DKCloudID_USB_Android

How to integrate into the project

Step 1. Add the JitPack repository to your build file

Open the document of build.gradle, to add the maven { url 'https://jitpack.io' }

at the end of repositories

```
allprojects {
    repositories {
    ...
    maven { url 'https://jitpack.io' }
    }
}
```

Step 2. Add implementation

'com.gitee.lochy:dkcloudid-usb-android-module:v1.0.2' to dependency

```
dependencies {
       implementation 'com.gitee.lochy:dkcloudid-usb-android-module:v1.0.2'
}
```

Step 3. Add the network permissions and USB access permission into

AndroidManifest.xml

```
   <uses-feature android:name="android.hardware.usb.UsbDevice" />
   <uses-feature android:name="android.hardware.usb.UsbManager" />

   <uses-permission android:name="android.permission.INTERNET" />
```

Step 4. UsbNfcDevice Initialization

```
    usbNfcDevice = new UsbNfcDevice(MainActivity.this);
    usbNfcDevice.setCallBack(deviceManagerCallback);
```

Step 5. Add the card read callback and card read code

```
    // Device operation class callback
    private DeviceManagerCallback deviceManagerCallback = new
DeviceManagerCallback() {
        @Override
        // card found callback
        public void onReceiveRfnSearchCard(boolean blnIsSus, int cardType,
byte[] bytCardSn, byte[] bytCarATS) {
            super.onReceiveRfnSearchCard(blnIsSus, cardType, bytCardSn,
bytCarATS);
            if (!blnIsSus || cardType == UsbNfcDevice.CARD_TYPE_NO_DEFINE) {
                return;
            }

            System.out.println("Activity Received active card callback: UID->"
+ StringTool.byteHexToSting(bytCardSn) + " ATS->" +
StringTool.byteHexToSting(bytCarATS));

            final int cardTypeTemp = cardType;
            new Thread(new Runnable() {
                @Override
                public void run() {
                    boolean isReadWriteCardSuc;
                    try {
                        isReadWriteCardSuc = readWriteCardDemo(cardTypeTemp);

                        // Turn on the buzzer to indicate that the card read is
finished
                        if (isReadWriteCardSuc) {
                            usbNfcDevice.openBeep(50, 50, 3);  //fast beeper 3
times when the card is successfully read and wrote
                        }
                        else {
                            //usbNfcDevice.openBeep(100, 100, 2); //slowly
beeper 2 times when failed to read and write the card
                            //fail read will turn off the antenna at once and tries
read automatically
                            usbNfcDevice.closeRf();
                        }
                    } catch (DeviceNoResponseException e) {
                        e.printStackTrace();
                    }
```

```
            }
        }).start();
    }
};

//Read Write Card Demonstration
private synchronized boolean readWriteCardDemo(int cardType) {
    if (cardType == DeviceManager.CARD_TYPE_ISO4443_B) {  // Find B cpu card,
ID card
        final Iso14443bCard iso14443bCard = (Iso14443bCard)
usbNfcDevice.getCard();
        if (iso14443bCard != null) {
            SamVIdCard samVIdCard = new SamVIdCard(usbNfcDevice);
            idCard = new IDCard(samVIdCard);

            int cnt = 0;
            do {
                try {
                    /**
                     * Get ID card data
                     * Note: This method is a synchronous blocking method,
and it will takes a couple time to return the ID card data, during that time,
the ID card cannot leave the reader!
                     */
                    IDCardData idCardData = idCard.getIDCardData();

                    /**
                     * Display ID card data
                     */
                    showIDCardData(idCardData);

                    //return read successfully
                    return true;
                } catch (DKCloudIDException e) {   // The server returns an
exception, repeats the parsing 5 times
                    e.printStackTrace();
                }
                catch (CardNoResponseException e) {    //Card read Abnormal
to exit, need retry again
                    e.printStackTrace();
                    return false;
                }
            }while ( cnt++ < 5 );  // If the server returns an exception, will
retry read 5 times until successful
```

```
            }
        }
        else if ( cardType == DeviceManager.CARD_TYPE_MIFARE ) {   //Find Mifare
card
                final Mifare mifare = (Mifare) usbNfcDevice.getCard();
                if (mifare != null) {
                    msgBuffer.delete(0, msgBuffer.length());
                    msgBuffer.append("Find Mifare card
->UID:").append(mifare.uidToString()).append("\r\n");
                    msgBuffer.append("Start verifying block 1 key\r\n");
                    handler.sendEmptyMessage(0);
                    byte[] key = {(byte) 0xff, (byte) 0xff, (byte) 0xff, (byte)
0xff, (byte) 0xff, (byte) 0xff};
                    try {
                        boolean anth = mifare.authenticate((byte) 1,
Mifare.MIFARE_KEY_TYPE_A, key);
