUT) { // Object leaving
         Log.i("firefly","EVENT_HANDLE_RADAR_OUT");
         return true;
      } else if (ret == RadarUtil.EVENT_HANDLE_NOTHING_HANDLED) { // No object
         Log.i("firefly","EVENT_HANDLE_NOTHING_HANDLED");
         return true;
      }

      return super.dispatchKeyEvent(event);
   }


// Monitor radar signals
// The radar signal is triggered in the form of KeyEvent and processed using handleEvent (KeyEvent
event)
public static final int RadarUtil.EVENT_HANDLE_NOTHING_UNHANDLE = -1; // Non-radar events are not
processed
public static final int RadarUtil.EVENT_HANDLE_NOTHING_HANDLED = 0; // Non-radar events processed

// Radar key-value pair
public static final int RadarUtil.KEYCODE_RADAR_IN = 305; // Something enters
public static final int RadarUtil.KEYCODE_RADAR_OUT = 306; // There is an object leaving

// Radar event
public static final int RadarUtil.EVENT_HANDLE_RADAR_IN = 1; // Something enters
public static final int RadarUtil.EVENT_HANDLE_RADAR_OUT = 2; // There is an object leaving
```