                        if (anth) {
                            msgBuffer.append("Key verification succeeded\r\n");
                            msgBuffer.append("Write
00112233445566778899001122334455 to block 1\r\n");
                            handler.sendEmptyMessage(0);
                            boolean isSuc = mifare.write((byte)1, new
byte[]{0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, (byte) 0x88, (byte) 0x99,
0x00, 0x11, 0x22, 0x33, 0x44, 0x55});
                            if (isSuc) {
                                msgBuffer.append("Write successfully! \r\n");
                                msgBuffer.append("Read the block 1 data\r\n");
                                handler.sendEmptyMessage(0);
                                byte[] readDataBytes = mifare.read((byte) 1);
                                msgBuffer.append("Block 1
data:").append(StringTool.byteHexToSting(readDataBytes)).append("\r\n");
                                handler.sendEmptyMessage(0);
                            } else {
                                msgBuffer.append("Write fail! \r\n");
                                handler.sendEmptyMessage(0);
                                return false;
                            }
                        }
                        else {
                            msgBuffer.append("Key verification failure\r\n");
                            handler.sendEmptyMessage(0);
                            return false;
                        }
                    } catch (CardNoResponseException e) {
```

```
                    e.printStackTrace();
                    return false;
                }
            }
        }
        else if ( cardType == DeviceManager.CARD_TYPE_ULTRALIGHT ) { //Find
Ultralight card
            String writeText = System.currentTimeMillis() + " Professional
contactless smart card reader solution provider! ";
            if (msgText.getText().toString().length() > 0) {
                writeText = msgText.getText().toString();
            }

            msgBuffer.delete(0, msgBuffer.length());

            final Ntag21x ntag21x = (Ntag21x) usbNfcDevice.getCard();
            if (ntag21x != null) {
                msgBuffer.delete(0, msgBuffer.length());
                msgBuffer.append("Find Ultralight Card
->UID:").append(ntag21x.uidToString()).append("\r\n");
                handler.sendEmptyMessage(0);
                try {
                    //Read and write a single block Demo
                    msgBuffer.append("Start to read the data of block0:\r\n");
                    handler.sendEmptyMessage(0);
                    byte[] readTempBytes = ntag21x.read((byte) 0);
                    msgBuffer.append("Return:
").append(StringTool.byteHexToSting(readTempBytes)).append("\r\n");
                    handler.sendEmptyMessage(0);
                    //Read and write demo of any length, with progress
callback
                    showReadWriteDialog("writing data", 1);
                    showReadWriteDialog("writing data", 1);
                    byte[] writeBytes = new byte[888];
                    Arrays.fill(writeBytes, (byte) 0x30);
                    msgBuffer.append("Start writing 888 bytes of data:
0x30").append("\r\n");
                    handler.sendEmptyMessage(0);
                    boolean isSuc =
ntag21x.longWriteWithScheduleCallback((byte) 4, writeBytes, new
Ntag21x.onReceiveScheduleListener() {
                        @Override
                        public void onReceiveSchedule(int rate) {
                            showReadWriteDialog("writing data", rate);
```

```
                        }
                    });
                    if (isSuc) {
                        msgBuffer.append("Write data successfully!
").append("\r\n");
                        handler.sendEmptyMessage(0);
                        msgBuffer.append("Start writing 888 bytes of
data").append("\r\n");
                        handler.sendEmptyMessage(0);
                        readTempBytes =
ntag21x.longReadWithScheduleCallback((byte) 4, (byte) (888 / 4), new
Ntag21x.onReceiveScheduleListener() {
                            @Override
                            public void onReceiveSchedule(int rate) {
                                showReadWriteDialog("Reading data", rate);
                            }
                        });
                        msgBuffer.append("Read successfully:
\r\n").append(StringTool.byteHexToSting(readTempBytes)).append("\r\n");
                        showReadWriteDialog("Reading dat", 100);
                    }
                    else {
                        msgBuffer.append("Date write fail!").append("\r\n");
                        showReadWriteDialog("Reading data", 0);
                    }
                } catch (CardNoResponseException e) {
                    e.printStackTrace();
                    msgBuffer.append(e.getMessage()).append("\r\n");
                    showReadWriteDialog("Writing data", 0);
                    return false;
                }
            }
        }

        return false;
    }
```